# UNIVERSITÉ FRANÇOIS RABELAIS DE TOURS

*École Doctorale MIPTIS*

*Laboratoire d'Informatique EA 6300*

**THÈSE** présentée par :

**Aymen CHERIF**

soutenue le : **17 juillet 2013**

pour obtenir le grade de : **Docteur de l'université François - Rabelais de Tours**

Discipline/ Spécialité : INFORMATIQUE

# Réseaux de neurones, SVM et approches locales pour la prévision des séries temporelles

**THÈSE DIRIGÉE PAR :**

M. CARDOT Hubert — Professeur, Université François Rabelais Tours

**RAPPORTEURS :**

M. GALLINARI Patrick — Professeur, Université Pierre et Marie Curie à Paris
M. SALOTTI Jean-Marc — Professeur, Institut Polytechnique de Bordeaux

**JURY :**

M. BONÉ Romuald — Professeur, Ecole Nationale d'Ingénieurs du Val de Loire à Blois
M. CARDOT Hubert — Professeur, Université François Rabelais Tours
M. CRUCIANU Michel — Professeur, Conservatoire National des Arts et Métiers à Paris
M. GALLINARI Patrick — Professeur, Université Pierre et Marie Curie à Paris
M. SALOTTI Jean-Marc — Professeur, Institut Polytechnique de Bordeaux

# Remerciements

Pour réaliser ce document et le travail qu'il présente, j'ai largement bénéficié de l'aide de nombreuses personnes. Je tiens à les remercier très sincèrement.

Je tiens avant tout à remercier mes directeurs de thèse Hubert CARDOT et Romuald BONÉ. Travailler avec vous a été enrichissant, merci de m'avoir encouragé durant toute ma thèse. Hubert pour ton aide précieuse sur le plan scientifique et administratif, merci pour ta patience et ta disponibilité. Romuald, merci pour toute l'aide que tu m'as apportée, pour ton enthousiasme permanent et ton incroyable disponibilité.

Je remercie Patrick GALLINARI et Jean-Marc SALOTTI pour l'honneur qu'ils me font en acceptant de rapporter sur mes travaux. Que Michel CRUCIANU trouve également l'expression de ma gratitude pour sa participation à mon jury.

Ce travail a été mené au sein de l'équipe Reconnaissance des Formes et Analyse d'Images, qui fait partie du laboratoire d'informatique de l'université de Tours, et s'est déroulé en majeure partie au département d'informatique de Polytech'Tours. Je suis très reconnaissant aux membres de cette équipe et de ce laboratoire pour leurs précieux conseils tout au long des années que nous avons passées ensemble.

Pour finir, je voudrais remercier les membres de ma famille (ma mère Safia, mon père Abdel Hamid et mon frère Mehdi) pour leurs soutiens sans failles qu'ils m'ont apportés tout au long de ces années.

# Résumé

La prévision des séries temporelles est un problème qui est traité depuis de nombreuses années. On y trouve des applications dans différents domaines tels que : la finance, la médecine, le transport, etc. Dans cette thèse, on s'est intéressé aux méthodes issues de l'apprentissage artificiel : les réseaux de neurones et les SVM. On s'est également intéressé à l'intérêt des méta-méthodes pour améliorer les performances des prédicteurs, notamment l'approche locale. Dans une optique de diviser pour régner, les approches locales effectuent le clustering des données avant d'affecter les prédicteurs aux sous ensembles obtenus. Nous présentons une modification dans l'algorithme d'apprentissage des réseaux de neurones récurrents afin de les adapter à cette approche. Nous proposons également deux nouvelles techniques de clustering, la première basée sur les cartes de Kohonen et la seconde sur les arbres binaires.

**Mots clés :** Réseaux de neurones, Perceptron multi-couche, réseaux de neurones récurrents, SVM (Support Vector Machines), prédiction des séries temporelles, régression, apprentissage artificiel supervisé et non supervisé.

# Abstract

Time series forecasting is a widely discussed issue for many years. Researchers from various disciplines have addressed it in several application areas : finance, medical, transportation, etc. In this thesis, we focused on machine learning methods : neural networks and SVM. We have also been interested in the meta-methods to push up the predictor performances, and more specifically the local models. In a divide and conquer strategy, the local models perform a clustering over the data sets before different predictors are affected into each obtained subset. We present in this thesis a new algorithm for recurrent neural networks to use them as local predictors. We also propose two novel clustering techniques suitable for local models. The first is based on Kohonen maps, and the second is based on binary trees.

**Keywords :** Neural networks, multi layer perceptron, recurrent neural networks, SVM (Support Vector Machines), time series forecasting, regression, machine learning, supervised learning, unsupervised learning.

# Table des matières

# Liste des tableaux

# Table des figures

# Chapitre 1

# Introduction

La prévision des séries temporelles est un problème qui a été abordé depuis longtemps. On en trouve des applications dans de nombreux domaines, par exemple l'économie, la météorologie, la médecine, la communication, le traitement de la parole, etc. Théoriquement, la prévision des séries temporelles nécessite de modéliser le système qui a généré les données de la série. En disposant d'un système d'équations mathématiques et déterministes et connaissant les conditions initiales, il serait possible de prévoir l'évolution du système. Cependant, les caractéristiques du phénomène qui a généré la série sont généralement inconnues. Les seules informations disponibles dans ce cas de figure sont les valeurs passées. La modélisation de la série se limite donc à imiter le mécanisme de génération des données sans retrouver explicitement les fonctions qui représentent ces mécanismes et ce uniquement à travers les valeurs passées.

Les problèmes que nous traiterons dans cette thèse concernent les séries mono-variées, c'est-à-dire que les séries sont composées d'une seule variable en entrée. En sortie, nous attendons du système une variable unique qui représente une estimation de la prévision selon un horizon $h$ à partir de l'instant de la valeur d'entrée. En pratique le système de prévision que nous modélisons prend M valeurs en entrées qui représentent des valeurs passées. La forme la plus répandue et celle qu'on utilisera dans cette thèse, appelée fenêtre temporelle, consiste à prendre en entrée une fenêtre temporelle des M vecteurs consécutifs pour produire une estimation des valeurs futures à un horizon de prévision $h$.

$$\hat{x}(t+h) = f((x(t), x(t-1), .., x(t-M+1))) \tag{1.1}$$

où $\hat{x}(t+h)$ représente la valeur de prévision à horizon $h$. $f$ représente la fonction du modèle de prévision et prend en entrée la fenêtre temporelle.

Dans cette thèse, on s'intéressera aux modèles adaptatifs issus du domaine de l'apprentissage automatique qui permettent d'apprendre les paramètres de la fonction $f$ à partir des exemples d'apprentissage. On retrouvera dans cette thèse trois méthodes connues : les réseaux de neurones à propagation avant, les réseaux de neurones récurrents et les *support vector regression*.

Les réseaux de neurones à propagation avant sont des méthodes d'inspiration biologique. En matière de chronologie, ils sont les premiers à être utilisés dans ce genre de problématiques (parmi les trois méthodes citées). Le réseau est constitué de neurones artificiels qui reçoivent des signaux, les modifient et les transmettent. Ces signaux peuvent provenir soit de neurones prédécesseurs soit directement des entrées, la sortie des neurones peut être soit vers d'autres neurones successeurs ou à l'extérieur du système. On parlera donc d'une architecture en couches, avec une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie qui dans notre cas contient un neurone unique.

Les réseaux de neurones récurrents sont une amélioration et sont caractérisés par la présence de cycles dans le graphe d'interconnexion. À la différence des architectures précédentes, le passage de l'information d'un neurone à l'autre n'est plus instantané. La présence de boucles permet une influence à plus long terme des signaux et permet aux réseaux de se passer de l'utilisation de la fenêtre temporelle.

Le troisième prédicteur utilisé dans cette thèse est le *support vector regression*. Il s'agit d'une adaptation de la méthode plus connue pour la classification de données. L'idée des support vector machines est de déterminer un séparateur linéaire qui divise l'espace en deux en passant par un espace de dimension supérieure : l'espace des exemples positifs et l'espace des exemples négatifs. L'adaptation des *support vector regression* remplace l'idée du séparateur par une courbe qui regroupe le maximum d'exemples autour d'une marge définie en utilisant la même technique de passage par un espace de dimension supérieure.

Nous verrons au cours de cette thèse qu'il existe plusieurs applications des méthodes citées ci-dessus pour la prévision des séries temporelles. Il existe également de nombreuses propositions d'améliorations et d'adaptations de ces méthodes dans des cas plus spécifiques et ou pour les rendre soit plus robustes soit plus génériques. L'objectif commun dans toutes ces propositions est de rendre l'erreur de prévision de plus en plus faible.

Parmi les techniques les plus performantes de la littérature, certaines peuvent être qualifiées de méta-méthodes. Il s'agit d'une utilisation bien spécifique des données pouvant être applicable avec n'importe quels prédicteurs et permettant à ces derniers de se renforcer. Un exemple des plus connus est la technique de boosting qui réitère le processus d'apprentissage en sélectionnant les données selon les résultats des prévisions des itérations précédentes. En donnant plus de probabilités d'être pris en compte aux exemples les plus difficiles à apprendre, cette technique a pour effet de renforcer le prédicteur, d'où le nom de boosting.

Notre attention se focalisera sur une autre méthode qu'on peut qualifier de méta-méthode, connue sous le nom de l'approche locale. Nous la considèrerons comme une méta-méthode puisqu'il s'agit d'un principe général qui opère sur les données et qui permet d'utiliser en théorie n'importe quel prédicteur. Le principe est de décomposer les données en sous-ensembles de données qu'on nommera *clusters* et dont les exemples au sein d'un même cluster sont les plus similaires possible. Ceci a pour conséquence de simplifier la tâche de

chaque prédicteur dédié à un cluster et théoriquement de permettre d'améliorer la précision de la prévision.

Nous avons remarqué à partir de l'état de l'art que les clusters sont le plus souvent formés par l'algorithme des cartes de Kohonen. Nous partirons de ce constat pour essayer d'adapter les réseaux de neurones récurrents à l'approche locale et évaluer l'intérêt de cette intégration. Nos expérimentations nous mèneront à constater l'importance du paramètre du nombre de clusters dans une telle approche. Cependant ce paramètre reste difficile à prévoir d'avance d'une série à une autre. Les méthodes actuelles proposées dans l'état de l'art sont incapables de faciliter la détermination du nombre de clusters adéquats pour un jeu de données.

Dans ce cadre nous proposons une adaptation des cartes de Kohonen afin de retrouver de manière automatique un clustering qui favorise une certaine distribution des données sur carte. Il s'agit de réitérer l'algorithme de clustering des cartes de Kohonen jusqu'à obtenir des groupements plus au moins homogènes, ce qui permet de réduire le nombre de clusters à quelques classes. La plus grande faiblesse de cette approche provient du fait que le clustering est indépendant des performances des prédicteurs. Pour cela nous proposons une approche qui prend en considération les résultats des prévisions locales lors de la formation des clusters. Il s'agit d'une approche hiérarchique de clustering construisant un arbre binaire et dont l'élagage est fait à partir des résultats obtenus dans chacun des sous-ensembles.

Ce manuscrit s'organise en 6 chapitres comme suit :

Le **chapitre 2** dresse un panorama des méthodes d'apprentissage artificiel les plus utilisées pour la prédiction des séries temporelles ainsi que des différentes variantes de l'état de l'art. Nous présenterons d'abord les réseaux de neurones à propagation avant, puis les réseaux de neurones récurrents et enfin les *support vector regression*. Le point commun de ces prédicteurs est qu'ils prennent en considération la globalité des données d'apprentissage. Pour cette raison les méthodes et les variantes présentées dans ce chapitre sont désignées comme étant des approches globales.

Le **chapitre 3** s'intéresse aux méthodes qui mettent en œuvre les prédicteurs vus dans le premier chapitre en les incluant dans des approches plus globales. Nous nous focaliserons sur les approches locales dans le sens où les prédicteurs utilisés sont paramétrés sur des sous-ensembles particuliers des données originales. Dans ce chapitre, nous partons d'une description des algorithmes de boosting tout en les comparant avec les approches locales qui utilisent le clustering des données. Nous établirons un état de l'art en prenant en compte le clustering partitionnel et le clustering hiérarchique. On établira aussi des comparaisons entre les différentes méthodes en matière d'efficacité.

Le **chapitre 4** présente les trois méthodes proposées dans cette thèse ainsi que des résultats préliminaires permettant de vérifier de premières hypothèses. L'intégration des RNN dans l'approche locale est présentée puis évaluée. Ensuite, la méthode de clustering auto-SOM mettant en œuvre un algorithme de carte de Kohonen adaptatif afin de trouver

une disposition de carte la plus adéquate est décrite puis évaluée. Enfin, nous présentons l'approche hiérarchique dont le principe est de réaliser un clustering en suivant un arbre binaire. C'est une approche qui reprend l'intérêt des méthodes précédentes tout en évitant les inconvénients remarqués dans les expérimentations préliminaires.

Le **chapitre 5** met en œuvre des expérimentations approfondies et qui sont faites sur les séries standard (taches solaires, Laser et MG-17). Dans ce chapitre, nous explorerons l'impact de certains paramètres les plus importants comme le nombre de clusters, la taille de la fenêtre temporelle et l'horizon de prévision. Pour chacune des méthodes, les meilleurs résultats obtenus sont comparés à ceux de l'état de l'art.

Le **chapitre 6** est une conclusion qui présente une synthèse des résultats de nos expérimentations et met en évidence les apports de notre travail. De nouvelles perspectives de recherche sont proposées.

# Chapitre 2

# Apprentissage et prévision des séries temporelles

## Introduction

Dans ce chapitre nous nous pencherons sur les méthodes les plus connues conçues pour la prévision des séries temporelles. Cette étude comportera des modèles et des algorithmes d'apprentissage provenant de différentes familles. On s'intéressa plus particulièrement aux réseaux de neurones artificiels ainsi qu'aux Support Vector Regression. Nous verrons qu'il existe de nombreuses optimisations et adaptations de ces méthodes.

## 2.1 Prévision des séries temporelles par apprentissage artificiel

L'apprentissage artificiel, en anglais *Machine Learning* (ML), est une branche de l'intelligence artificielle qui peut être mise à contribution pour appréhender les problèmes de prédictions [Bishop, 2006], [Mitchell, 1997]. On peut classer les méthodes d'apprentissage artificiel en deux grandes familles : l'apprentissage non supervisé et l'apprentissage supervisé. La prévision des séries temporelles est classiquement réalisée par la deuxième famille d'algorithmes mais nous verrons dans la suite que l'utilisation de l'apprentissage non supervisé est aussi possible.

Dans l'apprentissage non supervisé, l'algorithme va lui-même, sans information supplémentaire, catégoriser les variables d'entrées. Ce type d'apprentissage permet l'élaboration d'une représentation interne de l'espace des données d'entrée en identifiant une structure statistique sous-jacente des variables sous une forme plus ou moins simple.

De façon générale, l'apprentissage artificiel supervisé consiste à créer un modèle de prédiction (classification ou prévision) à partir d'une base d'apprentissage comprenant les exemples d'entrée ainsi que les sorties désirés associées. Les paramètres du modèle vont ainsi s'adapter en comparant à chaque fois les sorties obtenues et les sorties désirées, d'où l'appellation supervisé [Mitchell, 1997], [Nilsson, 2004]. Une fois le modèle obtenu par une

base d'apprentissage, l'utilisation d'une base de test, comprenant des nouveaux exemples non utilisés pendant l'apprentissage, permet de mesurer les performances de la méthode. Une des possibilités est de calculer l'Erreur Quadratique Moyenne, en anglais *Mean Square Error* (MSE) (voir équation 2.1).

L'algorithme d'apprentissage permet de "*prédire*" une valeur cible étant donnée une ou des valeurs d'entrées. Dans le cas où cette valeur cible est discrète (dans un ensemble fini), la tâche réalisée par l'algorithme est appelée classification supervisée puisqu'il s'agit de trouver la classe correspondant à un exemple donné en entrée. Quand la valeur cible appartient à un ensemble continu (par exemple $\mathbb{R}$ ou $[0, 1]$), la tâche est appelée régression. Elle représente le plus souvent la prévision d'une ou de plusieurs valeurs futures correspondant à une suite de valeurs passées. Dans cette thèse, on s'intéressera uniquement à cette dernière tâche, à savoir la prévision des séries temporelles.

Soit $E = \{(\mathbf{X}(t), y(t)) \in \mathbb{R}^d \times \mathbb{R}/0 \leq t \leq N\}$ l'ensemble de données qui peut être une base d'apprentissage ou une base de test. $\mathbf{X}(t)$ est le vecteur d'entrée à l'instant $t$, $\mathbf{X}(t) \in \mathbb{R}^d$. $y(t)$ représente la valeur cible correspondant à $\mathbf{X}(t)$ et $N$ le nombre d'exemples dans la base. Le but de l'algorithme d'apprentissage est de trouver une fonction $f$, la représentation mathématique du modèle obtenu, qui soit le plus proche possible de la fonction cible $\tilde{f}$. On notera que $\tilde{f}$ reste toujours inconnue et dans de nombreux cas hypothétique. Le système pourra être évalué à la fin par l'erreur MSE :

$$MSE = 1/N \sum_{t=0}^{N} (\hat{y}(t) - y(t))^2 \tag{2.1}$$

où $\hat{y}(t)$ est la valeur prédite c'est-à-dire $f(\mathbf{X}(t)) = \hat{y}(t)$ l'approximation de la valeur $y(t)$. Pour répondre à ce problème, plusieurs méthodes ont été développées. Une méthode peut être décrite par un modèle dans lequel on définit la fonction $f$ et un algorithme d'apprentissage par lequel cette fonction a été déterminée. On s'intéressera dans cette thèse à trois types de méthodes : les réseaux de neurones à propagation avant, les réseaux de neurones récurrents et les *support vector machines*.

Les Réseaux de Neurones à Propagation Avant sont des réseaux à deux ou plusieurs couches. La caractéristique principale de ces réseaux est que l'information traverse dans un sens unique des entrées vers la sortie. Cette famille de réseaux sera détaillée dans la section 2.4. Les Réseaux de Neurones Récurrents [Rumelhart *et al.*, 1986], [Haykin, 1999] et [Boné, 2000] sont, dans le cas général, des architectures sans couches (mais on peut tout à fait imaginer des architectures récurrentes assez proches des architectures à couches [Boné, 2000]) ; leur caractéristique principale est la possibilité d'avoir des boucles dans le graphe d'interconnexions. Cette famille d'architectures sera étudiée dans la section 2.5. Les *Support Vector Machines* (SVM) [Vapnik, 2000] représentent une autre méthode avec une famille d'algorithmes d'apprentissage différente des deux précédentes. Les *Support Vector Regression* (SVR) [Smola et Schölkopf, 2004] sont une adaptation de l'algorithme SVM au problème de régression dont le principe est de remplacer l'hyperplan séparateur à marge par un tube ou tunnel qui essaie au mieux d'englober le maximum de points de la série temporelle [Burges, 1998]. Cette famille d'algorithmes sera détaillée dans la section 2.6.

Avant de présenter ces méthodes, nous aborderons brièvement les méthodes statistiques

de modélisation des séries temporelles et en particulier les modèles linéaires. Ils représentent les premières applications pour la prévision des séries temporelles, et nous serviront plus tard comme une introduction aux modèles connexionnistes.

## 2.2 Prévision des séries temporelles par les méthodes statistiques

Il existe plusieurs méthodes qui abordent le problème de prévision des séries temporelles par des approches statistiques. Ces dernières reposent sur une modélisation mathématique de la série. Nous pouvons citer les travaux de Box & Jenkins [Box *et al.*, 1994] et [Hamilton, 1994] comme des références pour ces méthodes. Comme elles ne feront pas l'objet de notre étude, nous nous contenterons de les évoquer avec des références à des travaux qui leur sont associés. Nous évoquerons dans ce paragraphe les trois modèles linéaires qui nous serviront ultérieurement pour faire des comparaisons entre les résultats obtenus dans différentes méthodes.

### Le modèle Auto-Régressif

Le modèle Auto-Régressif, en anglais *Auto-Regressive* (AR), est l'un des premiers modèles utilisés. C'est un modèle linéaire [Box *et al.*, 1994] et [Hamilton, 1994] construit à partir d'une combinaison des valeurs passé situées dans une fenêtre temporelle à laquelle s'ajoute une erreur comme le montre l'équation (2.2).

$$x(t) = \sum_{i=1}^{p} w_i x(t-i) + \epsilon(t) \tag{2.2}$$

où $x(t)$ est la valeur de la série à l'instant $t$, $p$ est l'ordre du modèle, les $w_i$ sont les paramètres auto-régressifs et $\epsilon(t)$ représente le bruit blanc. On note le modèle auto-régressif d'ordre p AR(p).

### Le modèle Moyenne Mobile

Le modèle à moyenne mobile, en anglais *Moving Average* (MA), est aussi un modèle linéaire mais il se base sur le bruit blanc de la série. On appelle moyenne mobile d'ordre q, notée MA(q), le modèle défini par l'équation :

$$x(t) = \sum_{i=1}^{q} \theta_i \epsilon(t-i) + \epsilon(t) \tag{2.3}$$

où les $\theta_i$ sont les paramètres à moyenne mobile et les $\epsilon(t-i)$ sont les bruits blancs.

### Le modèle ARMA

Le modèle *ARMA* est une réunion des deux modèles précédents comme le montre l'équation (2.4)

$$x(t) = \sum_{i=1}^{p} w_i x(t-i) + \sum_{i=1}^{q} \theta_i \epsilon(t-i) + \epsilon(t) \tag{2.4}$$

Les modèles cités précédemment sont les plus basiques et les plus connus. On peut trouver dans la littérature des variantes de ces modèles. [Engle, 1982] introduit le modèle ARCH (*Auto-Regressive Conditional Heteroskedasticity*) qui a été généralisé par la suite par [Bollerslev, 1986], ou encore plus récemment dans une application liée aux réseaux informatiques [Anand, 2009] et dans une application de prédiction financière [Cheong, 2009]. Dans [Connor *et al.*, 1994], Connor utilise pour la prédiction robuste des séries temporelles une variante NARMA qui constitue une généralisation vers le cas non-linéaire récemment utilisée dans [Zemouri *et al.*, 2010].

Ces types de modèles sont, souvent appliqués appliqués dans le domaine financier. On peut citer [Engle, 1982], [Cheong, 2009], [Khashei *et al.*, 2009], [Khashei et Bijari, 2012] et d'autres encore. Dans [Chen *et al.*, 2001], Chen apporte un modèle de prévision de trafic routier. Dans [Dudul, 2005], ces types de modèle sont utilisés pour prédire une série synthétique.

## 2.3 Les Réseaux de Neurones Artificiels

Les réseaux de neurones artificiels, en anglais *Artificial Neural Networks* (ANN), constituent la première classe de méthodes qui nous intéressera dans cette thèse. Leur nom est dû aux neurones du cerveau humain. En effet, c'est la description des processus mentaux faite par les neurobiologistes qui est à l'origine des modèles théoriques des ANN. Ils sont aussi appelés réseaux connexionnistes ou réseaux neuromimétiques. Deux éléments caractérisent ces méthodes, une organisation appelée réseau comprenant un certain nombre d'automates aux fonctionnalités relativement simples appelés neurones. L'information se propage dans les réseaux sur des connexions pondérées par des paramètres souvent appelés poids. Le deuxième élément est l'algorithme d'apprentissage dont l'objectif est de faire évoluer les poids du réseau de neurones de manière à obtenir un comportement global intéressant. Dans cette section, nous passerons rapidement en revue les principes élémentaires des réseaux de neurones. Nous établirons aussi un lien entre les méthodes statistiques rencontrées dans 2.2 et ces méthodes connexionnistes.

### 2.3.1 Principes des Réseaux de Neurones Artificiels

Le neurone formel est l'entité de base qui compose un réseau de neurones. Ce paragraphe présente le modèle de neurones le plus souvent utilisé et qui sera considéré dans le reste de ce travail. Mathématiquement un neurone peut être représenté par une fonction à plusieurs variables avec une sortie unique. La figure 2.1 représente le schéma classique d'un neurone formel. Les valeurs $x_1, ..., x_n$ représentent les entrées du neurone $i$. Les valeurs $w_{ij}$ représentent les poids associés à chaque entrée $x_j$ avec $j \in \{1, .., n\}$.

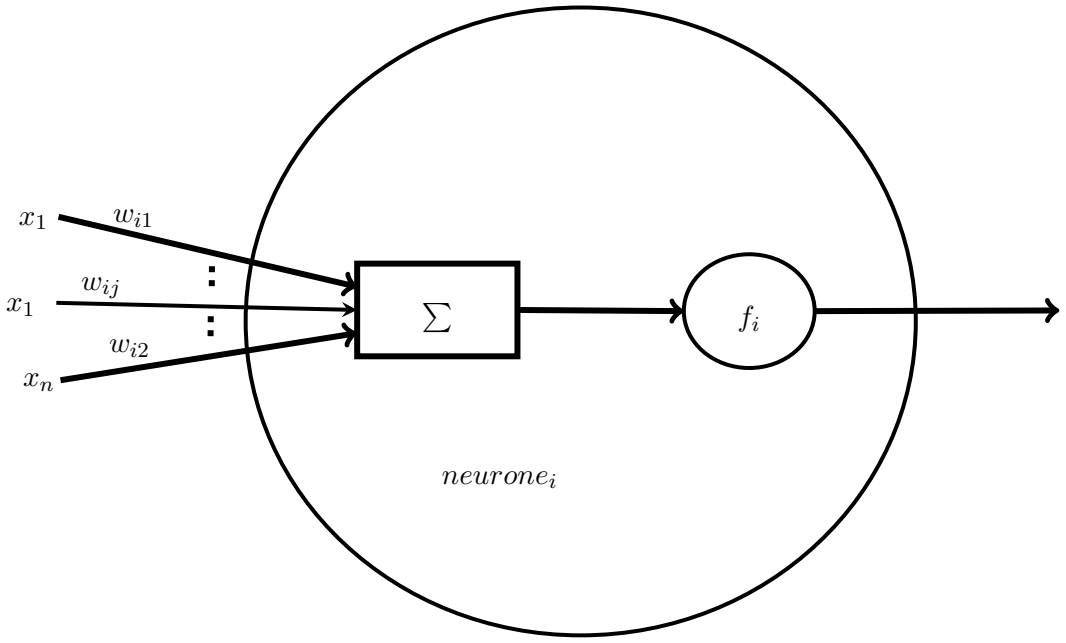


FIGURE 2.1 – architecture d'un neurone formel

Un neurone peut être également caractérisé par quatre aspects :
- La nature des entrées qui peut être soit binaire soit réelle ; dans notre cas ce sont toujours des réels.
- La fonction de sommation des entrées qui peut être linéaire ou non linéaire. En pratique, ainsi que dans nos expérimentations, c'est une combinaison pondérée par les poids $w_{ij}$ des entrées qui est utilisée.
- La nature de la fonction de transfert utilisée (linéaire ou non-linéaire). En pratique, on trouve des fonctions à seuil (équation 2.5) ou des fonctions sigmoïdes (équation 2.6) ou encore des fonctions gaussiennes (équation 2.7).
- La nature de la sortie (binaire ou réelle). Dans notre cas, elle sera toujours réelle.

$$\text{Fonction à seuil : } f(x) = \begin{cases} +1 & si\ x \geq a \\ -1 & sinon \end{cases} \tag{2.5}$$

$$\text{Fonction sigmoide : } f(x) = \frac{1}{1 + e^{(-ax)}} \tag{2.6}$$

$$\text{Fonction gaussienne : } f(x) = e^{(\frac{-ax^2}{2})} \tag{2.7}$$

Nous pouvons remarquer que la fonction de transfert sigmoïde possède un régime quasi-linéaire au voisinage de 0 et nonlinéaire ailleurs. Cette particularité offre au réseau la possibilité de s'adapter aussi bien aux problèmes linéaires qu'aux problèmes non linéaires.

### 2.3.2 Les connexions entre les neurones

Les connexions permettent de construire le réseau de neurones en liant les différentes entités ensemble. La plus simple et la plus classique des connexions est la connexion unidirectionnelle pondérée. Il s'agit d'un arc unidirectionnel pondéré par un poids $w_{ij}$ qui relie le neurone $j$ au neurone $i$ et qui correspond au poids utilisé dans la fonction de sommation (voir paragraphe 2.3.1). Ces connexions simples sont représentées dans la figure 2.2.

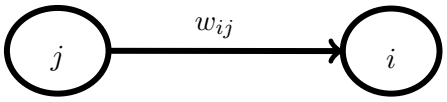


FIGURE 2.2 – Connexion simple entre deux neurones

Cependant, d'autres types de connexions plus complexes existent et ont déjà été utilisées dans le cadre de variantes de réseaux de neurones classiques (voir section 2.4.1.3). Parmi eux, on peut citer les connexions FIR et IIR qui vont être décrites ci-après.

#### Les connexions Finite Impulse Response (FIR)

Comme le montre la figure 2.3, il s'agit d'utiliser des blocs de retard. Un bloc de retard permet d'introduire un retard d'un pas de temps dans la propagation de l'information. Autrement dit, l'information ne se transmet pas aussitôt et met un pas de temps pour être transmise. Cumuler les blocs de retard permet de simuler une fenêtre temporelle à l'entrée d'un neurone.

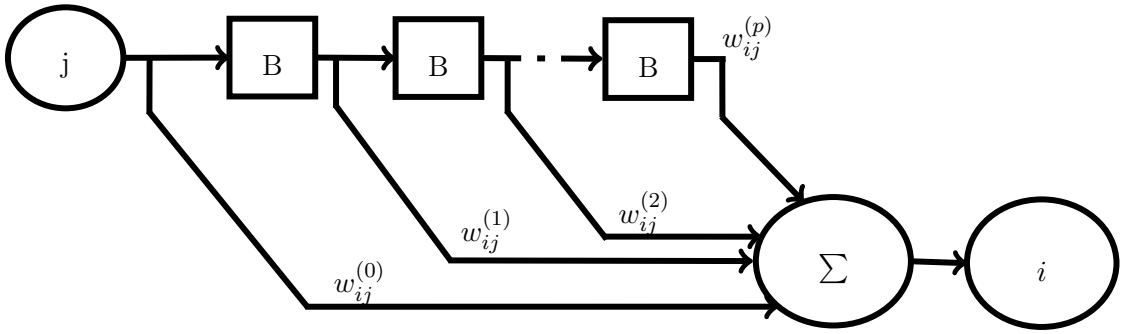


FIGURE 2.3 – Connexion FIR entre deux neurones

Ainsi une connexion FIR permet d'obtenir $p$ retards, à chaque retard correspond un poids $w_{ij}^{(r)}$, où $r$ est le degré du retard c'est-à-dire le nombre de blocs utilisés. ($1 \leq r \leq p$ ).

#### Les connexions Infinite Impulse Response (IIR)

Une connexion IIR est une connexion FIR à laquelle on ajoute à la sortie de la connexion une autre connexion FIR bouclée, comme le montre la figure 2.4.

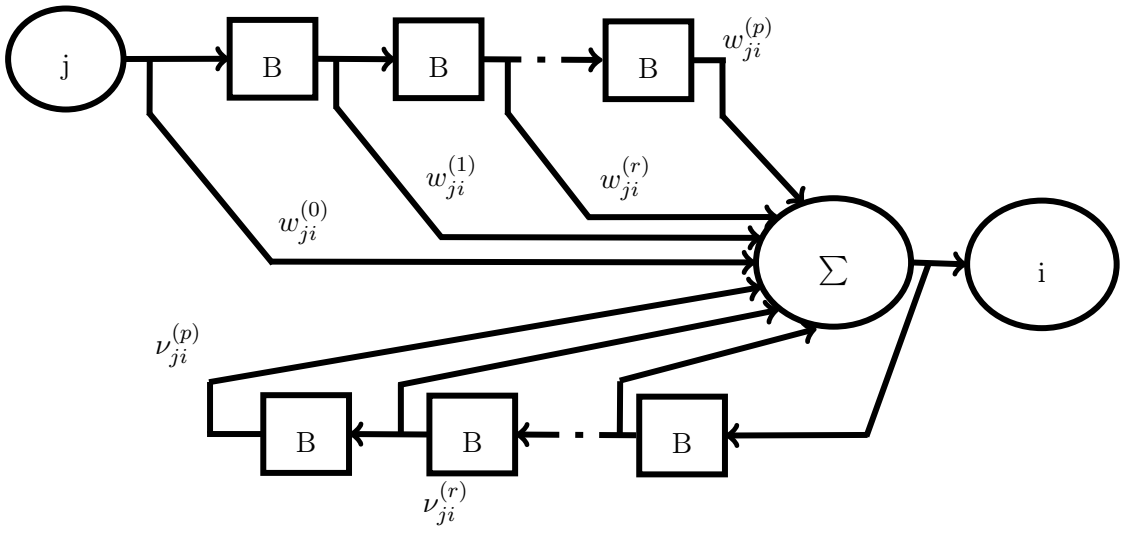


FIGURE 2.4 – Connexion IIR entre deux neurones

### 2.3.3 Les équivalents connexionistes des modèles statistiques

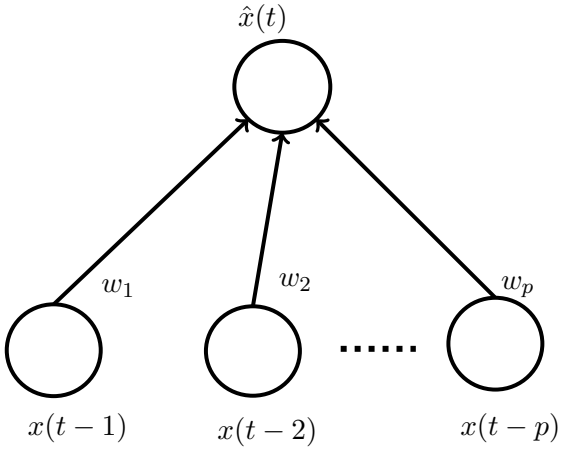


FIGURE 2.5 – architecture neuronale pour le modèle AR

Il est possible de traduire les modèles statistiques dans leurs équivalents connexionnistes [Connor *et al.*, 1994] et [Kouam et Fogelman, 1993]. Les architectures ainsi obtenues sont appelées Réseaux à Propagation Avant, en anglais *Feed Forward Neural Networks* (FFNN). Pour les modèles AR, les réseaux équivalents ressemblent à un réseau à propagation avant à deux couches, tandis que pour d'autres modèles comme le ARMA, les réseaux équivalents sont dans la famille des réseaux récurrents. La figure 2.5 représente l'architecture neuronale équivalente au modèle AR(p). Chaque neurone d'entrée est équivalent à une valeur de la fenêtre temporelle. La valeur prédite à l'instant $t-1$ est notée $\hat{x}(t)$ et elle est donnée par

l'équation (2.8).

$$\hat{x}(t) = \sum_{i=1}^{p} w_i x((t-i)) \tag{2.8}$$

L'erreur est définie par

$$e(t) = \hat{x}(t) - x(t) \tag{2.9}$$

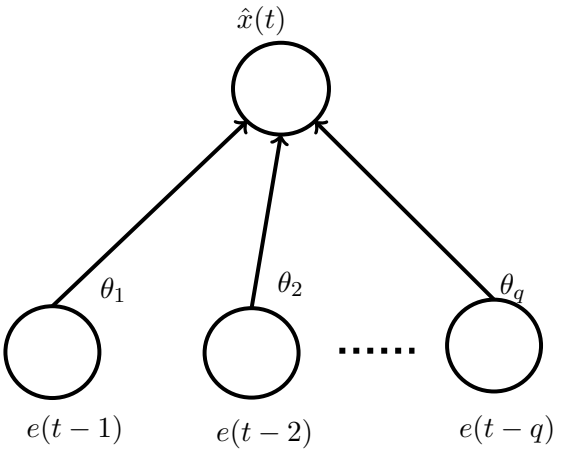


FIGURE 2.6 – architecture neuronale pour le modèle MA

Le modèle à moyenne mobile est défini par l'équation 2.3. En pratique, bien que $\epsilon(t) \neq e(t)$ puisque $e(t)$ est le reflet de la partie imprévisible de la série et de la limitation du modèle, les valeurs $\epsilon(t-i)$ sont remplacées par les résidus $e(t-i)$ pour estimer $\hat{x}(t+1)$, comme le montre l'équation 2.10. Le RNA équivalent est présenté dans la figure 2.6.

$$\hat{x}(t+h) = \sum_{i=0}^{q} \theta_i e(t-i) \tag{2.10}$$

L'architecture connexionniste équivalente au modèle ARMA ressemble aussi à une réunion des deux modèles précédents (voir figure 2.7), avec des entrées $e(t-i)$ placées par l'utilisateur et calculées avec $e(t) = \hat{x}(t) - x(t)$. D'après la figure 2.7, le réseau reçoit en entrée une fenêtre temporelle glissante qui contient les valeurs $x(t-1), x(t-2), .., x(t-p), e(t-1), e(t-2), .., e(t-q)$ tandis que le neurone de sortie fournit $\hat{x}(t)$.

Un autre modèle connexionniste a été proposé dans [Back et Tsoi, 1991] et dans [Crucianu, 1997]. Ces réseaux reposent sur deux neurones utilisant des connexions FIR (Finite Impulse Response) ou IIR (Infinite Impulse Response). Une connexion FIR est une sorte de méta-connexion contenant des connexions associées intégrant des retards consécutifs. Une connexion FIR permet de réaliser un modèle AR [Crucianu, 1997]. Une connexion IIR permet le bouclage local qui ajoute un comportement dynamique entre deux neurones. Une connexion IIR permet de mettre en œuvre le modèle ARMA [Back et Tsoi, 1991].

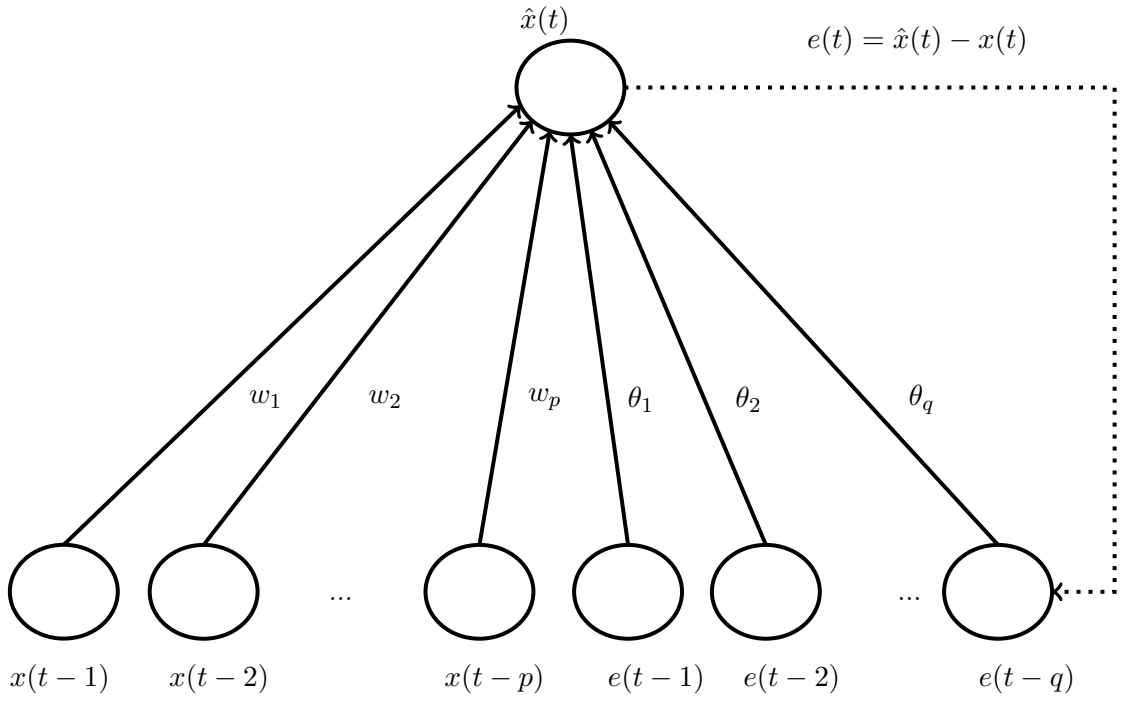


FIGURE 2.7 – architecture neuronale pour le modèle ARMA

## 2.4 Les Réseaux de Neurones à Propagation Avant

Nous avons vu ci-dessus des exemples de réseaux à propagation avant (section 2.3.3). La nécessité d'utilisation des FFNN nonlinéaires est venue naturellement de la limite des méthodes linéaires statistiques pour modéliser et prédire des séries nonlinéaires. Même en replaçant la fonction de transfert du neurone de sortie par une fonction nonlinéaire, ceci ne permet pas de dépasser efficacement cette limite. Obtenir un système totalement non linéaire se fait par l'ajout d'au moins une couche intermédiaire appelée généralement couche cachée.

Dans ce paragraphe, nous détaillerons principalement deux types de réseaux à propagation avant : le Perceptron Multi Couche, en anglais *Multi-Layer Perceptron* (MLP), et les Réseaux à Fonction à Base Radiale, en anglais *Radial Basis Function Networks* (RBFN).

### 2.4.1 Le perceptron multi couche

#### 2.4.1.1 Architecture de base

La figure 2.8 montre la composition générique d'un perceptron multi couche, en anglais *Multi-Layer Perceptron* (MLP). Dans ce type de réseaux, les neurones sont organisés de manière à constituer plusieurs couches. En pratique, le nombre de couches varie de 3 à 4 dont une couche d'entrée, une couche de sortie et une ou deux couches cachées.

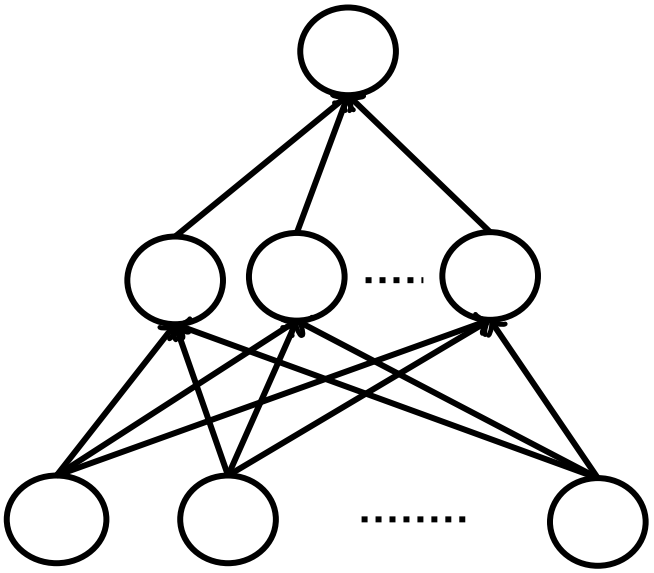

FIGURE 2.8 – Architecture générique d'un MLP

La couche d'entrée contient autant de neurones que la dimension de l'espace d'entrée. La couche de sortie contient un neurone unique pour la valeur de la prévision. Enfin, une ou plusieurs couches cachées contiennent un nombre variable de neurones.

#### 2.4.1.2 Caractéristiques de base

Du point de vue fonctionnel, tout comme les modèles linéaires, les MLP effectuent un certain nombre d'opérations algébriques. Grâce aux fonctions de transfert des neurones du réseau, les MLP réalisent une composition de fonctions sur les entrées. Ainsi, la sortie du réseau dépendra des entrées en cours et de la structure des connexions. La seule mémoire que possède un MLP est celle des exemples passés en entrée. On appellera fenêtre temporelle de dimension $d$, notée $\mathbf{x}(t)$, le vecteur colonne :

$$\mathbf{x}(t) = {}^{t}(x(t), x(t-1), ..., x(t-d)) \tag{2.11}$$

Il est mathématiquement démontré que les MLP possèdent un pourvoir d'approximation universelle [Cybenko, 1989]. Autrement dit, lorsque les données sont normalisées c'est-à-dire que la fonction objectif est comprise dans la boule unité, un MLP possédant une seule couche cachée est capable d'approximer n'importe quelle fonction mathématique. Il suffit d'avoir le nombre de neurones cachés suffisant. Cette caractéristique peut être suffisante pour expliquer le grand nombre de MLP à une seule couche cachée utilisée dans la littérature. Mais une autre explication est liée à de la nature de l'algorithme de rétropropagation du gradient utilisé. En effet, la valeur du gradient tend à disparaitre assez rapidement au fur et à mesure que l'on avance dans des couches profondes, rendant l'algorithme d'apprentissage inefficace ou trop lent.

#### 2.4.1.3 Variantes du MLP

La première application d'un MLP pour la prévision de séries temporelles remonte à [Lapedes et Farber, 1987]. La supériorité de tels réseaux sur les modèles statistiques a été démontrée à plusieurs reprises empiriquement, sur des données de différents types [De Almeida et Fishwick, 1991] [Sharda et Patil, 1992] [Hill *et al.*, 1996]. Dans [Weigend *et al.*, 1992] des expérimentations sont réalisées sur la série des taches solaires et sur des données financières. Dans [Tang et Fishwick, 1993], un *benchmark* est effectué sur les différents types de séries temporelles dont des séries à dépendances à long terme, des séries à changements de régime ainsi que des séries chaotiques. Il conclut que deux paramètres sont assez influents : la dimension d'entrée et le nombre de neurones cachés. Par expérimentation, la taille de l'ensemble d'apprentissage n'a pas d'impact majeur sur le résultat final. Les MLP de base ont été appliqués pour diverses problématiques de prévision. Citons la consommation électrique [Mori et Urano, 1996], certains phénomènes naturels (prédiction de débit d'eau dans des rivières, décomposition journalière de sédiments) [Atiya *et al.*, 1999] [Cigizoglu, 2004] ou encore le tourisme [Palmer *et al.*, 2006].

On retrouve aussi des variantes des MLP combinant la puissance de prédiction des réseaux et la clarté de modélisation des méthodes Box-Jenkins. On trouve, par exemple, dans [Zhang, 2003] un MLP combiné avec le modèle statistique ARIMA [Box *et al.*, 1994]. Le même principe est appliqué dans [Aburto et Weber, 2007] pour la prédiction des stocks des chaînes de production. Dans une application de prédiction météorologique, on retrouve dans [Niska *et al.*, 2005] une autre combinaison d'un MLP avec un modèle de prédiction météorologique (HIRLAM).

D'autres méthodes portent sur l'amélioration de l'algorithme d'apprentissage. Par exemple, avec des méthodes d'optimisation comme dans [Yudong et Lenan, 2009] en introduisant la technique *Bacterial chemotaxis optimization* (BCO) et dans [Wang et Lu, 2006] par les *Particle Swarm Optimization* (PSO). Dans [Cottrell *et al.*, 1995], un algorithme est proposé pour les MLP qui consiste à détecter et à éliminer les poids peu significatifs. [Wan, 1990] propose une adaptation de l'algorithme d'apprentissage afin d'intégrer les connexions à délais dans les MLP. Nous avons vu dans le paragraphe 2.3.2 que ce type de connexions ajoute une dimension temporelle au déroulement de l'algorithme. Dans [Duro et Reyes, 1999], une variante de l'algorithme MLP est proposée pour apprendre les délais.

La prise en compte des délais dans le réseau est une variante souvent proposée. Nous trouvons dans la littérature plusieurs travaux proposant d'intégrer dans l'architecture des réseaux de neurones des connexions à délais de type FIR ou IIR. Dans [Back *et al.*, 1995] est étudié l'impact d'ajout des connexions FIR sur les performances du MLP.

En termes de variante d'architecture, on trouve aussi des architectures à deux couches cachées bien qu'elles soient rares. Dans [Niska, 2004] des MLP à une et deux couches cachées ont été mis en œuvre pour la prédiction de la pollution atmosphérique. La méthode proposée permet la recherche d'une architecture optimale via un algorithme génétique.

### 2.4.2 Les Réseaux de Fonctions à Base Radiale

Les Réseaux de Fonctions à Base Radiale, en anglais *Radial Basis Function Network* (RBFN), sont des réseaux similaires aux MLP en termes d'architectures et d'algorithmes

d'apprentissage. La différence réside dans l'utilisation des fonctions de transfert à base radiale (gaussienne).

#### 2.4.2.1 Architecture de base

La figure 2.9 montre l'architecture générale d'un RBFN. Dans les neurones de la couche cachée sont généralement utilisées des fonctions de transfert gaussiennes qui jouent le rôle de fonction noyau de modèle. Dans les neurones de la couche d'entrée et la couche de sortie, le plus fréquent est d'utiliser les fonctions de transfert linéaires.

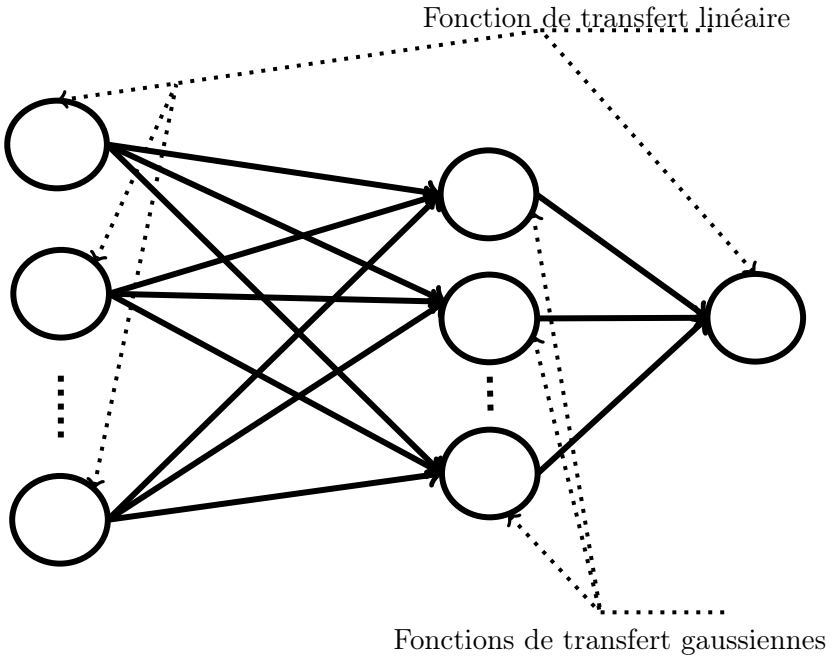


FIGURE 2.9 – architecture générique d'un RBFN

#### 2.4.2.2 Caractéristiques de base

Tout comme les MLP, les RBFN possèdent eux aussi la propriété d'approximation universelle [Haykin, 1999]. Les entrées sont propagées aux noyaux gaussiens $\Phi_j$ avec $j \in \{1, .., m\}$, $m$ étant le nombre de neurones cachés. Les valeurs du noyau sont ensuite pondérées par des coefficients $\lambda_j$ afin de produire la valeur de sortie $\hat{y}(t)$, approximation de la sortie désirée $y(t)$ :

$$\hat{y} = \sum_{j=1}^{m} \lambda_j \Phi_j(\mathbf{X}(t), c_j, \sigma_j) \tag{2.12}$$

Les noyaux gaussiens sont définis par :

$$\Phi_j(\mathbf{X}(t), c_j, \sigma_j) = \exp - \frac{\|\mathbf{X}(t) - c_j\|^2}{2\sigma_j} \tag{2.13}$$

Les paramètres $c_j$ et $\sigma_j$ désignent respectivement le centre et la largeur de la fonction gaussienne. L'intérêt principal provient du fait que les paramètres peuvent être appris indépendamment, avec un apprentissage relativement rapide pour les $\lambda_j$ comme solution d'un système d'équations linéaires. On peut trouver les détails de l'apprentissage du RBFN dans [Orr, 1998] et [Benoudjit et Verleysen, 2003]. En pratique, les paramètres $c_j$ influencent de façon relativement limitée le résultat final, alors que les paramètres $\sigma_j$ améliorent sensiblement l'approximation de la relation entrées-sortie. Théoriquement, il faut que les noyaux gaussiens soient assez larges et se recouvrent suffisamment pour que les parties de la distribution où aucun noyau n'est venu se placer soient prises en compte par le RBFN. En même temps, ils ne doivent pas être trop larges pour qu'ils ne recouvrent pas l'ensemble de la distribution. Les paramètres $\lambda_j$ sont uniques, une fois que $c_j$ et $\sigma_j$ sont fixés. Leur influence sur la sortie est directe, il s'agit du poids utilisé dans l'équation 2.12.

#### 2.4.2.3 Variantes du RBFN

On trouve dans la littérature de nombreuses applications directes des RBFN pour la prévision dans différents domaines [Benoudjit et Verleysen, 2003] [Kassam, 1993]. Par ailleurs, il est tout à fait possible de combiner de tels réseaux avec les modèles statistiques de séries temporelles comme dans le cas des MLP. Dans [Wedding et Cios, 1996], le RBFN donne en sortie la valeur de prédiction mais aussi une valeur de confiance. Les valeurs à faible confiance sont alors ignorées. Pour le reste, elles sont combinées avec les entrées dans un modèle de Box et Jenkins classique.

Dans [Harpham et Dawson, 2006], la fonction noyau classique gaussienne est remplacée par d'autres fonctions noyau. Une étude comparative est effectuée sur six autres fonctions noyaux différentes. Les résultats varient selon la nature des données. Les expérimentations ont été menées aussi bien sur des données réelles (Finance) que sur des données synthétiques (Mackey-Glass).

L'algorithme d'apprentissage est parfois amélioré par des algorithmes d'optimisation. Dans [Sheta et De Jong, 2001], un algorithme génétique est utilisé. [Rivas *et al.*, 2004] propose EvRBF où il introduit un algorithme évolutif dans l'apprentissage.

### 2.4.3 Algorithme d'apprentissage : cas du MLP

L'algorithme d'apprentissage le plus utilisé pour les FFNN ainsi que leurs variantes est l'algorithme de la Rétro-Propagation du gradient, en anglais *Back-Propagation* (BP) [Rumelhart *et al.*, 1986]. C'est un algorithme qui est souvent utilisé dans le cadre de nos expériences dont le principe est de rétro-propager le gradient de l'erreur de la couche de sortie vers l'entrée.

L'algorithme d'apprentissage peut s'appliquer de deux manières différentes : soit en considérant l'erreur instantanée $e(t)$, soit en considérant l'erreur globale $E(t_1, t_l)$ comprise entre les instants $t_1$ et $t_l$, avec :

$$E(t_1, t_l) = \sum_{t=t_1}^{t_l} e(t) \tag{2.14}$$

$e(t)$ est donnée généralement par la différence entre la sortie désirée et la sortie obtenue.

La première méthode nous donne un apprentissage temps réel dans lequel la modification de poids se fait à chaque exemple présenté

$$\Delta w_{ij}(t) = -\eta \frac{\delta e(t)}{\delta w_{ij}} \tag{2.15}$$

Dans la deuxième méthode, l'algorithme fonctionne en mode différé, c'est-à-dire que les poids sont modifiés à la fin de chaque cycle (quand tous les exemples de la base d'apprentissage ont été présentés)

$$\Delta w_{ij}(t_1, t_l) = -\eta \frac{\delta E(t_1, t_l)}{\delta w_{ij}} \tag{2.16}$$

Prenons le cas de l'apprentissage d'un MLP avec des connexions à délais (FIR) d'ordre $k$ avec $0 \leq k \leq p_i$. On suppose que l'apprentissage se fait avec l'algorithme BP en mode différé. On note $s_i(t)$ la sortie du neurone $i$,

$$s_i(t) = f_i(net_i(t)) \tag{2.17}$$

où $f_i$ est la fonction de transfert du neurone $i$ et $net_i$ est la valeur de sommation des entrées du neurone $i$ donnée par :

$$net_i(t) = \sum_{j \in Pred(i)} net_{ij}(t) \tag{2.18}$$

où $Pred(i)$ représente l'ensemble de neurones prédécesseurs du neurone $i$. Cette fois, $net_{ij}$ peut s'écrire en fonction des sorties des neurones prédécesseurs,

$$net_{ij}(t) = \sum_{k=0}^{p_i} w_{ij}^{(k)} s_j(t-k) \tag{2.19}$$

où $k$ représente le degré du retard (voir section 2.3.2) de la connexion entre deux neurones $i$ et $j$ (2.10).

Nous appliquons la descente du gradient sur la fonction d'erreur définie par l'équation 2.14. La variation totale des poids $w_{ij}$ est obtenue en appliquant une chaine de dérivation sur l'équation 2.16,

$$\frac{\delta E(t_1, t_l)}{\delta w_{ij}^{(k)}} = \sum_{t=t_1}^{t_l} \frac{\delta E(t_1, t_l)}{\delta s_i(t)} \frac{\delta s_i(t)}{\delta net_i(t)} \frac{\delta net_i(t)}{\delta w_{ij}^{(k)}} \tag{2.20}$$

On trouve facilement les deux derniers membres :

$$\frac{\delta s_i(t)}{\delta net_i(t)} = f_i'(net_i(t)) \tag{2.21}$$

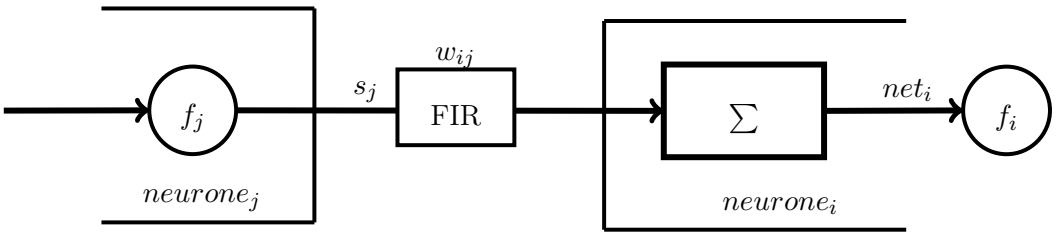


FIGURE 2.10 – Schéma de la connexion FIR simplifiée équivalente à celle de la figure 2.3 entre le neurone $i$ et son prédécesseur $j$

$$\frac{\delta net_i(t)}{\delta w_{ij}^{(k)}(t)} = s_j(t-k) \tag{2.22}$$

Le terme $\frac{\delta E(t_1,t_l)}{\delta s_i(t)}$ se calcule selon la position du neurone. Pour un neurone de la couche de sortie, il est obtenu directement en dérivant par rapport à $s_i(t)$,

$$\frac{\delta E(t_1,t_l)}{\delta s_i(t)} = -(s_i(t) - y(t)) \tag{2.23}$$

Pour les neurones de la couche cachée, on introduit le terme $\delta net_r(t^{'})$ pour tous les neurones successeurs.

$$\frac{\delta E(t_1,t_l)}{\delta s_i(t)} = \sum_{r \in Succ(i)} \sum_{t^{'}=t_1}^{t_l} \frac{\delta E(t_1,t_l)}{\delta net_r(t^{'})} \frac{\delta net_r(t^{'})}{\delta s_i(t)} \tag{2.24}$$

où $Succ(i)$ représente l'ensemble du neurones successeurs au neurone $i$.

Or, $net_r(t)$ peut s'écrire

$$net_r(t) = \sum_{r \in Succ(i)} \sum_{t_1}^{t_l} w_{ri}^{(k)} s_i(t-k) \tag{2.25}$$

et par conséquent,

$$\frac{\delta net_r(t^{'})}{\delta s_i(t)} = \begin{cases} w_{ri}^{(t^{'}-t)} & si\ 0 \leq (t^{'}-t) \leq p_r \\ 0 & sinon \end{cases} \tag{2.26}$$

L'équation générale de la modification des poids est décrite alors par les équations 2.27 et 2.28

$$\Delta w_{ij}^{(k)} = -\eta \sum_{t=t_1}^{t_l} \frac{\delta E(t_1,t_l)}{\delta s_i(t)} f_i^{'}(net_i(t)) s_j(t-k) \tag{2.27}$$

avec

$$\frac{\delta E(t_1,t_l)}{\delta s_i(t)} = \begin{cases} -(y_i(t) - s_i(t)) & si\ i\ est\ un\ neurone\ de\ sortie \\ \sum_{r \in Succ(i)} \sum_{t^{'}}^{t+p_r} \frac{\delta E(t_1,t_l)}{\delta net_r(t^{'})} w_{ri}^{(t^{'}-t)} & si\ i\ est\ un\ neurone\ caché \end{cases} \tag{2.28}$$

### 2.4.4 Diverses problématiques liées aux FFNN

De nombreuses études empiriques et exhaustives ont été menées sur les réseaux à propagation avant, notamment démontrant la supériorité de ces derniers sur les méthodes linéaires [Hill *et al.*, 1996] [Sharda et Patil, 1992]. Dans [Hoptroff, 1993] et [Moody, 1995], des études générales ont dressé un bilan sur l'utilisation des réseaux à propagation avant pour des applications financières tout en soulignant les diverses problématiques qui peuvent être rencontrées par ces méthodes. La plupart de ces problématiques peuvent être facilement exportées à d'autres domaines d'étude et ne sont pas exclusives à la prévision des séries financières. D'autres travaux se sont attachés à caractériser les capacités de ces réseaux et les conditions qui peuvent influencer les performances.

La nature des données est un des facteurs déterminants pour la performance finale du réseau de neurones. Dans [Gencay, 1994] est étudié l'impact du bruit sur la performance de la prévision. Il existe aussi plusieurs travaux mettant au point des prédicteurs robustes au bruit en se focalisant par exemple sur les données par élimination des valeurs aberrantes ou en remplaçant l'estimation moindre carrés des paramètres par une autre. La saisonnalité et le changement de régime sont d'autres aspects des données qui rendent la tâche de prédiction plus difficile. Le changement de régime a été étudié par [Gonzalez Mromera *et al.*, 2008], alors que [Zhang, 2005] étudie l'efficacité des prétraitements des séries saisonnières sur la performance des MLP. Il conclut que même avec les prétraitements, les performances des MLP en prédiction restent limitées.

Le choix de l'architecture est un facteur tout aussi important. Il touche notamment le nombre de neurones d'entrée [Mori et Urano, 1996] ainsi que le nombre de neurones cachés [Kaastra et Boyd, 1996]. Plusieurs approches tentent de trouver de manière automatique l'architecture adéquate des réseaux pour un problème donné. Par exemple, [Niska, 2004] propose d'utiliser un algorithme génétique en parallèle avec l'algorithme d'apprentissage d'un MLP. Ou encore pour les RBFN, le nombre de neurones cachés peut être déterminé en appliquant un algorithme de clustering [Lin et Chen, 2005] [Sun *et al.*, 2004]. Suite à une étude empirique, dans [Zhang *et al.*, 2001] il est conclu que le nombre d'entrées est un facteur qui s'est montré plus déterminant sur les performances du réseau que le nombre de neurones cachés, et ce en testant sur un grand nombre de séries temporelles. Ce résultat n'est pas forcément en contradiction avec le théorème d'approximation universelle [Cybenko, 1989] si on prend en compte l'effet "malédiction de la dimensionnalité" [Bellman et Kalaba, 1959] [Verleysen et François, 2005]. Ce principe mis en évidence expérimentalement. En augmentant la dimension des données, les réseaux deviennent plus rapidement inefficaces. Le même effet se produit en introduisant des variables exogènes peu significatives [Chakraborty *et al.*, 1992].

On remarque aussi que les performances des réseaux chutent lorsque l'on augmente l'horizon de prévision. Ce phénomène a été testé dans de nombreux travaux dont [Atiya *et al.*, 1999] et [Karunasinghe et Liong, 2006] pour la prévision des séries naturelles (débit de rivières).

La limite la plus spécifique aux réseaux à propagation avant pour la prévision est la fenêtre temporelle fixe qu'ils possèdent en entrée. Comme ces réseaux disposent d'une mémoire fixe sensiblement limitée, certains travaux ont considéré ce problème en modifiant l'architecture par l'ajout des connexions à délais directement dans le réseau ( [Back *et al.*,

1995], [Wan, 1990] et [Bone *et al.*, 2000]), d'autres ont modifié l'algorithme d'apprentissage en essayant d'apprendre les délais [Duro et Reyes, 1999] [Pi et Peterson, 1994]. Les réseaux à propagation avant restent les plus vulnérables à ces effets de dépendance à long terme face aux réseaux de neurones récurrents qui possèdent une mémoire interne théoriquement plus large.

## 2.5 Les Réseaux de Neurones Récurrents

La principale caractéristique des Réseaux de Neurones Récurrents, en anglais *Recurrent Neural Networks* (RNN), est l'existence de cycles dans le graphe orienté d'interconnexions des neurones qui représente l'architecture du réseau. La propagation du signal n'est plus unidirectionnelle (de l'entrée vers la sortie), contrairement aux réseaux à propagation avant. La conséquence immédiate est l'introduction du facteur temps dans le fonctionnement du réseau, notamment dans l'algorithme d'apprentissage associé. En effet, le passage du signal d'un neurone à un autre prend un pas de temps pour les connexions simples. L'avantage de ce type d'architectures est que le réseau réalise une mémoire interne qui peut remplacer la fenêtre temporelle utilisée dans les FFNN [Haykin, 1999], [Boné, 2000]. Nous passerons rapidement des exemples d'architecture de réseaux récurrents.

### 2.5.1 Architectures de réseaux récurrents

#### 2.5.1.1 Les réseaux localement récurrents globalement à propagation avant

Les réseaux localement récurrents globalement à propagation avant, en anglais *Locally Recurrent Globally feed Forward* (LRGF), sont une famille d'architectures qui peut être située entre les FFNN et les RNN. Les boucles existent au niveau des neurones et ne sont pas visibles dans l'architecture globale du réseau. Si l'on considère les neurones formels vus dans la section 2.3.1 (figure 2.1) dans laquelle le neurone est formé d'une unité de sommation et d'une unité de transfert, deux types d'architectures émergent. Le premier type est le bouclage de sortie présenté dans [Frasconi *et al.*, 1992] et dans [Poddar et Unnikrishnan, 1991], utilisant des neurones appelés à mémoire avec un bouclage local de la sortie vers l'unité de sommation. Le deuxième type est celui du bouclage de l'activation, voir dans [Back et Tsoi, 1991] pour plus d'information. Ces types de réseaux souffrent de lenteur dans l'exécution de leurs algorithmes d'apprentissage ainsi que d'une complexité pour la mise en œuvre. Ils sont aujourd'hui très peu utilisés dans la littérature, étant surpassés par d'autres architectures. Nous ne rentrerons pas dans les détails, toutefois de plus amples informations peuvent être trouvées dans les références citées.

#### 2.5.1.2 Architecture de Jordan et de Elman

Cette architecture particulière a été proposée par Jordan comme tout premier réseau récurrent [Jordan, 1986], initialement dédié aux traitements des séquences. Sa principale caractéristique est l'existence d'une mémoire de contexte (appelée aussi couche de contexte) liée à la couche de sortie et ayant le même nombre de neurones (2.11). Cette couche de contexte joue le rôle d'une mémoire qui garde l'état précédent du réseau. De plus, les boucles

locales dans les neurones de contexte permettent de garder une trace des valeurs antérieures. Il est à noter que l'apprentissage se faisait initialement uniquement sur les connexions standard (pas les boucles) ce qui permettait d'utiliser les méthodes d'apprentissage des réseaux à couches.

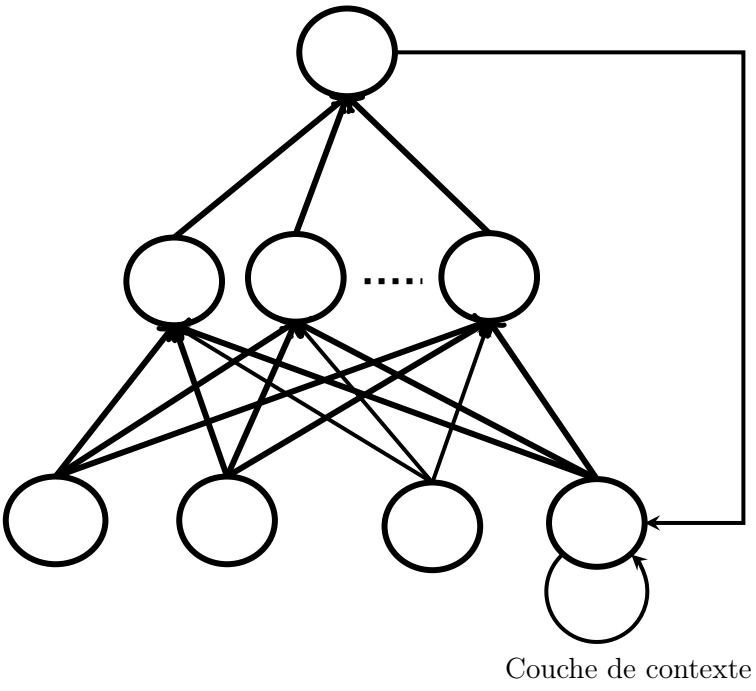


FIGURE 2.11 – Exemple d'architecture de Jordan

Inspiré du réseau de Jordan, le réseau de Elman [Elman, 1989] est très similaire dans son architecture (figure 2.12). La différence réside dans le bouclage qui ne se fait plus entre la couche de sortie et la couche de contexte, mais entre cette dernière et la couche cachée. A chaque instant $t$, la couche de contexte contient les valeurs de l'instant $t-1$ de la couche cachée, permettant ainsi d'injecter les anciennes valeurs de la couche cachée sans traitement. L'apprentissage se fait de la même manière.

Dans [Tung-Bo et Von-Wun, 1996], une autre architecture à une couche de contexte est proposée, il s'agit d'une combinaison des deux architectures de Elman et Jordan utilisant deux couches de contexte séparées. Elle est comparée avec celle de Elman et de Jordan sur des tâches de simulation de machines d'état ainsi que sur la détermination de la périodicité des séquences. Les résultats ont montré la supériorité de l'architecture proposée sur les deux autres. Une amélioration de Jordan est proposée dans [Song, 2011], l'objectif est de trouver une initialisation robuste du réseau par le biais d'un algorithme d'apprentissage à contraintes. Similairement, sa supériorité sur son prédécesseur est prouvée par un *benchmark* sur la prévision d'une série de trafic de transport [Ulbricht, 1994]. Dans [Guler *et al.*, 2005] une combinaison entre l'architecture de Elman et les exposants de Lyapunov est proposée dans le but d'analyser des signaux électro-encéphaliques.

Il existe diverses autres références bibliographiques dans lesquelles ce type d'architecture

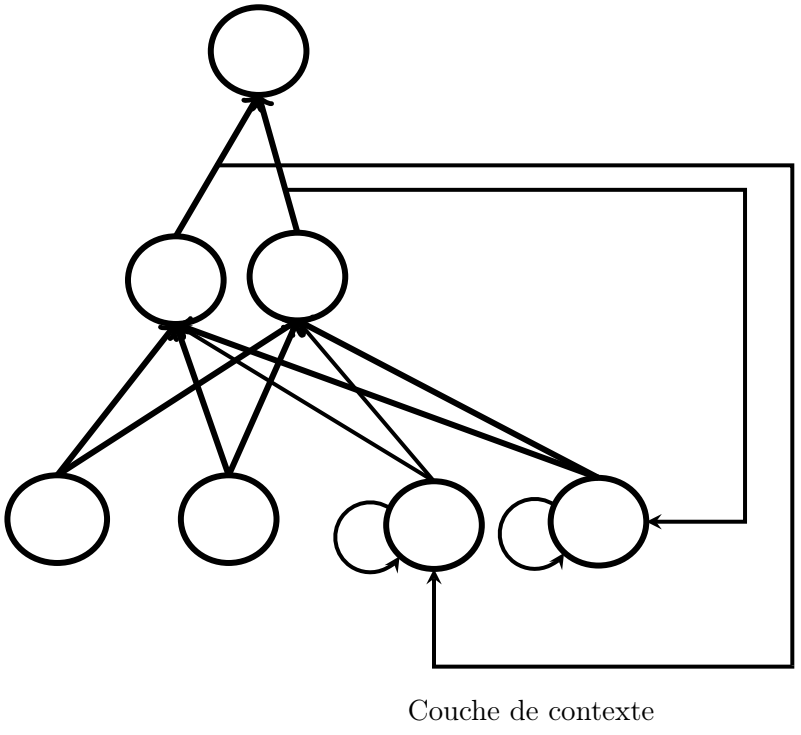


FIGURE 2.12 – Exemple d'architecture de Elman

est utilisé dans différents domaines. Citons par exemple [Pham et Karaboga, 1999] pour l'application à l'identification de systèmes, [Zhou *et al.*, 2011] et [Kelo et Dudul, 2012] pour l'application à la prévision des séries temporelles de consommation de gaz et d'électricité, [Köker, 2005] et [Liou *et al.*, 2008] pour des applications à la robotique, [Ekici *et al.*, 2009] et [Şeker *et al.*, 2003] pour les applications à la détection de défauts et le monitoring des systèmes.

Nous pouvons remarquer que ce type d'architecture est plus fréquent dans d'autres tâches d'analyse et de traitement de données séquentielles que dans la tâche spécifique de prévision de séries temporelles. Bien que ces architectures permettent de simuler une mémoire interne au réseau, et cela grâce à la couche de contexte, cette mémoire apparaît limitée dans le temps. En effet, utiliser le même algorithme d'apprentissage que les FFNN implique que les valeurs récupérées dans la couche de contexte seront utilisées avec les nouvelles valeurs uniquement dans l'itération en cours en produisant directement la valeur de sortie correspondante. Un autre algorithme d'apprentissage a été proposé initialement pour des réseaux de neurones récurrents plus génériques (comme les réseaux totalement récurrents) mais qui s'applique aussi pour les architectures d'Elman et de Jordan. Il offre aux réseaux une mémoire interne plus grande.

#### 2.5.1.3 Architectures générales

Nous désignons par réseaux totalement récurrents les architectures récurrentes génériques dans lesquelles les cycles peuvent exister entre n'importe quels neurones sans aucune contrainte.

Dans la figure 2.13, un réseau totalement récurrent sans couche est présenté. Tous les neurones sont connectés entre eux sans exception avec un neurone d'entrée et un neurone de sortie.

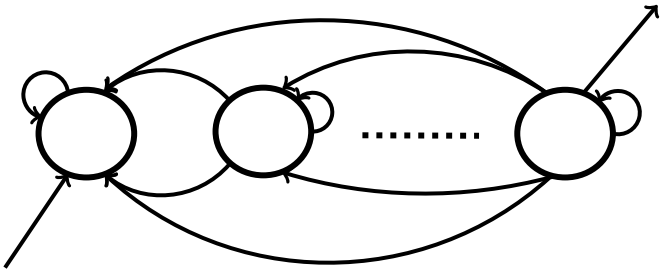

FIGURE 2.13 – Exemple d'architecture totalement récurrente

Dans la figure 2.14, un autre réseau totalement récurrent est présenté. Nous appellerons dans la suite ce type de réseaux "récurrents à couches". Cette architecture possède un neurone sur la couche d'entrée et un neurone sur la couche de sortie. La couche cachée est disposée de manière à ce que les neurones soient totalement connectés. Cette classe de réseaux a montré son efficacité et sa supériorité sur les précédentes dans [Boné, 2000]. Il est aussi montré expérimentalement qu'un neurone d'entrée est suffisant pour modéliser

et prédire les séries temporelles. Dans nos expérimentations, nous nous baserons sur cette catégorie quand il s'agit de RNN.

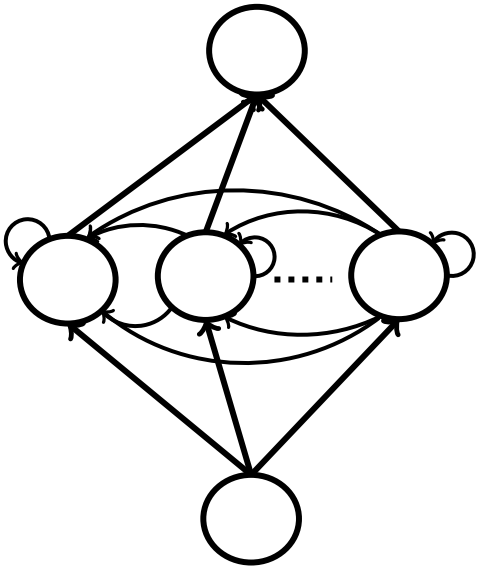

FIGURE 2.14 – Exemple d'architecture récurrente à couches

Il est important de noter que les RNN possèdent aussi un pouvoir d'approximation de fonctions mathématiques, ce qui a joué un rôle important dans le succès de ces architectures. Dans [Cybenko, 1989] cette propriété est démontrée pour un réseau totalement récurrent, dans [Funahashi et Nakamura, 1993] elle est démontrée par corolaire plus spécifiquement pour les réseaux récurrents à couches. Dans [Tenti, 1996] l'architecture totalement récurrente est utilisé pour prédire les taux de change sur certaines devises. Les résultats ont été comparé à des FFNN et montrent que ces architectures sont capables de surpasser des réseaux à propagation avant dans de nombreuses occasions. Il est aussi à noter que les réseaux à fonction radiale (RBFN) vus précédemment dans 2.4.2 peuvent être repris sous une forme récurrente [Zemouri *et al.*, 2003].

Nous avons vu dans les paragraphes précédents les architectures de base qui peuvent être décrites en tant que récurrentes. Certaines de ces architectures sont assez proches de la classe des FFNN, comme par exemple les réseaux localement récurrents (LRGF). On trouve aussi une équivalence entre certains modèles linéaires et des architectures récurrentes, par exemple, entre un RNN et un modèle ARMA [Saxen, 1997], ou encore avec le modèle NARX [Lin *et al.*, 1996].

Il existe une variété d'approches originales proposant de nouvelles architectures plus ou moins différentes. Dans [Ulbricht, 1994] une architecture appelée Multi-Recurrent Neural Networks est utilisée pour la prédiction du trafic routier. Ce réseau se base sur une architecture à couches tout en autorisant des boucles entre les couches sur des neurones spécifiques. Une autre variante combine globalement plusieurs réseaux de manière hiérarchique [Hihi, 1996]. Plus récemment, une amélioration d'architecture de RNN a été proposée par [Ohta

et Gunji, 2006] destinée à l'apprentissage incrémental, dans laquelle les représentations internes existantes ne sont pas automatiquement écrasées à chaque itération. Outre la structure de l'architecture des RNN, cette classe d'architecture diffère des FFNN par les algorithmes d'apprentissage proposés et leurs variantes. On ne s'intéressera dans la suite qu'aux algorithmes d'apprentissage des réseaux globalement récurrents.

### 2.5.2 Algorithmes d'apprentissage des RNN

Cette partie est dédiée aux algorithmes d'apprentissage des réseaux globalement récurrents. On s'intéressera plus spécifiquement à l'algorithme de la rétro-propagation dans le temps. Cependant nous verrons qu'il existe d'autres méthodes d'apprentissage ainsi que des variantes qu'on présentera dans les parties suivantes.

#### 2.5.2.1 Rétro-propagation dans le temps

L'algorithme de la rétro-propagation dans le temps, en anglais *Back Propagation Through Time* (BPTT), a été proposé par Rumelhart [Rumelhart *et al.*, 1986]. L'idée principale de cet algorithme est de déplier dans le temps le RNN de façon à obtenir un réseau à $l$ couches, comme le montre la figure 2.16, correspondant au réseau totalement récurrent (figure 2.15 à titre d'exemple). Le nombre $l$ est directement lié à la taille de l'ensemble de données, chaque couche représente une nouvelle itération dans laquelle une nouvelle donnée est présentée.

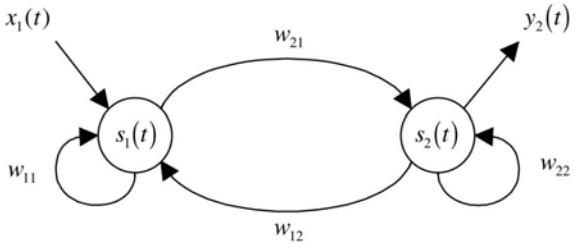


FIGURE 2.15 – Exemple de réseau de neurones récurrent associé au réseau déplié de la figure 2.16

Dans le réseau d'origine, chaque neurone peut être un neurone caché, d'entrée, de sortie ou d'entrée et sortie à la fois. Si on applique le principe de la rétro-propagation du gradient de l'erreur sur le réseau déplié, on obtiendra des neurones cachés avec une sortie désirée. Dans ce cas, il faudra tenir compte, pour ces neurones, de la rétro-propagation issue des neurones suivants mais aussi de la rétro-propagation de l'erreur locale entre la sortie désirée et la sortie obtenue de ce neurone caché.

Dans le cas d'un réseau récurrent, un neurone $i$ peut être décrit par l'expression mathématique suivante :

$$s_i(t) = f_i(net_i(t-1)) + x_i(t) \tag{2.29}$$

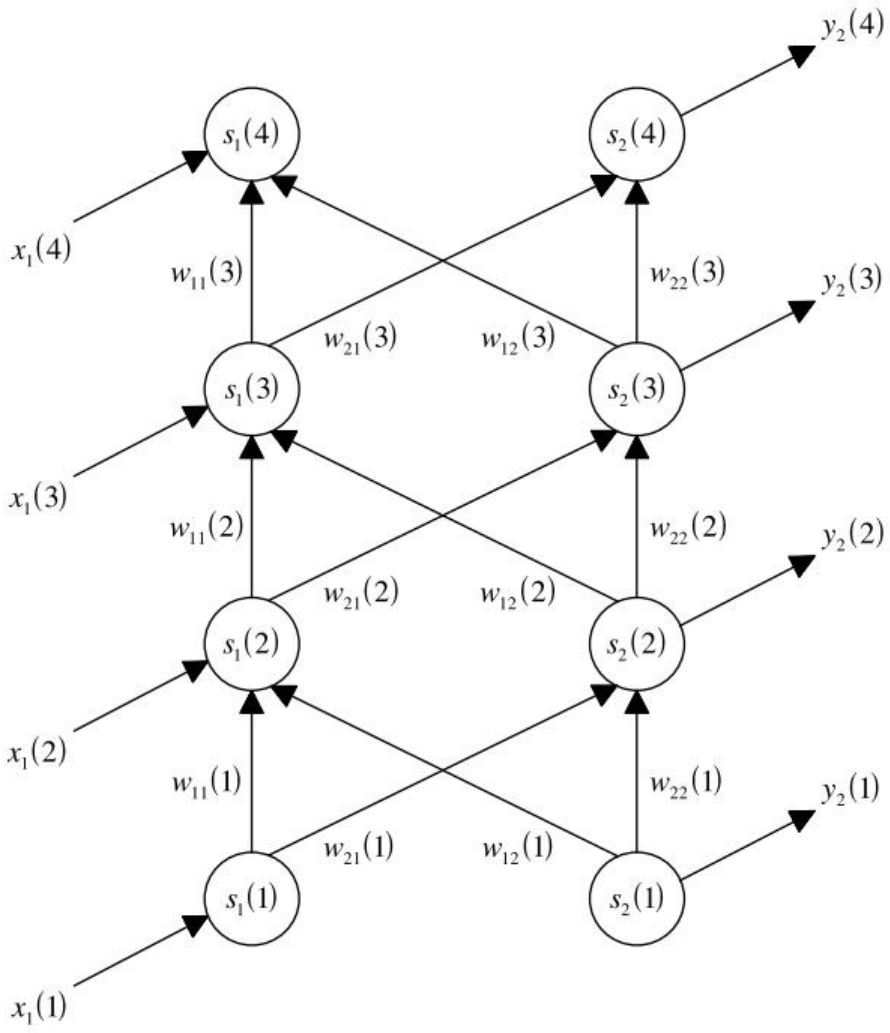


FIGURE 2.16 – dépliement du réseau récurrent de la figure 2.15 en un réseau à propagation avant multi couche

avec

$$net_i(t-1) = \sum_{j \in Pred(i)} w_{ij}(t-1)s_j(t-1) \tag{2.30}$$

où $w_{ij}(t-1)$ est la duplication du poids $w_{ij}$ correspondant au réseau récurrent d'origine à l'instant $t-1$.

La variation d'un poids $\Delta w_{ij}$ se calcule sous forme de la somme de toutes les variations. En appliquant la descente du gradient sur l'erreur totale, on obtient l'équation (2.31),

$$\Delta w_{ij}(t_1, t_l - 1) = -\eta \frac{\delta E(t_1, t_l)}{\delta w_{ij}} = -\eta \sum_{\tau=t_1}^{t_l-1} \frac{\delta E(t_1, t_l)}{\delta w_{ij}(\tau)} \tag{2.31}$$

En appliquant la chaîne de dérivation, on obtient :

$$\frac{\delta E(t_1, t_l)}{\delta w_{ij}(\tau)} = \frac{\delta E(t_1, t_l)}{\delta net_i(\tau)} \frac{\delta net_i(\tau)}{\delta w_{ij}(\tau)} \tag{2.32}$$

Pour évaluer le terme $\frac{\delta E(t_1,t_l)}{\delta s_i(\tau+1)}$, nous devons considérer la situation du neurone $i$. Cette situation dépend de la variable $\tau$. En effet, pour $\tau = t_l - 1$ le neurone peut être considéré comme un neurone de la couche de sortie. On obtient donc

$$\frac{E(t_1, t_l)}{\delta s_i(\tau+1)} = \frac{\delta e(t_l)}{\delta s_i(t_l)} = \begin{cases} s_i(t_l) - d_i(t_l) & si\ i \in O(\tau+1) \\ 0 & sinon \end{cases} \tag{2.33}$$

où $O(\tau+1)$ représente l'ensemble de neurones de sortie à l'instant $\tau+1$. Le deuxième cas correspond à $t_1 \leq \tau \leq t_l$, ce qui signifie que le neurone $i$ se positionne dans une couche cachée. Nous obtenons

$$\frac{\delta E(t_1, t_l)}{\delta s_i(\tau+1)} = \frac{\delta e(\tau+1)}{\delta s_i(\tau+1)} + \sum_{j \in Succ(i)} \frac{\delta E(t_1, t_l)}{\delta net_j(\tau+1)} \frac{\delta net_j(\tau+1)}{\delta s_i(\tau+1)} \tag{2.34}$$

or selon (2.30) $\frac{\delta net_j(\tau+1)}{\delta s_i(\tau+1)}$ donne

$$\frac{\delta net_j(\tau+1)}{\delta s_i(\tau+1)} = w_{ij}(\tau+1) \tag{2.35}$$

d'où

$$\frac{\delta E(t_1, t_l)}{\delta s_i(\tau+1)} = \begin{cases} s_i(\tau+1) - d_i(\tau+1) + \sum_{j \in Succ(i)} \frac{\delta E(t_1,t_l)}{\delta net_j(\tau+1)} w_{ji}(\tau+1) & si\ i \in O(\tau+1) \\ \sum_{j \in Succ(i)} \frac{\delta E(t_1,t_l)}{\delta net_j(\tau+1)} & sinon \end{cases} \tag{2.36}$$

Nous pouvons ainsi conclure à la forme finale de modification des poids pour l'algorithme BPTT

$$\Delta w_{ij}(t_1, t_l - 1) = -\eta \sum_{\tau=t_1}^{t_l-1} \frac{\delta E(t_1, t_l)}{\delta net_i(\tau)} s_j(\tau) \tag{2.37}$$

avec

- pour $\tau = t_l - 1$

$$\frac{\partial E(t_1, t_l)}{\partial net_i(\tau)} = \begin{cases} (s_i(\tau) - d_i(\tau)) f_i^{'}(net_i(\tau)) & \text{si } i \in O(\tau) \\ 0 & \text{sinon} \end{cases} \tag{2.38}$$

- pour $t_1 \leq \tau \leq t_l - 1$

$$\frac{\partial E(t_1, t_l)}{\partial net_i(\tau)} = \begin{cases} \left[ (s_i(\tau) - d_i(\tau)) + \sum_{j \in succ(i)} \frac{\partial E(t_1, t_l)}{\partial net_j(\tau+1)} w_{ij}(\tau+1) \right] f_i^{'}(net_i(\tau)) \\ \text{si } i \in O(\tau) \\ \sum_{j \in succ(i)} \frac{\partial E(t_1, t_l)}{\partial net_j(\tau+1)} w_{ij}(\tau+1) f_i^{'}(net_i(\tau)) \\ \text{sinon} \end{cases} \tag{2.39}$$

où $d_i(\tau)$ est la sortie désirée à l'instant $\tau$.

L'algorithme a une complexité en temps pour un réseau totalement récurrent de $O(N^2)$ avec $N$ le nombre total neurones. L'inconvénient de cet algorithme est dans sa complexité en mémoire où elle est en $O(lN^2)$ ce qui peut être conséquent avec des ensembles de données de grande taille, d'autant plus que l'algorithme doit charger en mémoire toute la séquence de données pour pouvoir calculer la variation des poids. Pour éviter cela, il suffit de limiter la profondeur de la rétro-propagation.

#### 2.5.2.2 Apprentissage récurrent temps réel

L'apprentissage récurrent temps réel, en anglais *Real Time Recurrent Learning* (RTRL), utilise une stratégie différente de modification des poids durant l'apprentissage. Dans cet algorithme, les poids sont modifiés à chaque pas de temps. Le principe de l'algorithme est toujours le même, c'est-à-dire calculer les variations des poids par la descente du gradient minimisant l'erreur totale, mais en appliquant ses variations à chaque nouvel exemple présenté comme l'indique l'équation 2.40.

$$\Delta w_{ij} = \sum_{t=t_1}^{t_l} \Delta w_{ij}(t) \tag{2.40}$$

avec

$$\Delta w_{ij}(t) = -\nu \frac{\delta e(t)}{\delta w_{ij}} = \nu \sum_{t \in T(t)} (d_r(t) - s_r(t)) \frac{\delta s_r(t)}{\delta w_{ij}} \tag{2.41}$$

Par le même raisonnement appliqué avec BPTT, on retrouve la formule de modification des poids à l'instant $t$ en fonction de l'instant $t-1$

$$\frac{\delta s_r(t)}{\delta w_{ij}} = f^{'}(net_r(t-1))(\delta_{ir} s_j(t-1) + \sum_{q \in Pred(r)} \frac{\delta s_q(t-1)}{\delta w_{ij}}) \tag{2.42}$$

avec comme condition initiale

$$\frac{\delta s_r(t)}{\delta w_{ij}} = 0 \tag{2.43}$$

Cet algorithme se caractérise par une plus faible occupation mémoire par rapport à BPTT mais une complexité en temps plus élevée ($O(N^4)$ comparée à $O(N^3)$).

#### 2.5.2.3 Variantes d'algorithmes

Dans la classe des RNN, il existe un très grand nombre de variantes et d'améliorations des algorithmes de base, comme BPTT ou RTRL. Par exemple, dans [Boné, 2000], à partir de BPTT, un algorithme constructif permet de rajouter en cours d'apprentissage et de manière sélective des connexions à délais. Dans [Schmidhuber, 1992], les algorithmes BPTT et RTRL sont combinés dans une même approche stratégique qui consiste à diviser le calcul par bloc au sein du réseau. Chaque bloc réalise un calcul avec BPTT puis RTRL est utilisé pour relier les résultats des différents blocs entre eux. Dans [Han *et al.*, 2004] un algorithme inspiré de BPTT est adapté pour la prévision à long terme en réalisant une série de prévisions multi-pas appelé Recurrent Predictor Neural Networks. Dans [Schuster et Paliwal, 1997] une approche bidirectionnelle d'apprentissage est présentée, elle consiste à faire apprendre un RNN en lui présentant les données dans le sens positif (de la première valeur présentée à l'instant $t_1$ à la dernière présentée à l'instant $t_l$) et négatif (c'est-à-dire le sens inverse du temps, de $t_l$ à $t_1$).

D'autres approches se focalisent sur l'amélioration du calcul du gradient dans l'algorithme d'apprentissage. En effet, d'une part, selon [Bengio *et al.*, 1993] le gradient tend à s'annuler rapidement au sein du réseau pour la plupart des variables d'entrées, c'est-à-dire que les dérivées des sorties à un instant $t$ par rapport aux poids tends vers zéro plus on avance dans $t$. D'autre part, le temps de convergence est relativement élevé selon la taille des données. Deux facteurs impliquent cette augmentation en temps de calcul : l'existence des boucles dans le réseau et le traitement temporel des algorithmes d'apprentissage. Dans [Toomarian et Barhen, 1992] un algorithme d'apprentissage par renforcement utilise une méthode d'apprentissage en ligne qui permet d'obtenir le gradient récursivement (Fast Forward Propagation). Dans [Sun *et al.*, 1992], une approche par la fonction de Green est proposée, permettant une représentation mathématique plus efficace ainsi qu'une implémentation numérique plus rapide. Il s'agit de faire sortir de la formulation en ligne du gradient une équation différentielle linéaire dont la fonction de Green est une solution. Les expérimentations montrent la rapidité de cette approche par rapport à RTRL. Dans [Srinivasan *et al.*, 1994] un algorithme d'apprentissage permettant de réévaluer efficacement le gradient sur des interconnexions ciblées est proposé (Targeted Retro Propagation).

Les Echo State Network (ESN) peuvent être considérés comme un type de réseau à part avec un algorithme d'apprentissage particulier [Jaeger, 2001]. La caractéristique de la couche cachée est d'être très faiblement connectée. Les connexions et poids du réseau sont initialisés de manière aléatoire. La couche cachée est aussi appelée réservoir, elle représente un système non linéaire. Les paramètres finaux du système sont uniquement les poids de la couche de sortie. On peut trouver deux types d'algorithme d'apprentissage, en ligne ou hors ligne, mais de façon générale, l'apprentissage peut être caractérisé en trois étapes principales. La première est la génération aléatoire du réservoir avec les connexions et les poids. La deuxième consiste à présenter les données d'apprentissage au réservoir sur une période et à appliquer la règle d'apprentissage pour corriger la sortie. Enfin le calcul des poids de sortie est effectué. On trouve aussi des variantes et des améliorations assez

nombreuses comme par exemple dans [Jiang *et al.*, 2008] où est présentée une version évolutive de l'algorithme.

En termes de performances, plusieurs études expérimentales comparatives ont été menées sur les RNN. Dans [Tung-Bo et Von-Wun, 1996] une comparaison est réalisée entre trois architectures de RNN à couche de contexte. Dans [Saad *et al.*, 1998] une comparaison est effectuée entre plusieurs classes de réseaux de neurones parmi lesquelles des RNN, des réseaux à délais et des réseaux stochastiques pour la prévision des données financières des cours monétaires. Dans [Ho *et al.*, 2002] une comparaison est réalisée entre un modèle Box-Jenkins (ARIMA) et des architectures récurrentes et à propagation avant. Les résultats montrent la supériorité des RNN et des ARIMA sur les FFNN dans la prévision des séries temporelles. Ils montrent aussi qu'en cherchant les paramètres optimaux pour les RNN (tels que le nombre de neurones cachés, les paramètres d'apprentissage ou même les poids de départ), ces derniers permettent d'obtenir des résultats plus satisfaisants. Les RNN sont également comparés à des FFNN dans [Connor et Atlas, 1991] ; les expérimentations montrent à nouveau la supériorité des RNN sur les FFNN.

Bien que les RNN ont été, à plusieurs reprises, évoqués en tant que prédicteurs puissants et performants pour la prévision des séries temporelles, ceci n'empêche pas que ces réseaux passent un certain nombre de problèmes. Selon Bengio, l'un des problèmes les plus importants est la dépendance à long terme par rapport aux données. Dans [Bengio *et al.*, 1993] il est montré que l'apprentissage des dépendances à long terme par les RNN avec les approches de la descente du gradient n'est pas facile. Dans [Boné *et al.*, 2002] un algorithme d'apprentissage est présenté permettant d'améliorer la robustesse du RNN face aux dépendances à long terme en rajoutant des connexions à délais. Dans [Hihi, 1996] une approche différente est proposée pour ce même problème, il s'agit d'une architecture hiérarchique de réseaux. Dans [Lin *et al.*, 1998] est montré qu'un réseau récurrent basé sur le modèle NARX était capable d'améliorer la prise en compte des dépendances à long terme.

Il existe de nombreuses études qui se sont penchées sur l'aspect dynamique des réseaux récurrents. On ne s'intéressera pas particulièrement à cet aspect dans notre travail mais nous pouvons citer à titre d'exemple [Bertschinger et Natschläger, 2004] où deux types de dynamique sont identifiées (chaotique ou ordonnée). Ces dynamiques sont étudiées en fonction de la connectivité du réseau, dans le but de déterminer la limite dans l'espace des paramètres séparant le régime ordonné et le régime chaotique. Dans [Casey, 1996], les RNN sont étudiés pour modéliser et extraire des machines d'états finies. Dans [Zuo *et al.*, 2010] une étude de stabilité est réalisée sur des RNN introduisant des délais variables.

Nous avons abordé jusqu'à présent la famille de prédicteurs basés sur les réseaux de neurones artificiels de deux types : les réseaux à propagation avant et les réseaux récurrents. Dans ce qui suit, on s'intéressera à la deuxième famille de prédicteurs que nous avons utilisée dans cette thèse, appelée les Support Vector Regression.

## 2.6 Support vector regression

Les Support Vector Regression (SVR) sont une adaptation du problème de la classification au problème de la régression de l'algorithme des Machines à Supports Vecteurs, en

anglais *Support Vector Machines* (SVM). Nous ne rentrerons pas dans ce qui suit dans les détails théoriques de ces méthodes, des informations plus détaillées peuvent être trouvées dans [Vapnik, 2000].

Les SVM et les SVR sont appliqués dans divers domaines. Historiquement l'application majeure des SVM est la reconnaissance de caractères [Scholkopf *et al.*, 1997], [Boser *et al.*, 1992] et [Guyon *et al.*, 1992], mais aussi la reconnaissance de la parole [Schmidt et Gish, 1996] et la détection de visages [Osuna *et al.*, 1997]. Ceux qui nous intéressent, les SVR, sont utilisés dans la prévision des séries temporelles [Müller *et al.*, 1997] et [Drucker *et al.*, 1997] ainsi que dans l'estimation de densité [Weston *et al.*, 1999].

### 2.6.1 Présentation générale des SVM

Les SVM (ainsi que les SVR) sont une classe d'algorithmes d'apprentissage supervisé. Les SVM sont basés sur les mêmes principes d'apprentissage que les réseaux de neurones. Toutefois, ils possèdent l'avantage d'être plus simples à configurer.

Le modèle du classifieur est construit à partir d'un ensemble d'apprentissage de $N$ exemples étiquetés $(\mathbf{x}_i, y_i)$ avec $\mathbf{x}_i \in \mathcal{R}^p$ et $y_i \in \{1; -1\}$ selon la classe ($p$ représente la dimension des vecteurs d'entrée ou encore le nombre de caractéristiques dans les exemples d'entrées). Il est à remarquer que la version originale des SVM traite uniquement le cas binaire à deux classes, mais des extensions existent pour le multi-classe. L'apprentissage permet de construire la fonction $f$ de décision appelée aussi hyperplan séparateur

$$f(\mathbf{x}) = signe(< \mathbf{w}, \mathbf{x} > +b) \tag{2.44}$$

avec $\mathbf{w} \in \mathcal{R}^p$ et $b$ les paramètres permettant de déterminer dans quelle partie de l'hyperplan se trouve l'exemple $\mathbf{x}$. La figure 2.17 présente le cas linéairement séparable, où la marge $\Delta$ est définie par la distance minimale entre les deux points des différentes classes.

### 2.6.2 Support Vector Regression

Le principe des SVM est repris et adapté par les SVR pour modéliser un problème de régression. Le but est d'approximer un ensemble de données $(\mathbf{x}_i, y_i)$ par une fonction $f$

$$f(\mathbf{x}) =< \mathbf{w}, \mathbf{x} > +b \tag{2.45}$$

telle que

$$|f(\mathbf{x}_i) - y_i| \leq \epsilon \tag{2.46}$$

avec $i \in \{1, .., N\}$.

L'idée consiste à minimiser le terme $\mathbf{w}$ tout en étant sous la contrainte de ne pas dépasser un taux d'erreur $\epsilon$. De point de vue graphique, ceci revient à trouvez une zone du plan qui contient tous les exemples $\mathbf{x}_i$ de largeur $2\epsilon$ appelé tube, (voir figure 2.18). Si on considère la minimisation de $\|\mathbf{w}\|^2$ on obtient le problème quadratique suivante :

$$\begin{gathered} minimiser \, \|\mathbf{w}\|^2 \\ tel \; que \begin{cases} y_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - b \leq \epsilon \\ \langle \mathbf{w}, \mathbf{x}_i \rangle + b - y_i \leq \epsilon \end{cases} \end{gathered} \tag{2.47}$$

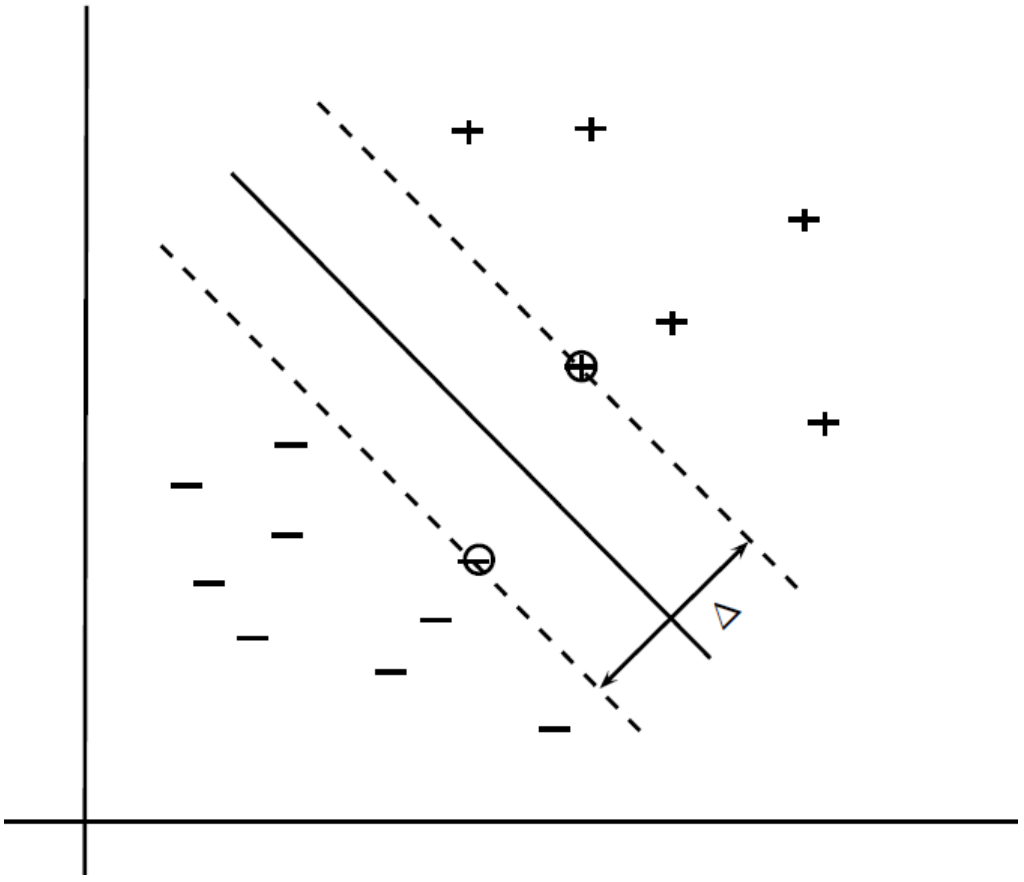


FIGURE 2.17 – Hyperplan linéaire séparant avec la marge $\Delta$ les exemples positifs des exemples négatifs

En passant par la formulation duale et l'équation de Lagrange, la fonction obtenue peut être écrite sous la forme

$$f(\mathbf{x}) = \sum_{i=1}^{N} (\alpha_i - \alpha_i^*) \langle \mathbf{x}_i, \mathbf{x} \rangle + b, \tag{2.48}$$

où $\alpha_i$ et $\alpha_i^*$ sont les multiplicateurs de Lagrange issus de la formulation duale et à déterminer. Il est à noter qu'en pratique, il est difficile de satisfaire la contrainte stricte du tube $\epsilon$. Afin de résoudre ce problème, il est possible d'assouplir les contraintes comme le montre la figure 2.18.

L'idée fondamentale des SVM est de trouver la fonction objectif tout en minimisant un certain risque. Théoriquement le risque peut être défini en fonction de la fonction perte et de la fonction $f$ trouvée comme le montre l'équation 2.49. On peut voir dans la figure 2.18 que les exemples à l'intérieur du tube possèdent une valeur de la fonction perte nulle. Pour les exemples en dehors du tube, la valeur de la fonction perte est positive et linéairement croissante.

$$R[F] = \int l(\mathbf{x}, y, f(\mathbf{x})) dP(\mathbf{x}, y) \tag{2.49}$$

où $l(.,.,.)$ représente la fonction perte. En pratique, comme la distribution est inconnue, on se contente de minimiser le risque empirique définit par l'équation 2.50.

$$R_{emp}[F] = \frac{1}{N} \sum_{i=0}^{N} l(\mathbf{x}, y, f(\mathbf{x})) \tag{2.50}$$

Cependant, dans un espace assez riche en caractéristiques, utiliser un petit nombre de données pour définir le risque n'est pas adapté et peut mener à un sur-apprentissage. L'idée est d'utiliser un version régularisée du risque en y introduisant un terme de contrôle. Le risque régularisé peut ainsi être défini par l'équation 2.51 :

$$R_{reg}[F] = R_{emp}[F] + \frac{\lambda}{2} \|\mathbf{w}\|^2 \tag{2.51}$$

où $\lambda > 0$ est appelée constante de régularisation.

Afin de résoudre le problème non linéaire, l'idée consiste à passer par l'intermédiaire d'une fonction de *mappage* vers un espace de caractéristiques de plus grande dimension. Une régression linéaire est alors appliquée dans ce nouvel espace. Le problème est que le calcul dans cet espace peut être long voire même impossible. Pour cela, la fonction de *mappage* doit être choisie de manière à transformer le produit scalaire de l'espace des caractéristiques en un produit dans l'espace d'origine. Si on considère comme exemple la fonction de *mappage* $\Phi$

$$\begin{array}{rccc} \Phi : & \mathbb{R}^2 & \longrightarrow & \mathbb{R}^3 \\ & (x_1, x_2) & \longmapsto & (x_1^2, \sqrt{2} x_1 x_2, x_2^2), \end{array} \tag{2.52}$$

on peut facilement vérifier la propriété suivante

$$\left\langle (x_1^2, \sqrt{2} x_1 x_2, x_2^2), (x'^2_1, \sqrt{2} x'_1 x'_2, x'^2_2) \right\rangle = \left\langle \mathbf{x}, \mathbf{x}' \right\rangle^2, \tag{2.53}$$

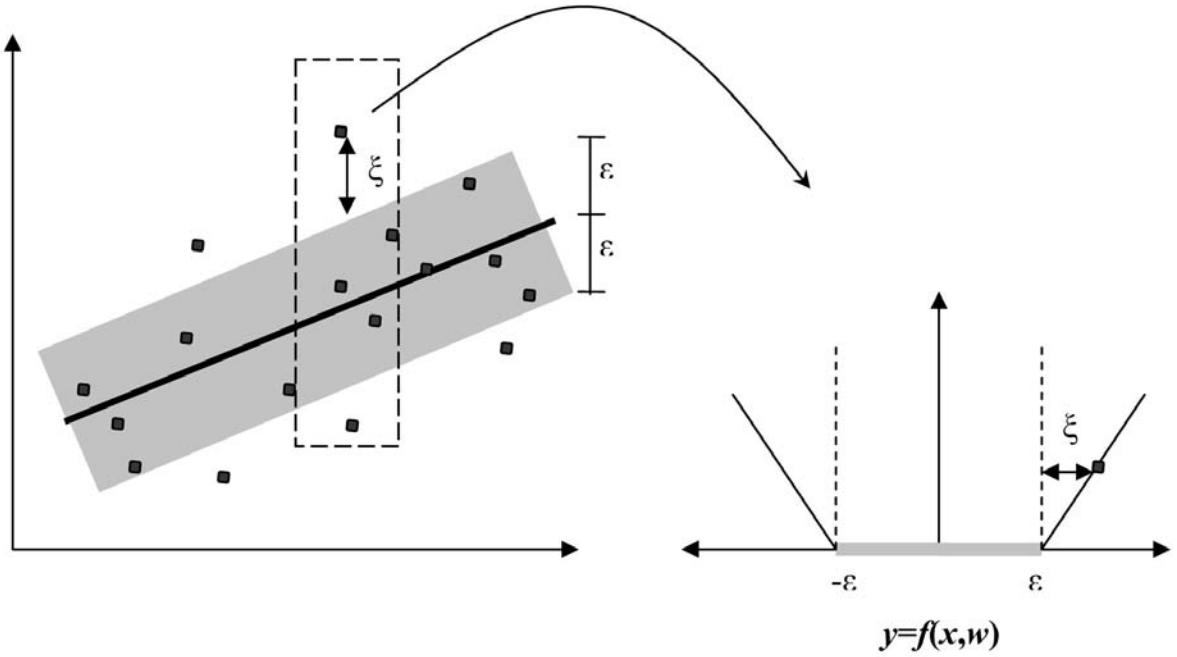


FIGURE 2.18 – Fonction perte avec assouplissement des contraintes ($\epsilon$-*insensitive*)

Ainsi la résolution d'un problème non linéaire est effectuée en passant par une fonction noyau qu'on notera $k$ et définie par $k(x_1, x_2) = \langle \Phi(x_1), \Phi(x_2) \rangle$. La fonction objectif peut alors s'écrire sous la forme

$$f(\mathbf{x}) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}) + b \tag{2.54}$$

### 2.6.3 Variantes et optimisations

#### 2.6.3.1 SVR par programmation linéaire

Dans les SVR par programmation linéaire, en anglais Linear Programming-SVR (LP-SVR) la norme $L_2$ est remplacée par la norme $L_1$ à la fois dans la régularisation et dans la fonction perte. Ceci a pour conséquence de transformer le programme quadratique en un programme linéaire.

La formulation LP possède des avantages par rapport à la formulation classique du Quadratic Programming (QP) [Vapnik, 2000]. Elle permet à la fois d'assurer la parcimonie des vecteurs supports et d'utiliser des noyaux plus généraux [Mangasarian, 2002]. D'autre part, [Smola *et al.*, 2000] montre que cette approche donne un meilleur taux d'erreur de généralisation pour la classification sur les bords de chaque classe. Dans [Smola *et al.*, 1999], l'utilisation de la norme $L_1$ avec la fonction perte $\epsilon$-*insensitive* donne un problème de programmation linéaire à $4N + 1$ variables. [Mangasarian, 2002] simplifie la résolution du programme linéaire en réduisant le nombre de variables à $3N + 1$, diminuant ainsi

considérablement le temps de calcul. Dans [Weston *et al.*, 1999], des LP-SVR sont utilisés pour résoudre un problème d'estimation de densité.

#### 2.6.3.2 Utilisation d'hyper-paramètres pour SVR

Dans la méthode standard des Support Vectors, le choix de la fonction de coût adéquate est un critère assez important pour la bonne performance des SVM et des SVR. Plus particulièrement, lorsqu'il s'agit de la fonction $\epsilon$-*insensitive*, le choix concerne la largeur adéquate $2\epsilon$ du tube. Smola [Smola *et al.*, 1999] montre que ce paramètre est lié à la nature du bruit dans les données d'apprentissage. Or, cette information est indisponible dans les applications réelles et dans le cas général. La technique $\nu$-SVR permet d'auto-régler le seuil $\epsilon$ durant l'apprentissage. Cela consiste à définir $\epsilon$ en tant que variable de l'équation du risque à minimiser (2.55) du problème d'optimisation et de rajouter un hyper-paramètre $\nu$. On obtient

$$Minimiser\ R_\nu[f] = R_{emp} + \lambda/2\,\|\mathbf{w}\|^2 + \nu\epsilon \tag{2.55}$$

Il est aussi possible d'utiliser cette approche avec la formulation en programmation linéaire [Mangasarian, 2002]. Cette approche a pour avantage de permettre à la fois de définir le pourcentage d'erreur autorisé et le pourcentage des SV à la fin de l'apprentissage [Schölkopf *et al.*, 2000].

#### 2.6.3.3 Les SVR à moindre carré

Les SVR à moindre carré, en anglais *Least Square-SVR* (LS-SVR), représentent une autre simplification de l'algorithme classique des SVR. Il s'agit de remplacer la fonction coût $\epsilon$-intensive par l'erreur quadratique, ce qui ramène à résoudre un système d'équations linéaires [Suykens *et al.*, 2002] en suivant le même principe que les SVR classiques. Le problème primal correspondant est le suivant :

$$\begin{aligned} &minimiser\ 1/2\,\|\mathbf{w}\|_2^2 + 1/2C\sum_{i=1}^{N} e_i^2 \\ &t.q.\ y_i = \langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b + e_i\ avec\ i = 1, .., N \end{aligned} \tag{2.56}$$

Un premier inconvénient de cette méthode est la perte de la parcimonie des SV qui était garantie par la fonction perte $\epsilon$-insensitive. Dans [Suykens *et al.*, 2000], il est proposé de résoudre ce problème en réduisant les SV par élagage des paramètres. La technique consiste à réapprendre avec l'algorithme classique LS-SVM en enlevant à chaque fois les exemples ayant les petites valeurs de $\alpha_i$.

Un deuxième inconvénient est le manque de robustesse face au bruit dans les données due à l'erreur quadratique. Ce problème est traité dans [Suykens, 2002] en remplaçant la fonction d'erreur classique par une fonction régulée en pondérant les erreurs des exemples. Nous obtenons la formulation du problème suivante

$$\begin{aligned} &minimiser\ 1/2 + \|\mathbf{w}\|_2^2 + 1/2C\sum_{i=1}^{N} v_i e_i^2 \\ &t.q.\ y_i = \langle \mathbf{w}, \Phi(\mathbf{x}_i)\rangle + b + e_i\ avec\ i = 1, .., N \end{aligned} \tag{2.57}$$

où $v_i$ est le poids associé à $e_i$. Pour déterminer les $v_i$, la première étape consiste à utiliser la formulation LS-SVM de l'équation (2.56) pour trouver les $e_i$. L'estimation se fait par la formule heuristique :

$$v_i = \begin{cases} 1 \ si \ \ |e_i/\hat{s}| \leq c_1 \\ \frac{c_2 - |e_i/\hat{s}|}{c_2 - c_1} \ si \ c_1 \leq |e_i/\hat{s}| \leq c_2 \\ 10^{-4} \ sinon \end{cases} \tag{2.58}$$

où $\hat{s}$ représente une estimation de l'écart type de l'erreur $e_i$. Une fois les $v_i$ déterminés, le modèle est calculé en utilisant l'équation (2.57).

Les SVR ont été appliqués dans divers domaines, majoritairement en finance, comme dans [Tay et Cao, 2002] pour la prévision d'indice boursier (S&P 500) ou encore [Sansom *et al.*, 1998] pour prédire les prix du marché de l'électricité. Ils ont aussi été appliqués dans des domaines environnementaux, comme dans [Leung et Yuen, 2002] pour la prévision de la qualité de l'air ou pour la consommation électrique [Chang *et al.*, 2001].

En ce qui concerne les études expérimentales et les comparaisons avec d'autres méthodes, les SVR ont montré dans la plupart des cas de meilleurs résultats face aux FFNN. Dans [Trafalis et Ince, 2000], le SVR est comparé à des architectures neuronales du type FFNN et a montré une supériorité dans les prévisions des marchés d'échanges des devises et ce pour une fenêtre temporelle assez petite et un horizon court. Les mêmes résultats sont confirmés dans [Tay et Cao, 2002] où les SVR ont montré une meilleure capacité à éviter le sur-apprentissage par rapport aux MLP. Dans [Thissen *et al.*, 2003] et [Ao et Palade, 2011] des SVR ont été comparés avec des RNN du type Elman. Dans [Thissen *et al.*, 2003] les résultats ont montré la supériorité des SVR face aux réseaux de Elman sur des données artificielles comme MG-17. Par contre, les résultats étaient plus proches lors de l'utilisation des données réelles. La raison était liée à l'impossibilité de faire un apprentissage exhaustif avec les SVR afin d'exploiter les spécificités mathématiques et les différents paramétrages possibles de la méthode vue les larges volumes de données. En passant à des sous-ensembles de basses résolutions (environ 10% des tailles réelles), les SVR étaient plus performants que les RNN d'Elman.Une autre expérimentation réalisée dans [Ao et Palade, 2011], les deux prédicteurs ont été "boostés" via les méthodes d'ensemble ( [Schapire, 1990] et [Freund et Schapire, 1996]) et ce sur des séries temporelles de "*microarray*" (génétique), dans laquelle les réseaux d'Elman ont été généralement meilleurs. Cependant, nous n'avons pas trouvé dans la littérature des comparaisons entre les SVR et les RNN à architectures génériques à couches comme celui déjà présenté dans la figure 2.14. Puisque nous savons que ces derniers sont, au moins dans les séries standard utilisées dans cette thèse, meilleurs que les réseaux d'Elman, nous effectuerons cette comparaison dans nos expériences.

Nous trouvons aussi dans la littérature des versions modifiées et avancées des SVR sous des formes plus au moins complexes. Citons des propositions visant à améliorer la gestion des paramètres. Dans [Tay et Cao, 2002] une approche C-ascendante est proposée où la constante de régulation $C$ est changée de manière à augmenter l'importance des erreurs les plus récentes et à diminuer les plus éloignées (dans le temps). Similairement, il est proposé dans [Cao *et al.*, 2000] de varier la largeur du tube $\epsilon$ en s'inspirant du même principe. Il est aussi possible de combiner les deux approches [Cao et Tay, 2003]. Liang et Sun [Liang et Sun, 2003] ont proposé une version de SVR permettant de modifier le noyau pendant l'apprentissage.

D'autres propositions avancées combinent deux (ou plus) domaines différents en une seule méthode, par exemple, dans [Yu-Kun *et al.*, 2005], un rapprochement entre la logique floue et les SVR pour la prévision des taux de change, ou encore dans [Pai et Hong, 2005] les techniques SVR et les algorithmes génétiques dans le but de trouver les paramètres optimaux.

## Conclusion du chapitre

Dans ce chapitre, nous avons vu deux grandes familles de méthodes d'apprentissages artificiels pour la prévision des séries temporelles : les réseaux de neurones et les Support Vecteur Machines. Nous avons vu deux types d'architectures neuronales : les FFNN et les RNN, avec souvent un avantage pour les RNN pour la prévision des séries temporelles. Par contre, entre les RNN et les SVR, nous avons vu que les résultats de la littérature étaient proches et partagés selon les données, sauf pour les réseaux de types Elman où les SVR était plus performants. Nous essaierons dans le futur d'effectuer cette comparaison de plus près dans nos expérimentations.

L'optimisation des prédicteurs peut être faite selon deux approches différentes. La première est l'amélioration de la méthode en elle-même. C'est ce que représente l'ensemble des techniques présentes dans ce chapitre. Nous verrons par la suite une deuxième approche qui consiste à modifier peu ou pas du tout le prédicteur, mais à adopter une stratégie différente dans la gestion des données d'apprentissage. Si la première approche peut être décrite comme une approche globale dans le sens où le prédicteur gère les données de manière globale, la deuxième serait définie comme une approche locale, elle sera l'objet du chapitre suivant.

# Chapitre 3

# Méthodes locales pour la prévision des séries temporelles

## Introduction

Nous avons abordé dans le chapitre précédent différentes méthodes d'apprentissage avec une focalisation sur les réseaux de neurones et les machines à supports vecteurs. Nous avons vu également différentes améliorations et variantes des algorithmes classiques. Dans ce chapitre, nous nous intéresserons à une forme de plus haut niveau d'apprentissage. Il s'agit des méta-méthodes qui utilisent les données de manière partielle, c'est-à-dire sur une sous-partie des données, dans le but d'améliorer les résultats globaux. Nous commencerons par détailler l'intérêt de telles approches, ensuite nous verrons de manière rapide les méthodes d'ensemble mais nous nous focaliserons plus sur les méthodes locales par segmentation (*clustering*).

## 3.1 Intérêt

Les méthodes d'apprentissage du chapitre précédent possèdent toutes des limitations dues à certaines erreurs ou biais qui doivent être considérés. Nous pouvons définir de manière théorique ces erreurs et les décomposer en deux principales entités. Les détails peuvent être trouvés dans [Geman et Bienenstock, 1992] et [Bishop, 1995]. En effet, l'erreur de prédiction pour un modèle donné se décompose théoriquement en deux : le biais et la variance. Le biais peut être vu comme une erreur d'approximation moyenne résultant du modèle construit sur les différents ensembles d'apprentissage possibles. La variance représente la différence entre toutes fonctions approximées résultant de tous les ensembles d'apprentissage possibles et peut être vu comme la sensibilité des résultats au choix de l'ensemble d'apprentissage. Ces deux composantes sont complémentaires. On est toujours amené à réaliser un compromis entre les deux, sous peine de tomber entre deux cas extrêmes : le sous-apprentissage et le sur-apprentissage. Les modèles qui sont fortement entrainés sont largement proches des données et ont par conséquent une variance très forte. Inversement, chercher à diminuer la variance peut être décrit comme lisser la fonction du modèle mais

pas trop afin de ne pas augmenter l'erreur de généralisation.

Un des aspects pouvant amener un modèle à être surentrainé est l'augmentation du nombre de paramètres (degré de liberté), ce qui peut résulter des approches qui tentent d'améliorer la méthode d'apprentissage en soi. Pour les réseaux de neurones, par exemple, ceci peut être la conséquence de l'ajout de neurones et de connexions comme les connexions à délais. Une alternative est d'essayer de garder le modèle le plus simple possible et de procéder par une approche plus haut niveau, qui prend principalement en considération le bon choix des données.

L'amélioration du modèle en se focalisant sur les données peut prendre deux formes. La première est une approche dans laquelle le traitement des données est fait indépendamment de l'apprentissage. Dans la deuxième, les deux phases sont dépendantes.

Généralement la phase de traitement de données est nécessaire pour pouvoir adapter les données à l'algorithme d'apprentissage. Par exemple, il est connu que les algorithmes de type descente du gradient sont sensibles à l'échelle des données, ils sont plus efficaces avec des données dans les mêmes ordres de valeur, d'où l'idée de centrer et réduire les données avant tout apprentissage. Mais dans d'autres cas, les données peuvent subir des transformations vers un nouvel espace de caractéristiques de dimension plus ou moins grande, par exemple en déduisant une nouvelle caractéristique par une combinaison d'anciennes. Cette dernière opération peut être liée au domaine d'application. Dans les deux cas, que ce soit en centrant et réduisant les données ou par construction d'espace de caractéristiques différent, l'amélioration du résultat du prédicteur est focalisée sur l'adaptation des données.

Une autre approche souvent utilisée est la sélection de caractéristiques. En effet, le résultat d'un apprentissage peut être amélioré en ajoutant des variables exogènes. Cette approche nécessite aussi une expertise afin de sélectionner les variables exogènes à ajouter pour éviter la dégradation des performances sous l'effet de la sur-dimensionnalité. La sélection de caractéristiques peut être réalisée indépendamment de l'apprentissage, tout comme le prétraitement des données, ou d'une manière dépendante des résultats du prédicteur et itérative appelée *wrapper*. Nous ne rentrerons pas dans les détails de ces techniques, le lecteur pourra consulter [Guyon *et al.*, 2006].

Le principe des méthodes d'ensemble est d'extraire un sous-ensemble des données et de répéter le processus d'apprentissage jusqu'à convergence. Le processus est itératif, l'idée est de choisir à chaque fois un sous-ensemble constitué des exemples les plus difficiles à apprendre. Nous présenterons brièvement cette approche par la suite.

Le principe de l'approche locale est de décomposer la méthode en deux phases principales. Dans la première, les données sont segmentées en différents sous ensembles. Cette segmentation se fait selon un ou plusieurs critères possibles, mais le plus fréquemment utilisé est l'indice de similarité entre les exemples (en utilisant la notion de distance). Dans la deuxième phase, un prédicteur est assigné pour chaque sous ensemble. Généralement ce processus n'est pas itératif. Par rapport aux méthodes d'ensemble, si on considère la sélection du sous-ensemble comme une segmentation avec deux classes, la différence porte sur le critère de segmentation qui est dans ce cas basé sur la difficulté de l'exemple, et sur la nature itérative de l'algorithme.

Notre intérêt se portera sur cette dernière approche, nous essaierons dans ce qui suit de dresser un bilan des différents travaux réalisés pour la segmentation des données tem-

porelles.

## 3.2 Aperçus des méthodes d'ensemble

Nous présenterons dans ce qui suit une brève description des méthodes d'ensemble. Ces méthodes consistent à combiner plusieurs hypothèses résultant des modèles d'apprentissage (par exemple les réseaux de neurones) pour obtenir l'estimation finale. Il existe plusieurs catégories de méthodes.

Le boosting, présenté dans [Schapire, 1990] et [Freund et Schapire, 1996], utilise un algorithme d'apprentissage classique comme algorithme de base. Il peut être résumé en deux étapes. La première est l'échantillonnage, c'est-à-dire la sélection des exemples qui vont être inclus dans la base d'apprentissage. Cette sélection se fait par une pondération liée à la qualité de prédiction des phases précédentes et ce en privilégiant les exemples les plus difficiles. Au départ les pondérations sont égales. Ceci est suivi d'un nouvel apprentissage et d'une mise à jour des poids à la lumière des nouvelles performances. La deuxième étape est l'intégration finale des résultats qui consiste en une combinaison faite soit par vote majoritaire ou par vote pondérée dans le cas de la classification [Freund et Schapire, 1996] soit par médiane pondérée ou par moyenne pondérée dans le cas de la régression [Drucker et Branch, 1997] [Drucker, 1999] (figure 3.1).

L'algorithme de bagging appelé aussi *bootstrap aggregating* [Breiman, 1996a] utilise plusieurs modèles de classifieurs ou régresseurs, chacun apprend sur un sous-ensemble de l'ensemble de données d'origine qui est obtenu par un tirage aléatoire avec remise. À la fin de l'apprentissage, les modèles résultant sont combinés par vote majoritaire ou par moyenne selon la nature du problème (classification ou régression) comme le montre la figure 3.2.

Le bagging et le boosting représentent les méthodes les plus utilisées dans la littérature. Le fait que la procédure d'apprentissage est appliquée sur des sous-ensembles de l'original est un point commun avec les approches locales. La principale différence réside dans la manière de former les sous-ensembles. En effet, dans les approches locales, l'idée est de créer ces sous-ensembles à partir du principe de maximum de vraisemblance.

Nous pouvons citer d'autres méthodes comme le *stacking* proposé dans [Wolpert, 1992] et [Breiman, 1996b] qui est une méthode organisée en couches. Dans le premier niveau, plusieurs modèles sont entraînés parallèlement, produisant ainsi des hypothèses différentes. Dans le second niveau, un nouveau modèle est utilisé avec l'ensemble de données plus les différentes hypothèses produites afin d'obtenir l'hypothèse finale. Le cascading [Gama, 1998] est une autre méthode qui ressemble à la précédente et qui entraine séquentiellement les modèles par niveau en utilisant un modèle à la fois. Des détails plus complets peuvent être trouvés dans [Tuv, 2006].

## 3.3 Principe des approches locales

Nous avons vu précédemment que des méthodes de type boosting et bagging effectuent un apprentissage sur un sous-ensemble des données, ce qui constitue un point commun

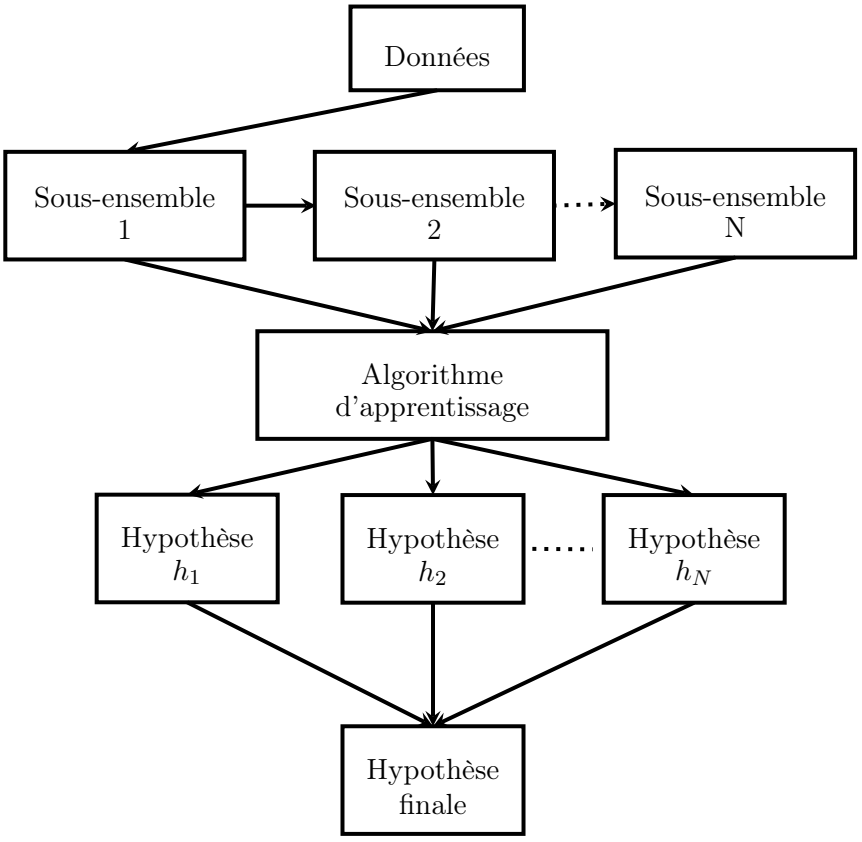


FIGURE 3.1 – Schéma de l'algorithme de boosting

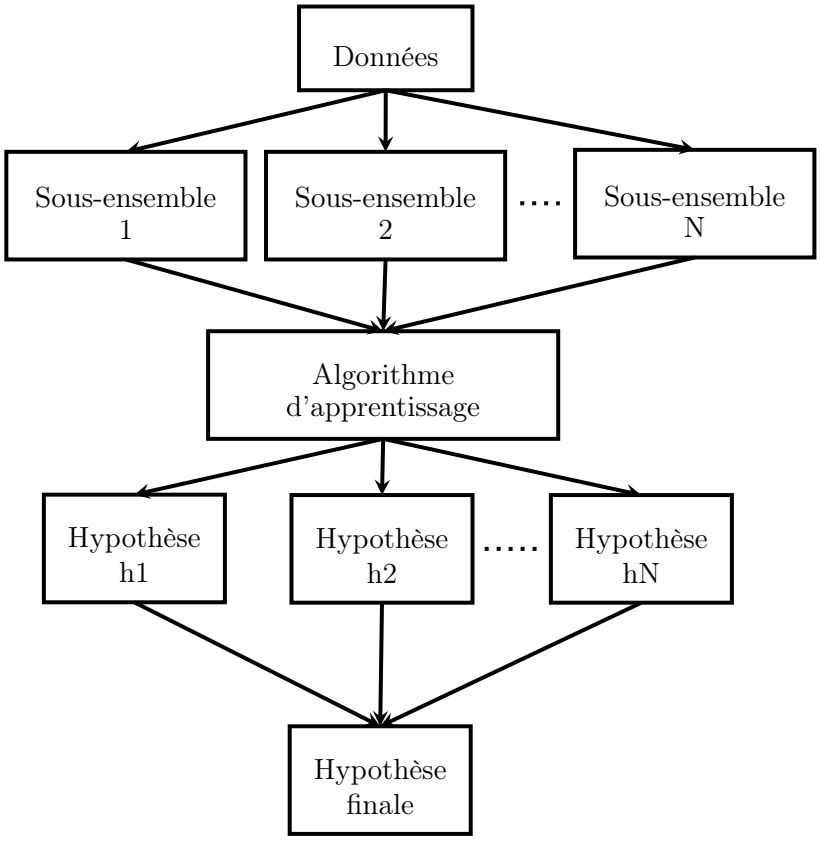


FIGURE 3.2 – Schéma de l'algorithme de bagging

avec les approches locales. Ces dernières peuvent être décrites sous forme d'un processus en trois étapes.

Dans la première, un ensemble de dimension $M$ est construit à partir de la série temporelle d'origine. Ce nouvel espace vectoriel est obtenu en faisant glisser une fenêtre temporelle sur les données. La deuxième étape consiste à segmenter (*clustering*) l'ensemble d'apprentissage en se basant sur le principe de similarité, par exemple en utilisant des algorithmes de quantification vectorielle. La technique de quantification vectorielle permet la réduction de larges volumes de données à un nombre réduit de prototypes. La troisième et dernière étape est celle de la prédiction. Un prédicteur est associé à chacun des clusters pour effectuer l'apprentissage. Le processus est décrit par la figure 3.3.

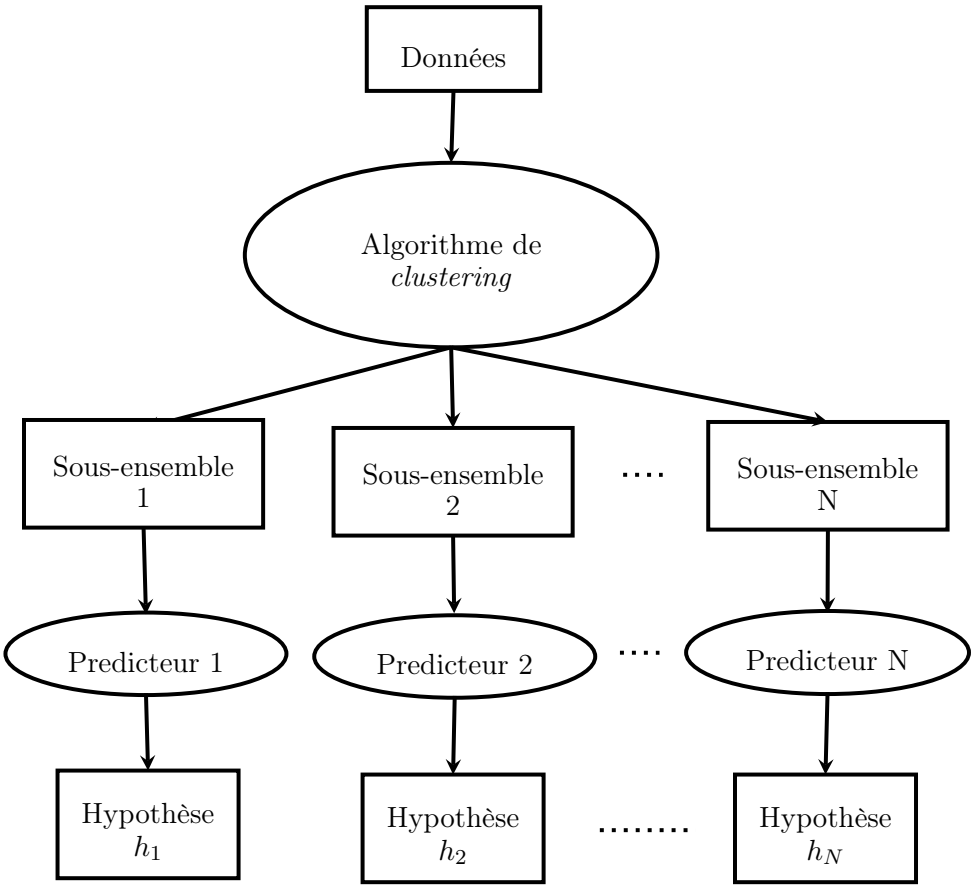


FIGURE 3.3 – Schéma de l'algorithme de l'approche locale

Contrairement aux méthodes de boosting et de bagging, l'hypothèse finale est construite par l'union des hypothèses obtenues pour chaque cluster. Ce rassemblement se fait d'une manière naturelle puisque tous les clusters sont naturellement disjoints. Le principal avantage d'une telle approche réside dans la simplification de la fonction à estimer en la divisant en plusieurs fonctions émanant de groupes plus au moins homogènes. Cette technique permet aussi d'identifier des comportements saisonniers de la série temporelle.

Dans la suite nous distinguerons deux grandes familles d'approche locale en se basant sur la nature de l'algorithme de segmentation appliqué la segmentation partitionnel ou la segmentation hiérarchique.

## 3.4 Clustering partitionnel

Par algorithme de segmentation partitionnel nous faisons référence aux algorithmes produisant des clusters se situant sur un seul niveau horizontal sans structure apparente entre les clusters. Ces techniques de partitionnement produisent des clusters par l'intermédiaire d'un critère local défini à partir des sous-ensembles ou d'un critère global défini à partir de tous les exemples. En pratique ce sont des algorithmes itératifs, on s'intéressera uniquement à ceux basés sur l'erreur quadratique tels que les *k-Means* ou les Cartes de Kohonen. Nous pouvons formaliser l'erreur quadratique d'un clustering $\mathcal{K}$ contenant $K$ clusters et d'un ensemble de vecteurs $\mathcal{E}$ par l'équation 3.1.

$$e^2(\mathcal{E}, \mathcal{K}) = \sum_{i=1}^{K} \sum_{j=1}^{n_i} \left\| \mathbf{x}_i^{(j)} - \mathbf{c}_j \right\|^2 \tag{3.1}$$

où $\mathbf{x}_i^{(j)}$ est le $j^{ieme}$ vecteur dans le cluster $i$, $n_i$ sa taille et $c_j$ le vecteur prototype du $j^{ieme}$ cluster.

### 3.4.1 Quantification Vectorielle

La quantification vectorielle est une méthode classique dont l'objectif initial est la réduction des grands volumes de données [Gersho et Gray, 1992] [Grossberg, 1987]. Elle permet, d'une part, de réduire un ensemble des données vectorielles tout en limitant la perte d'information subie lors de cette réduction et, d'autre part, de créer des partitions chacune représentée par un noyau appelé aussi prototype. Ainsi, le résultat d'une quantification vectorielle peut être résumé à un ensemble de vecteurs disposés à l'intérieur d'une distribution de données de manière à représenter à eux seuls l'ensemble de la distribution initiale.

Nous pouvons formaliser l'opération en considérant un ensemble de départ $\mathcal{D}$ contenant $N$ exemples de dimension $d$, défini par l'équation 3.2 :

$$\mathcal{D} = \left\{ \mathbf{x}_i | \mathbf{x}_i = \left( x_i^1, x_i^2, \ldots, x_i^d \right) \in \mathbb{R}^d, 1 \leq i \leq N \right\}, \tag{3.2}$$

où les $x_i^j$ sont les différentes composantes du vecteur $\mathbf{x}_i$ avec $1 \leq j \leq d$.

Après quantification vectorielle de l'ensemble $\mathcal{D}$, on dispose d'un ensemble $\mathcal{P}$ de $n$ vecteurs prototypes où $n$ est un nombre très inférieur à $N$. Bien évidemment, ces prototypes sont de même dimension que les exemples qu'ils représentent. L'ensemble $\mathcal{P}$ peut être formalisé par l'équation 3.3.

$$\mathcal{P} = \left\{ \mathbf{y}_i | \mathbf{y}_i = \left( y_i^1, y_i^2, \ldots, y_i^d \right) \in \mathbb{R}^d, 1 \leq i \leq n \right\}, \tag{3.3}$$

où les $y_i^j$ sont les différentes composantes du vecteur $\mathbf{y}_i$ avec $1 \leq j \leq d$.

Un des algorithmes les plus connus et simples qui implémentent la technique de quantification vectorielle est l'apprentissage compétitif, en anglais *Competitive Learning*, appelé aussi *K-means*. Cet algorithme d'apprentissage non supervisé est en mesure d'adapter la position d'un ensemble de prototypes à une distribution d'un ensemble de données $\mathcal{D}$ par un procédé itératif.

Au cours d'une itération, un exemple $x_i$ est traité, le prototype le plus proche au sens d'une mesure de distance $dist(.,.)$, noté $\mathbf{p}_k$, est sélectionné :

$$\mathbf{p}_k = \min_{1 \leq j \leq n} dist(\mathbf{p}_j, \mathbf{x}_i) \tag{3.4}$$

où $dist(\mathbf{p}_j, \mathbf{x}_i)$ est la mesure de distance entre les vecteurs $\mathbf{x}_i$ et $\mathbf{p}_j$, le plus souvent la distance euclidienne.

Les positions des prototypes $\mathbf{p}_k$ sont ensuite modifiées au sein de la distribution pour qu'ils soit plus représentatifs des données considérées. En d'autre termes, on rapproche les prototypes des données pour que les nouvelles positions tiennent compte des informations contenues dans la distribution des données (voir équation 3.5).

$$\mathbf{p}_k(t+1) = \mathbf{p}_k(t) + \alpha(t)(\mathbf{x}_i - \mathbf{p}_k(t)) \tag{3.5}$$

où $t$ représente l'itération en cours et $\alpha$ le taux d'apprentissage, généralement choisi comme une fonction décroissante du nombre d'itérations, afin d'assurer la convergence de l'algorithme.

Les deux étapes sont répétées sur l'ensemble de données $\mathcal{D}$. À la fin de l'exécution de cet algorithme, pour chaque vecteur $\mathbf{p}_j \in \mathcal{P}$, on définit un sous-ensemble de $\mathcal{D}$ regroupant toutes les données $\mathbf{x}_i$ qui sont plus proches de ce prototype que de tous les autres. Ces groupes de données sont appelés cluster et noté $C_j$ où l'indice $j$ correspond au prototype $\mathbf{p}_j$ correspondant. Formellement pour un prototype $\mathbf{p}_j$, un cluster est défini par l'équation 3.6 :

$$C_k = \left\{ \mathbf{x}_i \in \mathcal{D} | \forall j \in [1..n], j \neq k, dist(\mathbf{p}_k, \mathbf{x}_i) \leq dist(\mathbf{p}_j, \mathbf{x}_i) \right\} \tag{3.6}$$

Notons que la création des clusters $C_j$ correspondant aux différents prototypes $\mathbf{p}_j$ de $\mathcal{P}$ revient à partitionner $\mathcal{D}$.

L'algorithme des K-means et certainement la méthode la plus simple pour le clustering des données mais elle possède certains défauts. Le résultat final est fortement dépendant de l'initialisation d'où une très forte chance de tomber dans des minima locaux. Il est inadapté aux données séquentielles comme les séries temporelles. Ces problèmes sont décrits dans [Zhang *et al.*, 2004] où des K-Means sont combinés avec un algorithme génétique. Une autre approche présentée par [Vlachos *et al.*, 2003] consiste à utiliser les K-Means sur des séries temporelles après une décomposition en ondelettes. L'idée est de profiter de la propriété multirésolution de cette transformation pour effectuer un clustering par niveau. Au départ un clustering est effectué sur une résolution grossière à partir des données. Les

résultats du clustering sont utilisés pour l'initialisation de l'étape suivante à un niveau de décomposition supérieur. Ce processus est répété jusqu'à stabilisation ou jusqu'à atteindre les données originales.

Les propositions décrites ci-dessus traitent plutôt la problématique de l'initialisation dans le cadre du clustering des séries temporelles mais sans prendre en compte la nature séquentielle qui sont des données temporellement liées. Dans [Chakrabarti *et al.*, 2006] une proposition de K-Means adaptée est présentée, modifiant le calcul des nouvelles valeurs des centroïdes. En effet, après chaque itération, ces derniers sont calculés en combinant les centroïdes obtenus par le K-Means à l'itération en cours avec ceux des itérations précédentes. Le tout est pondéré en fonction de la taille des clusters et par un paramètre utilisateur d'échange *cp*.

Pour une description plus formelle, on définit le coût historique entre clustering $C$ et $C^{'}$ par l'équation 3.7 :

$$hc(C, C^{'}) = \min_{f} \left\| \mathbf{c}_i - \mathbf{c}^{'}_{f(i)} \right\| \tag{3.7}$$

où $f$ est une fonction qui réalise *mappage* entre les centroïdes du clustering $C$ et les centroïdes de $C'$ de la meilleure manière possible. On désigne par $\mathbf{c}_j^t$ le centroïde relatif au cluster $j$ à l'instant $t$ et par $\mathbf{c}_{f(j)}^{t-1}$ le centroïde le plus proche de $\mathbf{c}_j^t$. De la même manière, $n_j^t$ représente le nombre de points dans le cluster $j$ à l'instant $t$ et $n_{f(j)}^{t-1}$ est le nombre correspondant au cluster $j$ à l'instant $t-1$. La mise à jour des centroïdes peut être formulée par l'équation 3.8.

$$\mathbf{c}_j^t = (1-\gamma)cp.\mathbf{c}_{f(j)}^{t-1} + \gamma(1-cp)\mathbf{p}_t \tag{3.8}$$

où

$$\gamma = n_j^t/(n_j^t + n_{f(j)}^{t-1}) \tag{3.9}$$

et $\mathbf{p}_t$ est le prototype découvert par l'algorithme de K-Means au cours d'une itération selon l'équation 3.5.

### 3.4.2 Présentation des cartes auto-organisées

Tout comme les K-Means, l'algorithme des cartes auto-organisées, en Anglais *Self Organizing Maps* (SOM), est un algorithme itératif de classification non supervisée proposé par Kohonen [Kohonen, 1982]. Les SOM ont été utilisées avec succès dans de très nombreuses applications comme la classification de texte, la classification d'image et la robotique. Nous aborderons dans ce qui suit les principaux concepts ainsi qu'une description du fonctionnement de l'algorithme. Les fondements théoriques de cette méthode peuvent être trouvés dans [Horowitz et Alvarez, 1995], [Cottrell, 1998] et [Flanagan, 1998].

L'algorithme SOM est très proche de l'apprentissage compétitif. Les prototypes se déplacent au sein de la distribution de données de l'ensemble $\mathcal{D}$ mais la particularité de cet algorithme consiste en l'utilisation d'une grille reliant les différentes unités entre elles. Ces unités (ou neurones par correspondance aux réseaux de neurones) possèdent des prototypes qui sont représentés sous la forme de vecteurs poids.

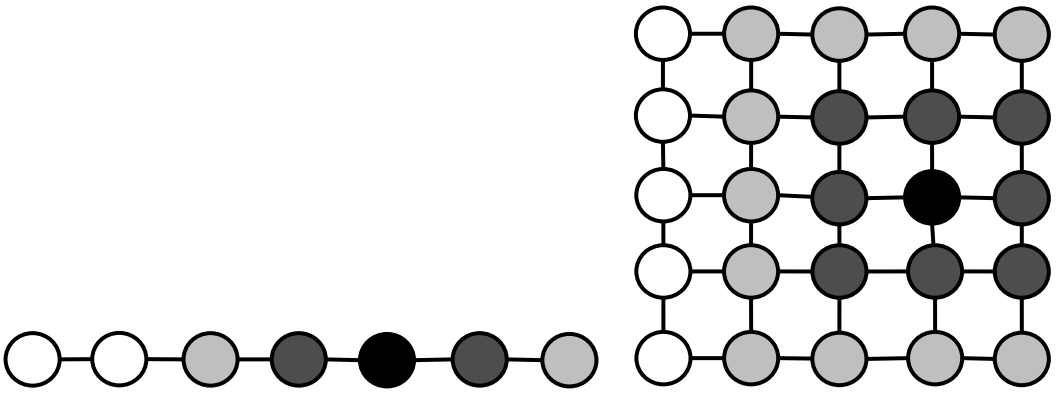

FIGURE 3.4 – Exemples de grille unidimensionnelle et bidimensionnelle, et les relations de voisinage entre les unités

La figure 3.4 représente deux topologies de grilles parmi les plus utilisées. La première est une grille unidimensionnelle linéaire, la deuxième est une grille bidimensionnelle rectangulaire. Dans les deux exemples, des ronds reliés entre eux par des segments de droite représentent les neurones ou unités. L'intérêt principal d'une telle structure est de pouvoir définir une relation de voisinage entre les vecteurs prototypes. La relation de voisinage peut être décrite de différentes manières. Prenons le cas où le voisinage est limité à deux neurones les plus proches (figure 3.4). L'unité en noir est notre centre d'intérêt. Dans le cas linéaire, les deux unités directement liées à cette unité (en gris foncé) représente le voisinage d'ordre 1 avec une distance égale à 1. Les unités en gris clair représentent le voisinage d'ordre 2 avec une distance égale à 2, tandis que les unités en blanc sont considérées en dehors du voisinage. Similairement, dans le modèle de grille bidimensionnelle, les unités en gris foncé aux alentours et qui encerclent directement l'unité en noir, le centre d'intérêt, sont le premier voisinage à une distance de 1. Le niveau 2 est représenté en gris clair et les unités en blanc sont considérés en dehors du voisinage. Dans cet exemple, nous avons imaginé aussi un système de coordonnées dans un espace de dimension 1 ou 2 où la distance se calcule par la mesure euclidienne classique. Ceci n'est pas l'unique façon de considérer le voisinage, une des représentations possibles qui ont été faites dans la littérature est la carte circulaire dont les deux extrémités sont reliées. La notion de distance doit être adaptée afin de prendre en compte cette géométrie. On peut considérer, à titre d'exemple, le nombre minimal de connexions entre les unités. La notion de limite de voisinage est arbitraire et peut être considérée comme un paramètre utilisateur ou ignorée dans de nombreux cas, en guise de simplification du modèle.

L'algorithme des SOM se déroule en deux phases : une phase d'apprentissage et une phase de test (ou d'utilisation). Durant la phase d'apprentissage, les données de l'ensemble $\mathcal{D}$ sont présentées, itération par itération, successivement à l'algorithme. La première opération est la même que pour le *competitive learning* (voir section 3.4.1). Elle consiste à trouver le vecteur poids $\mathbf{w}_k$ le plus proche de l'exemple $\mathbf{x}_i$ (équation 3.10).

$$\mathbf{w}_k = \min_{1 \leq j \leq n} dist(\mathbf{w}_j, \mathbf{x}_i) \tag{3.10}$$

L'opération suivante est la mise à jour du prototype $\mathbf{w}_k$, mais dans le cas des SOM, les autres vecteurs poids voisins $\mathbf{w}_j$ sont eux aussi adaptés en fonction des propriétés du voisinage imposées par la grille. L'équation 3.11 représente la formule d'apprentissage généralement utilisée pour la mise à jour des unités de la grille :

$$\mathbf{w}_j(t+1) = \mathbf{w}_j(t) + \alpha(t) h_{jk}(t)(\mathbf{x}_i - \mathbf{w}_j(t)) \tag{3.11}$$

où $\mathbf{w}_j$ est le $j^{ieme}$ vecteur poids de la grille, $\mathbf{x}_i$ est le $i^{ieme}$ exemple de $\mathcal{D}$.

Les composantes $\alpha(t)$ et $h_{jk}(t)$ sont respectivement le taux d'apprentissage et la fonction de voisinage du SOM.

Le taux d'apprentissage est une fonction décroissante qui assure la convergence de l'algorithme. Dans la plupart des applications, elle suit une tendance décroissante exponentielle. L'idée derrière ce choix est que le déplacement des prototypes de la grille vers la donnée $\mathbf{x}_i$ est autant plus important que la valeur de $\alpha(t)$ est importante. Au départ, la carte est initialisée aléatoirement, ce qui justifie le fait d'effectuer des modifications importantes au début. Plus l'algorithme avance dans le temps, plus il approche d'une solution, d'où la nécessité d'affiner le pas d'apprentissage.

La fonction de voisinage $h_{jk}(t)$ est une fonction qui dépend directement de la distance entre les deux unités de la carte (l'unité du prototype le plus proche d'indice $k$ et l'unité à modifier d'indice $j$), mais aussi de la borne de la limite qui est supposée être grande au départ de l'algorithme, puis tend à diminuer et éventuellement à être nulle dans les dernières itérations. En effet, les vecteurs poids étant initialisés aléatoirement, il est utile d'influencer le plus de prototypes possibles. Par la suite, les prototypes changerons avec un écart de plus en plus petit, de même les ensembles obtenus varient de moins en moins jusqu'à devenir locaux ou quasi locaux. Une des manières d'implémenter cette fonction est de se baser sur une gaussienne dont la largeur décroît au cours du temps :

$$h_{jk}(t) = \exp^{-(\frac{dist(u_j, u_k)^2}{\sigma(t)^2})} \tag{3.12}$$

À la fin de l'exécution de l'algorithme SOM, on obtient un ensemble de prototypes $\mathcal{P}$, comme dans la section 3.4.1. Ce nouvel ensemble respecte bien à la fois la caractéristique de quantification vectorielle et celle de topologie. La première est respectée car, d'une part, le nombre de prototypes est largement plus petit que le nombre total de données dans $\mathcal{D}$. D'autre part, chacun des vecteurs poids $w_i$ est assimilé à un centroïde d'un cluster (voir 3.4.1). La deuxième caractéristique réside dans la forme de la grille et la notion de voisinage. En matière de résultat, deux données similaires au sein de la distribution initiale se trouveront, après convergence, soit dans le même cluster soit dans deux clusters voisins.

La qualité des résultats est influencée par deux paramètres principaux : la topologie de la carte et le nombre de prototypes. Les formes les plus souvent utilisées sont la grille linéaire unidimensionnelle et la grille bidimensionnelle rectangulaire (voir figure 3.4). Le choix est guidé par la connaissance de la distribution de données. Le nombre de prototypes est aussi un critère important. Il influence l'exécution de l'algorithme SOM : plus le nombre de

prototypes augmente, plus le temps de convergence augmente, plus on a de chance d'avoir un grand nombre de clusters de petite taille. Inversement, avoir un petit nombre de clusters peut nous amener à obtenir des clusters peu homogènes. Ceci est bien évidemment très dépendant de la connaissance à priori de la distribution des données.

### 3.4.3 Adaptation des SOM pour les séries temporelles

Pour la prévision des séries temporelles, le but est de prédire dans chacun des clusters. Il est tout à fait logique de viser un compromis entre la taille du cluster et l'homogénéité des données. Dans cette partie, nous nous intéresserons tout particulièrement aux applications des SOM sur les données séquentielles.

Dans [Walter *et al.*, 1990] une approche pour la prévision de séries fortement non linéaires par le modèle linéaire $AR(p)$ est présentée. Cette technique appelée Local Linear Model (LLM) consiste à utiliser les SOM pour le clustering des séries temporelles, le but étant d'obtenir des clusters suffisamment petits pour que la condition linéaire soit le plus valable.

Dans [Vesanto, 1997] la même technique a été reprise, utilisant les SOM et le modèle linéaire $AR(p)$ pour la prévision des séries chaotiques. A titre d'illustration des capacités de prévision sur de telle données, des expérimentations sur la série Mackey-Glass ont été effectuées. Les résultats ont montré que les modèles localement linéaires étaient compétitifs et même parfois meilleurs que d'autres prédicteurs plus complexes.

Une modification de cette approche est proposée dans [Barreto *et al.*, 2004] et appelée KSOM. Les deux approches cités précédemment utilisent l'algorithme de clustering uniquement dans le but de séparer les données. KSOM construit les modèles locaux en utilisant uniquement les $K$ vecteurs des unités gagnantes comme vecteur d'entrée, puisque ces derniers représentent les prototypes des données d'apprentissage. Selon l'auteur cette technique est moins couteuse en temps de calcul et procure une plus grande généralisation et robustesse.

Dans [Fu *et al.*, 2001], l'algorithme des cartes de Kohonen est utilisé afin d'identifier des modèles (*patterns*) dans les séries temporelles. Le principe est d'utiliser des SOM avec des grilles bidimensionnelles de taille variable entre $5 \times 5$ et $10 \times 10$ sur des échantillons de séries temporelles composés de 30 valeurs passées.

Dans [Noelia Sánchez-Maroño *et al.*, 2003], la même approche est utilisée en remplaçant les $AR(p)$ par des réseaux fonctionnels ( [Castillo *et al.*, 1998]) pour la phase de prédiction. Les expérimentations ont été réalisées sur les séries standard MG-17 et Laser.

Dans [Martin-Merino et Roman, 2006], des prédicteurs locaux, SVM, sont utilisés pour la prévision de données de consommation électrique. La méthode utilise des SOM classiques pour le partitionnement des données avec l'utilisation d'une carte unidimensionnelle circulaire, dont les deux extrémités sont reliées. Ce choix est guidé par la nature cyclique des données de consommation électrique, fortement influencées par les saisons de l'année.

Proposée dans [Simon *et al.*, 2004] ou encore dans [Simon *et al.*, 2007], la double quantification vectorielle est une autre approche qui utilise les SOM pour la prévision des séries locales temporelles. Dans cette méthode, deux grilles (ou cartes) sont utilisées séparément. La première quantification vectorielle est réalisée sur un vecteur $\mathbf{x}(t)$ contenant $d + 1$ va-

leurs antérieures (équation 3.13). La deuxième est faite sur le même vecteur auquel on ajoute une valeur future, c'est-à-dire une quantification dans un espace de dimension $d+2$ (équation 3.14).

$$\mathbf{x}(t) = (x(t), x(t-1), .., x(t-d)) \tag{3.13}$$

$$\mathbf{x}^{'}(t) = (x(t+1), x(t), x(t-1), .., x(t-d)) \tag{3.14}$$

Ceci permet d'établir une matrice de transition, qu'on notera $T = [t_{ij}]$, calculée selon l'équation 3.15. Le but est d'établir une relation entre le clustering $C$ de la première carte de Kohonen et le deuxième clustering $C^{'}$ effectué sur les vecteurs $\mathbf{x}^{'}(t)$.

$$t_{ij} = \frac{Card(\{\mathbf{x}(t) \in c_i | \mathbf{x}^{'}(t) \in c_j\})}{Card(\{\mathbf{x}(t) \in c_i\})} \tag{3.15}$$

où $Card(.)$ signifie le cardinal de l'ensemble. La partie au numérateur traduit le nombre d'éléments $\mathbf{x}(t)$ appartenant à $c_i$ (issu du premier clustering) tel que $\mathbf{x}^{'}(t)$ appartient à $c_j$ (issu du second clustering). Ensuite, pour chaque cluster $c_j$ de $C^{'}$ on associe un prédicteur et on y effectue un apprentissage. Les prédictions de l'ensemble de tests sont déterminées alors par combinaison linéaire des valeurs prédites par chacun de ces prédicteurs pondérées par les valeurs $t_{ij}$. Par exemple, pour un vecteur $\mathbf{x}(t) \in c_i$ la valeur prédite est déterminée par l'équation 3.16.

$$x(t+1) = \sum_{j=1}^{N^{'}} t_{ij} x_i^j(t+1) \tag{3.16}$$

où $x_i^j(t+1)$ représente la valeur obtenue par le prédicteur correspondant aux cluster $c_i$ et $c_j$ et $N^{'}$ est le nombre de clusters de $c_j$.

Même si la valeur de prédiction finale est obtenue par combinaison des prédictions locales, contrairement aux méthodes précédentes, on considère qu'on est toujours dans le cadre de prédiction locale puisque chaque prédicteur apprend sur un sous-ensemble des données.

Toutes les méthodes citées ci-dessus ont en commun l'utilisation de l'algorithme des SOM. Il paraît clair que l'élément clé de l'utilisation de cet algorithme est la recherche de clusters qui soient les plus homogènes dans leurs compositions. Cependant, bien que les SOM s'avèrent un algorithme très utile et efficace pour ce type d'application, ils sont conçus à la base pour des données qui ne possèdent pas de spécificité temporelle. Pour cette raison, on trouve plusieurs adaptations de l'algorithme pour les séries temporelles ou plus généralement les données séquentielles. Dans la section suivante, nous ferons le tour de certaines de ces adaptations des SOM.

### 3.4.4 Variantes des cartes auto-organisées

En cherchant les adaptations de l'algorithme SOM pour le traitement de données séquentielles, la littérature nous fait émerger deux approches principales, chacune donnant

naissance à une famille de méthodes. Une première consiste à effectuer un traitement récursif du signal entrant. Donc les résultats des itérations précédentes sont prises en compte pour déterminer le prototype gagnant. Une deuxième approche consiste à faire un *mappage* des dépendances temporelles vers des corrélations spatiales. Ainsi la relation de séquentialité entre des données est traduite par des relations spatiales entre les éléments de la carte en ajoutant de nouvelles conditions ou de nouvelles informations.

#### 3.4.4.1 Variantes à traitement récursif

**Temporal Kohonen Map**

Les cartes de Kohonen temporelles, en anglais *Temporal Kohonen Map* (TKM), sont la première proposition de modification de l'algorithme SOM spécialement pour les séquences temporelles [Chappell et Taylor, 1993]. L'algorithme utilise une valeur d'activation afin de déterminer le prototype le plus proche pour un exemple, à l'image de l'algorithme SOM classique :

$$u_k(t,d) = \min_{i \in V} \|u_i(t-1,d)\| \tag{3.17}$$

Cette valeur d'activation, notée $u_k(t,d)$, est définie par une somme récursive comme le montre l'équation 3.18 :

$$u_i(t,d) = d.u_i(t-1,d) - \frac{1}{2}\|\mathbf{x}(t) - \mathbf{w}_i(t)\|^2 \tag{3.18}$$

où $\mathbf{x}(t)$ représente l'exemple d'entrée à l'instant $t$ et $\mathbf{w}_i(t)$ représente le vecteur de poids correspondant à l'unité $i$ de la carte. $d$ est un paramètre utilisateur, $0 \leq d \leq 1$, de manière à ce que plus $d$ tend vers 0, plus on se ramène à l'algorithme classique des SOM. Inversement, plus $d$ tend vers 1, plus les anciennes valeurs d'activation sont influentes dans le choix du prototype le plus proche de l'exemple $\mathbf{x}(t)$. On peut trouver une application de cet algorithme pour la prédiction de données séquentielles dans [Gonçalves *et al.*, 2010], pour déterminer des défauts dans des valves électriques.

**Recurrent Self Organizing Maps**

Les cartes auto-organisées récurrentes [Varsta *et al.*, 2001], en anglais *Recurrent Self Organizing Maps* (RSOM), sont une conséquence directe des TKM. L'idée est de remplacer la valeur d'activation (qui est un scalaire) par un vecteur différentiel noté $\mathbf{y}_i(t,\alpha)$ défini par l'équation 3.19.

$$\mathbf{y}_i(t,\alpha) = (1-\alpha)\mathbf{y}_i(t-1,\alpha) + \alpha(\mathbf{x}(t) - \mathbf{w}_i(t)) \tag{3.19}$$

où $\alpha$ joue un rôle analogue à $(1-d)$ dans la TKM. Ainsi, le critère pour déterminer le prototype le plus proche à chaque itération doit lui aussi changer. Dans le cas des TKM c'était le minimum des activations, ici c'est le maximum des normes des vecteurs différentiels de la carte :

$$y_k(t,\alpha) = \min_{i \in V} \|\mathbf{y}_i(t-1,\alpha)\|^2 \tag{3.20}$$

RSOM représente une amélioration de TKM. En effet les expériences montrent que dans TKM les exemples ont tendance à se concentrer dans les extrémités de la carte. Alors que RSOM, selon des résultats expérimentaux obtenus dans [Varsta *et al.*, 2001], utilise une règle d'apprentissage plus simple et plus adaptée au fonctionnement du modèle (voir l'équation 3.19). On peut trouver dans la littérature des applications de RSOM dans une expérience sur des données EEG [Varsta *et al.*, 1997], ou encore pour la prévision des séries temporelles MG-17 et Laser avec comme prédicteur des modèles localement linéaires [Koskela *et al.*, 1997].

#### Cartes Auto-Organisée Récursives

Les cartes auto-organisée récursives [Voegtlin, 2002], en anglais *RECursive Self-Organizing Maps* (RecSom), représentent une autre variante des cartes de Kohonen. Malgré une appellation similaire à RSOM (voir section 3.4.4.1), une approche tout à fait différente est mise en œuvre. Cette méthode utilise l'algorithme classique SOM tout en gardant à chaque itération une copie de la carte obtenue à l'itération précédente, comme le montre la figure 3.5.

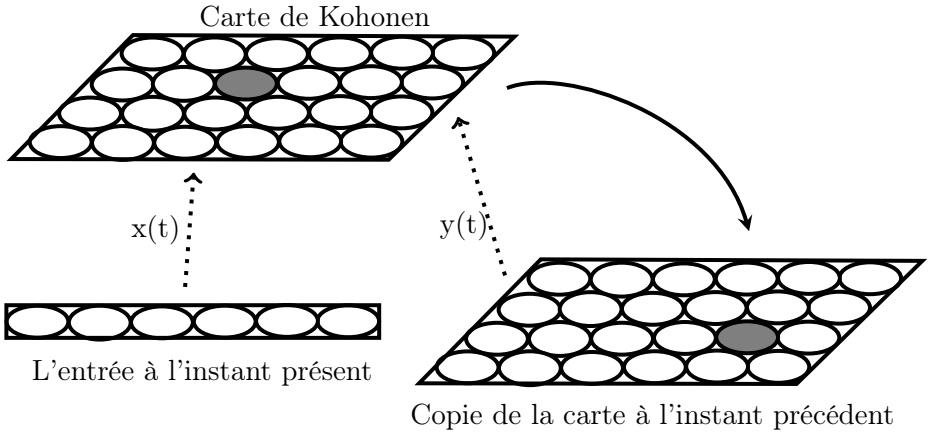


FIGURE 3.5 – Schéma de disposition des cartes dans RecSOM

La carte située au-dessus est la carte habituellement utilisée dans SOM, celle à droite représente une copie de la première à l'itération précédente. Le vecteur $\mathbf{x}(t)$ représente le vecteur exemple en entrée à l'instant $t$. Le vecteur $\mathbf{y}(t)$, dénommé vecteur d'activation de la carte à l'instant $t$, a, pour des raisons de stabilité, ses valeurs calculées en fonction de l'erreur de quantification $E_i$. Ainsi pour chaque unité, on définit deux vecteurs poids $\mathbf{w}_i^x$ et $\mathbf{w}_i^y$ représentant respectivement la valeur du vecteur poids sur la carte de l'itération en cours et sur la carte de l'itération précédente (voir figure 3.5).

L'algorithme RecSOM utilise la même règle d'apprentissage pour la modification des vecteurs poids, cependant la manière de trouver le neurone le plus proche pour chacun des

exemples diffère. L'unité gagnante à un instant $t$ est déterminée en minimisant l'erreur de quantification $E_i$ définie par l'écart entre les exemples et les prototypes (équation 3.21).

$$E_i = \alpha \left\| \mathbf{x}(t) - \mathbf{w}_i^x \right\|^2 + \beta \left\| \mathbf{y}(t-1) - \mathbf{w}_i^y \right\|^2 \tag{3.21}$$

avec $\alpha \geq 0$ et $\beta \geq 0$. Le choix de ces paramètres est important pour la qualité de l'apprentissage ainsi que pour la stabilité de l'algorithme.

L'avantage de cette méthode réside dans l'utilisation de valeurs de feedback pour représenter le temps tout en gardant la nature auto-organisatrice de l'algorithme. Cela permet à la méthode d'être implicite et autonome. La principale difficulté se trouve dans le choix de la fonction de transfert qui est crucial pour les performances. Les expériences faites avec la série temporelle MG-17 montrent que RecSOM disposent d'une erreur de quantification significativement inférieure comparativement à SOM ou encore à RSOM.

### Feedback SOM

Les cartes auto-organisatrices à couche de contexte [Mizushima, 2006], appelée en anglais *Feedback SOM* (FSOM), s'inspirent des réseaux de neurones récurrents d'Elman. L'architecture utilisée rappelle celle de RecSOM (voir figure 3.5) vue précédemment par l'utilisation des deux grilles. La première représente la couche compétitive, c'est-à-dire celle où est déterminée l'unité gagnante à chaque exemple, en anlgais *Best Muching Unit* (BMU). La deuxième joue le rôle de la couche de contexte par analogie au réseau d'Elman.

Il est aussi à noter que chaque connexion entre les unités est pondérée à la manière des réseaux de neurones et que les unités possèdent une valeur scalaire et non un vecteur poids (ou prototype) comme dans les approches SOM classiques (voir figure 3.6). Les principales différences par rapport à la méthode RecSOM précédemment citée se trouvent dans la manière de trouver l'unité gagnante à chaque itération ainsi que dans la mise à jour des deux couches, et plus particulièrement de la couche de contexte.

L'unité gagnante de la carte compétitive est déterminée en minimisant une combinaison linéaire constituée de l'exemple d'entrée $X(t)$ pondéré par les poids sur les connexions correspondantes et des valeurs dans la couche de contextes pondérées par les poids sur les connexions correspondantes. Une fois l'unité gagnante déterminée, les deux cartes sont mises à jour de manière parallèle et similaire à la formule classique des SOM. De plus, les valeurs de la couche de contexte sont atténuées à la fin de chaque itération par un ratio $\beta$ compris entre 0 et 1.

Des détails supplémentaires sur l'algorithme et la simulation effectuée peuvent être trouvés dans [Mizushima, 2006]. Cette approche semble être plus adaptée à des applications de traitement de langage qu'à les séries temporelles ou séquentielles. En observant les architectures de ces deux dernières méthodes (figure 3.5, et 3.6) on remarque bien une ressemblance avec les RNN, avec un *feedback* de l'information utile pour le traitement des données séquentielles.

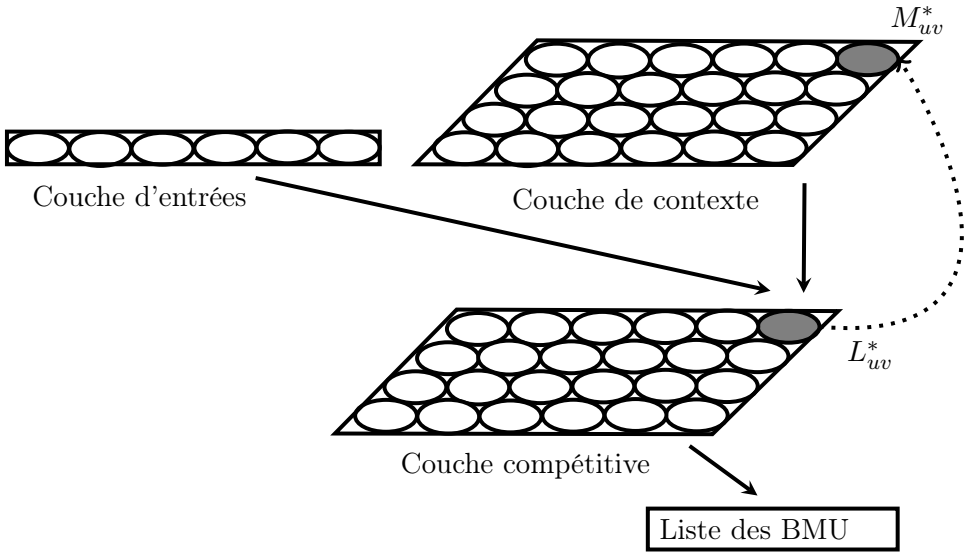


FIGURE 3.6 – Schéma de disposition des cartes dans FSOM

**SARDNET**

SARDNET [James, 1995], *Sequential Activation Retention and Decay NETwork*, est une méthode qui utilise des sous-ensembles de carte avec un principe de rappel et d'oubli d'activation permettant de représenter des séquences de vecteurs.

L'architecture est composée d'une couche d'entrée constituée de $n$ nœuds, chacun de ces nœuds est associé à une composante du vecteur d'entrée. La carte possède $m \times m$ unités avec une valeur d'activation $o_{ij}$ et un vecteur poids (ou prototype, pour garder la même dénomination des SOM) de même dimension que les vecteurs d'entrées.

Le choix de l'unité gagnante de la carte se fait comme pour les SOM, à une exception près. Une unité, une fois sélectionnée, n'est plus éligible pour les itérations suivantes (c'est-à-dire pour être gagnante), de plus sa valeur d'activation décroît progressivement.

De même la règle d'apprentissage est identique à celle utilisée par l'algorithme classique pour modifier le vecteur prototype gagnant et son voisinage. La valeur d'activité du prototype gagnant est réinitialisée à 1, cette valeur est mise à jour à chaque itération de façon à diminuer progressivement selon l'équation :

$$o_{ij}(t+1) = d.o_{ij}(t) \tag{3.22}$$

où $d$ représente un facteur d'oubli tel que $0 \leq d \leq 1$.

L'avantage de cette méthode est de permettre l'impact d'un large nombre de catégories dans une carte assez petite. L'application de cette méthode a été faite sur des séquences d'éléments discrets ; nous n'avons pas pu trouver d'application la concernant sur les séries temporelles.

#### 3.4.4.2 Variantes à mappage spatio-temporel

**Merge SOM**

Merge SOM (MSOM) [Strickert et Hammer, 2005] est une autre variante de SOM destinée au traitement des données séquentielles. L'algorithme permet de prendre en compte l'historique des exemples présentés d'une manière intuitive. Dans MSOM, chaque unité $u_i$ de la carte possède un vecteur appelé vecteur poids $\mathbf{w}_i \in \mathcal{R}^d$ (équivalant au vecteur prototype utilisé dans SOM) et un vecteur contexte noté $\mathbf{c}_i \in \mathcal{R}^d$. Pour chaque exemple présenté, l'unité gagnante $u_k$ est déterminée en minimisant la distance $d_i(t)$ définie par l'équation 3.23 :

$$d_i(t) = (1-\alpha)\left\|\mathbf{x}(t) - \mathbf{w}_i\right\|^2 + \alpha\left\|\mathbf{c}(t) - \mathbf{c}_i\right\|^2 \tag{3.23}$$

$\mathbf{c}(t)$ est un vecteur de $\mathcal{R}^d$ appelé descripteur de contexte, déterminé de manière récursive comme le montre l'équation 3.24, où $\alpha$ est un paramètre qui permet de doser entre la distance euclidienne classique et l'influence du contexte.

$$\mathbf{c}(t) = (1-\beta)\mathbf{w}_{u_k(t-1)} + \beta\mathbf{c}_{u_k(t-1)} \tag{3.24}$$

L'équation 3.24 représente une combinaison linéaire des propriétés de l'unité gagnante à l'instant $t-1$. $\beta$ est le paramètre de fusion, $0 \leq \beta \leq 1$, d'où le nom de l'approche. Le terme $u_k(t-1)$ se réfère à l'unité gagnante à l'étape précédente. La règle d'apprentissage est ensuite appliquée sur la carte pour la mise à jour des unités. Ceci est fait séparément sur les vecteurs poids et les vecteurs contextes comme le montre l'équation 3.25.

$$\begin{aligned} \Delta\mathbf{w}_i &= \gamma h_\sigma(T(u_i, u_k))(\mathbf{x}(t) - \mathbf{w}_i) \\ \Delta\mathbf{c}_i &= \gamma' h_\sigma(T(u_i, u_k))(\mathbf{c}(t) - \mathbf{c}_i) \end{aligned} \tag{3.25}$$

où $\gamma$ et $\gamma'$ sont les deux pas d'apprentissage pour respectivement le vecteur poids et le vecteur de contexte. $T(.)$ est la fonction topologique de la carte comme pour la SOM classique.

La méthode MSOM ressemble par certains aspects à celle utilisée par RSOM. Elle peut être décrite comme une implémentation alternative de l'encodage fait par RSOM. Mais selon [Strickert et Hammer, 2005] elle dépasse RSOM en flexibilité et en capacité à simuler des automates finis grâce à la représentation explicite du contexte : dans MSOM, chaque neurone gère à la fois les exemples d'entrée et les anciennes unités gagnantes des itérations précédentes. Ceci est permis par l'intermédiaire d'une règle récursive qui représente une sorte de mémoire sur la carte en elle-même, contrairement au RecSOM où l'opération se fait de manière séparée.

**SOMTAD**

Les cartes auto-organisées à diffusion d'activités temporaires, en anglais *Self Organizing Maps with Temporal Activity Diffusion* (SOMTAD) [Eulianoa et Principe, 1999], sont une méthode inspirée d'un concept biologique de diffusion d'activité. Il s'agit d'une équation de réaction-diffusion qui était utilisée pour expliquer la formation des *patterns* naturels

par l'interaction de composants chimiques. Ses différentes composantes interagissent entre elles et se diffusent dans la substance selon une formule précise (pour les détails voir dans [Eulianoa et Principe, 1999]). La diffusion d'activité s'inspire de la diffusion de l'oxyde de nitrate dans le cerveau.

La méthode consiste à garder les bases de l'algorithme SOM en ajoutant une auto-organisation dans l'espace et le temps. Cela est fait en créant des unités temporellement reliées entre elles. À chaque itération, l'unité ou le groupe d'unités activées modifient leur entourage de manière qu'il soit plus susceptible d'être activé dans les prochaines itérations. L'activité se définit selon l'équation 3.26.

$$act_i(t) = act_k(t-1)(1-\mu) + \mu dist(\mathbf{x}(t), \mathbf{w}_i) \tag{3.26}$$

où $dist(\mathbf{x}(t), \mathbf{w}_i)$ représente la distance entre l'exemple d'entrée et l'unité $u_i$. Les valeurs d'activité sont alors propagées à travers les unités voisines comme le montre la figure 3.7. La dégradation progressive de l'activité est modélisée par une multiplication avec un facteur $\nu \leq 1$ (voir l'équation 3.27 et la figure 3.7).

$$en_i(t) = \nu act_i(t-1) \tag{3.27}$$

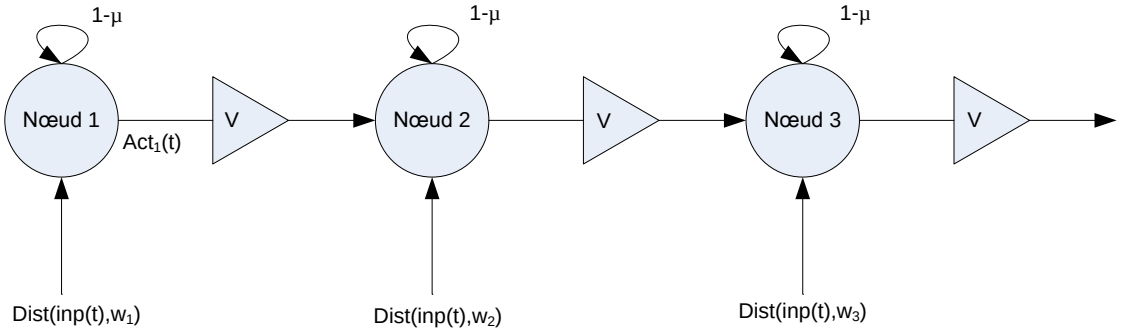


FIGURE 3.7 – Modèle de diffusion d'activité dans SOMTAD

Comme tous les algorithmes de type SOM, à chaque itération il est nécessaire de déterminer l'unité gagnante. Dans cette méthode, ceci est fait d'une manière spécifique décrite par l'équation 3.28 :

$$u_k = arg \min_{i \in V} \left( dist(\mathbf{x}(t), \mathbf{w}_i) + \beta en_i(t) \right) \tag{3.28}$$

où $\beta$ représente un paramètre spatio-temporel qui détermine l'ampleur de la valeur en diffusion des unités voisines. Augmenter $\beta$ donne plus de chances que les unités dans le voisinage soient gagnantes dans une itération future. D'un autre côté, diminuer $\beta$ rapproche l'algorithme d'un SOM classique. Enfin, la partie mise à jour de la carte est celle de l'approche classique de Kohonen.

La particularité de cette méthode est qu'elle permet de créer une mémoire distribuée spatialement sur la carte. SOMTAD a été utilisée dans des applications liées à la robotique dans [Kulzer, 1996] et dans [Euliano *et al.*, 1996], ou encore dans le clustering de séquences de phonèmes en temps réel [Ruwisch *et al.*, 1997].

#### SOMSD

La méthode des cartes auto-organisées pour les données structurées, en anglais *Self Organizing Maps for Structured Data* (SOMSD) [Hagenbuchner, 2003], a été proposée principalement pour des structures de données hiérarchiques sans pour autant exclure la possibilité d'utiliser les données séquentielles. Cette approche ressemble à RecSOM et à MSOM, qui utilisent deux vecteurs pour chaque unité, un vecteur de poids noté $\mathbf{w}_i$ et un vecteur de contexte noté $\mathbf{c}_i$ associé à une unité $u_i$.

Dans SOMSD, le vecteur $\mathbf{c}_i$ contient les indices de la dernière unité gagnante enregistrée, ce qui veut dire que $\mathbf{c}_i$ est un vecteur toujours égal à la dimension de la carte. Ainsi pour une grille bidimensionnelle, on a $\mathbf{c}_i \in [1..n_1] \times [1..n_2]$, où $n_1$ et $n_2$ sont respectivement le nombre d'unités sur l'axe horizontal et le nombre d'unités sur l'axe vertical.

L'unité gagnante est déterminée en minimisant une distance $dist(\mathbf{x}(t), u_i)$, où $\mathbf{x}(t)$ est l'exemple d'entrée et $u_i$ est l'unité comparée. Cette distance est définie par l'équation 3.29.

$$dist(\mathbf{x}(t), u_i) = \alpha \left\| \mathbf{x}(t) - \mathbf{w}_i \right\|^2 + \beta T(I(t-1), \mathbf{c}_i) \tag{3.29}$$

où $I(t-1)$ représente l'indice de l'unité gagnante à l'instant $t-1$ et $T$ est la fonction topologique utilisée entre les unités de la carte.

Une amélioration de SOMSD est proposée dans [Strickert et Hammer, 2003], appelée SOM pour les séquences (SOM-S), et qui transpose l'idée de SOMSD sur d'autres types de cartes bidimensionnelles rectangulaires. Il s'agit de décrire la topologie des unités sous forme de graphe dans lequel les connexions entre les neurones sont réalisées en associant à chaque unité un nombre fixe de voisins directement liés. Ce qui a permis d'utiliser de nouvelles forme de topologie (hyperboliques). Ce type de plan s'est expérimentalement montré plus adéquat pour la structures des données séquentielles sur la série temporelle MG-17. Les comparaisons sont faites avec des structures classiques de SOM sur la base de l'erreur de quantification, sans aller jusqu'à une application pour la prévision des séries temporelles.

### 3.4.5 Neural Gas

Neural Gas (NG) [Martinetz *et al.*, 1993] est une méthode de clustering qui s'inscrit, comme les précédentes, dans le cadre du clustering partitionnel. Cet algorithme permet de réaliser la tâche de quantification vectorielle tout en assurant une convergence rapide et d'obtenir une erreur de distorsion minimale. NG est aussi basé sur la règle d'adaptation soft-max, qui est une variante de l'algorithme des K-means qui prend en compte un classement du voisinage. Cependant cet algorithme n'utilise pas la distance euclidienne mais un rangement du voisinage issu du positionnement d'un exemple aux prototypes. Chaque fois qu'un exemple est présenté, il ne suffit pas de déterminer l'unité la plus proche mais

de classer tous les vecteurs poids du plus proche au plus distant. La règle d'apprentissage, définie par l'équation 3.30, rappelle celle utilisée par les cartes de Kohonen.

$$\Delta \mathbf{w}_i = \alpha h_\lambda(k_i)(\mathbf{x}(t) - \mathbf{w}_i) \tag{3.30}$$

où $\alpha$ est le pas d'apprentissage et $k_i$ représente le rang du vecteur $\mathbf{w}_i$ à l'issue du classement de ce dernier par rapport à la distance de l'exemple concerné. $h_\lambda$ est une fonction similaire à la topologie dans SOM même si, dans le cas des NG, les liens spatiaux entre les unités sont inexistants. La valeur de $h_\lambda$ varie en fonction du rang $k_i$ : elle est égale à l'unité quand la valeur du rang $k_i$ est égale à 0, et tend vers zéro quand la valeur de $k_i$ augmente. Ceci peut être facilement obtenu en adoptant une fonction exponentielle :

$$h_\lambda(k_i) = \exp^{-\frac{k_i}{\lambda}} \tag{3.31}$$

Il est montré dans [Martinetz *et al.*, 1993] que lors de l'exécution, l'algorithme NG obéit à une descente du gradient sur une fonction de coût définie dans la forme générale par l'équation 3.32.

$$E_{ng} = \int_t dist(\mathbf{x}(t), \mathbf{w}_k) P_k(\mathbf{x}(t))(\mathbf{x}(t) - \mathbf{w}_k) \tag{3.32}$$

où $P_k(\mathbf{x}(t))$ est la probabilité que l'exemple $\mathbf{x}(t)$ soit dans le cluster $k$, $k$ étant l'indice de l'unité gagnante correspondante à $\mathbf{x}(t)$.

L'appellation Neural Gas provient de la formule de modification moyenne des vecteurs poids qui correspond à un mouvement sur-amortis de particules dans l'espace. Les gradients appliqués sur les particules ressemblent à une force qui pousse ces particules vers une direction de l'espace où la densité est faible. Cette force ressemble à un couplage répulsif entre les particules (représentées par les vecteurs de références), rappelant ainsi une force entropique qui tend à distribuer les vecteurs dans l'espace comme des particules d'un gaz en diffusion.

Les performances ont été comparées avec les algorithmes K-Means et SOM. Selon les expérimentations, NG converge plus rapidement et atteint une erreur de quantification minimale, mais ceci avec une complexité en temps plus élevée, qui demande une implémentation en parallèle pour atteindre en temps raisonnables ces niveaux de performances.

L'algorithme NG est aussi utilisé dans le problème de prévision de séries temporelles dans le cadre d'une approche locale. La série testée est celle de Mackey-Glass avec un modèle localement linéaire pour la phase de prévision et des ensembles d'apprentissage de différentes tailles. Les expérimentations montrent que les performances de l'approche locale avec NG combinée avec un modèle linéaire pour la prévision dépassent celles de K-Means combinées avec un réseau de neurones RBF. D'une part, NG peut atteindre les mêmes performances avec un moindre nombre de paramètres dans le modèle. D'autre part, NG dépasse aussi les K-means en capacité de généralisation. De plus, les approches classiques basées sur les réseaux RBF demandent un ensemble d'apprentissage plus large pour atteindre les mêmes performances.

Tout comme les cartes de Kohonen, NG dans sa version orignale est un algorithme de clustering de données quelconques et il peut être envisagé de faire des modifications pour

l'adapter aux données séquentielles. En d'autres termes, les variantes proposées dans les travaux de la section 3.4.4 pour l'algorithme SOM ont également été proposées pour la modification de NG.

Une adaptation des cartes de Kohonen appelée SOMTAD, présentée dans la section 3.4.4.2, est appliquée à l'algorithme NG. Cet algorithme, rebaptisé GASTAD, fonctionne sous le même principe que SOMTAD, sauf que NG n'utilise en général pas de grille qui prédéfinit la topologie et l'ordre des neurones. Cela permet d'avoir une flexibilité intéressante pour une structure de diffusion plus adaptable aux données. Dans SOMTAD, deux neurones voisins sur la grille doivent être relativement proches dans le temps, ce qui n'est plus une contrainte dans le cas des GASTAD. Ceci rend le couplage entre le temps et l'espace contrôlable et permet de se focaliser sur les corrélations temporelles.

## 3.5 Clustering hiérarchique

Dans le cas général, il existe plusieurs autres catégories de clustering. Parmi ces catégories on s'intéressera au clustering hiérarchique. L'opération de clustering hiérarchique est illustrée par les figures 3.8 et 3.9. Dans la figure 3.8, il est présenté sept exemples (nommés A, B, C, D, E, F et G) organisés dans trois clusters différents.

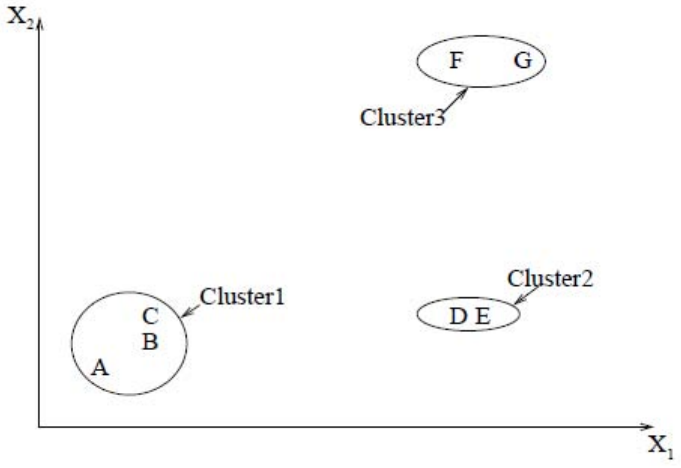


FIGURE 3.8 – Un exemple d'application pour le clustering hiérarchique

L'algorithme de segmentation hiérarchique permet d'obtenir le dendrogramme de la figure 3.9. Ce dendrogramme représente l'emboitement des groupes de *patterns* et les niveaux de similarités sous forme de palier. On remarque que selon le placement dans un niveau de palier, le clustering qu'on obtient est différent. Plus on se positionne vers le haut plus le nombre de clusters diminue et inversement.

L'intérêt principal est que ce type d'algorithme ne demande pas un nombre prédéfini de clusters par l'utilisateur, ce qui n'est clairement pas le cas des algorithmes partitionnels que nous venons de voir dans la section 3.4. Ce genre de méthode possède aussi l'avantage

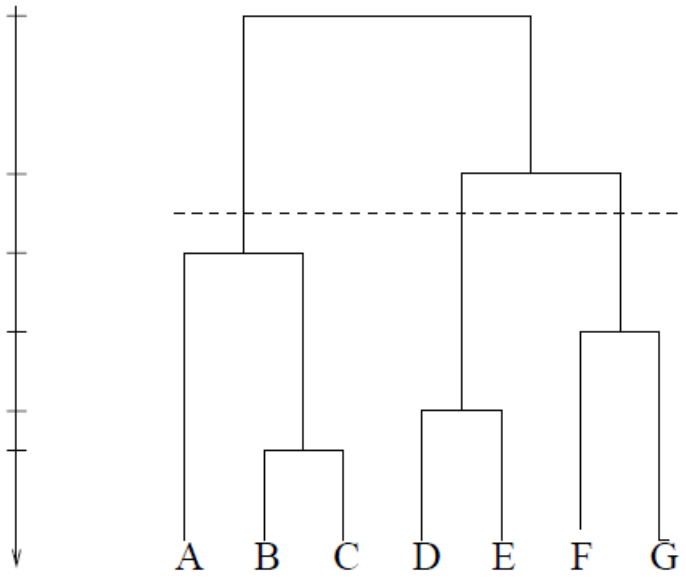


FIGURE 3.9 – Le dendogramme obtenu par un algorithme de clustering hiérarchique sur l'exemple de la figure 3.8

de ne pas dépendre d'une hypothèse sur la distribution des données ainsi que de n'avoir besoin comme information explicite que d'une matrice de dissimilarité.

La matrice de dissimilarité doit contenir les distances entre les éléments de l'ensemble $\mathcal{D}$. La méthode DAISY proposée dans [Kaufman et Rousseeuw, 1990] permet de faire ce calcul, qui peut être appliqué aussi bien sur les données numériques que sur les données discrètes. Il n'existe pas de restriction spécifique sur la nature de la mesure de distance utilisée autre que les caractéristiques de base des métriques. Cependant, le calcul peut se révéler lourd quand l'ensemble de données est très grand et tout particulièrement pour les types de données que nous envisageons. Pour cela, d'autres moyens peuvent être envisagés. Par exemple, à chaque niveau de la hiérarchie, une division est faite par le biais de l'algorithme des K-means ou encore l'algorithme SOM (dans [Villmann et Erzsébet, 2010], et dans [Pampalk *et al.*, 2003]).

Il existe deux grandes familles pour le clustering hiérarchique : l'approche par division et l'approche par agrégation. La première peut être décrite comme traitement du haut vers le bas (*top-down*) alors que la deuxième est un traitement du bas vers le haut (*bottom-up*).

### 3.5.1 Approche par division

L'approche par division [Kaufman et Rousseeuw, 1990], en anglais *DIvisive ANAlysis* (DIANA), est un algorithme de segmentation hiérarchique qui procède selon une stratégie *top-down*. L'algorithme peut être décrit par quelques étapes. Au départ, toutes les données de $\mathcal{D}$ sont considérées comme un unique cluster qui regroupe les $N$ exemples. Ensuite, à chaque itération, parmi tous les sous-ensembles déjà obtenus, un est choisi pour être divisé

en deux. Le critère de choix de ce cluster se fait selon le diamètre le plus grand, c'est-à-dire la distance la plus grande entre deux exemples d'un même cluster. Cette opération est répétée jusqu'à ce que chaque groupe ne comporte qu'un seul exemple, ce qui fait un total de $2^{N-1}-1$ possibilités de division d'un cluster en deux sous-groupes. DIANA utilise l'algorithme 1 afin d'effectuer cette division.

**Algorithm 1** Description de l'algorithme DIANA

1. Sélectionner le cluster $C_k$ ayant le plus grande diamètre.
2. Soit $D^k = \{d(x_i, x_j) | x_i, x_j \in C_k \; et \; x_j \in C_s\}$
   et soit $D_i^{ks} = \{d(x_i, x_j) | x_i, x_j \in C_k \; et \; x_j \notin C_s\}$.
3. Créer le nouveau cluster $C_s = \{x_s\}$ où $x_s = arg\max_{x_i}(D_i^k)$.
4. Soit $D_i^{kk} = \{d(x_i, x_j) | x_i, x_j \in C_k\}$
   et soit, pour un $x_i$ fixé, l'ensemble $D_i^k = \{d(x_i, x_j) | x_i, x_j \in C_k - \{x_i\}\}$.
5. $x_h = arg\max_{x_i \in C_k - C_s}(D_i^{kk} - D_i^{ks})$.
6. Si $(D_h^{kk} - D_h^{ks}) > 0$ alors $x_h$ est plus proche de $C_s$
   Mettre $x_h$ dans $C_s$.
7. Répéter les étapes 4 à 6 jusqu'à $(D_h^{kk} - D_h^{ks}) < 0$.
8. S'il reste encore au moins un cluster avec plus que 1 élément, aller à 1.

Il est aussi possible de fixer un critère d'arrêt pour ne pas être obligé d'aller en profondeur comme pour l'algorithme 1. Le critère d'arrêt le plus utilisé est un coefficient qui mesure la qualité du clustering trouvé par l'algorithme. À titre d'exemple, dans [Kaufman et Rousseeuw, 1990], on considère le ratio $dd(i)$ entre le diamètre du cluster auquel l'exemple $x_i$ a appartenu avant la séparation et le diamètre global. Le critère d'arrêt $CR$ est donné par l'équation 3.33.

$$CR = 1 - \sum_{i=1}^{n} \frac{dd(i)}{n} \tag{3.33}$$

D'autres moyens que ceux de DIANA peuvent être utilisés pour la division des clusters. Par exemple dans [Dittenbach *et al.*, 2000] des cartes de Kohonen sont à la base d'une méthode appellée *Growing Hierarchical Self-Organization Maps* (GHSOM). Son principe consiste à utiliser de manière hiérarchique plusieurs grilles indépendantes, chacune d'entre elles représentant un algorithme SOM à part. L'exécution se fait du haut de l'arbre vers le bas. Dans le premier niveau un seul SOM est d'abord utilisé. Ensuite, pour chaque unité de la carte sélectionnée pour être développée dans le prochain niveau de l'arbre, une nouvelle grille est générée qui utilise uniquement le sous-ensemble associé à cette unité (voir l'algorithme 2).

**Algorithm 2** Description de l'algorithme GHSOM

1- Initialiser la grille de SOM
2- Apprendre en utilisant l'ensemble apprentissage correspondant à la carte
3- Pour chaque unité à étendre, créer une nouvelle carte et aller à 2

Une amélioration de GHSOM [Pampalk *et al.*, 2003] propose un moyen de déterminer quelle région de la carte est préférable à détailler, que certaines régions des données se voyant décomposée plus que d'autres. L'algorithme propose aussi des mécanismes afin de garantir la préservation de la topologie dans SOM.

### 3.5.2 Approche par agrégation

La fusion par agrégation, en anglais *AGglomerative NESting* (AGNES), a aussi été proposée comme une alternative à DIANA dans [Kaufman et Rousseeuw, 1990]. À l'inverse de cette dernière, AGNES procède par les feuilles de l'arbre, qui sont des éléments singletons de $\mathcal{D}$, en les fusionnant deux à deux à chaque itération jusqu'à obtenir un seul ensemble global à la racine de l'arbre.

À chaque niveau, $\frac{n(n-1)}{2}$ possibilités de fusion sont disponibles, $n$ étant le nombre de nœuds dans le niveau considéré. La fusion se fait sur la base des deux clusters les plus proches dans la matrice de dissimilarité selon une distance déterminée. Plusieurs types de métriques peuvent être envisagées pour l'approche hiérarchique. Par exemple, *Single linkage* prend en compte la plus petite dissimilarités, *complete linkage* prend en compte la plus grande des dissimilarités, *average linkage* prend en compte la moyenne des dissimilarité. L'algorithme 3 décrit de manière générale cette approche.

**Algorithm 3** Description de l'algorithme AGNES

1- Choisir entre les clusters les deux plus proches $C_i$ et $C_j$, selon la distance choisie.
2- Grouper les exemples des deux clusters en un seul nouvel ensemble nommé $C_k$, et remplacer $C_i$ et $C_j$ par $C_k$.
3- Si le nombre de clusters est supérieur à 1, aller à l'étape 1.

### 3.5.3 Autres modèles hiérarchiques pour les séries temporelles

À notre connaissance, il n'existe pas d'approche locale pour la prévision des séries temporelles utilisant la segmentation hiérarchique. Cependant, nous pouvons trouver dans la littérature des modèles qui utilisent, d'une manière ou d'une autre, une structure hiérarchique sur des séries temporelles. Celui proposé dans [Jordan et Jacobs, 1994] peut être considéré comme un des plus proches des approches locales.

Le principe est inspiré des travaux de [Jacobs *et al.*, 1991]. Ils proposent pour la première fois une architecture d'apprentissage supervisé composée de plusieurs prédicteurs parallèles, appelés *Expert Networks*, et un prédicteur spécifique qui joue le rôle d'aiguilleur appelé *Gating Network* comme le montre la figure 3.10.

Ce modèle ne ressemble pas à ce que nous avons vu dans les approches locales. Toutefois, le réseau aiguilleur permet en quelque sorte de simuler la fonction du clustering en utilisant le même vecteur d'entrée pour déterminer l'expert sélectionné. Il utilise pour ce faire une technique adaptée du maximum de vraisemblance.

Dans [Jordan et Jacobs, 1994], ce même modèle est utilisé de manière hiérarchique en parallélisant plusieurs experts locaux sous une forme fractale, comme le montre la figure

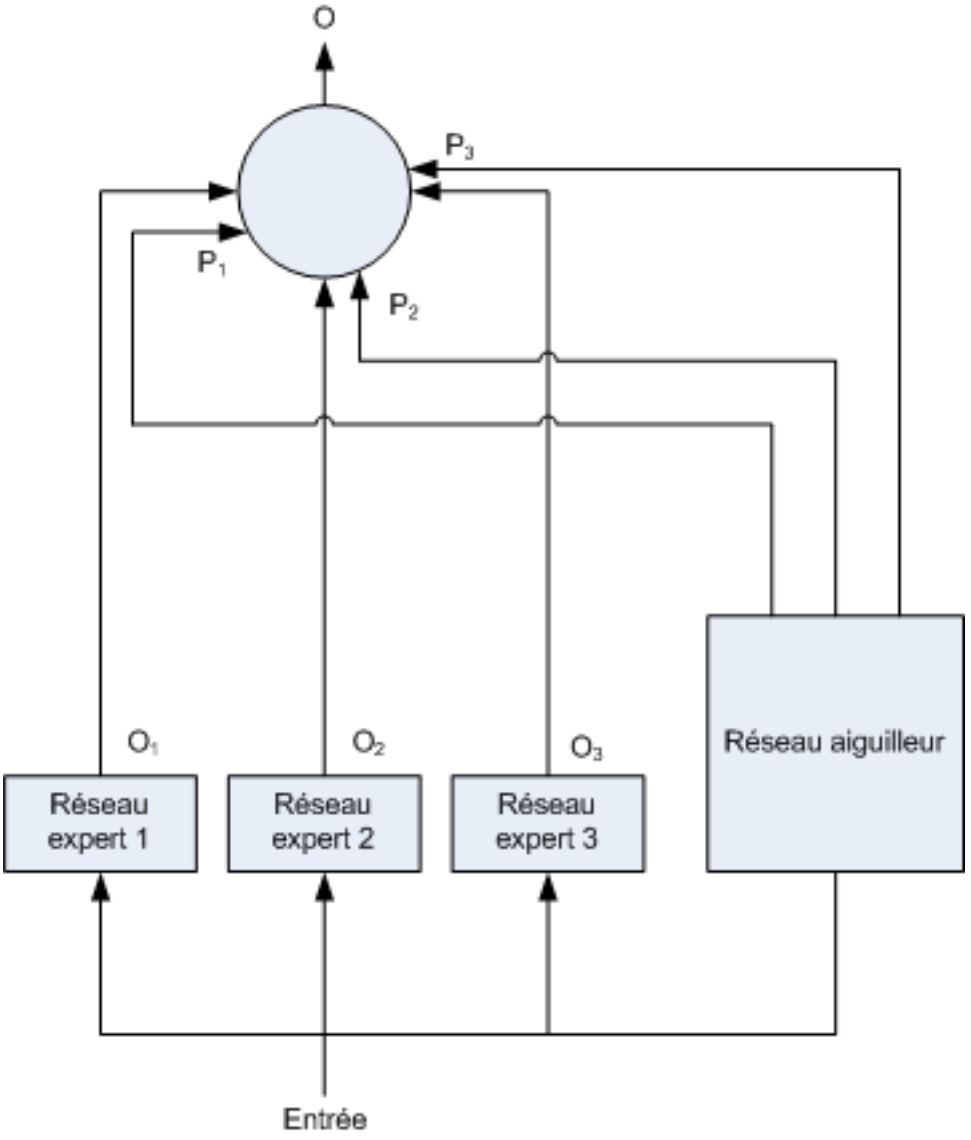


FIGURE 3.10 – Schéma de l'architecture des experts locaux

3.11. Ce principe a été utilisé récemment dans d'autres applications liées à la modélisation des séries temporelles [Huerta, 2003] [Prado *et al.*, 2006].

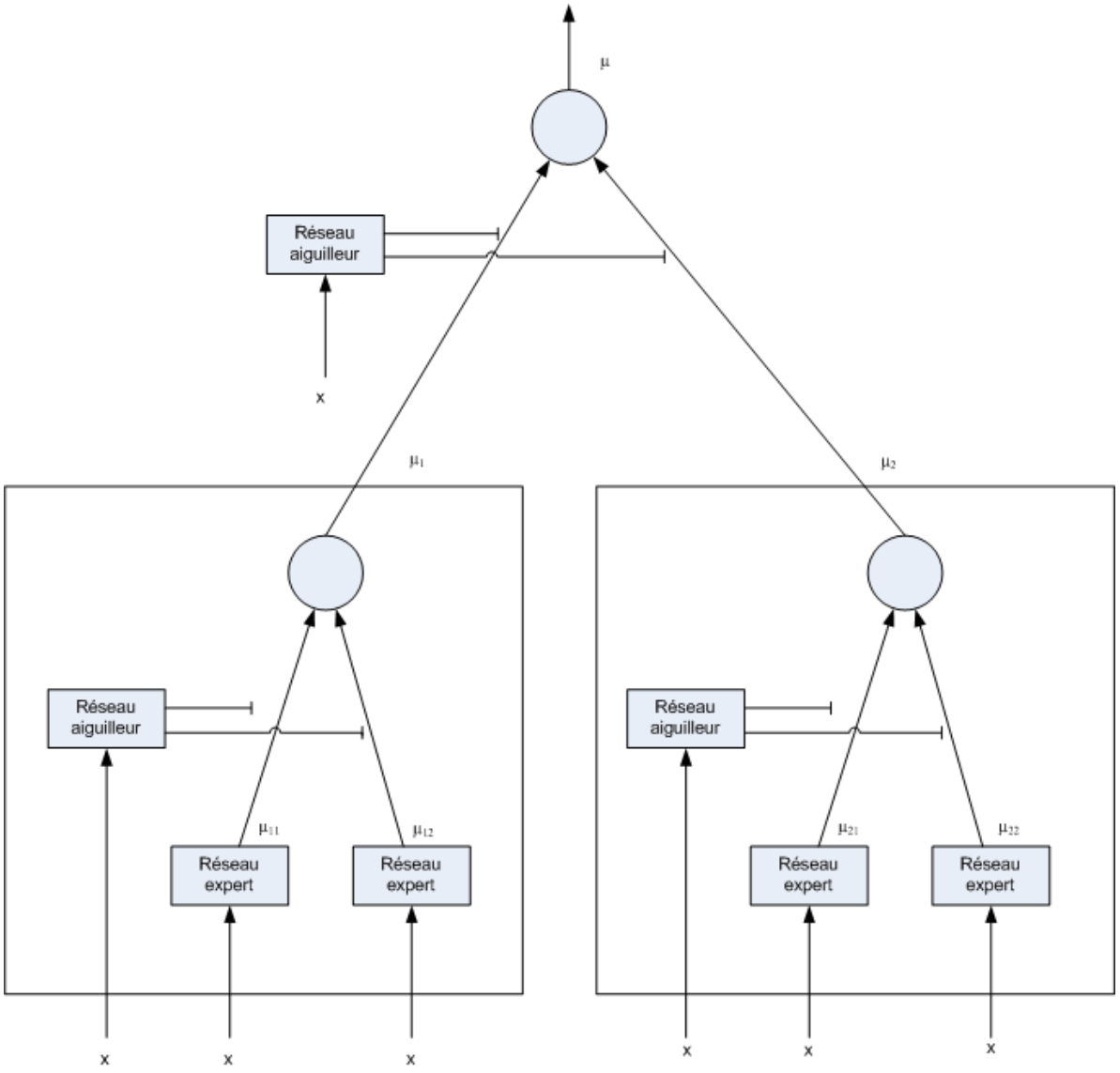


FIGURE 3.11 – Schéma de l'architecture hiérarchique des experts locaux

Dans [Koenig et Youn, 2011] un algorithme de clustering hiérarchique est proposé pour des données génétiques. L'algorithme est modifié de sorte à ce qu'il soit le plus adapté pour ce type de données. Une nouvelle notion de distance est aussi utilisée. Le dendrogramme hiérarchique permet d'effectuer des comparaisons rapides entre les gènes.

Dans [Hyndman *et al.*, 2011] une méthode de prévision des séries temporelles particulières est présentée. Il s'agit de données pouvant être agrégées sous différents niveaux en groupes (par exemples le produit, la géographie ou d'autres caractéristiques). Ce n'est

clairement pas du clustering ou une approche locale. Cependant les prédicteurs sont disposés de manière hiérarchique séparés par niveaux. Ensuite, les résultats sont combinés et reconsidérés.

### 3.5.4 Récapitulatif des méthodes présentées

Nous avons cité, dans ce chapitre, plusieurs travaux classés dans la famille des approches locales ou tout simplement des méthodes pour le clustering des données séquentielles. Comme nous avons étendu l'état de l'art sur de nombreux domaines, certains des travaux présentés ne s'adressent pas aux séries temporelles et plus spécifiquement la prévision. C'est pour cette raison que dans ce paragraphe nous présentons un récapitulatif sur les propositions de la littérature qui ont traité le clustering des données séquentielles, ou dans le meilleur des cas la prévision des séries temporelles.

Le tableau 3.1 résume les résultats obtenus sur ces méthodes que nous avons sélectionnées. Il y est présenté, pour chacune des méthodes, les performances correspondant à chacune des séries temporelles (si elles sont disponibles) et avec éventuellement un commentaire. Il s'agit de l'erreur NMSE sur l'ensemble test.

On remarque que les séries les plus utilisées sont principalement MG-17 et Laser. Le modèle local avec une combinaison des cartes de Kohonen et $AR(p)$ [Vesanto, 1997] est le plus performant, dépassant ainsi le boosting. Ce dernier est plus performant sur laser dépassent les deux approches locales [Noelia Sánchez-Maroño *et al.*, 2003] et [Varsta *et al.*, 2001]. Cependant, pour les techniques de clustering NG [Martinetz *et al.*, 1993], SOMSD [Strickert et Hammer, 2003] et MSOM [Strickert et Hammer, 2005] ont été évalués visuellement par rapport au clustering obtenu.

Nous évaluerons nos propositions sur les séries précédemment citées par rapport à deux types de résultats. D'un côté par rapport à la littérature en se référant aux résultats obtenus dans le tableau 3.1. Et d'un autre côté par rapport aux autres méthodes que nous avons développées.

| Méthode | Résultats (NMSE) | | commentaires |
|---|---|---|---|
| Boosting | Sunspots | $78 \times 10^{-3}$ | |
| | MG-17 | $0,13 \times 10^{-3}$ | |
| | Henon | $1,9 \times 10^{-3}$ | |
| | Laser | $1,9 \times 10^{-3}$ | |
| SOM + AR(p) [Vesanto, 1997] | MG-17 | $0,0019 \times 10^{-3}$ | |
| | MG-17 | $0,144 \times 10^{-3}$ | prédiction à $(t + 84)$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | Laser | $11 \times 10^{-3}$ | |
| RSOM [Varsta *et al.*, 2001] | Laser | $8410^{-3}$ | |
| MSOM [Strickert et Hammer, 2005] | MG-17 | | Graphe : Entropie/Temps |
| SOMSD [Strickert et Hammer, 2003] | MG-17 | | Graphe : Erreur de quantification/Temps |
| Neural Gas [Martinetz *et al.*, 1993] | MG-17 | | Graphe de l'erreur de prédiction |

TABLE 3.1 – Tableau récapitulatif des résultats sur les séries temporelles les plus communes

## Conclusion du chapitre

Nous avons abordé dans ce chapitre des techniques décrites comme des méta-méthodes qui utilisent les algorithmes d'apprentissage présentés dans le chapitre précédent. Ces méthodes permettent d'améliorer l'efficacité des modèles classiques. Nous avons présenté la technique du boosting et nous avons focalisé notre attention sur l'approche locale dont le principe est d'effectuer une segmentation des données sur la base de similarité. Nous avons aussi divisé ces approches en deux familles en considérant la nature de l'algorithme de segmentation, hiérarchique et partitionnel, tout en illustrant par des approches provenant de la littérature et utilisées sur des séries temporelles ou des données séquentielles.

Dans le prochain chapitre, nous évoquerons notre apport qui concerne principalement la prévision des séries temporelles par les approches locales. Ces apports seront aussi classés dans les deux catégories selon la nature de clustering effectué, le clustering partitionnel et le clustering hiérarchique. Nous comparerons nos méthodes avec plusieurs autres propositions trouvées dans la littérature, y compris avec celles que nous avons citées dans le tableau 3.1.

# Chapitre 4

# Propositions sur les méthodes de prévisions locales

## Introduction

Le chapitre précédent donne un aperçu de l'état de l'art des méthodes de prévision locales. Nous avons aussi vu que les cartes de Kohonen sont les plus utilisées dans ce cadre pour le clustering des données et que les approches locales permettent clairement d'améliorer les performances par rapport aux méthodes classiques qui traitent les données dans leur globalité.

Dans ce chapitre, nous présenterons nos propositions afin d'améliorer les performances et/ou de simplifier l'usage des approches locales pour la prévision des séries temporelles. Le premier apport est une méthode pour intégrer les réseaux de neurones récurrents dans l'approche locale. Nous présenterons ensuite deux améliorations de la phase de *clustering* avec des nouveaux algorithmes de segmentations à partitions et de segmentations hiérarchiques.

## 4.1 Intégration de réseaux de neurones récurrents

Dans cette section, on considérera que les données sont déjà segmentées. Nous nous focaliserons sur la phase prédiction de l'approche locale et l'utilisation des prédicteurs locaux, plus particulièrement les réseaux de neurones récurrents.

Nous avons précédemment montré que les réseaux de neurones récurrents (RNN) dépassaient souvent les MLP ou les SVM en matière de performance dans la prévision des séries temporelles. Cependant, ni les MLP, ni les SVM ne posent problème pour l'application en tant que prédicteurs locaux. En effet, la perte du séquencement temporel des données suite à la phase de clustering n'affecte pas l'utilisation des MLP ou des SVM dans la phase de prédiction.

Pour les MLP, ceci peut être expliqué par trois raisons principales. La première raison est l'utilisation d'une fenêtre temporelle qui est la même que celle utilisée dans le cadre du clustering ou qui lui est au plus égale en taille. La deuxième est que dans l'approche locale,

les valeurs sont propagées unilatéralement de l'entrée du réseau vers sa sortie, c'est-à-dire sans boucles internes. La troisième raison est le traitement indépendant des vecteurs de données dans l'algorithme de rétropropagation, c'est-à-dire que le calcul de la valeur de sortie ainsi que la modification des poids des connexions se font dans une même itération du processus d'apprentissage.

Pour SVM, il suffit de remarquer que la fonction objectif peut être calculée indépendamment pour tout sous ensemble des données.

Ceci n'est pas le cas des RNN où les connexions peuvent former des cycles ; nous rappelons à titre d'exemple le RNN totalement connecté décrit par la figure 2.15 que nous avons décrit dans le chapitre 2.

De ce fait, les RNN ne peuvent pas être appliqués de la même manière que les MLP ou les SVM dans l'approche locale. A l'instar des réseaux à propagation avant où l'information est transmise de couche en couche, les RNN utilisent aussi des informations provenant des itérations antérieures (dues aux cycles). Ces raisons suffisent pour affirmer que les algorithmes utilisés pour l'apprentissage des RNN ne sont plus adaptés pour l'utilisation avec l'approche locale.

### 4.1.1 Cas de l'algorithme de rétropropagation dans le temps

Nous avons vu que l'algorithme BPTT consiste à déplier dans le temps le réseau récurrent d'origine, donnant naissance à un réseau à $l$ couches, où $l$ est le nombre d'exemples. La figure **??** est le réseau déplié correspondant à celui de la figure 2.16 du chapitre 2 en appliquant l'algorithme d'apprentissage entre les instants $t_1$ et $t_l - 1$.

On note, pour chaque neurone $i$, sa fonction de transfert $f_i$ et sa sortie à l'instant $t$ $s_i(t)$ avec :

$$s_i(t) = f_i(net_i(t-1) + x_i(t)) \tag{4.1}$$

L'algorithme décrit dans la section 2.5.2.1 explique la formule utilisée pour calculer la variation des poids en utilisant la technique de descente du gradient :

$$\Delta w_{ij}(t_1, t_l - 1) = \sum_{\tau=t_1}^{t_l-1} \Delta w_{ij}(\tau) = -\eta \sum_{\tau=t_1}^{t_l-1} \frac{\partial E(t_1, t_l)}{\partial w_{ij}(\tau)} \tag{4.2}$$

Il est important de rappeler ici, qu'après la quantification vectorielle, l'ordre séquentiel entre les vecteurs n'est plus nécessairement assuré. Les boucles internes du réseau créent une mémoire implicite qui contribue à la prédiction en plus de l'entrée du réseau. Or, dans ce cas, présenter les données dans un ordre séquentiel est obligatoire pour un fonctionnement cohérent et des prédictions correctes. Ceci peut être observé dans l'équation 4.2 par l'utilisation du terme $\tau$.

Pour cette raison nous proposons de présenter pour chaque prédicteur local la totalité des données à laquelle nous rajoutons une information supplémentaire d'appartenance au cluster sous la forme du terme $\delta_{tk}$ défini par :

$$\delta_{tk} = \begin{cases} 1 & \text{si le vecteur } X(t) \in C_k \\ 0 & \text{sinon} \end{cases} \tag{4.3}$$

En considérant $\delta_{ik}$, la formule de modification des poids peut être décrite par l'équation 4.4

$$\Delta w_{ij}^{(k)}(t_1, t_l - 1) = \sum_{t_1}^{t_l - 1} \Delta w_{ij}(\tau) = -\eta \sum_{t_1}^{t_l - 1} \frac{\partial E^{(k)}(t_1, t_l)}{\partial w_{ij}(\tau)} \tag{4.4}$$

En effectuant les mêmes calculs dans la section 2.5.2.1, la règle d'apprentissage finale pour un RNN du $k^{\text{ième}}$ cluster est définie par les équations 4.5 et 4.6. La différence réside dans le calcul du gradient $\frac{\partial E^{(k)}(t_1,t_l)}{\partial net_i(\tau)}$ qui doit prendre en compte l'appartenance de l'élément au cluster traité par le prédicteur, d'où l'introduction du terme $\delta_{tk}$.

- pour $\tau = t_l - 1$

$$\frac{\partial E^{(k)}(t_1, t_l)}{\partial net_i(\tau)} = \begin{cases} (s_i(\tau) - d_i(\tau))\delta_{tk} f_i'(net_i(\tau)) & \text{si } i \in O(\tau) \\ 0 & \text{sinon,} \end{cases} \tag{4.5}$$

- pour $t_1 \leq \tau \leq t_l - 1$

$$\frac{\partial E^{(k)}(t_1, t_l)}{\partial net_i(\tau)} = \begin{cases} \left[ (s_i(\tau) - d_i(\tau))\delta_{tk} + \sum_{j \in succ(i)} \frac{\partial E(t_1, t_l)}{\partial net_j(\tau + 1)} w_{ij}(\tau + 1) \right] f_i'(net_i(\tau)) \\ \text{si } i \in O(\tau) \\ \sum_{j \in succ(i)} \frac{\partial E(t_1, t_l)}{\partial net_j(\tau + 1)} w_{ij}(\tau + 1) f_i'(net_i(\tau)) \\ \text{sinon,} \end{cases} \tag{4.6}$$

où $O(\tau)$ est l'ensemble de neurones de sortie à l'instant $\tau$ et $d_i(\tau)$ est la sortie désirée à l'instant $\tau$.

On voit bien que l'introduction du terme $\delta_{ik}$ permet d'appliquer la mise à jour uniquement pour le cluster $C_k$. L'information circule toujours dans le réseau, même si elle n'appartient pas au cluster $X(t) \in C_k$.

### 4.1.2 Cas de l'apprentissage récurrent temps réel

Bien que nous ne l'ayons pas exploité pour les expérimentations, il est utile de rappeler la possibilité d'utiliser les RNN en mode local avec d'autres algorithmes d'apprentissage. Aussi le même raisonnement est applicable avec l'algorithme RTRL pour l'apprentissage des

réseaux de neurones récurrents. Nous avons vu dans la section 2.5.2.2 que pour l'algorithme temps réel, les modifications de poids se font à chaque itération de l'apprentissage. Tout comme avec BPTT, on introduit le terme $\delta_{ik}$ qui permet de déterminer si l'élément à l'instant $t$ est bien dans le cluster $k$. La formule de modification des poids s'écrit alors selon l'équation 4.7.

$$\Delta w_{ij}(t) = -\nu \frac{\delta e(t)}{\delta w_{ij}} = \nu \sum_{t \in T(t)} (d_r(t) - s_r(t)) \frac{\delta s_r(t)}{\delta w_{ij}} \tag{4.7}$$

avec $\frac{\delta e(t)}{\delta w_{ij}}$ calculée de manière récursive comme suit :

$$\frac{\delta s_r(t)}{\delta w_{ij}} = f^{'}(net_r(t-1)) = (\delta_{ir} s_j(t-1) + \sum_{q \in Pred(r)} \frac{\delta s_q(t-1)}{\delta w_{ij}}) \tag{4.8}$$

avec comme condition initiale

$$\frac{\delta s_r(0)}{\delta w_{ij}} = 0 \tag{4.9}$$

### 4.1.3 Résultats préliminaires

Dans ce paragraphe, nous présenterons quelques résultats préliminaires concernant l'intégration des RNN dans l'approche locale. L'objectif est de comparer l'utilisation des RNN en tant que prédicteurs locaux et en tant que prédicteurs globaux. Nous effectuerons aussi une comparaison rapide avec d'autres méthodes utilisées.

Ces expérimentations sont menées sur trois séries temporelles différentes et de référence pour l'évaluation de méthodes de prévision par la communauté scientifique. La série des taches solaires, représente la moyenne annuelle des taches solaires (Sunspots) allant de 1700 à 1979. La série Mackey-Glass est une série générée par un modèle non linéaire qui a été établi à l'origine pour modéliser la production des globules blancs chez les patients atteints de leucémie. La série Laser mesure l'intensité des pulsations chaotiques d'un laser $NH_3 - FIR$, c'est une des séries employées dans la compétition de l'institut de Santa Fe. Nous présenterons ces données plus en détails dans le chapitre suivant. Pour chacune des méthodes citées, nous évaluerons l'erreur quadratique moyenne normalisée, en anglais *Normalized Mean Square Error* (NMSE).

Pour les expérimentations qui vont suivre, nous avons utilisé les mêmes configurations pour les prédicteurs, ce qui permettra de faire des comparaisons préliminaires. Pour les MLP nous avons utilisé une fenêtre temporelle de taille 7 avec une couche cachée de 4 éléments. Pour les RNN nous avons aussi utilisé une couche cachée de 7 neurones. Pour les SVR nous avons utilisé les paramètres par défaut de la librairie libSVM (qui peuvent être retrouvés dans la documentation de la librairie). Ces configurations des prédicteurs seront aussi utilisées dans les autres expérimentations préliminaires dans ce chapitre. Pour le clustering, nous nous sommes basés sur une configuration standard des cartes de Kohonen, algorithme SOM classique avec une carte linéaire de taille fixée à une quinze d'unités. Nous étudierons plus tard, dans les expérimentations approfondies, les comportements de la carte, en particulier en fonction du nombre de clusters définis au départ.

Le tableau 4.1 présente les résultats des prévisions sur la série des taches solaires. L'introduction des RNN en tant que modèles locaux est testée et comparée avec d'autres prédicteurs. Deux ensembles de tests ont été utilisés (test1 et test2). Le tableau 4.1 nous permet de conclure que tous les prédicteurs utilisés (MLP, RNN et SVM) deviennent plus performant avec l'utilisation de l'approche locale. Par exemple, pour les RNN l'erreur passe de 0.041 à 0.022, soit une réduction de presque 50% sur une moyenne à partir de 10 expérimentations. Le meilleur résultat obtenu sur le premier ensemble de tests est le modèle local avec les MLP tandis que les RNN locaux apparaissent les meilleurs sur le deuxième ensemble de tests et toujours sur une moyenne de 10 expérimentations.

| Méthode | NMSE Test set 1 | NMSE Test set 2 |
|---|---|---|
| MLP | $0,0059$ | $0,0191$ |
| SVM | $0,0094$ | $0,0256$ |
| RNN | $0,041$ | $0,0871$ |
| Local MLP | $0,0055$ | $0,018$ |
| Local SVM | $0,0081$ | $0,0207$ |
| Local RNN | $0,0224$ | $0,0271$ |

TABLE 4.1 – Tableau des résultats préliminaires sur la série temporelle sunspots

Le tableau 4.2 présente les résultats des prévisions sur la série Laser. Les résultats sur cette série apparaissent un peu différents, nous pouvons dire que les données sont plus difficiles à apprendre pour les RNN par rapport à l'exemple précédent. D'un coté, si on observe l'évolution de la série (figure 5.5, chapitre 5), on remarque un changement brusque dans le régime. D'un autre côté, les RNN sont des méthodes qui utilisent la chronologie dans l'apprentissage, le fait de décomposer la série Laser rend l'apprentissage moins efficace. C'est pour cette raison que nous trouvons les RNN largement moins performants même en introduisant l'approche locale avec des performances qui se dégradent de $0,008$ à $0,031$. Dans ces tests préliminaires, les meilleurs résultats obtenus correspondent au modèle local utilisant les MLP en tant que prédicteur. Nous ramenons donc ces résultats à la fois à la nature de la série Laser et aux caractéristiques des RNN.

| Méthode | NMSE Test set |
|---|---|
| MLP | $0,023$ |
| SVM | $0,055$ |
| RNN | $0,008$ |
| Local MLP | $0,0048$ |
| Local SVM | $0,040$ |
| Local RNN | $0,031$ |

TABLE 4.2 – Tableau des résultats préliminaires sur la série temporelle Laser

Le tableau 4.3 présente les résultats d'expérimentations préliminaires des prévisions sur la série Mackey-Glass (17). Les tests ont été menés sur les 3 prédicteurs utilisés précédemment ainsi que sur les modèles locaux associés. Comme pour les autres expérimentations, nous

nous sommes basés sur des prédicteurs avec des configurations choisies arbitrairement. Nous remarquons que pour ces jeux de données les modèles locaux permettent d'améliorer les résultats, allant jusqu'à diviser l'erreur quadratique moyenne approximativement par 10. D'autre part, les RNN qui étaient supérieurs en approche globale se voient améliorés en approche locale (de $0,235$ à $0,041$). Par nature la série Mackey-Glass est chaotique, mais contrairement à Laser elle ne possède pas un changement brusque dans le régime. De plus, le grand nombre de valeurs disponibles est un élément en faveur des RNN surtout avec l'approche locale et le clustering des données.

| Méthode | NMSE Test set $\times 10^3$ |
|---|---|
| MLP | $0,321$ |
| SVM | $0,52$ |
| RNN | $0,235$ |
| Local MLP | $0,023$ |
| Local SVM | $0,037$ |
| Local RNN | $0,041$ |

TABLE 4.3 – Tableau des résultats préliminaires sur la série temporelle Mackey-Glass (17)

## 4.2 Algorithme Auto-SOM

Nous avons vu précédemment, outre l'intérêt de l'intégration des RNN dans les approches locales en fonction des types de données, que l'utilisation des approches locales donnait généralement une amélioration des prédicteurs. En effet les expériences préliminaires ont montré que dans la plupart des cas (si en prend en compte les trois prédicteurs), les erreurs de prévision peuvent chuter jusqu'à être divisées par 10 et ce, en utilisant des prédicteurs de même type et sous les mêmes configurations. Par contre l'approche locale introduit plusieurs paramètres, ce qui nous ramène à deux problèmes. Comment segmenter efficacement des données séquentielles qui ne possèdent pas une structure apparente de classes ? Comment prédire efficacement ? Quel est le lien entre ces deux problèmes ?

Parmi les paramètres supplémentaires, le nombre de clusters est l'un des plus importants. Toutefois, il est difficile en examinant une série temporelle de pouvoir en déduire le nombre adéquat de clusters, d'abord à cause de la nature continue de la série mais aussi parce que cette information peut être dépendante d'autres paramètres, particulièrement la taille de la fenêtre temporelle utilisée pour le clustering.

Les expérimentations que nous présenterons montrent que les clusters obtenus influencent la valeur de l'erreur locale et par suite l'erreur globale. De ce fait, le nombre de clusters maximum, c'est-à-dire le nombre d'unités de la carte de Kohonen, est un facteur important car il va impacter la composition finale des clusters. Par exemple, plus le nombre de clusters est petit plus les clusters finaux seront volumineux et donc moins homogènes, plus l'erreur locale va augmenter. Inversement, plus le nombre de clusters augmente, plus leurs tailles diminueront, ce qui affectera par la suite la qualité de l'apprentissage. Il apparaît clair que l'enjeu est de trouver le bon équilibre.

On peut aussi se demander, comme nous partons de l'algorithme des cartes de Kohonen, quel est l'intérêt de trouver le bon nombre de clusters puisqu'il suffit de choisir dès le départ un nombre très grand. En pratique, les expérimentations montrent que les vecteurs vont se concentrer sur quelques régions de la carte avec une grande majorité des unités qui resteront vides. La réponse est qu'une telle stratégie peut présenter des inconvénients reliés à l'aspect aléatoire de l'algorithme. En effet, la carte de Kohonen, tout comme les réseaux de neurones, est initialisée aléatoirement au début de l'algorithme, ce qui a pour conséquence de ramener la solution retenue vers des minimum locaux et plus rarement vers le minimum global. Plus le nombre d'unités (et de centroïdes associés) dans la carte augmente, plus le nombre de minima locaux va augmenter, de même que la probabilité de tomber dans un de ces minima locaux.

Ce phénomène peut être illustré par une expérimentation en exécutant l'algorithme des cartes de Kohonen sur plusieurs instances. on remarque quelques fois des NMSE relativement élevées comparée à celles de la majorité des instances effectuées. Le tableau 4.4 montre par instance la valeur de la NMSE globale, la moyenne des NMSE des clusters et la NMSE minimale et maximale parmi les clusters. On remarque que l'erreur globale varie entre $0,0146$ et $0,0047$ malgré les autres valeurs qui restent à peu près proches. Ceci implique que le clustering ainsi que la structure finale de la carte sont eux aussi variables et impactent l'erreur globale. En pratique, plus on choisit des grandes cartes plus on obtient des clusters vides et des régions composées différemment.

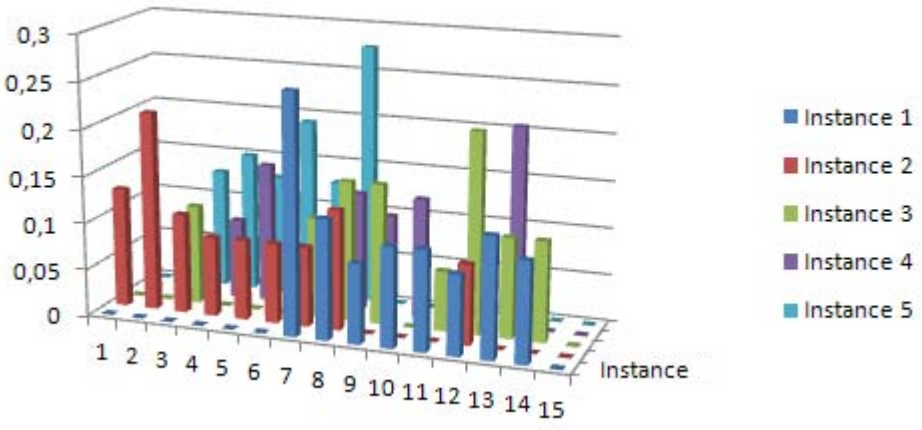


FIGURE 4.1 – Exemple de différentes instances de clustering, composition des carte par rapport à chaque instance

L'inconvénient de ce phénomène est l'augmentation de la partie de l'erreur associée à la variance du modèle. Dans cette partie nous cherchons à introduire un algorithme de clustering qui soit capable en même temps et automatiquement de s'adapter aux données et de trouver le nombre adéquat de classes, limitant ainsi un des problèmes cités ci-dessus.

| Instance | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| NMSE globale | 0,0146 | 0,0047 | 0,0101 | 0,0052 | 0,0049 | 0,0049 | 0,0081 | 0,0056 | 0,0095 | 0,0064 |
| NMSE Moyenne | 0,0387 | 0,0352 | 0,0376 | 0,0354 | 0,0356 | 0,0354 | 0,0369 | 0,0360 | 0,0377 | 0,0364 |
| Min cluster | 0,0007 | 0,0015 | 0,0011 | 0,0011 | 0,0014 | 0,0034 | 0,0013 | 0,0009 | 0,0015 | 0,0014 |
| Max cluster | 0,2127 | 0,2553 | 0,2127 | 0,2127 | 0,2765 | 0,2553 | 0,2127 | 0,2340 | 0,2340 | 0,2127 |

TABLE 4.4 – Tableau des différentes valeurs des NMSE obtenues sur un même cluster dans 10 expériences

Nous partons des cartes de Kohonen pour plusieurs raisons. La première est la simplicité de cet algorithme. Il rappelle les K-means avec une notion de voisinage qui a l'avantage de faciliter la représentation et l'interprétation. La deuxième raison est que les algorithmes de Kohonen sont parmi les plus utilisés pour l'approche locale, que ce soit la version classique de l'algorithme ou les versions modifiées. Facilitant les équations comparatives de performances. La troisième raison est que nous avons voulu profiter de la caractéristique de topologie des cartes de Kohonen, tous les centroïdes des unités étant au même niveau liés entre eux par la notion de voisinage. Dans la distribution finale obtenue, chaque cluster tient compte de la totalité des données grâce à la notion de voisinage.

### 4.2.1 Description de la méthode

L'idée principale de la méthode est de guider l'algorithme de Kohonen jusqu'à ce que la carte obtenue soit la plus proche d'une configuration souhaitée. Cela est assuré en apportant des modifications sur la carte de manière à favoriser la convergence de cette configuration, en exécutant itérativement l'algorithme et en le laissant s'adapter automatiquement aux jeux de données proposés.

Si on suppose que les données possèdent une structuration en classes, on peut imaginer diverses distributions possibles comme par exemple la distribution gaussienne, très présente dans les phénomènes naturels et réels. Or dans notre cas, nous traitons des séries temporelles dont les valeurs viennent souvent d'un phénomène réel, qu'il soit naturel, économique ou social.

Notre hypothèse est que la distribution des données dans la carte obtenue tendra à être proche d'une distribution gaussienne. Si on considère un cluster de la carte qui représente une classe de données, dans une distribution gaussienne des éléments, on remarque une concentration autour du centroïde qui représente la moyenne des éléments de la classe. Ensuite, plus on s'approche des bords plus les éléments seront éparpillés et peu nombreux. En remarquant que les centroïdes ne sont pas indépendants mais calculés par la notion de voisinage, quand une classe est assez large ou que les centroïdes sont assez proches les uns des autres ou les deux, on devrait voir le même phénomène sur la carte ; le centroïde le plus proche du centre de la classe aura la concentration de vecteurs la plus forte, avec une tendance décroissante pour les centroïdes aux voisinages de moins en moins proches. La figure 4.2 utilise cet exemple sur une carte du type linéaire.

La figure 4.3 présente le nombre de vecteurs associés à chaque cluster pour représenter la distribution. Nous observons dans cet exemple deux régions distinctes avec une concentration maximale autour des unités 8 et 14. Les deux régions sont séparées par un grand nombre de clusters "vides" faisant apparaitre deux classes distincte. Pour la phase de prévision, nous avons deux choix possibles : soit on considère chaque unité comme un cluster pour la prévision locale, soit on utilise les groupement d'unités dans la carte pour constituer des classes qui serviront pour la prévision locale.

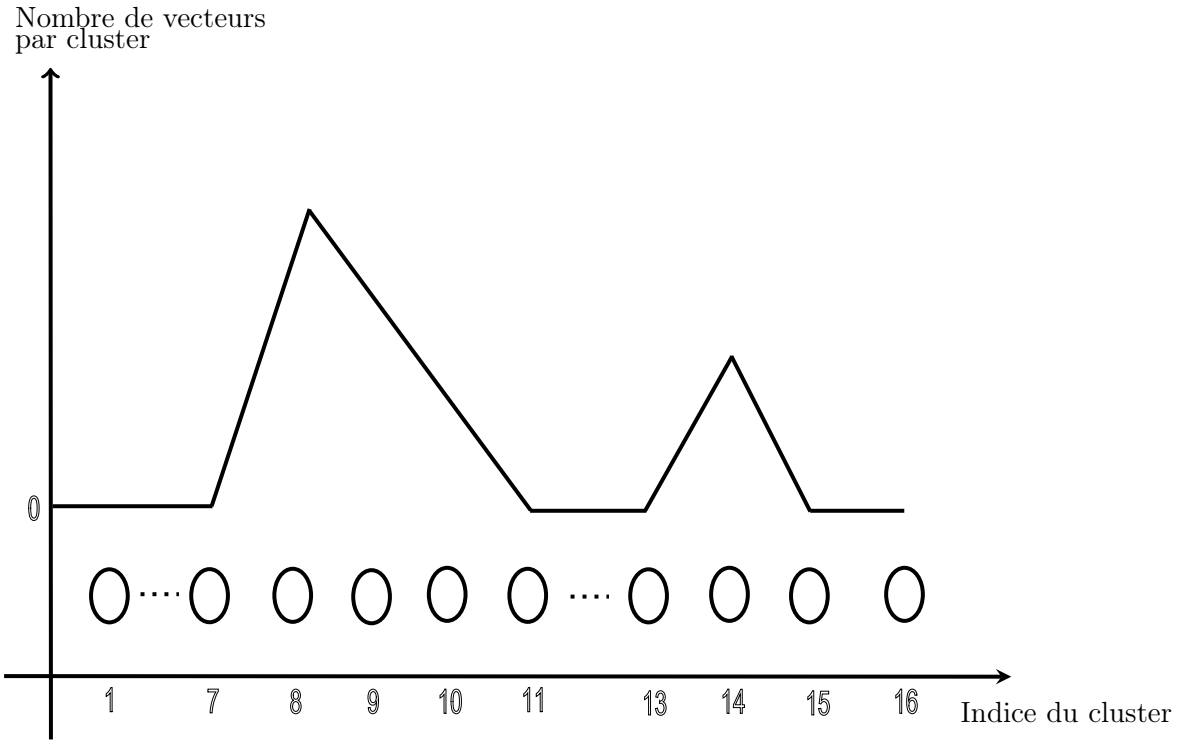


FIGURE 4.2 – Résultat de clustering dans une carte linéaire respectant la distribution gaussienne

Notre proposition consiste à répéter itérativement l'utilisation des algorithmes des cartes de Kohonen en introduisant à chaque nouvelle itération de nouvelles unités jusqu'à obtenir une configuration semblable à la figure 4.2. Le nombre d'unités à ajouter est paramétrable mais c'est un paramètre moins important que la taille de la carte dans l'algorithme classique. La figure 4.3 montre une étape de la procédure d'ajout de neurone.

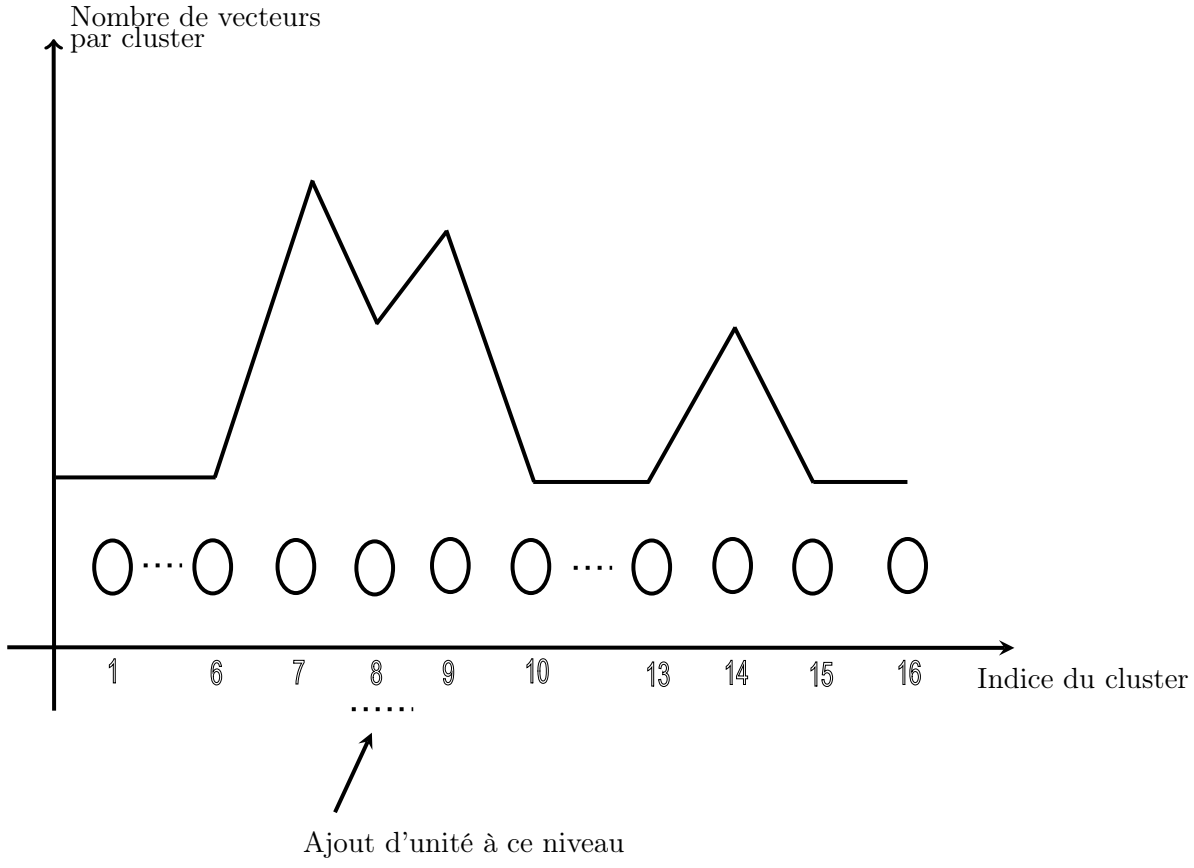


FIGURE 4.3 – Exemple d'ajout d'unités dans une carte linéaire

Une façon de modéliser mathématiquement ce problème sur une carte linéaire consiste à suivre l'allure de la distribution dans le voisinage de la carte. On considère une fonction théorique notée $\tilde{f}$ qui représente la valeur d'un critère $c$ en fonction de l'indice du cluster sur la carte. Cette fonction peut être vue comme une superposition de distributions représentant les différentes classes comme décrites dans l'équation 4.10.

$$\tilde{f}(x) = \sum_{i=1}^{t} \tilde{f}_i^c(x) \tag{4.10}$$

où $t$ est le nombre de classes qui apparaissent dans la carte, avec

$$\tilde{f}_i^c(x) = a_i e^{\frac{-(x-b_i)^2}{2\sigma_i^2}} \tag{4.11}$$

où l'indice $c$ représente le critère pris en compte par la fonction, $b_i$ représente l'indice du cluster le plus proche du vrai centre de la classe, $a_i$ est l'amplitude de la gaussienne et $\sigma_i$ représente la largeur de la gaussienne.

Nous pouvons approximer cette allure en considérant une marge $\Delta$ entre la fonction théorique $\tilde{f}$ et la fonction approchée $f$ (voir la figure 4.4). Cette représentation possède l'avantage d'être généralisable à plusieurs autres formes de distribution. Le principal inconvénient de cette méthode est le fait qu'elle soit à forte contrainte sur la distribution de la carte. De plus, dans le cas général il n'est pas possible d'estimer à l'avance les informations sur les paramètres de la gaussienne telle que la largeur ou l'amplitude.

$$\left| f(i) - \tilde{f}(i) \right| \leq \Delta \tag{4.12}$$

Quand la condition n'est plus valide sur un indice $i$, ce dernier devient une position d'insertion de nouvelles unités.

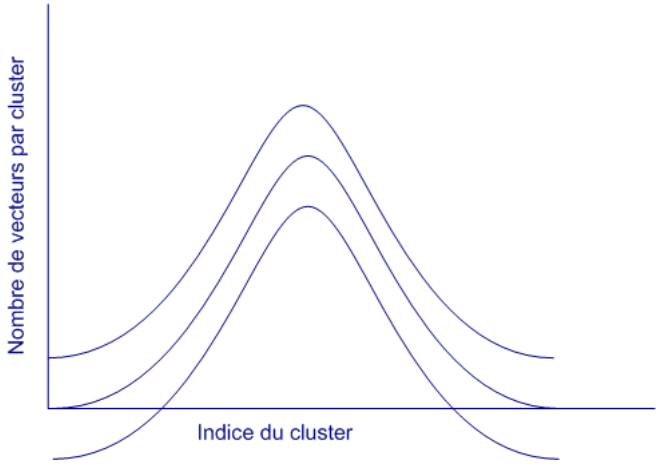


FIGURE 4.4 – Fonction de nombre de vecteurs par cluster délimité par une marge $\Delta$

Nous proposons une autre représentation mathématique du problème, qui se base uniquement sur les variations de la fonction approchée $f$. L'algorithme développé fonctionne d'une manière simple. Pour chaque nouvelle carte obtenue, on vérifie la variation des nombres d'éléments dans la carte. L'objectif est d'avoir des concentrations fortes près du centre et moins sur l'extrémité. Il suffit de chercher des variations dans cette fonction qui ne passent pas par une zone vide comme le cas de l'unité 8 de la figure 4.2. Ensuite, dans chacune de ces positions, nous ajoutons un certain nombre de neurones. Les neurones ajoutés sont

initialisés aléatoirement, ce qui va "exciter" plus ou moins les neurones dans le voisinage le plus proche et beaucoup moins voire pas du tout les clusters les plus éloignés.

On pose

$$f^{'}(i) = f(i) - f(i-1) \tag{4.13}$$

une approximation de la dérivée de $f$. Le signe de $f^{'}$ permet de facilement conclure la variation de $f$. Afin de détecter les positions d'insertions, il suffit à l'algorithme de trouver une position $i$ qui vérifie les trois conditions suivantes :

$$f^{'}(i+1) > 0 \tag{4.14}$$

$$\exists k \ / \ f^{'}(k) < 0 \ et \ \forall j \in ]k, i] \ f^{'}(j) \leq 0 \tag{4.15}$$

$$f(i) \neq 0 \tag{4.16}$$

La condition 4.14 implique que $f$ entre $i+1$ et $i$ est strictement croissante. La condition 4.15 implique que dans l'intervalle $[k, i]$ $f$ est décroissante. Donc la position $i$ indique clairement un changement de variation, la condition 4.16 vient vérifier si le changement de variation est fait après un cluster vide (ou une zone d'unités non affectées).

**Algorithm 4** Description de l'algorithme des Auto-SOM

1- Initialiser la grille de SOM
2- Lancer l'apprentissage SOM sur la carte
3- Vérifier les distributions sur la carte
4- Si les distributions ne sont pas séparées, insérer de nouveaux neurones et aller à 2.
5- Fin de l'algorithme

De plus, afin d'assurer la convergence, nous pouvons limiter le nombre d'itérations. L'algorithme s'arrêtera même si la condition sur la carte n'est toujours pas valide. D'après la description de l'algorithme, la méthode utilise uniquement la sortie des SOM c'est-à-dire la carte. Nous pouvons ainsi utiliser une grande variété d'algorithmes. Nous utiliserons des TKM et des RSOM, les plus simples à mettre en œuvre.

La réflexion de base a été faite sur le modèle linéaire de la carte, mais ceci peut être facilement étendu vers d'autres types de topologies et plus spécifiquement les cartes rectangulaires.

### 4.2.2 Expérimentations préliminaires

Nous présentons dans ce paragraphe des résultats préliminaires de la méthode auto-SOM. On se contentera dans ce chapitre de remarquer l'intérêt et l'apport de cette approche locale par rapport à un prédicteur global et de comparer avec les algorithmes classiques SOM. On

utilisera l'algorithme comme méthode de clustering de base combinée à chaque fois avec un des prédicteurs habituellement utilisés.

Nous utiliserons les mêmes configurations arbitrairement choisies pour le prédicteur, ce qui nous permettra d'évaluer l'impact de l'introduction de l'approche locale. Ces configurations engloberont les paramètres spécifiques du prédicteur, c'est-à-dire de l'algorithme d'apprentissage, mais aussi des paramètres qui ne lui sont pas reliés, en particulier la fenêtre temporelle. On prendra à titre d'exemple une fenêtre temporelle de taille 6 dans toutes les expérimentations sauf dans le cas des RNN où il n'est pas nécessaire d'en utiliser une. Les expérimentions ont été réalisées sur les trois séries de références : les taches solaires, Laser et MG-17. Nous utiliserons la NMSE comme mesure de performance.

Le tableau 4.5 présente les résultats des prévisions sur la série des taches solaires. On remarque que les performances de tous les prédicteurs ont été améliorées par rapport aux prédicteurs globaux. On voit bien que le MLP a été nettement amélioré avec une erreur divisée par presque 4 en passant de $0,0059$ à $0,0016$ et dépasse par la même occasion les performance obtenues en utilisant les SOM classiques. La seule différence avec ces dernières c'est que dans l'expérimention, nous avons utilisé une carte de taille arbitraire (on a choisi 15 unités avec une carte linéaire) alors qu'auto-SOM finissait toujours avec une carte d'une soixantaine de nœuds.

Dans le cas des SVM, nous enregistrons à peu près la même amélioration de l'erreur qui est divisée par plus de 3 qui touche à la fois les ensembles test1 et test2, en enregistrant un passage de l'erreur NMSE de $0,0094$ à $0,0029$ et dépassant de loin l'erreur obtenue en utilisant l'approche locale SOM classique. Il est important de remarquer que les SVM étaient les moins performants car nous avons choisi dans ces expérimentations préliminaires de passer outre le grid search qui est une phase importante pour trouver les meilleures performances. Nous pouvons ramener ce résultat au fait que le clustering le plus adapté crée des vecteurs de plus en plus proches et homogènes, ce qui simplifie la tâche des SVM pour la régression.

| Méthode | NMSE Test set 1 | NMSE Test set 2 |
|---|---|---|
| MLP | $0,0059$ | $0,0191$ |
| SVM | $0,0094$ | $0,0256$ |
| Local MLP | $0,0016$ | $0,0197$ |
| Local SVM | $0,0029$ | $0,0189$ |

TABLE 4.5 – Tableau des résultats préliminaires sur la série temporelle Sunspots

Le tableau 4.6 présente les résultats des prévisions sur la série Laser. Dans le cas des approches locales par SOM, les MLP locaux réalisaient les meilleures performances, ce que l'on retrouve dans ces expérimentations. Les MLP réalisent de meilleures performances que celles obtenues en conjonction avec les SOM classiques $(0,002)$. Ceci montre encore une fois que le nombre de classes est un facteur important pour l'efficacité de l'approche locale où nous obtenons dans la plupart des cas des cartes assez grandes de l'ordre de plusieurs centaines de nœuds (nous aborderons cet aspect plus en détails dans le chapitre suivant).

Les performances des SVM voient eux aussi une nette amélioration de $0,055$ à $0,037$, dépassant aussi les résultats obtenus par les SOM classiques, en utilisant une même configuration non optimisée.

| Méthode | NMSE Test set |
|---|---|
| MLP | $0,023$ |
| SVM | $0,055$ |
| Local MLP | $0,002$ |
| Local SVM | $0,037$ |

TABLE 4.6 – Tableau des résultats préliminaires sur la série temporelle Laser

Le tableau 4.7 présente les résultats d'expérimentations préliminaires des prévisions sur la série Mackey-Glass (17). Les meilleures performances sont celles des MLP avec une NMSE de $0,019$, mais en général ces résultats ne diffèrent pas énormément des résultats obtenus avec les SOM avec une amélioration peu significative. Nous remarquons que pour tous les prédicteurs les auto-SOM permettent d'obtenir un résultat meilleur que les SOM classiques lorsque les deux méthodes sont soumises aux même conditions.

| Méthode | NMSE Test set $\times 10^3$ |
|---|---|
| MLP | $0,321$ |
| SVM | $0,743$ |
| Local MLP | $0,019$ |
| Local SVM | $0,520$ |

TABLE 4.7 – Tableau des résultats préliminaires sur la série temporelle Mackey-Glass (17)

Toutefois, il faut remarquer que les SOM classiques peuvent conduire à des résultats qui sont de l'ordre de ce que donne auto-SOM, à condition de connaitre au préalable la taille adéquate de la carte. Nous verrons dans le chapitre suivant comment ce paramètre peut influencer les performances du prédicteur. En pratique plus la taille de la carte augmente plus les données seront distribuées dans la carte, d'où l'intérêt de cet apport.

L'avantage de la méthode auto-SOM est de permettre au système d'être adaptable vis-à-vis du nombre de clusters. Toutefois, on remarque que l'écart type pour certaines données peut être plus élevé. Ceci soulève un problème de stabilité des performances de la méthode. En effet, utiliser nombre de vecteurs comme information par cluster est une heuristique qui permet de guider l'algorithme mais qui reste imprécise. C'est pour cela que dans certains cas, l'algorithme auto-SOM s'arrête d'une manière assez rapide et obtient ainsi un nombre de clusters inférieur à celui qui pourrait être optimal.

Pour cette raison, nous pouvons imaginer des moyens d'améliorer cette approche, par exemple en remplaçant et/ou combinant d'autres paramètres optimisant l'utilisation de la carte. Mais dans la suite de nos travaux, nous avons choisi d'opter pour une autre démarche, consistant à relier le clustering aux performances des prédicteurs locaux, selon une approche locale hiérarchique.

## 4.3 Approche locale hiérarchique

Les motivations exposées précédemment sont à l'origine de cette nouvelle approche. Nous cherchons à mettre en place une approche locale qui soit la plus souple possible par rapport données auxquelles elle peut être confrontée. Pour cela, nous voulons que l'algorithme cherche automatiquement à s'adapter aux données et trouve le clustering et le nombre de classes adéquat.

Nous avons fait le choix d'adopter des cartes de Kohonen. Cependant cette dernière approche possède plusieurs inconvénients. Dans certaines expérimentations, nous obtenons des cartes de très grande dimension et avec un grand nombre de clusters vides, ce qui peut créer un gaspillage de mémoire qui peut devenir rapidement gênant sur des machines disposent de ressources limitées. Un autre inconvénient qui peut survenir est l'obtention des clusters de très faibles tailles, surtout lorsque la carte est très grande.

Nous proposons des techniques d'approches locales hiérarchiques basées sur les arbres binaires. Plusieurs avantages peuvent justifier l'utilisation de telles approches. Nous pouvons contrôler la profondeur de l'arbre en fonction de plusieurs paramètres. Nous pouvons considérer, par exemple, la taille des clusters ; nous éviterons ainsi d'obtenir des clusters de petite taille. Nous pouvons aussi lier la profondeur du clustering aux résultats de prédicteurs afin d'éviter une explosion combinatoire.

L'utilisation des approches locales hiérarchiques ouvre la possibilité de nouvelles stratégies. Par exemple, si notre objectif est d'obtenir un assez bon résultat tout en évitant de passer un temps d'apprentissage très grand, nous pouvons alors imaginer une stratégie d'arrêt dans laquelle chaque niveau de clustering est évalué sur les performances des prédicteurs, nous permettant ainsi de décider de continuer en profondeur ou de s'arrêter à ce niveau.

Inversement, si notre objectif principal est de chercher le clustering qui donne les meilleures performances sans prêter attention à la durée d'apprentissage, une approche du type élagage peut être la plus adéquate, d'où l'idée de trouver les meilleurs clusters possibles pour obtenir les meilleures performances en partant des feuilles de l'arbre.

Quelle que soit la stratégie adoptée, le développement de l'arbre est effectué par l'algorithme des *k-Means*. Autrement dit, pour chaque nœud de l'arbre, un k-means à deux centroïdes ($k = 2$) est applique sur les vecteurs associés à ce nœud. Le choix de l'algorithme *k-Means* est lié à sa rapidité et sa simplicité. En plus nous avons opté pour des arbres binaires ($k = 2$). De plus, dans un premier temps nous souhaitons tester une méthode simple.

### 4.3.1 Description des méthodes

Un arbre binaire est un graphe. Nous pouvons dans le cas général définir un graphe par le triplet :

$$G = (V, E; \gamma) \tag{4.17}$$

avec $V$ l'ensemble de sommets. Dans notre cas, les sommets sont les nœuds de l'arbre. $E$ est l'ensemble des arêtes qui relient les sommets entre eux et $\gamma$ la fonction d'incidence qui associe à chaque arête une paire de sommets.

$$\begin{array}{rccc} \gamma & : & E & \rightarrow & V \times V \\ & & a & \mapsto & (v_1, v_2) \end{array} \tag{4.18}$$

Pour simplifier, on notera l'arbre $T = (V, E)$, on appellera les sommets nœuds et on supposera les arêtes orientées. Ainsi l'exemple de l'équation 4.18 est orienté du sommet $v_1$ vers le sommet $v_2$.

Nous pouvons discerner trois types de nœuds dans un arbre binaire : un nœud racine, un nœud feuille et un nœud quelconque.

Le nœud racine, noté $r$, ne peut posséder que deux arêtes représentant deux nœuds fils. Cette condition est définie par 4.19. Il existe un seul nœud racine par arbre.

$$\exists!(a_1\, a_2) \in E^2 \ tel\ que\ \gamma(a_1) = (r, v_1)\ et\ \gamma(a_2) = (r, v2) \tag{4.19}$$

avec $v_1$ et $v_2 \in V$ et $v_1 \neq v_2$.

Les nœuds feuilles notés $f$ sont définis par la relation 4.20, avec $\mathcal{F}$ l'ensemble des nœuds feuilles d'un arbre.

$$\forall f \in \mathcal{F},\ \exists!a \in E\ tel\ que\ \gamma(a) = (v_1, f) \tag{4.20}$$

c'est-à-dire qu'un nœud feuille ne possède qu'une seule arête qui le relie au nœud père.

Les nœuds génériques sont les nœuds de l'arbre qui ne sont ni des feuilles ni une racine, que nous pouvons définir comme étant des nœuds à trois arêtes :

$$\forall n \in V\ tel\ que \begin{cases} \gamma(a_1) = (n, v_1) \\ \gamma(a_2) = (n, v_2) \\ \gamma(a_3) = (v_3, n) \end{cases} \tag{4.21}$$

avec $v_1 \neq v_2 \neq v_3 \in V$. Selon l'orientation des arêtes, les nœuds $v_1$ et $v_2$ représentent les descendants de $n$, tandis que le nœud $v_3$ représente le père.

Les arbres que nous générons sont tous des arbres complets dont les nœuds respectent les définitions citées ci-dessus. Nous pouvons vérifier qu'un arbre binaire est complet lorsqu'il n'existe qu'un seul nœud de type racine qui vérifie 4.19.

On rappelle que pour un arbre binaire complet de hauteur $h \geq 1$, le nombre de nœuds suit une croissance exponentielle avec

$$Card(V) = \sum_{i=0}^{h} 2^i \tag{4.22}$$

On associe à chaque nœud $v_i$ de l'arbre un ensemble de vecteurs de l'ensemble d'apprentissage $A$. Ces sous-ensembles qu'on notera $A(v_i)$ représenteront les clusters obtenus par les *k-Means*.

On définit une coupe horizontale de l'arbre comme étant un clustering de l'ensemble A tel que

$$A = \bigcup_{i=1}^{k} A(v_i) \; et \; \bigcap_{i=1}^{k} A(v_i) = \emptyset \tag{4.23}$$

où $k$ est le nombre de nœuds dans la coupe la horizontale. Le nœud $r$ peut être considéré comme une coupe horizontale également (puisque $A(r) = A$), tout comme l'ensemble $\mathcal{F}$ est une coupe horizontale.

Nous utiliserons ce formalisme pour décrire les deux méthodes d'approche locale hiérarchiques qui peuvent être vues comme deux stratégies différentes, une stratégie de *pré-pruning* dans laquelle l'élagage est fait au fur et à mesure que l'arbre est développé, et une stratégie post-pruning dans laquelle le pruning est fait une fois l'arbre développé.

### Approche pré-pruning

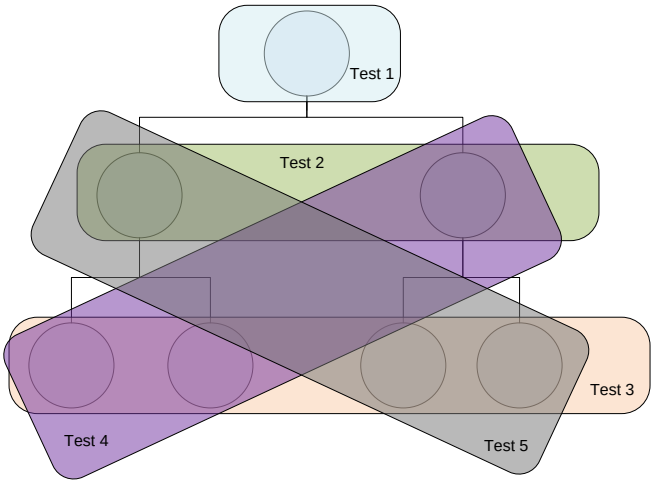


FIGURE 4.5 – Exemple de stratégie pré-pruning

Dans cet algorithme le développement de l'arbre se fait en parallèle avec l'exécution des prédicteurs. Plus exactement, à chaque itération et pour chaque nœud qui n'a pas encore été traité par le prédicteur, on développe successivement deux niveaux de l'arbre.

Nous obtenons donc à chacun nouvelle itération un nouvel arbre binaire complet à deux niveaux ($h = 2$) avec un nœud racine, deux nœuds quelconques et quatre nœuds feuilles que nous noterons respectivement $r$, $n_1$, $n_2$ et $f_i$ avec $1 \leq i \leq 4$, comme le montre la figure 4.5.

Les prédicteurs sont séparément appliqués sur les 7 nœuds, calculant ainsi les erreurs de prévision sur l'ensemble d'apprentissage et de validation.

A la lumière de cette décomposition, des évaluations sont effectuées, comme le montre la figure 4.5, définissant des coupes horizontales de l'arbre. Pour un arbre de hauteur $h = 2$, il existe cinq combinaisons possibles pour des coupes horizontales :

$$C_1 = \{r\} \tag{4.24}$$

$$C_2 = \mathcal{F} \tag{4.25}$$

$$C_3 = \{\forall n \in V \ tel \ que \ n \neq r \ et \ n \notin \mathcal{F}\} \tag{4.26}$$

$$C_4 = \{n_1 \in V \ et \ v_1, v_2 \in \mathcal{F}, \ tel \ que \ n_2 \in C_3 \ avec \ \left\{ \begin{array}{l} \gamma(a_1) = (n_2, v_1) \\ \gamma(a_2) = (n_2, v_2) \\ n_1 \neq n_2 \end{array} \right\}\} \tag{4.27}$$

$$C_5 = \{n_2 \in V \ et \ v_1, v_2 \in \mathcal{F}, \ tel \ que \ n_1 \in C_3 \ avec \ \left\{ \begin{array}{l} \gamma(a_1) = (n_1, v_1) \\ \gamma(a_2) = (n_1, v_2) \\ n_1 \neq n_2 \end{array} \right\}\} \tag{4.28}$$

où la coupe $C_1$ représente la racine, la coupe $C_2$ représente les feuilles, la coupe $C_3$ représente le niveau 1 de l'arbre, les coupes $C_4$ et $C_5$ représentent une combinaison d'un nœud du niveau 1 de l'arbre et les deux fils du deuxième nœud du niveau 1.

L'algorithme proposé utilise une liste $L_c$ de nœuds à développer. Au fur et à mesure, des nouveaux nœuds sont ajoutés et d'autres sont retirés.

En comparant les performances en erreur de prédiction globale associées aux cinq coupes horizontales montrées explicitement dans la figure 4.5. La coupe ayant l'erreur de prédiction la plus faible est sélectionnée et les feuilles associées sont ajoutées à la liste $L_c$. L'algorithme répète ainsi les opérations décrites ci-dessus jusqu'à ce que la liste $L_c$ devienne vide, ne contenant plus de nœuds à développer. À ce moment-là, les nœuds de la liste $L_c$ n'appartenant pas à $\mathcal{F}$ composent la coupe finale.

Parmi les nœuds sélectionnés, les feuilles sont insérées dans la liste des nœuds à développer $L_c$, les autres nœuds sont insérés dans la liste des nœuds finaux $L_f$. Cet algorithme est décrit plus formellement dans la description l'algorithme ci-dessous :

L'avantage de cette approche est que le clustering est guidé par les résultats de la prévision. Nous pouvons aussi envisager la taille des clusters comme critère supplémentaire, ce qui éviterait d'avoir des clusters trop petits, faussant ainsi les résultats. L'inconvénient de cette approche est l'optimisation. En effet, à chaque itération l'algorithme étant évalué sur une profondeur limitée à deux niveaux, il existe un risque que l'algorithme s'arrête sur un niveau qui ne représente pas la solution optimale.

Pour cette raison, lorsque le temps d'apprentissage n'est pas le souci principal, nous avons aussi proposé l'approche *Bottom-Up* (exploration en profondeur) dans laquelle l'arbre est exploré à fond.

**Algorithm 5** Description de la stratégie pré-pruning

$L_c \leftarrow R$
**while** $\neg vide(L_c)$ **do**
  $n \leftarrow L_c(0)$
  $Developper(n)$
  $Predict(n)$
  $C \leftarrow trouver\_coupe\_min(n)$
  $L_c \leftarrow m \in C$ , *tel que* $m \in \mathcal{F}$
  $L_f \leftarrow m \in C$ , *tel que* $m \notin \mathcal{F}$
**end while**
retouner $L_f$

### La stratégie Post-pruning

Dans la section précédente, nous avons vu comment l'algorithme sélectionne la coupe horizontale optimale d'un arbre de hauteur $h = 2$. Dans ce paragraphe l'algorithme permet de trouver la coupe optimale dans l'arbre décomposé au préalable au maximum.

Nous appelons cette stratégie *post-pruning* car l'élagage se fait à la fin de la construction de l'arbre, en essayant de trouver la coupe horizontale optimale sur la totalité de l'arbre. L'algorithme se caractérise par trois phases : une phase de construction de l'arbre (ou clustering), une phase d'apprentissage des prédicteurs et une phase d'élagage.

Dans la phase de construction de l'arbre, ce dernier est développé au maximum selon un ou plusieurs critères d'arrêts possibles. Nous pouvons par exemple utiliser la profondeur pour contrôler l'explosion combinatoire de l'arbre ou encore utiliser la taille des clusters pour éviter le sous-apprentissage.

Une fois la phase de construction de l'arbre terminée, les prédicteurs sont lancés sur la totalité des nœuds de l'arbre, de la racine jusqu'aux feuilles. C'est la phase la plus coûteuse en temps et c'est la raison pour laquelle les critères d'arrêt peuvent jouer un rôle important.

Pour avoir la meilleure évaluation possible d'un noeud de l'arbre, il faut classiquement prendre en considération l'erreur de prédiction sur des données différentes de l'ensemble d'apprentissage. C'est pour cette raison que nous avons laissé la possibilité d'effectuer une validation croisée, utile pour les jeux de données de moyenne ou petite taille.

La troisième phase revient à réaliser l'élagage à la lumière des performances réalisées sur chaque nœud. Pour cela nous proposons un algorithme en $O(n)$, avec n le nombre de noeuds (7).

L'algorithme utilise une liste qui représente la coupe horizontale finale de l'arbre, notée $L_f$, et une liste temporaire $L_t$.

Au départ la liste est initialisée par une coupe horizontale quelconque et peut être la racine ou l'ensemble des feuilles $\mathcal{F}$. À chaque itération un nœud est sélectionné, le sens de parcours est arbitraire. Chaque nœud sélectionné, remplace tous ses descendants et/ou ancêtres qui se trouvent dans la liste $L_f$.

Pour cela, on définit deux opérateurs, $Descendant(v_i)$ et $Parent(v_i)$, qui permettent d'obtenir respectivement l'ensemble des nœuds descendants de $v_i$ et l'ensemble des nœuds

parents de $v_i$ définis par les conditions :

$$v_j \in parent(v_i) \ si \quad \begin{array}{l} \gamma(a) = (v_j, v_i) \ avec \ a \in E \\ ou \\ \exists v_k \in V \ tel \ que \ \gamma(b) = (v_j, v_k) \ et \ v_k \in parent(v_i) \ avec \ b \in E \end{array} \tag{4.29}$$

$$v_j \in descendant(v_i) \ si \quad \begin{array}{l} \gamma(a) = (v_i, v_j) \ avec \ a \in E \\ ou \\ \exists v_k \in V \ tel \ que \ \gamma(b) = (v_k, v_j) \ et \ v_k \in descendant(v_i) \ avec \ b \in E \end{array} \tag{4.30}$$

Si on prend en compte le parcours des nœuds du plus bas au plus haut, c'est-à-dire jusqu'à la racine, il est clair qu'il suffit de se baser uniquement sur l'opérateur descendant de chaque nœud.

Les performances globales en erreur de prédiction sont alors comparées entre les deux coupes horizontales associées à $L_c$ et $L_t$. Si l'erreur de $L_t$ est inférieure, cette dernière remplace la liste $L_c$. Ces traitements sont répétés jusqu'à avoir parcouru tous les nœuds de l'arbre, y compris la racine. L'algorithme 7 donne une description plus formelle du procédé d'élagage.

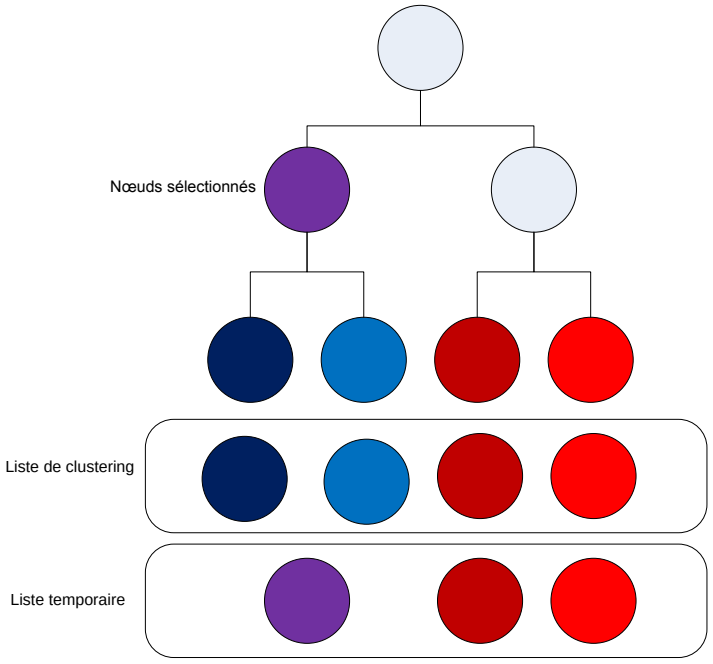


FIGURE 4.6 – Exemple de stratégie post-pruning

**Algorithm 6** Procédure d'élagage du post-pruning

$L_f \leftarrow \mathcal{F}$
$MSE \leftarrow calculErreurGlobale(L_f)$
**for all** $v_i \in \mathcal{V} - \mathcal{F}$ **do**
  $L_t \leftarrow L_f - Descendent(v_i) + \{v_i\}$
  $MSE_t \leftarrow calculErreurGlobale(L_t)$
  **if** $MSE_t < MSE$ **then**
    $MSE \leftarrow MSE_t$
    $L_f \leftarrow L_t$
  **end if**
**end for**
retouner $L_f$

**Algorithm 7** Elagage post-pruning

1- Phase de clustering : développer l'arbre au maximum.
2- Phase de prédiction : effectuer la prédiction sur tous les nœuds de l'arbre.
3- Phase d'élagage : recherche du clustering optimal.

**Une troisième stratégie : le mapping des feuilles**

La stratégie précédente consiste à évaluer les coupes possibles en partant du bas de l'arbre. Elle vise à favoriser une décomposition minimale des données et à obtenir des clusters séparés, comme toutes les approches locales.

Cependant, avec les expériences que nous avons menées, nous avons remarqué l'apparition de configurations particulières. Il s'agit du cas où un nœud décomposé en deux fils, l'erreur de prévision obtenue avec le père étant meilleure que celle de la composition des deux fils, les stratégies proposées vont sélectionner le père, ignorant la performance des deux fils séparément. Par exemple, le cas où un nœud père possède deux fils dont l'un donne un prédicteur avec une très bonne performance et l'autre donne un prédicteur avec une très mauvaise performance. La contribution de ce dernier à l'erreur globale est tellement grande que l'erreur combinée des deux fils est supérieure à celle du père.

L'idée de la stratégie de mapping est d'exploiter tous les prédicteurs des nœuds obtenus de l'arbre. Pour cela, nous associons à chaque nœud certaines caractéristiques. Ainsi on définit pour chaque nœud l'erreur de prévision locale notée $E_i$ et $\tau_i$ le taux d'exemple affecté et déterminé par :

$$\tau_i = \frac{Card(n_i)}{Card(r)} \tag{4.31}$$

où $Card(n)$ représente le nombre d'exemples affectés au nœud $n$.

Cela nous permet d'établir, pour chaque nœud, la contribution $c_i$ du nœud à l'erreur de prévision totale, donnée par l'expression suivante :

$$c_i = \tau_i \times E_i \tag{4.32}$$

L'erreur de prévision finale s'écrit sous la forme suivante :

$$NMSE = \sum_{x_i \in r} \frac{1}{Card(r)} (x_i - \hat{x}_i)^2 \tag{4.33}$$

en remarquant que les classes qui composent la coupe $C$ sont disjointes et complémentaires, la NMSE peut encore être décomposée sous cette forme.

$$NMSE = \frac{1}{Card(r)} \sum_{j=0}^{H} \sum_{x_i \in C_j} (x_i - \hat{x}_i)^2 \tag{4.34}$$

où $H$ représente le nombre de clusters dans la coupe. En faisant sortir les NMSE locales de l'expression précédente, nous pouvons écrire

$$NMSE = \sum_{j=1}^{H} \sum_{x_i \in C_j} \frac{Card(n_j)}{Card(r)} \frac{1}{Card(n_j)} (x_i - \hat{x}_i)^2 \tag{4.35}$$

Ce qui permet d'établir que l'erreur totale d'une coupe horizontale $C$ de l'arbre peut être écrite par la somme des contributions.

$$NMSE = \sum_{n_i \in C} c_i \tag{4.36}$$

En remarquant que les contribution sont des éléments positifs, minimiser l'erreur de prévision globale d'un modèle peut se ramener à sélectionner les nœuds donnant les plus faibles contributions. Dans ce cadre $c_i$ représente aussi l'efficacité d'un prédicteur associé à un nœud particulier. Dans cette stratégie, on vise à exploiter cette information comme le montre la figure 4.7.

Cette stratégie nous ramène donc à associer à chaque feuille de l'arbre le nœud de la même branche ayant la plus faible contribution. Avec cette stratégie nous exploitons tous les prédicteurs exploités dans l'arbre et évitons le cas où un des deux fils impacte négativement sur la performance globale. Dans la figure 4.7, les nœuds en gris représentent les prédicteurs sélectionnés. Les feuilles grises possèdent la contribution minimale de la branche à laquelle elles appartiennent. La feuille blanche indique qu'elle ne possède pas la contribution minimale de sa branche. Si un vecteur dans l'ensemble test est affecté à cette feuille, le prédicteur associé au nœud supérieure qui est indiqué en gris dans la figure sera utilisé pour la prévision. On peut dire que le prédicteur ayant obtenu une plus faible contribution par rapport à la feuille par rapport à l'ensemble l'apprentissage, est statistiquement le meilleur nœud à utiliser le père.

### 4.3.2 Expérimentation préliminaire

Nous présenterons quelques expérimentations préliminaires afin de confirmer l'intérêt des approches hiérarchiques. Nous reprenons les conditions des expérimentations précédentes, toujours sur les mêmes séries standard. Nous fournissons à chaque fois les performances obtenues par les prédicteurs globaux, les prédicteurs avec pré-pruning et les prédicteurs avec post-pruning.

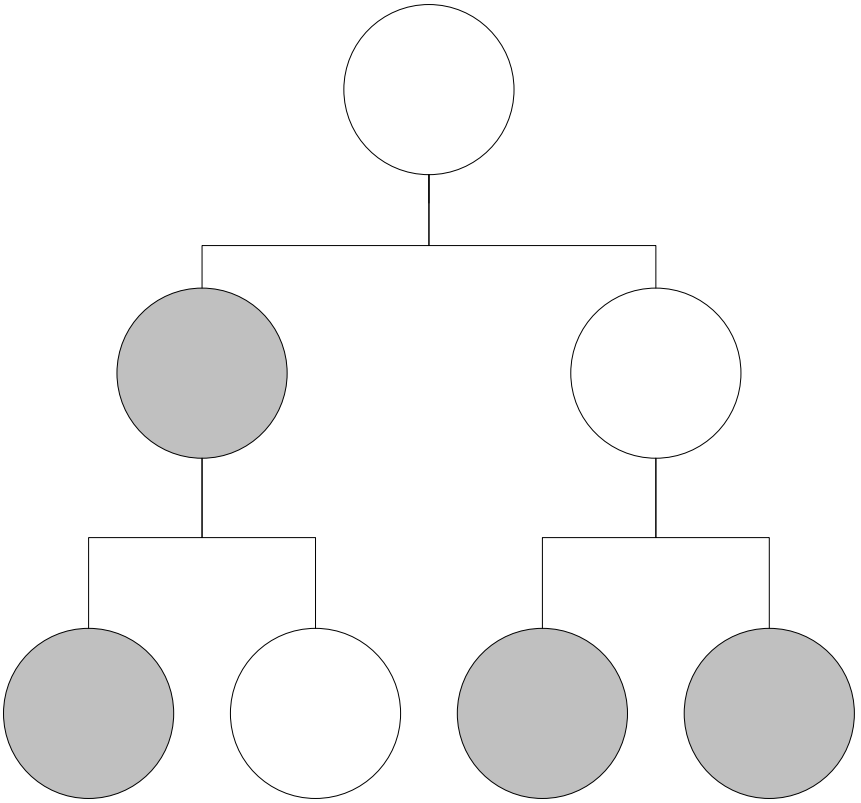

FIGURE 4.7 – Exemple de stratégie post-pruning

De par la nature même des algorithmes décrits précédemment, il est tout à fait logique que les performances obtenues par pré-pruning soient généralement compris entre celles des approches globales et celles du post-pruning et que cela soit l'inverse en matière de temps de calcul. Les détails des expérimentations tels que l'influence des résultats (tailles des fenêtres temporelles, hauteur maximale de l'arbre) ou les structures des arbres obtenus seront décrits dans le chapitre sur les expérimentations approfondies. On se contentera ici d'évoquer les résultats finaux de la NMSE.

Le tableau 4.8 présente les résultats des prévisions sur la série des taches solaires. La meilleure NMSE obtenue $0,00103$ correspond à post-pruning MLP et au mapping avec $0,0037$, ce qui peut arriver puisque le pruning s'effectue sur l'ensemble de validation. En pratique, le post-pruning permet d'avoir une NMSE plus petite que les SOM. Pour la série des taches solaires, il n'est pas nécessaire de définir des arbres très profonds (généralement la hauteur maximale était fixée aux alentours de 6). Le pré-pruning s'arrête relativement rapidement et généralement sans dépasser les approches locales SOM.

| Méthode | NMSE Test set 1 | NMSE Test set 2 |
|---|---|---|
| MLP | $0,0059$ | $0,0191$ |
| SVM | $0,0094$ | $0,0256$ |
| RNN | $0,041$ | $0,0871$ |
| post-pruning MLP | $0,0010$ | $0,0162$ |
| pré-pruning MLP | $0,0048$ | $0,0198$ |
| mapping MLP | $0,0037$ | $0,0256$ |
| post-pruning SVM | $0,0069$ | $0,0216$ |
| pré-pruning SVM | $0,0094$ | $0,0144$ |
| mapping SVM | $0,0081$ | $0,088$ |

TABLE 4.8 – Tableau des résultats préliminaires sur la série temporelle Sunspots

Le tableau 4.9 présente les résultats des prévisions sur la série Laser. La meilleure NMSE observée correspond au modèle post-pruning MLP avec $0,0015$ tandis qu'avec l'algorithme pré-pruning, on obtient une erreur plus élevée que les SOM $(0,004)$. On remarque aussi une amélioration des SVM par l'approche hiérarchique. Pour les RNN, les résultats ne différent pas de l'approche locale SOM classique. Nous avons rattaché ce problème, comme pour les expérimentations des SOM classiques, à la nature de la série.

Le tableau 4.10 présente les résultats d'expérimentations préliminaires multipliées par $\times 10^3$ sur la série Mackey-Glass. Les meilleures performances obtenues sont celles des RNN en conformité avec les résultats des cartes de Kohonen. La MSE équivalente est de $0,009$. Les RNN se sont montrés moins bons lorsque la taille de l'ensemble d'apprentissage est réduite, ce qui explique ces résultats. Toutefois, pour les autres prédicteurs, l'approche hiérarchique permet aussi d'améliorer la performance ; Pour les SVM et les MLP, l'erreur diminue presque de moitié.

Ces expérimentations nous confirment que la stratégie pre-pruning est la moins coûteuse en temps de calcul vu qu'on n'est pas obligé d'effectuer l'apprentissage dans tous les nœuds.

| Méthode | NMSE Test set |
|---|---|
| MLP | 0.023 |
| SVM | 0.055 |
| RNN | 0.008 |
| post-pruning SVM | 0, 029 |
| pré-pruning MLP | 0, 004 |
| mapping MLP | 0, 0053 |
| post-pruning MLP | 0, 0015 |
| pré-pruning SVM | 0, 041 |
| mapping SVM | 0, 044 |

TABLE 4.9 – Tableau des résultats préliminaires sur la série temporelle Laser

| Méthode | NMSE Test set $\times 10^3$ |
|---|---|
| MLP | 0.321 |
| SVM | 0, 743 |
| RNN | 0.235 |
| pré-pruning MLP | 0, 289 |
| post-pruning MLP | 0, 017 |
| mapping MLP | 0, 022 |
| pré-pruning SVM | 0, 69 |
| post-pruning SVM | 0, 378 |
| mapping SVM | 0, 046 |

TABLE 4.10 – Tableau des résultats préliminaires sur la série temporelle Mackey-Glass

Toutefois, les expérimentations préliminaires ne suffisent pas à étudier en profondeur les algorithmes proposés. A ce stade nous ne pouvons avoir de conclusions ni sur les structures de l'arbre ni sur l'influence des paramètres utilisateur.

## 4.4 Généralisation des méthodes proposées

Nous venons de présenter, dans les sections précédentes, trois apports dans le cadre de la prévision locale des séries temporelles. Un premier apport se situe au niveau des prédicteurs : nous avons introduit une modification de l'algorithme d'apprentissage des réseaux de neurones récurrents afin de les rendre adaptables aux approches locales. Sans cela, il était impossible d'utiliser la méthode de prévision basée sur les réseaux de neurones récurrents. Ensuite, nous nous sommes intéressés à la phase de clustering des données temporelles et nous avons proposé des nouvelles méthodes plus adaptables à ce type de données. Leurs particularité est de permettre d'obtenir le nombre de clusters convenable. Ceci nous permet non seulement de proposer un nouvel algorithme basé sur les cartes de Kohonen et ses variantes, mais aussi de proposer un nouveau type d'approche locale hiérarchique, utilisant à la fois les arbres binaires et les K-means. L'avantage de cette dernière approche est de pouvoir choisir le clustering des sous-ensembles en prenant en compte les performances des prédictions. Ce point en particulier nous rappelle une caractéristique des algorithmes de

boosting que nous avons vue dans le chapitre précédent.

En effet, il y a une certaine similitude avec le boosting lorsqu'on considère la succession d'apprentissages sur des sous-ensembles de l'espace des données. Nous pouvons évoquer quelques différences élémentaires. D'abord, dans le mécanisme de construction des sous ensembles où dans le boosting il s'agit d'un filtrage des données selon un critère tout en rajoutant de l'aléatoire, alors que pour l'approche locale, l'opération se résume en un clustering des données dans l'optique de construire des sous ensembles composés de vecteurs les plus proches possible les uns des autres. Ensuite, une différence majeure apparait dans la phase finale de rassemblement des résultats. Dans le boosting, une fusion des différents résultats est effectuée, autrement dit les résultats peuvent être combinés, alors qu'en approche locale, une concaténation suffit puisque la segmentation implique que les sous-ensembles sont complémentaires. Finalement, nous pouvons citer une dernière différence concernant la procédure globale où le boosting effectue un traitement itératif en boucle alors que dans l'approche locale le traitement est décrit comme étant séquentiel.

La combinaison de ces apports permet de constituer un système de prévision complet qui est décrit dans la figure 4.8. Deux parties peuvent facilement être distinguées : une partie clustering, non supervisée, et une partie prédiction, supervisée. Pour la partie clustering, deux grandes familles d'algorithmes sont envisageables. Pour la partie prédiction, trois algorithmes peuvent être utilisés pour l'apprentissage : les SVM, les MLP et les RNN. Dans la figure 4.8, les traits en pointillés qui relient les prédicteurs avec le clustering hiérarchique symbolisent la liaison entre ces deux parties.

Le résultat final obtenu est présenté en bas de la figure 4.8. Il s'agit d'un système composé de $n$ prédicteurs. Chaque nouvelle donnée, appartenant à un ensemble de tests, est d'abord réparti vers le prédicteur adéquat. Les résultats de prévision locaux sont ensuite collectés pour constituer les résultats globaux.

Tout le système est prévu à la base pour être adapté à la tâche de prévision des séries temporelles. Cependant, nous pouvons nous demander comment nos méthodes peuvent être adaptées pour les problèmes d'apprentissage dans le cas général (prévision ou classification). Est-il possible pour un tel système de résoudre des problèmes de classification tels que, par exemple, la reconnaissance de caractères ? Ce genre de problèmes nécessite dans la plupart des cas une phase préalable qui consiste à construire une base de caractéristiques numériques à partir des données originales, qui peuvent être, par exemple, des images associées aux caractères. Cette phase d'extraction de caractéristiques ne peut pas être réalisées par notre système. Elle permet d'obtenir un ensemble défini de couples comme suit :

$$\mathcal{D} = \{(V_i, c_i),\ \textit{tel que}\ V_i \in \mathbb{R}^d\ \textit{et}\ c_i \in [1, N]\} \tag{4.37}$$

où $V_i$ représente un vecteur de caractéristiques et $c_i$ représente la classe associée, $N$ étant le nombre de classes dans le problème.

Effectuer un apprentissage non supervisé sur les vecteurs $V_i$ permet d'obtenir un ensemble de clusters qui peuvent être homogènes par rapport aux classes $c_i$ ou hétérogènes

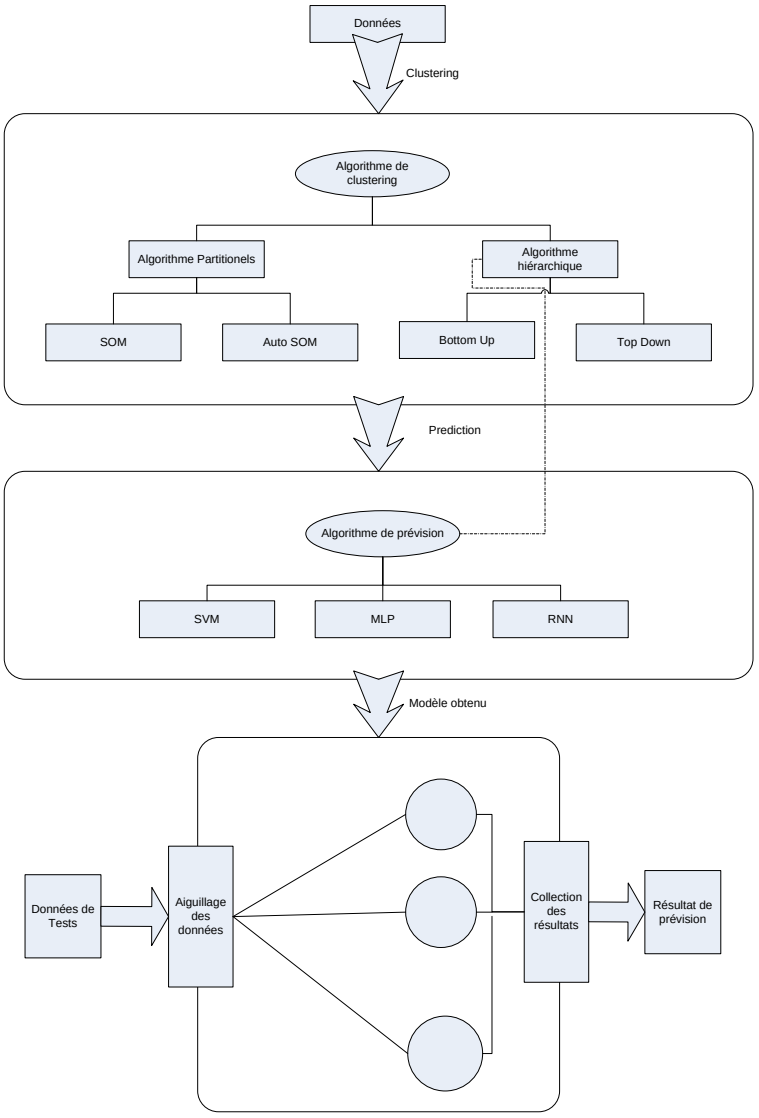


FIGURE 4.8 – Schéma global des propositions présentées dans ce chapitre

avec des clusters composés de plusieurs classes. Un apprentissage peut être envisagé sur chacun des clusters obtenus. De plus, à partir des informations de composition des clusters, nous pouvons mesurer des degrés de confiance en fonction de l'appartenance à de la classe $c_i$ et au cluster $j$.

## Conclusion du chapitre

Dans ce chapitre, nous avons présenté les apports que nous avons élaborés durant cette thèse. Nous avons d'abord introduit l'intégration des réseaux de neurones récurrents dans le cadre de l'approche locale pour la prévision des séries temporelles via une modification de l'algorithme d'apprentissage. Ensuite nous nous sommes intéressés à la partie clustering en proposant des nouvelles méthodes de clustering, l'une partitionelle et l'autre hiérarchique. Nous proposons un nouvel algorithme de carte de Kohonen afin de donner à l'algorithme plus d'adaptabilité face aux données. L'approche hiérarchique par arbre binaire est, à notre connaissance, une nouveauté par rapport aux approches locales. Une telle approche permet de choisir le degré de granularité des données (via les niveaux de l'arbre) le plus adéquat par rapport aux performances des prédicteurs sur ces données.

Nous avons également effectué des expérimentations préliminaires en utilisant des configurations similaires pour les méthodes de base dans le but d'évaluer rapidement l'impact de l'introduction du boosting et des approches locales sur les performances des différents prédicteurs. Ceci nous a permis de conclure à un résultat encourageant dans lequel l'erreur de prévision subit une chute remarquable dans certains cas, simplement en ajoutant le clustering des données et sans toucher aux prédicteurs. Toutefois de telles expérimentations ne sont pas suffisantes. Nous proposons de les étendre, dans le prochain chapitre, afin de pouvoir étudier l'impact des différents paramètres sur les performances de ces méthodes.

# Chapitre 5

# Expérimentions

## Introduction

Dans ce chapitre nous synthétiserons les expérimentations en mettant en œuvre différent apports de cette thèse. Pour ce faire, nous utiliserons des séries temporelles de références : les taches solaires, la série Laser et la série synthétiques Mackey-Glass. Nous présenterons chacune des ces séries. Ensuite nous présenterons l'apport des méthodes proposées ainsi que l'impact des paramètres les plus influents. En premier lieu on présentera l'évaluation de l'intégration des RNN dans l'approche locale. Ensuite, nous étudierons la proposition des cartes de Kohonen itératif (Auto-SOM) et ainsi que l'approche locale hiérarchique via les arbres binaires. Nous effectuerons des comparaisons avec d'autres méthodes de l'état de l'art.

## 5.1 Procédure employée

Les expérimentations ont été réalisées sur deux types de séries temporelles, constituées de données synthétiques, comme la série Mackey-Glass(17), et de données naturelles, comme la série des taches solaires (*Sunspots*) et la série Laser.

Dans la plupart des expériences, nous avons utilisé trois ensembles de données. Le premier est l'ensemble d'apprentissage, il est utilisé de manière itérative par les différentes méthodes. Le deuxième est l'ensemble de validation, qui sert à valider les résultats de l'apprentissage et aide à trouver les paramètres optimaux pour chaque jeu de données. Dans le cas des réseaux de neurones, il est aussi particulièrement utile afin d'éviter le sur-apprentissage, pour éviter une perte en pouvoir de généralisation. La détection de l'augmentation de l'erreur de validation, permet d'arrêter l'apprentissage. Le dernier ensemble est l'ensemble de tests, utilisé afin de mesurer les performances de la méthode sur de nouvelles données non présentées durant l'apprentissage ou la validation. Afin de permettre la meilleure évaluation possible, l'ensemble de test n'est présenté qu'à la fin, après avoir fixé tous les paramètres grâce aux deux premiers ensembles.

Dans le cadre de l'évaluation de nos méthodes d'approches locales, nous avons utilisé trois techniques différentes pour effectuer le clustering des séries temporelles. La première est la quantification vectorielle usuelle par le biais des cartes de Kohonen sous leur forme classique (SOM) ou sous forme des deux variantes temporelles, à savoir TKM et RSOM. La deuxième est la variante que nous proposons, Auto-SOM qui permet la recherche du nombre de clusters adéquats. Et enfin, la troisième méthode proposée est une approche locale hiérarchique appliquée à la prévision des séries temporelles avec trois variantes : l'approche pré-pruning, l'approche post-pruning et le mapping des feuilles de l'arbre.

Avec ces trois techniques de clustering, nous avons la possibilité d'utiliser trois types de prédicteurs différents : les réseaux à propagation avant, les réseaux récurrents et les Support Vector Regression.

Les réseaux de neurones à propagation avant utilisés dans ces expérimentations sont des MLP à trois couches (voir figure 5.1). La couche d'entrée est une couche qui compte autant de neurones que de valeurs dans la fenêtre temporelle. Pour simplifier, nous avons choisi la même fenêtre temporelle que dans la phase de clustering des données. La couche de sortie possède un seul neurone qui correspond à la valeur prédite. Le nombre de neurones dans la couche cachée est déterminé expérimentalement en testant sur les données globales. Une fois le réseau choisi sera appliqué sur toutes les approches locales suivantes. Chaque neurone de la couche de sortie et de la couche cachée est connecté à un neurone de biais. Nous utiliserons la fonction de transfert tangente hyperbolique pour les neurones de la couche cachée et les fonctions de transfert linéaire pour les autres neurones.

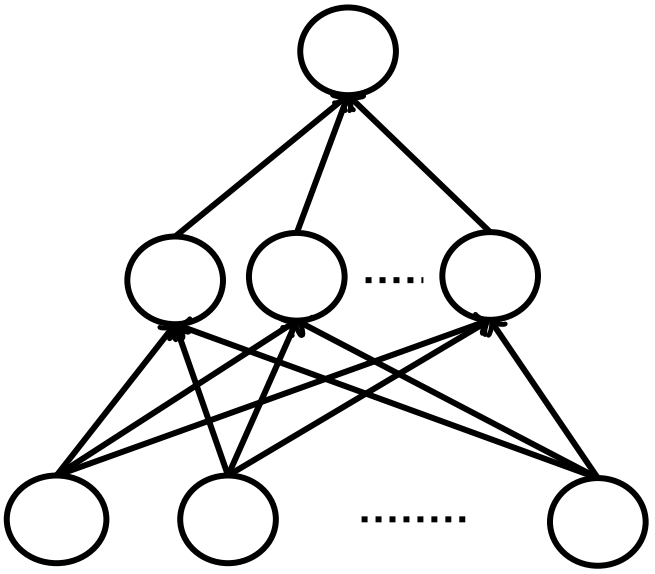

FIGURE 5.1 – Architecture générique d'un MLP pour les expérimentations

L'architecture retenue pour les réseaux de neurones récurrents est la plus répandue, contenant trois couches comme le montre la figure 5.2. La première couche reçoit les entrées, elle se résume à un neurone unique. La couche de sortie possède aussi un unique neurone

qui correspond aux valeurs prédites. La couche cachée est totalement récurrente. Chacun des neurones de cette couche est connecté au neurone d'entrée, au neurone de sortie ainsi qu'à tous les neurones de la même couche y compris lui-même. Tout comme pour le MLP, chaque neurone de la couche de sortie et de la couche cachée est connecté à un neurone de biais. Le nombre de neurones de la couche cachée sera déterminé par des expérimentations sur les données globales pour chacune des séries. Les neurones de la couche cachée possèdent des fonctions de transfert tangente hyperbolique, les autres une fonction linéaire.

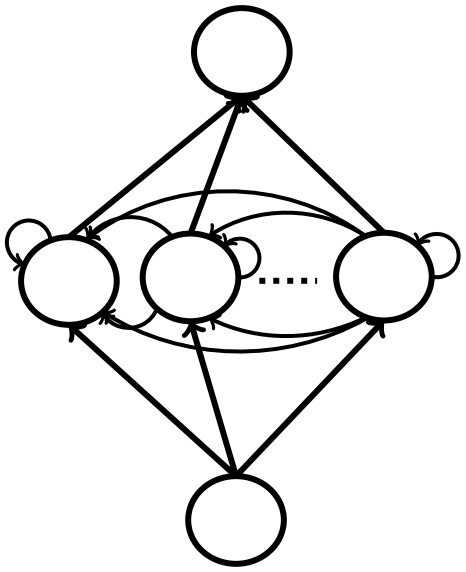

FIGURE 5.2 – Architecture générique d'un RNN pour les expérimentations

Pour les expérimentations avec les SVM, nous nous sommes basés sur la variante $\nu$-SVM (voir dans le chapitre 2.6.3.2). Pour chaque jeu de données, nous effectuons une recherche des paramètres optimaux via un *grid search*. Une fois trouvés, ces paramètres seront utilisés dans les différentes approches locales.

Pour toutes les expériences, nous avons calculé l'erreur quadratique moyenne normalisée, en anglais *Normalized Mean Square Error* (NMSE), afin d'évaluer et de comparer les différentes méthodes. La NMSE est définie par l'équation :

$$NMSE = \frac{\sum_{t=1}^{N} (x(t) - \hat{x}(t))^2}{\sum_{t=1}^{N} (x(t) - \bar{x})^2} = \frac{\sum_{t=1}^{N} (x(t) - \hat{x}(t))^2}{N\sigma^2} \tag{5.1}$$

où $\hat{x}(t)$ est la prévision obtenue à l'instant $t$ par le prédicteur, $\bar{x}$ et $\sigma$ sont respectivement la valeur moyenne et l'écart type estimés de la série. La normalisation par la variance a l'avantage de permettre de comparer des séries d'amplitudes différentes. L'utilisation de la

moyenne comme valeur de prévision correspond à une NMSE de 1. Il est aussi à noter que la NMSE et la MSE d'une série centrée et réduite sont identiques.

Pour la robustesse des résultats, nous exécutons plusieurs instances de la même méthode sur un même jeu de données, nous permettant d'obtenir une NMSE moyenne ainsi que l'écart type sur le nombre d'instances. Ce nombre varie en fonction de la tailles des ensembles d'apprentissage et des différents temps de calcul nécessaire avec.

Nous avons essayé dans chacune de nos expériences d'estimer et d'interpréter l'impact de certains paramètres utilisateur sur les performances des méthodes proposées. Parmi les paramètres que nous avons testés, ceux qui nous ont semblé les plus importants sont la taille de la fenêtre temporelle, le nombre de clusters et l'horizon de prévision.

La taille de la fenêtre temporelle est un paramètre utilisateur qui est dépendant de la nature des données sur lesquelles est appliquée la méthode. Il joue un double rôle. Le premier, lors du clustering, où en changeant la taille de la fenêtre, on obtient un espace vectoriel différent et donc un résultat de clustering différents. Le deuxième, lors de la prévision (dans le cas des MLP et les RNN), où le nombre de valeurs passées peut influencer positivement ou négativement le résultat. Nous essaierons expérimentalement de trouver à chaque fois le paramètre idéal.

Le nombre de clusters est un paramètre utilisateur important. Dans le cas des séries temporelles, il est difficile de déterminer *a priori* (ou même *a posteriori* dans certains cas) le nombre de classes adéquats de la série. Ces données sont généralement des variations dans des phonèmes naturels, elles obéissent parfois à certaines règles saisonnières qui peuvent donner à des classifications dans les données. Mais vu le recouvrement dans les données et la présence des valeurs aberrantes il est difficile de classifier les données discrètement comme on peut le faire dans d'autres applications (exemple reconnaissance de caractères).

L'horizon de prévision est un paramètre qui influence la qualité des prévisions. En effet, il est connu que plus on augmente l'horizon de prévision plus les résultats se dégradent. Il a été montré que les méthodes du type Boosting permettent d' améliorer les performances des prédicteurs [Assaad *et al.*, 2008]. Nous étudierons l'impact de ce paramètre sur nos méthodes.

En ce qui concerne les conditions expérimentales, pour les préditeurs utilisés nous avons décrit les paramètres fixés ainsi que le protocole employé lisé pour les fixer dans la section 5.2.4 et ce pour les différentes données que nous avons utilisées. Pour les autres paramètres qui sont spécifiques aux méthodes locales que nous avons proposées, nous les avons décrits au fur et à mesure dans ce chapitre. Les expérimentations ont été menées de manière répétitive où nous avons effectué une dizaine d'expérimentions (variant selon les types de données utilisées). Ce qui nous a permis d'avoir plusieurs évaluations de performances pour une seule configuration. Nous avons pris pour les comparaisons les meilleures performances obtenues. Avec une étude sur les écarts-types obtenus à la fin de ce chapitre section 5.5.4.

## 5.2 Présentation des données

Nous présentons les séries temporelles retenues pour les expérimentations des méthodes proposées et les comparaisons avec d'autres méthodes, soit en se référant à la littérature soit avec les méthodes que nous avons développées. Ces séries sont reconnues de référence et donc souvent utilisées dans les *benchmarks*. Elles sont disponibles par internet ou par calcul (pour le cas des séries MG) et les conditions telles que la taille des ensembles, et l'échantillonnage sont fournies ce qui permet d'effectuer des comparaisons les moins biaisés possible.

### 5.2.1 Présentation de la série des taches solaires

Les taches solaires sont des taches sombres liées à l'activité du champ magnétique du soleil, qui influencent la propagation des ondes électromagnétiques dans l'ionosphère. Ces taches ont été observées pour la première fois vers 1610, mais les moyennes annuelles de leurs nombres n'ont été enregistrées qu'à partir de 1700. La figure 5.6 représente graphiquement les différentes observations. L'apparition des taches est pseudo-cyclique, atteignant un maximum environ tous les 10 ou 11 ans, comme l'indique le corrélogramme de la figure 5.4. Actuellement il n'existe pas de modèle physique capable d'expliquer ces fluctuations.

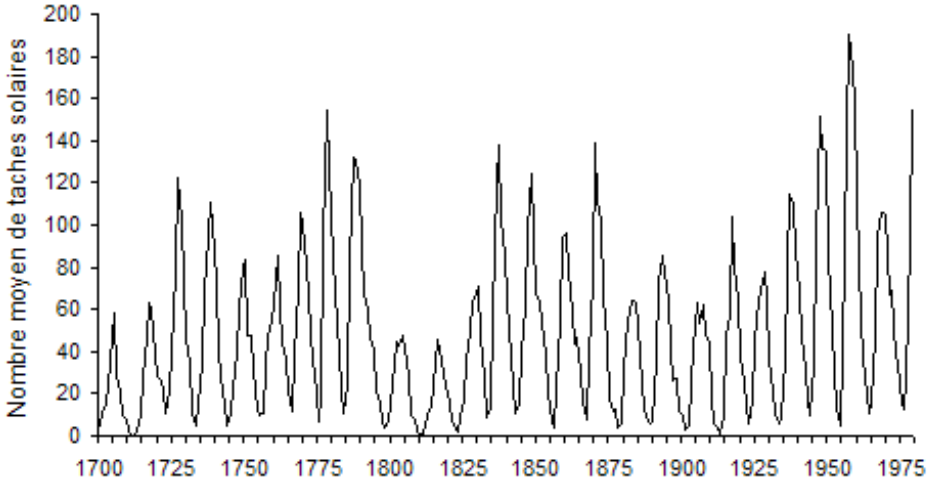


FIGURE 5.3 – Moyennes annuelles du nombre des taches solaires (source [Boné, 2000])

La série de références qui en résulte couvre la période allant de 1700 à 1979. C'est une série non stationnaire et non linéaire. Cependant [Akaike, 1978] la suppose localement stationnaire. La première étude de cette série revient à Yule [Yule, 1927] qui a utilisé un modèle autorégressif. Ensuite, différents modèles ont été testés [Tong et Lim, 1980] et [Rao et Gabr, 1984] incluant les réseaux de neurones à couches [Weigend *et al.*, 1990], [Svarer *et al.*, 1993] ou récurrents [McDonnell et Waagen, 1994] [Aussem *et al.*, 1995] [Boné, 2000]

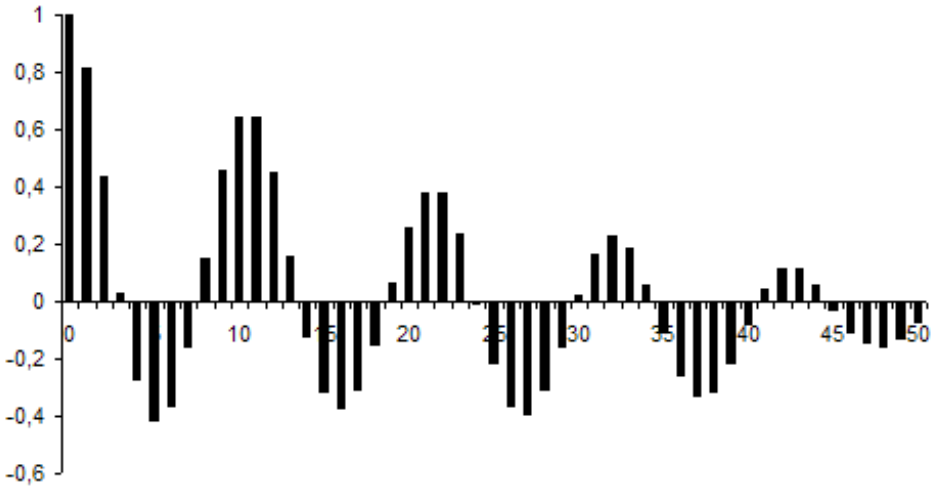


FIGURE 5.4 – Corrélogramme des taches solaires (source [Boné, 2000])

et les SVR [Cao, 2003], [Rubio *et al.*, 2011].

La décomposition habituelle des données consiste à former l'ensemble d'apprentissage avec les valeurs de la période $1700 - 1920$ et à utiliser pour l'évaluation des performances deux ensembles possédant des variances différentes. Le premier est constitué des valeurs sur la période $1921 - 1955$ (ensemble test1). Le second est constitué avec les valeurs issues de la période $1956 - 1979$ (ensemble test2)). L'ensemble test2 est considéré dans la littérature comme plus difficile car à variance plus élevée que test1.

| Modèle | NMSE Test1 | NMSE test2 |
|---|---|---|
| Copie carbone | $0,427$ | $0,966$ |
| MLP [Weigend *et al.*, 1992] | $0,086$ | $0,350$ |
| MLP [Kouam et Fogelman, 1993] | $0,082$ | $0,320$ |
| MLP [Czernichow, 1993] | $0,078$ | $0,283$ |
| MLP | $0,0058$ | $0,018$ |
| RNN | $0,0092$ | – |
| SVM | $0,0047$ | $0,0298$ |
| SVR [Cao, 2003] | $0,1544$ | – |

TABLE 5.1 – Tableau récapitulatif des résultats sur la série des taches solaires

Le tableau 5.1 donne les performances de plusieurs modèles appliqués à cette série. Pour chaque modèle est indiquée l'erreur quadratique moyenne normalisée (NMSE) sur l'ensemble test. La copie carbone consiste à prendre comme valeur future estimée la valeur actuelle. Les MLP prennent en entrée une fenêtre temporelle des 12 dernières valeurs de la série. Weigend [Weigend *et al.*, 1990] utilise une technique d'élimination de poids par

pénalisation, réduisant le réseau de 8 à 3 neurones cachés. Dans [Kouam et Fogelman, 1993] un réseau à 3 neurones cachés est directement proposé. Czernichow [Czernichow, 1993] utilise une technique de sélection de variables d'entrée pour obtenir un MLP à 5 entrées et 5 neurones cachés. Dans [Cao, 2003] un modèle local mixte entre prédiction locale déterminé par SOM et SVM experts a été utilisé. Nous présentons aussi un MLP que nous avons utilisé constitué de 9 neurones d'entrées et de 3 neurones cachés, un RNN à 5 neurones cachés et un SVM dont les paramètres ont été fixés via *grid search*.

### 5.2.2 La série Laser

Une compétition de l'institut de Santa Fe sur l'analyse et la prévision de séries temporelles (*The Santa Fe Institute Time Series Prediction and Analysis Competition*) a été lancée en 1991 pour évaluer les nouvelles techniques de prévision de séries temporelles. Les séries présentées lors de cette compétition provenaient de domaines variés : physique, biologie, finance, astrophysique et musique. Les compétiteurs ont eu accès aux données au premier août 1991, et étaient libres de choisir une ou plusieurs séries. Pour chacune de ces séries, la provenance ainsi que l'évolution future étaient tenues secrètes. Les compétiteurs n'avaient donc aucun moyen de vérifier la qualité de leurs solutions.

Les 14, 15 et 17 mai 1992 eut lieu le congrès, le *NATO Advanced Research Workshop*, pour présenter les différentes contributions à la compétition. Pour la série "A", qui contient le plus de soumissions, le vainqueur fut Eric Wan [Weigend, 1993] [Wan, 1993], avec un réseau de neurones à propagation avant et à connexions FIR. C'est sur cette série que nous avons testé nos algorithmes.

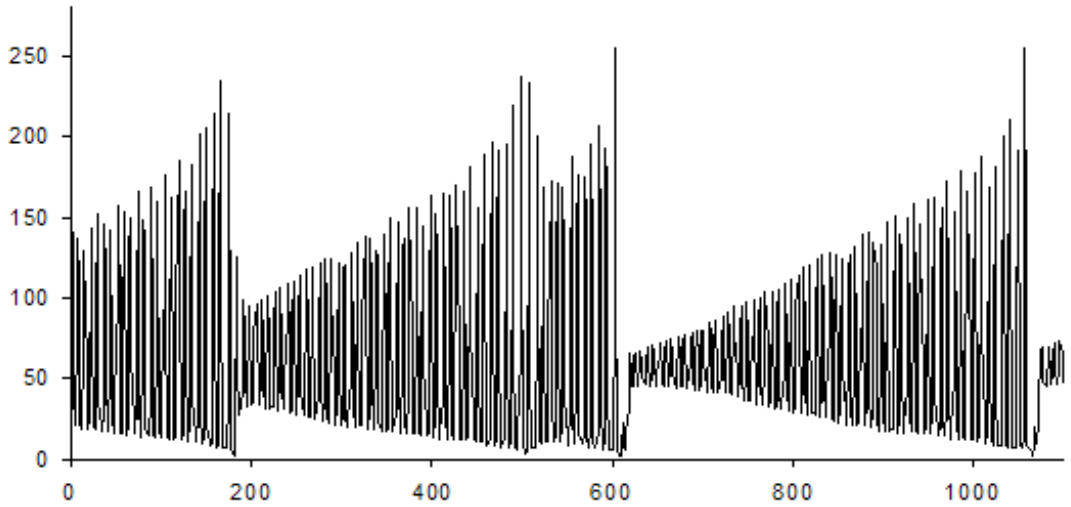


FIGURE 5.5 – Graphique de la série temporelle Laser (source [Boné, 2000])

La série "A" (figure 5.5) mesure l'intensité des pulsations chaotiques d'un laser $NH_3$-

FIR (la désignation FIR n'a pas de relation avec les connexions neuronales du même nom). Le comportement observé du laser peut être approché par un modèle relativement simple, les équations de Lorentz [Weiss *et al.*, 1995] :

$$\begin{aligned} \frac{dx}{dt} &= -\sigma x + \sigma y \\ \frac{dy}{dt} &= -xz + rx - y \\ \frac{dz}{dt} &= xy - bz \end{aligned} \tag{5.2}$$

Les chercheurs ne disposaient que des 1000 premières valeurs de la série, et les performances des méthodes furent ensuite évaluées sur les 100 valeurs suivantes. Nous nous sommes concentré sur la prévision à un pas de temps et avons appliqué les mêmes prétraitements que E. Wan : les données ont été centrées réduites. De même, nous avons repris sa constitution des ensembles d'apprentissage et de validation. L'ensemble d'apprentissage contient les 900 premiers points et l'ensemble de validation contient les points situés entre les instants 550 et 1000. La principale difficulté de la série réside dans la diminution soudaine d'intensité (aux instants 200, 510 et 600). L'ensemble stop devait contenir un de ces changements de régime tout en ne restreignant pas l'ensemble d'apprentissage.

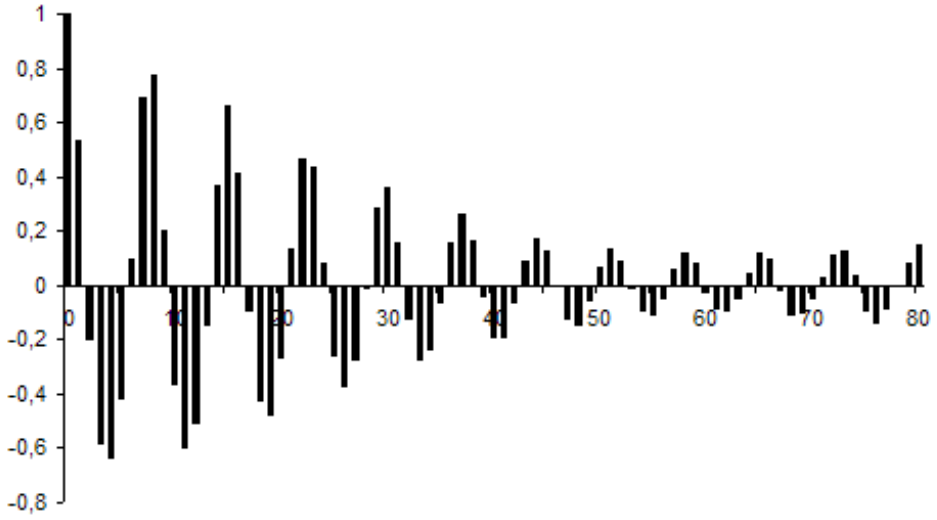


FIGURE 5.6 – Corrélogramme de la séries temporelle Laser (source [Boné, 2000])

Le modèle vainqueur de la compétition pour la série "A" est un réseau à propagation avant avec deux couches cachées contenant chacune 12 neurones. Le neurone d'entrée est relié à la première couche par des connexions FIR d'ordre 5.

Le tableau 5.2 donne les performances des différents modèles appliqués à la série Laser.

| Modèle | NMSE Test |
|---|---|
| Copie carbone | $2,17$ |
| MLP FIR [Wan, 1993] | $0,023$ |
| RNN [Boné, 2000] | $0,00792$ |
| Boosting [Assaad *et al.*, 2008] | $0,00494$ |
| RSOM [Koskela *et al.*, 1997] | $0,084$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $0,011$ |

TABLE 5.2 – Tableau récapitulatif des résultats sur la série Laser

Pour chaque modèle est indiqué l'erreur quadratique moyenne normalisée (NMSE) sur l'ensemble test. Deux modèles globaux sont présentés : dans [Boné, 2000] un RNN avec un algorithme d'ajout de connexion nommé EBPTT et un MLP à connexion FIR présenté dans [Wan, 1993]. Dans [Assaad *et al.*, 2008] un modèle de boosting utilisant des RNN avec l'algorithme BPTT est présenté. Et nous trouvons enfin des modèles locaux à travers [Koskela *et al.*, 1997] et [Noelia Sánchez-Maroño *et al.*, 2003].

### 5.2.3 La série Mackey-Glass

Les séries de Mackey-Glass [Mackey et Glass, 1977] sont des séries de références pour l'évaluation de nombreux systèmes de prévision [Farmer et Sidorowich, 1988], [Casdagli, 1989], dont ceux à base de réseaux de neurones [Lapedes et Farber, 1987], [Hakim *et al.*, 1991], [Wan, 1993], [Svarer *et al.*, 1993], [Back et Tsoi, 1994], [Aussem *et al.*, 1995], [Duro et Reyes, 1999] et des SVR [Müller *et al.*, 1997], [Thissen *et al.*, 2003], [Herrera *et al.*, 2007], [Rubio *et al.*, 2011]. Les séries sont générées par un modèle non linéaire, qui a été établi à l'origine pour décrire la production des globules blancs chez les patients atteint de leucémie. Il est défini par l'équation :

$$\frac{dx(t)}{dt} = -0,1x(t) + \frac{0,2x(t-\tau)}{1+x^{10}(t-\tau)} \tag{5.3}$$

En fonction des valeurs de $\tau$, la dynamique du modèle passe par le point fixe, un comportement périodique jusqu'au chaos déterministe pour des valeurs de $\tau > 16,8$. Les systèmes chaotiques déterministes sont caractérisés par une évolution de leurs vecteurs d'état selon une trajectoire chaotique, assez difficile à décrire, n'étant ni un point d'équilibre, ni périodique, ni même quasi périodique. L'état du système est ici l'ensemble des informations permettant de déterminer sans ambiguïté son évolution future. La définition du chaos déterministe étant assez controversée, nous nous bornerons à considérer comme chaotiques les systèmes dynamiques ayant une dépendance exponentielle aux conditions initiales. Pour de tels systèmes déterministes, il existe généralement un sous-espace dans lequel l'évolution du système est confinée après une phase de transition. L'objet géométrique dans l'espace d'état vers lequel les trajectoires sont attirées porte le nom d'attracteur étrange, lorsque l'objet possède une structure géométrique à dimension fractionnaire (non entière).

La dimension fractale permet, dans une certaine mesure, de caractériser les attracteurs étranges et donc les systèmes chaotiques. Plusieurs approches ont été proposées pour la

calculer (pour une présentation voir [Parker et Chua, 1989] par exemple). Devant l'impossibilité pratique d'accéder au vecteur de l'état interne, ces approches se basent sur une conséquence du théorème de Takens [Rand et Young, 1981] : Pour les systèmes dynamiques déterministes sans bruit, la trajectoire du vecteur contenant la fenêtre temporelle possède un comportement équivalent à celui de l'état du système. La largeur de la fenêtre, qui reste à déterminer, est la dimension interne de la série.

Les deux valeurs habituellement retenues pour le paramètre $\tau$ sont 17 et 30 (figure 5.7). Pour $\tau = 17$, le graphique en 3 dimensions des points $(x(t), x(t+1), x(t+2))$ dans l'espace des phases permet de visualiser l'attracteur de la série (figure 5.8).

La génération des données se fait suivant une technique de Runge-Kutta du $4^{ième}$ ordre [Press *et al.*, 1992], avec pour conditions initiales $x(t) = 0,9$ pour $0 \leq t \leq \tau$. Les données ainsi obtenues sont enfin échantillonnées à la fréquence 6. Pour $\tau = 17$ et $\tau = 30$, les deux dimensions internes, évaluées par [Casdagli, 1989], sont respectivement de 4 et 6.

La plupart des résultats de la littérature correspondent à une tâche de prévision à un pas de temps des séries générées, ce qui, en considérant l'échantillonnage, correspond en réalité à la valeur 6 pas de temps plus tard. Les premiers 500 points sont utilisés pour l'apprentissage et les 100 pour le test.

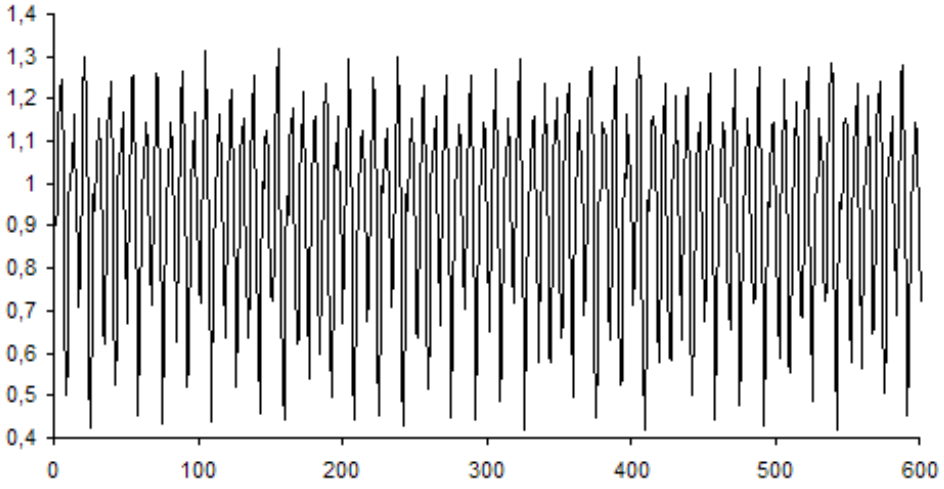


FIGURE 5.7 – Graphique de la série temporelle MG-17 (source [Boné, 2000])

Le tableau 5.3 donne les performances des différents modèles appliqués à la série MG-17. Pour chacun de ces modèles présentés l'erreur est évaluée par la NMSE sur l'ensemble de tests. Dans le tableau figurent des modèles globaux tels que le modèle linaire, MLP, RBFN et RNN. Par exemple, dans [Zemouri *et al.*, 2003] il est proposé un réseau récurrent de fonctions radiale (RBFN). Assad [Assaad *et al.*, 2008] utilise un réseau de neurones récurrents avec une méthode de boosting. Nous retrouvons aussi des approches locales comme dans [Koskela *et al.*, 1997] qui met RSOM avec un modèle linéaire. Dans [Barreto *et al.*, 2004], une autre approche locale combine les cartes de Kohonen et des réseaux de

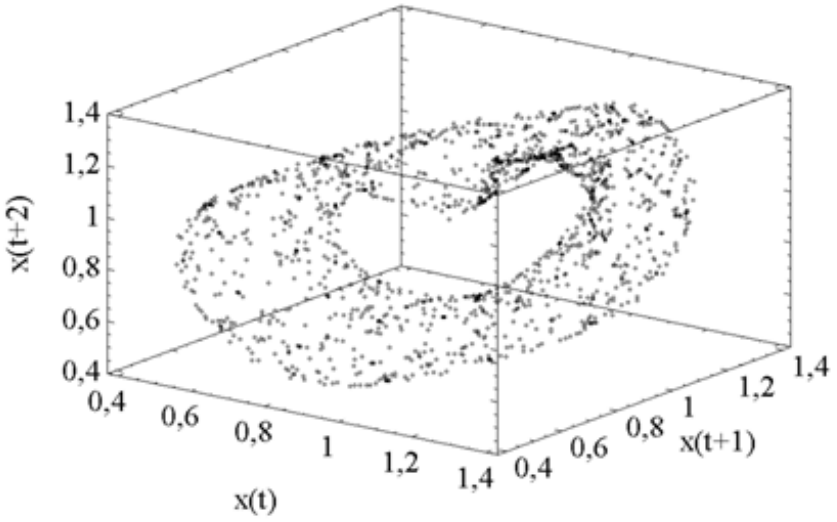


FIGURE 5.8 – Attracteur de la série MG-17 (source [Boné, 2000])

| Modèle | NMSE Test $\times 10^3$ |
|---|---|
| Linéraire | 269 |
| RBFN | $10,7$ |
| MLP | 10 |
| MLP FIR | $4,9$ |
| RNN (BPTT) | $0,216$ |
| [Zemouri *et al.*, 2003] | 38 |
| [Assaad *et al.*, 2008] | $0,11$ |
| [Koskela *et al.*, 1997] | $3,11$ |
| [Barreto et Araujo, 2004] | $0,75$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $0,000574$ |
| [Vesanto, 1997] | $0,17$ |

TABLE 5.3 – Tableau récapitulatif des résultats sur la série Mackey-Glass (source [Boné, 2000])

neurones. On retrouve dans [Noelia Sánchez-Maroño *et al.*, 2003] une approche locale avec SOM et réseaux fonctionnels. Enfin, dans [Vesanto, 1997] une approche locale avec SOM et MLP est présentée.

### 5.2.4 Le choix des prédicteurs

Souvent, la phase de paramétrage est la phase la plus coûteuse en temps. Dans notre cas, il ne s'agit pas uniquement des paramètres des prédicteurs mais du modèle incluant la phase de clustering. Afin de simplifier les expérimentations et surtout d'évaluer les méthodes proposées, nous avons essayé de paramétrer les prédicteurs de manière globale sur chacune des séries temporelles puis d'utiliser les mêmes configurations pour les différents prédicteurs locaux.

Nous avons appliqué cette procédure pour les MLP, RNN et les SVR. Dans les deux premiers cas, nous nous sommes intéressés au nombre de neurones cachés qui permettent d'obtenir le compromis entre erreur moyenne et écart type. Nous nous sommes intéressés également à la taille de la fenêtre d'entrée pour le cas des SVM et des MLP. Le tableau 5.4 détaille ces valeurs pour les différentes séries temporelles ainsi que les paramètres spécifiques aux SVM utilisés par libSVM.

| Modèle | SunSpots | Laser | MG-17 |
|---|---|---|---|
| taille de la fenêtre (MLP) | 9 | 8 | 8 |
| nombre de neurones de cachée (MLP) | 3 | 7 | 10 |
| nombre de neurones cachée (RNN) | 5 | 7 | 7 |
| taille de la fenêtre (SVM) | 12 | 8 | 8 |
| paramètre c SVM (SVM) | 4 | 64.0 | 64 |
| paramètre g SVM (SVM) | 0.5 | 0.0625 | 0.5 |

TABLE 5.4 – Tableau récapitulatif des paramètres optimaux en approche globale

## 5.3 Intégration des RNN dans l'approche locale SOM

La première partie de notre évaluation concernera l'intégration des RNN dans l'approche locale. Nous considèrerons dans cette partie, pour la phase clustering de l'approche locale, l'algorithme de cartes de Kohonen et ses variantes. L'intérêt étant de valider ou non l'apport des réseaux de neurones récurrents en tant que prédicteurs locaux. Nous étudierons cela en trois étapes. On s'intéressera d'abord à l'impact du nombre de clusters. Puis à la taille de la fenêtre temporelle utilisée. Enfin, nous étudierons l'effet de l'horizon de prévision sur le modèle.

### 5.3.1 Étude sur le nombre de clusters

Dans une première phase nous étudions l'impact du nombre de clusters sur la performance globale (NMSE finale du système). Pour le phase clustering, on se basera sur les algorithmes SOM, TKM et RSOM, pour la phase de prédiction nous utiliserons les RNN.

La figure 5.9 montre l'évolution de l'erreur globale (NMSE) sur l'ensemble de test en fonction de la taille de carte utilisée. Pour une carte unidimensionnelle, elle représente sa longueur, pour une carte bidimensionnelle elle représente la largeur des cotés. La figure est décomposée en trois schémas : le schéma (a) relatif aux cartes linéaires, le schéma (b) relatif aux cartes circulaires et le schéma (c) est relatif aux cartes bidimensionnelles. Chacun de ces schémas comporte trois courbes associées aux algorithmes de clustering utilisés (SOM, TKM et RSOM).

Nous remarquons dans ces différentes configurations que dans le cas des taches solaires, les meilleures performances se situent dans les plus petits nombres de clusters (souvent deux). Ceci peut paraître très logique lorsqu'on considère la taille réduite des données d'apprentissage. Souvent, deux clusters seulement permettent de décomposer la base d'apprentissage en deux sous-ensembles assez homogènes tout en gardant un volume de données suffisant pour construire le modèle local, décomposer encore plus les données risque de créer de plus petits clusters insuffisant à éviter le sous-apprentissage. Nous pouvons également remarquer que la carte bidimensionnelle est la moins performante puisque les courbes associées se trouvent généralement au dessus des autres. Une première explication peut être reliée à la précédente. En effet, dans le cas d'une carte bidimensionnelle, le nombre de clusters à considérer est autant plus important. On peut estimer, d'après ce résultat, qu'une carte bidimensionnelle n'est pas adaptée à des données de type séries temporelles. Même si les données utilisées pour le clustering sont des données multidimensionnelles, l'origine de ces données est unidimensionnelle.

Le cas des séries Laser et MG-17 est similaires. Nous pouvons y retenir les mêmes conclusions concernant la carte rectangulaire qui semble la moins performantes dans cette approche. Nous remarquons qu'il existe une valeur de la taille de la carte (en moyenne) pour laquelle les performances semblent être les meilleures et se dégradent après. Ce point se situe : aux alentours de 10 pour la série Laser et 15 pour la série MG-17. S'il impossible de déterminer une valeur fixe, c'est parce que l'algorithme SOM se base sur une initialisation aléatoire qui rend les résultats finaux variables d'une instance à une autre.

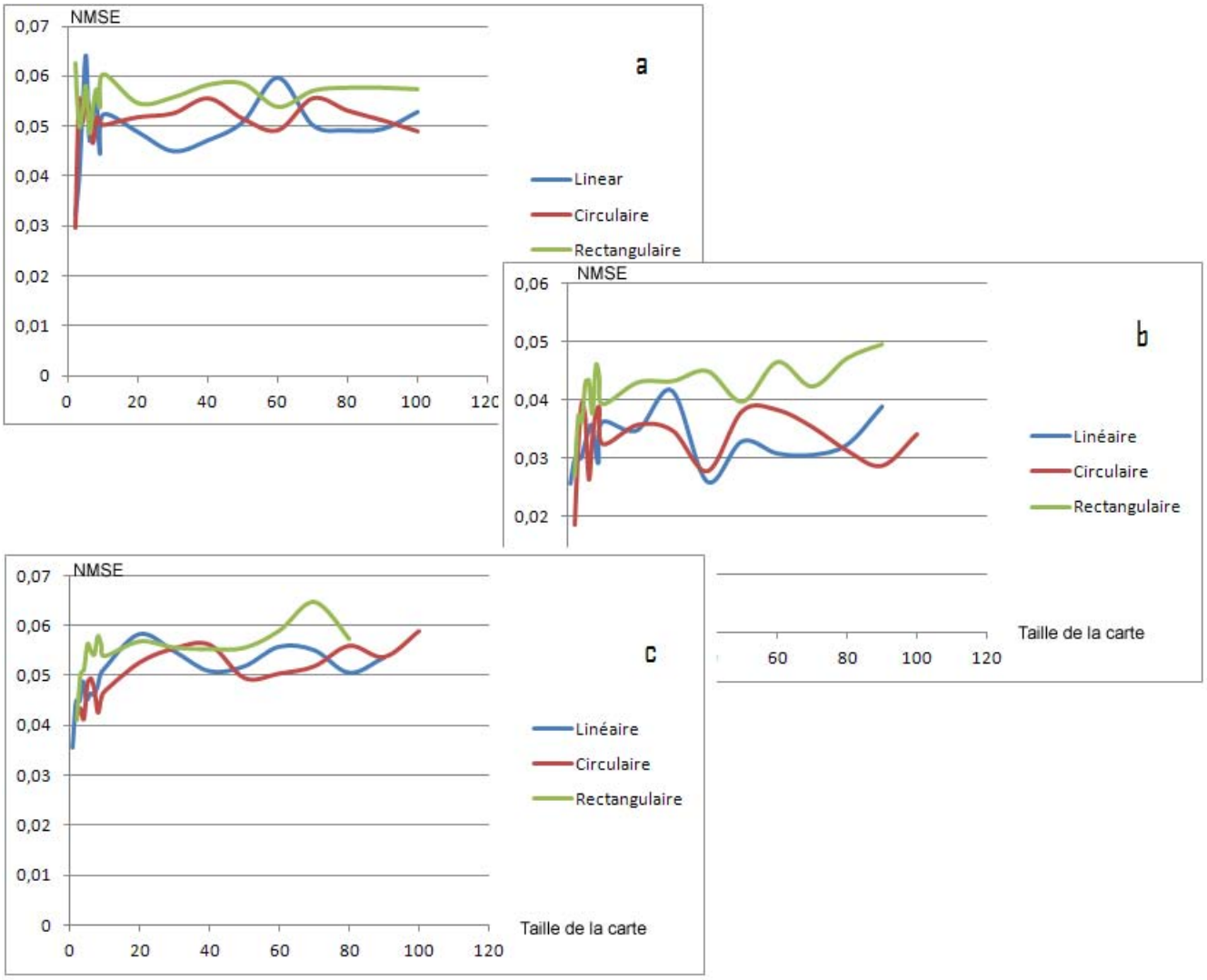


FIGURE 5.9 – Évolution de l'NMSE en fonction de la taille des clusters pour la séries *Sunspots*. (a) pour l'algorithme classique SOM, (b) pour l'algorithme TKM et (c) pour l'algorithme RSOM

Il y a aussi une différence entre la taille de la carte et le nombre de clusters obtenus sur une base d'apprentissage. Une fois l'apprentissage sur la carte terminés, les noeuds vide sont "bridé". Ce qui veut dire que les exemples de validations et de tests ne peuvent être affectés dans ces clusters. La figure 5.10 présente, pour les trois algorithmes de clustering utilisés, la relation entre le nombre de cluster et la taille d'une carte linéaire sur les trois séries utilisées. Le schéma (a) correspond à la série des taches solaire, le schéma (b) correspond à la série Laser et le schéma (c) correspond à la série MG-17.

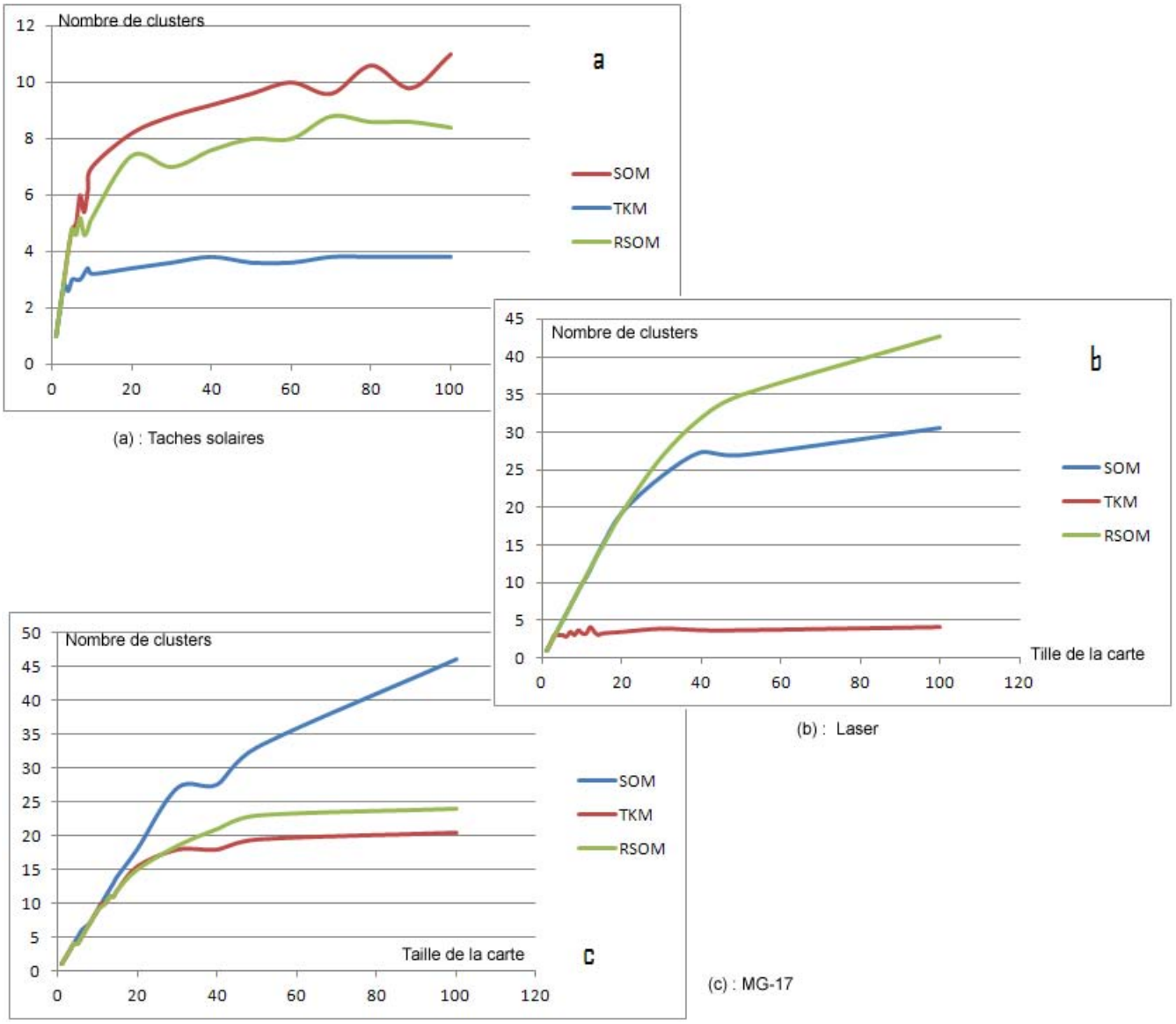


FIGURE 5.10 – Nombre de clusters obtenus en fonction de la taille de la carte

D'après la figure 5.10, les courbes qui représentent le nombre de clusters obtenus possèdent deux régimes. Le premier correspond aux cartes de petite taille dans lequel les

courbes sont linéaires confondues avec la première bissectrice. Dans cette zone le nombre des prototypes est assez petit pour qu'ils soient tous pris comme des clusters. Selon les données utilisées, cette période peut plus ou moins durer, ensuite le nombre de clusters retenus se stabilise, la carte devient de plus en plus large et, que tous les prototypes n'auront pas des exemples affectés. Cela se traduit par une autre partie de la courbe linéaire, ou quasi-linéaire, dont la pente est visiblement plus faible. Nous remarquons également que l'algorithme TKM a tendance à se restreindre à un plus petit nombre de clusters que les autres algorithmes, notamment dans le cas des séries des taches solaires et Laser alors que les algorithmes SOM et RSOM sont assez similaires.

Nous pouvons ensuite étudier les performances des prévisions en fonctions des clusters obtenus. Par exemple, on pourra trouver un lien entre les caractéristiques et la qualité de prévision d'un même cluster. On pourra ainsi lui associer un indice de confiance. Les figures 5.11, 5.12 et 5.13 nous montrent cette relation pour, respectivement, les taches solaires, Laser et MG-17. Chacune de ces figures contient deux schémas : le premier pour une carte de petite taille et le second pour une carte de grande taille.

Dans la figure 5.11, deux expérimentations sur la série des taches solaires sont mises en avant, la première avec une carte composée de quatre unités, la deuxième avec une carte composée de 20 unités. La première carte permet d'obtenir une erreur NMSE de $0,040$ devant $0,030$ pour la deuxième carte. La raison peut se trouver dans la distribution des exemples dans la carte 2. En effet, on remarque qu'un cluster en particulier (18 dans la carte) est le plus volumineux en nombre d'exemples d'une part, et d'autre part possède la NMSE la plus petite, ce qui a joué un rôle important afin d'améliorer les performances globales. On peut donc dire que le clustering a joué un rôle primordial dans cet exemple. Cependant, puisque le processus est un processus aléatoire, car la distribution finale dépend de l'initialisation de la carte, on trouve que cette configuration ne donne pas la meilleure performance en moyenne.

Nous retrouvons les mêmes conclusions pour les séries Laser et MG-17, à part la différence dans la taille de la base d'apprentissage. Ils peuvent êtres résumés dans les figures 5.12 et 5.13.

Quelque soit le prédicteur utilisé, la phase de clustering est une phase importante car son rôle consiste à préparer au mieux les données. Il se trouve que dans notre cas, nous avons plusieurs possibilités, à la fois dans l'algorithme de clustering et dans la topologie de la carte. On se propose d'étudier l'impact de ces différents choix. La figure 5.14 décrit l'évolution de l'erreur de quantification en fonction du temps pour les trois algorithme SOM, TKM et RSOM, dans respectivement, les schémas (a), (b) et (c).

D'après la figure 5.14, on peut remarquer que la vitesse de convergence est plus rapide pour l'algorithme classique, une vingtaines d'itérations par rapport à une quarantaine pour TKM et RSOM. Les expériences précédentes ont aussi montré les faibles résultats obtenus à partir du clustering avec la carte rectangulaire. Toutefois, l'évolution de l'erreur de quantification ne permet pas de conclure, à part le fait que la carte linéaire effectue parfois plus d'oscillations avant de converger, ce qui augmente les chances d'aboutir à un meilleur minimum local. C'est peut être une deuxième raison expliquant les meilleurs résultats en terme de NMSE obtenus avec les cartes linéaires.

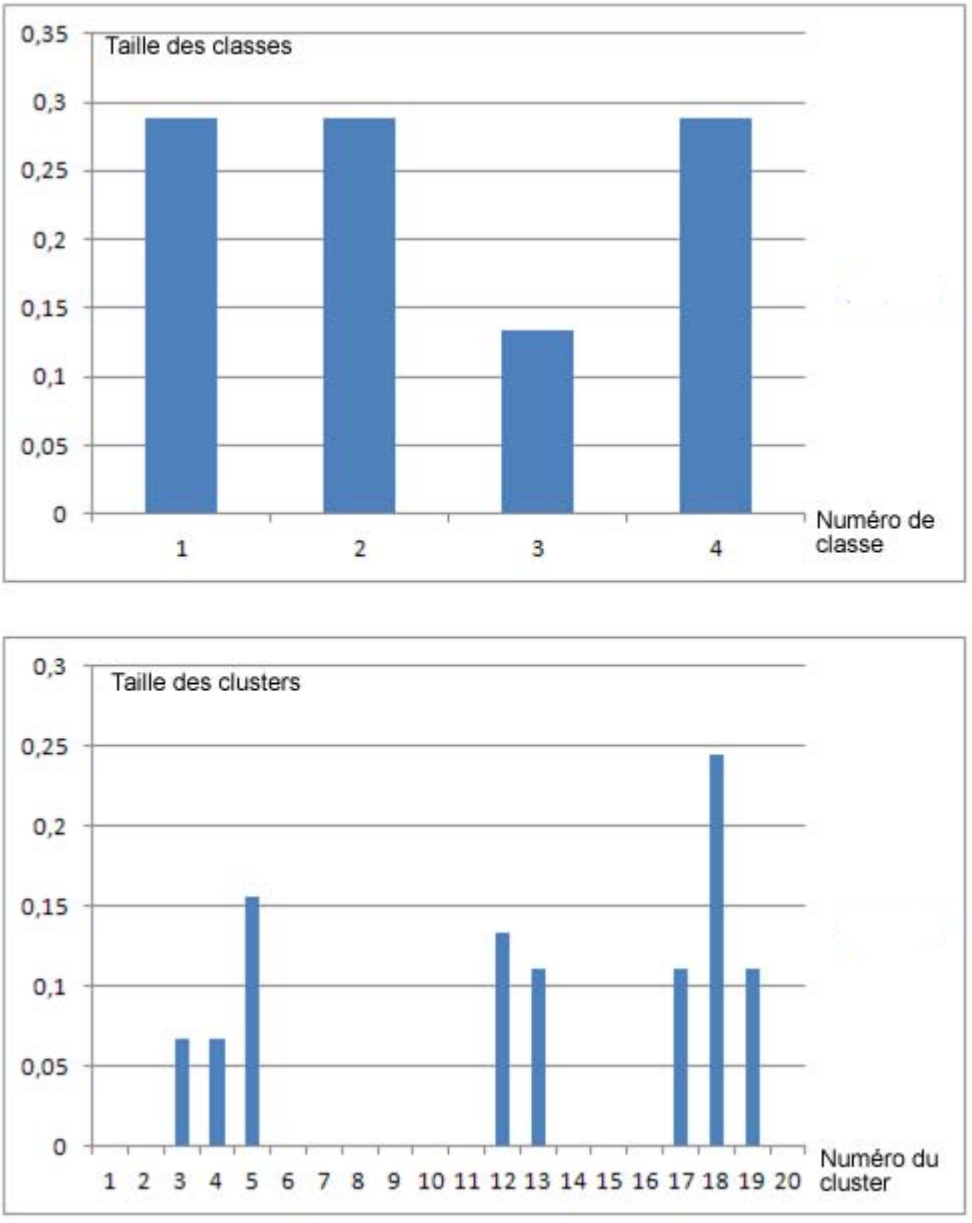


FIGURE 5.11 – Disposition d'une carte linéaire après clustering avec les valeurs de prévisions locales pour la série des taches solaires. (au dessus cas d'un carte de taille 4, au dessous cas d'une carte de taille 20)

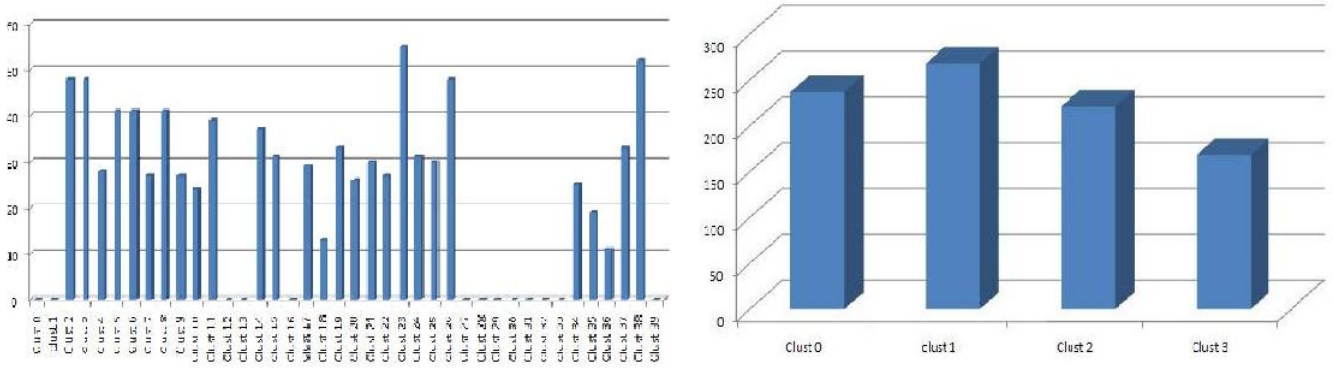


FIGURE 5.12 – Disposition d'une carte linéaire après clustering avec les valeurs de prévisions locales pour la série Laser. (à gauche cas d'une carte de taille 20, à droite cas d'un carte de taille 4)

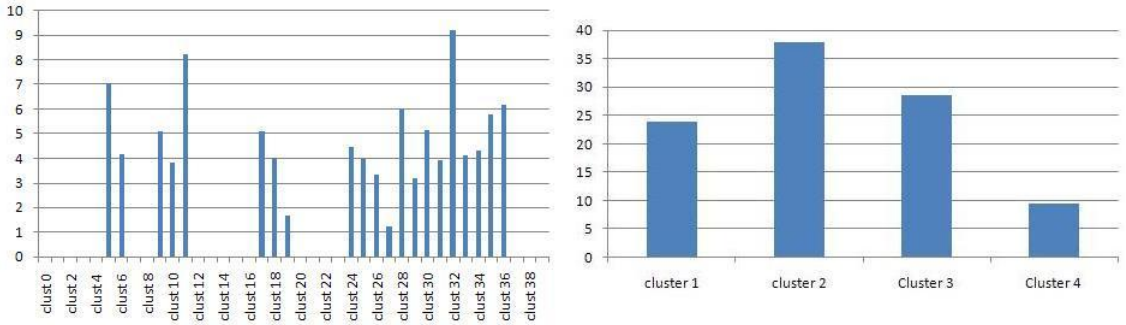


FIGURE 5.13 – Disposition d'une carte linéaire après clustering avec les valeurs de prévisions locales pour la série MG-17. (à gauche cas d'une carte de taille 20, à droite cas d'un carte de taille 4)

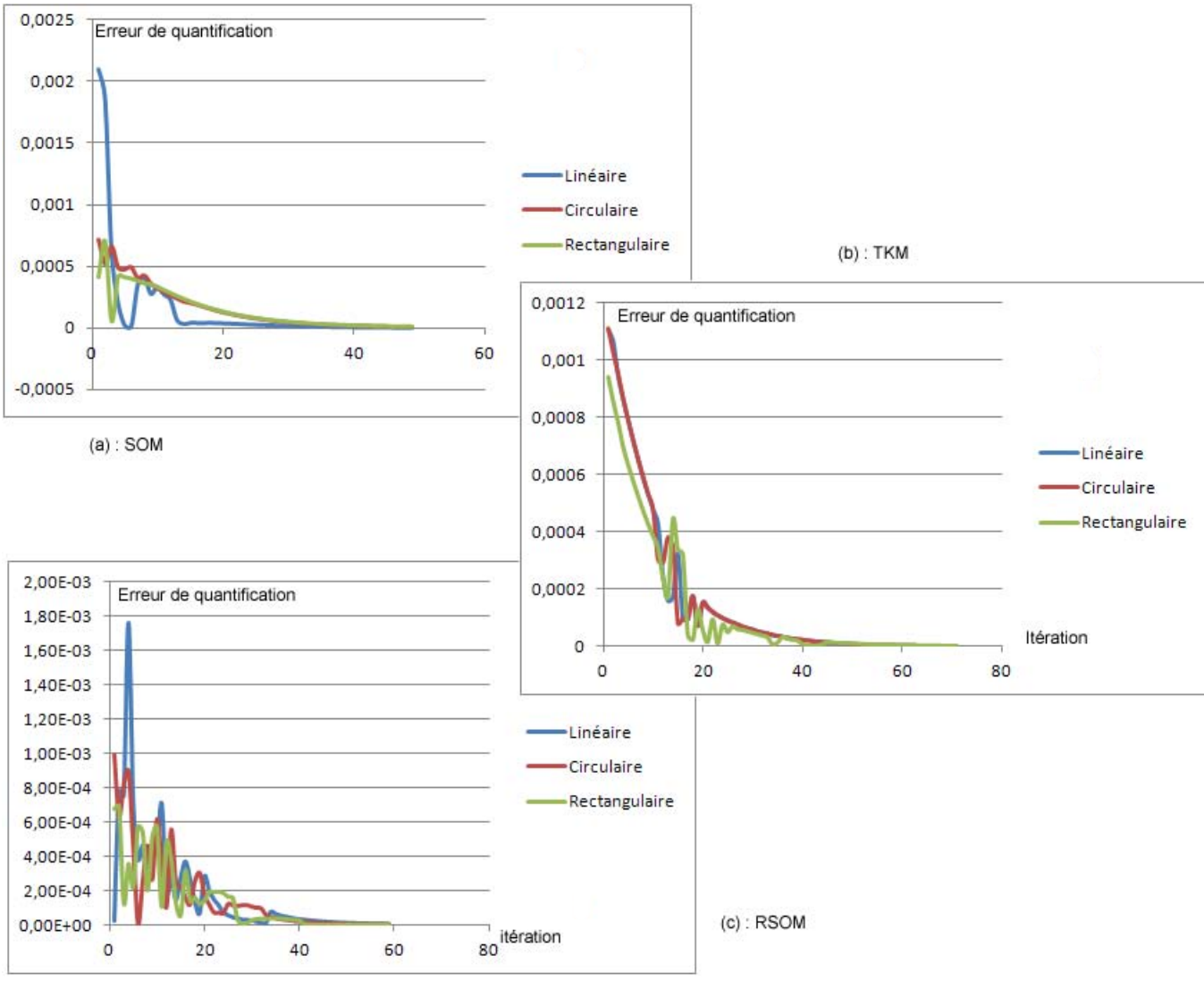


FIGURE 5.14 – Évolution de l'erreur de quantification pour les taches solaires

Le tableau 5.3.1 récapitule les meilleures performances obtenues sous les différentes configurations testées et pour les séries d'évaluation.

| NMSE | SunSpots | Laser | Mackey-Glass $\times 10^3$ |
|---|---|---|---|
| SOM + RNN | $0,017$ | $0,031$ | $0,037$ |
| TKM + RNN | $0,011$ | $0,021$ | $0,035$ |
| RSOM + RNN | $0,027$ | $0,036$ | $0,040$ |
| RNN | $0,065$ | $0,008$ | $0,235$ |

TABLE 5.5 – Récapitulatif des meilleures performances obtenues

Selon le tableau 5.3.1, le meilleur résultat obtenu pour la série des taches solaires correspond à l'algorithme TKM à 2 clusters dans la carte linéaire avec une NMSE de $0,011$ et pas loin derrière on trouve l'algorithme classique SOM à 2 clusters avec une NMSE de $0,017$, enfin viens RSOM avec toujours seulement deux cluster et une NMSE de $0,027$. On remarque que quelque soit l'algorithme de clustering utilisé les meilleurs performances se situent toujours avec la carte de plus petite taille. Une amélioration par rapport à l'approche globale est notée.

Dans le cas de la série MG-17, les meilleurs résultats obtenus sont relatifs à l'algorithme TKM, très proches de ceux de l'algorithme classique avec une NMSE de $0,035 \times 10^{-3}$. Quand à la taille de la carte, elle varie de 15 à 25 en fonction des instances d'exécutions. Nous pouvons aussi remarquer que dans tous les cas, le modèle local apporte une amélioration par rapport au prédicteur global.

Selon le tableau 5.3.1, Les meilleurs résultats obtenu par la séries Laser sont similaires au cas précédent. La NMSE varie de $0,021$ à $0,031$ avec un avantage pour l'algorithme TKM. Les tailles des cartes utilisées varient de 6 à 13 unités. Toutefois, les modèles locaux ne surpassent pas le prédicateur global dans ce cas. Ceci peut être associé à la nature de la série Laser qui présente un changement de régime assez brusque. Cet élément rend la série difficile à prédire pour les modèles globaux. Ceci est encore plus vrai pour l'approche locale, le phénomène ne se produisant qu'une seule fois.

### 5.3.2 Impact de la taille de la fenêtre temporelle

Comme pour le nombre de clusters, nous déterminerons l'impact de la taille de la fenêtre utilisée dans le clustering et la prévision. Il s'agit d'étudier l'évolution de l'erreur de prévision en fonction du nombre $M$ d'éléments passés séquentiels dans le vecteur d'entrée utilisé par l'algorithme de clustering SOM, TKM et RSOM.

La figure 5.15 montre l'évolution de l'erreur globale moyenne en fonction de la taille de la fenêtre temporelle qui est comprise entre 5 et 15. Pour que le réseaux de neurones n'utilisent qu'une seule valeur en entrée, ce test évalue réellement l'effet de l'algorithme de clustering. Le schéma présente ces valeurs pour la série des taches solaires. Les trois courbes synthétisent les expérimentations menées avec trois types d'algorithmes de clustering (SOM, TKM et RSOM).

D'après la figure 5.15, on peut remarquer que l'algorithme de clustering TKM est remarquablement meilleur que les deux autres et ce quelque soit la fenêtre temporelle

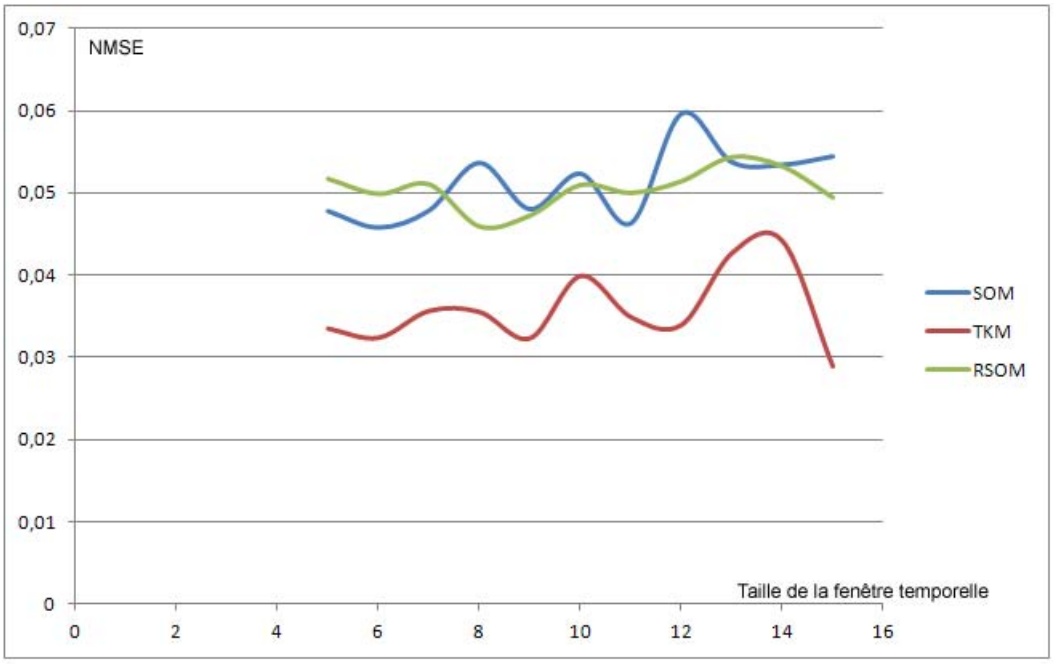


FIGURE 5.15 – Évolution de l'erreur NMSE en fonction de la taille de la fenêtre temporelle pour les taches solaires

utilisée. Nous avons vu dans les expérimentations précédentes que TKM a tendance à ne pas disperser les exemples d'apprentissage dans la carte. Cette particularité suffit à expliquer les meilleures performances obtenues par le prédicteur RNN par rapport aux autres algorithmes de clustering. Nous pouvons également remarquer pour les trois cas, en dépit de l'apparence oscillatoire des courbes, qu'il existe une tendance haussière, une chose tout à fait logique lorsqu'on prend en compte l'effet de la sur-dimensionnalité sur le prédicteur. Les mêmes conclusions sont valables pour les séries Laser et MG-17.

Le tableau 5.3.2 récapitule les meilleures performances obtenues en variant la taille de la fenêtre temporelle. Pour la série des taches solaires, la meilleure performance est celle de l'algorithme TKM en concordance avec le schéma précédent. Vient ensuite l'algorithme classique SOM puis RSOM avec respectivement $0,016$ pour une fenêtre temporelle de 5, $0,029$ pour une fenêtre temporelle de 6 et $0,036$ pour une fenêtre temporelle de 8.

Dans le cas de la série MG-17, nous obtenons avec l'algorithme TKM une NMSE de $0.036 \times 10^{-3}$ et une fenêtre temporelle de taille 14, et $0.040 \times 10^{-3}$ avec une fenêtre de taille 9 pour l'algorithme de clustering classique SOM. Pour RSOM, $0.045 \times 10^{-3}$ et pour une fenêtre optimale obtenue de 15. Ainsi, nous confirmons que les performances des RNN par l'approche locale sont toujours améliorées.

Dans le cas de la série Laser, la fenêtre temporelle optimale varie de 3 à 15 en fonction de l'algorithme d'apprentissage utilisé. Nous obtenons pour l'algorithme TKM une NMSE de $0,023$ avec 3 valeur passées, suivie de $0,037$ pour l'algorithme classique SOM avec une

fenêtre temporelle de 6 et enfin pour RSOM, la NMSE obtenue est de $0,033$ avec une fenêtre temporelle composée de 13 valeurs passées. On remarque, pour tous les résultats présentés, la supériorité du prédicteur RNN global.

Les trois séries confirment la supériorité de TKM comme algorithme de clustering quand il s'agit de prédire avec un réseau récurrent sur les bases locales obtenues.

| | SunSpots | Laser | MG-17 $\times 10^3$ |
|---|---|---|---|
| SOM | $0,029$ | $0,037$ | $0,040$ |
| TKM | $0,016$ | $0,023$ | $0,036$ |
| RSOM | $0,036$ | $0,033$ | $0,045$ |

TABLE 5.6 – Meilleurs résultats obtenus en faisant varier la taille de la fenêtre temporelle

### 5.3.3 Étude sur l'impact de l'horizon de prévision

L'horizon de prévision $h$ est un facteur important dans tout système de prévision. On sait que la qualité de prévision se dégrade quand on augmente $h$. Dans le but de voir quelle configuration résiste le mieux, nous étudions l'impact de ce facteur sur l'erreur de prévision finale.

La figure 5.16 nous donne le résultat de cette expérience pour la série temporelle des taches solaire. On trouve dans ce schéma trois courbes associées aux algorithmes de clustering (SOM, TKM et RSOM) ainsi qu'une quatrième courbe qui représente l'évolution de l'erreur d'un prédicteur RNN global.

Selon la figure 5.16 qui dresse l'évolution des NMSE moyennes obtenues, l'algorithme TKM est plus performant que SOM et RSOM comme algorithme de clustering pour tous les horizons de prévisions testés. Dans le cas de l'approche globale, l'erreur de prévision commence proche de celle obtenue par TKM sur un horizon de prévision $h = 1$ puis se dégrade rapidement et vient au niveau de l'approche locale par clustering SOM et RSOM. A partir d'un horizon assez élevé $h = 10$, l'erreur de prévision augmente considérablement pour dépasser toutes les autres. On peut dire que le clustering, et plus spécifiquement TKM, procure une meilleure robustesse en ce qui concerne la prévision à long terme.

Ce même résultat est similaire pour les séries temporelles Laser et MG-17.

### 5.3.4 Comparaison avec l'état de l'art

Dans cette section nous effectuons une comparaison des résultats obtenus par l'intégration des RNN dans l'approche locale avec les résultats de l'état de l'art des prévisions des séries temporelles. Le tableau 5.7 prend en compte le cas de la séries des tâches solaires. Les résultats de l'état de l'art présenté dans ce tableau sont classés dans l'ordre décroissant de la NMSE. Dans [Cao, 2003], un ensemble de prédicteurs expert composée de SVM est appliqué. CPA-NN [Oh *et al.*, 2005] est une approche locale qui utilise l'analyse en composante principale pour le clustering. [Weigend *et al.*, 1992], [Kouam et Fogelman, 1993] et [Czernichow, 1993] utilisent des MLP en tant que prédicteurs locaux dans des approches

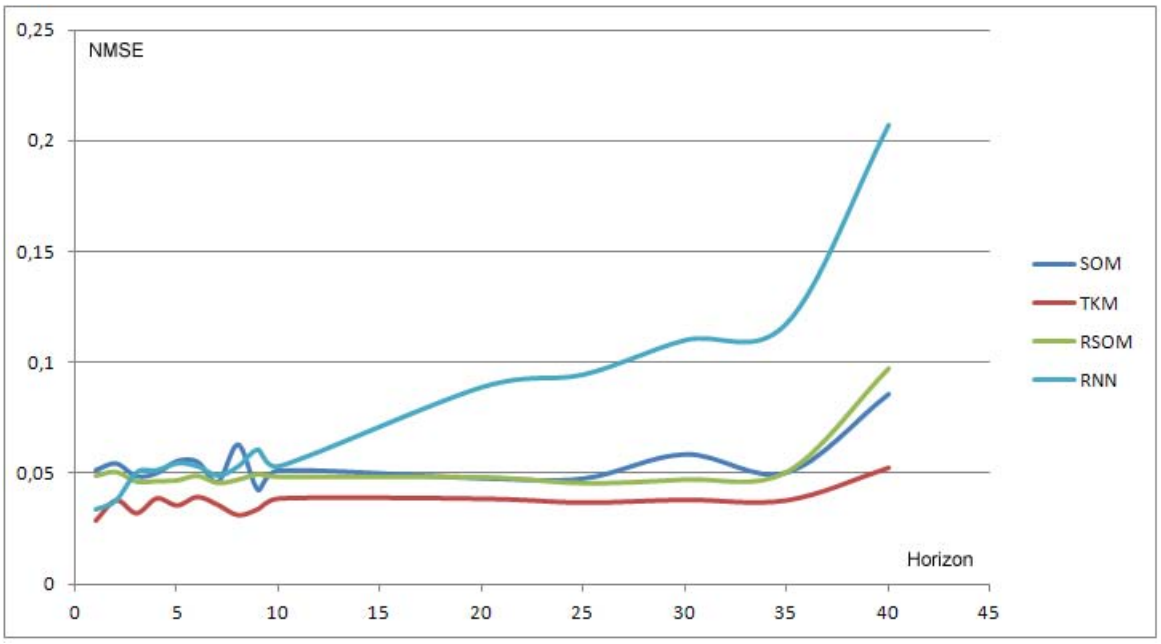


FIGURE 5.16 – Évolution de l'erreur NMSE pour les taches solaires en fonction de l'horizon de prévision

locales plus au moins semblables. Dans [Boné, 2000], un réseau de neurones récurrent est utilisé de manière globale sur la série. Dans [Assaad *et al.*, 2008], un algorithme de Boosting met en œuvre les réseaux de neurones récurrents. Enfin on trouve, un SVM avec des noyaux composites dans [Jiang *et al.*, 2007].

Les modèles proposés intégrant les RNN dans l'approche locale donnent les meilleurs résultats. Comme vu précédemment, TKM est l'algorithme de clustering le plus performant avec au meilleur des cas une NMSE de $0,011$. L'approche locale combinée au réseau récurrent permet d'améliorer en général la performance du prédicteur. Nous obtenons ainsi une NMSE plus petite en comparant les meilleurs résultats.

Le tableau 5.11 présente un comparatif avec l'état de l'art sur la série Laser. Les méthodes sont présentées en ordre décroissant de la NMSE. Dans [Koskela *et al.*, 1997], une approche locale utilisant RSOM comme algorithme de clustering et MLP en tant que prédicteur est mise en œuvre. Dans [Wan, 1993], un MLP à connexions FIR est utilisé. Dans [Noelia Sánchez-Maroño *et al.*, 2003], une combinaison entre SOM comme algorithme de clustering et des réseaux fonctionnels est retenue.

Le meilleur RNN local est celui utilisé après le clustering TKM $(0,021)$. Mais les meilleurs résultats obtenus dans l'état de l'art sont liées à l'utilisation des réseaux de neurones récurrents, d'abord en tant que prédicteur global [Boné, 2000] $(0,008)$ et ensuite en l'intégrant avec l'algorithme de Boosting [Assaad *et al.*, 2008] $(0,005)$.

Le tableau 5.17 présente un comparatif avec l'état de l'art sur la série Mackey-Glass. Les méthodes sont présentées dans l'ordre décroissant des erreurs NMSE. Un réseau RBFN

| Modèle | NMSE Test |
|---|---|
| [Cao, 2003] | $0,1544$ |
| CPA-NN [Oh *et al.*, 2005] | 0.99 |
| MLP [Weigend *et al.*, 1992] | $0,086$ |
| RNN [Boné, 2000] | $0,084$ |
| MLP [Kouam et Fogelman, 1993] | $0,082$ |
| MLP [Czernichow, 1993] | $0,078$ |
| Boosting [Assaad *et al.*, 2008] | $0,078$ |
| SVM [Jiang *et al.*, 2007] | $0,039$ |
| | |
| SOM+RNN | $0,017$ |
| TKM+RNN | $0,011$ |
| RSOM+RNN | $0,027$ |

TABLE 5.7 – Tableau comparatif des meilleurs résultats sur la série des taches solaires

| Modèle | NMSE Test |
|---|---|
| RSOM [Koskela *et al.*, 1997] | $0,084$ |
| FIR MLP [Wan, 1993] | $0,023$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $0,011$ |
| RNN [Boné, 2000] | $0,008$ |
| Boosting [Assaad *et al.*, 2008] | $0,005$ |
| | |
| SOM+RNN | $0,031$ |
| TKM+RNN | $0,021$ |
| RSOM+RNN | $0,036$ |

TABLE 5.8 – Tableau comparatif des meilleurs résultats sur la série Laser

est utilisé dans [Zemouri *et al.*, 2003] par une approche globale. Dans [Koskela *et al.*, 1997] une approche locale utilisant RSOM comme algorithme de clustering et MLP en tant que prédicteur est mise en œuvre. Dans [Barreto *et al.*, 2004] un modèle local est utilisé en se basant sur des réseaux de neurones compétitifs. Dans [Vesanto, 1997] une autre approche locale est présentée en se basant sur les réseaux de neurones et SOM comme algorithme de clustering. Un RNN avec boosting est utilisé dans [Assaad *et al.*, 2008].

L'intégration des RNN dépasse en performance les méthodes citées ci-dessus. Toutefois, la meilleure performance collectée dans l'état de l'art est celle de [Noelia Sánchez-Maroño *et al.*, 2003], dépassant de loin toutes les autres.

| Modèle | NMSE Test $\times 10^3$ |
|---|---|
| [Zemouri *et al.*, 2003] | 38 |
| [Koskela *et al.*, 1997] | $3,11$ |
| [Barreto et Araujo, 2004] | $0,75$ |
| [Vesanto, 1997] | $0,17$ |
| Boosting [Assaad *et al.*, 2008] | $0,11$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $5,75 \times 10^{-4}$ |
| SOM+RNN | $0,037$ |
| TKM+RNN | $0,035$ |
| RSOM+RNN | $0,040$ |

TABLE 5.9 – Tableau comparatif des meilleurs résultats sur la série MG-17

Les expérimentations effectuées ci-dessus nous permettent de formuler quelques conclusions qui nous serons utiles dans les expérimentations qui vont suivre. D'abord, on remarque amélioration des performances des prédicteurs RNN par l'approche locale. En effet, deux éléments rentrent en jeu, à savoir la taille et la saisonnalité des données. Dans le cas des taches solaires, des données qui ne sont pas assez volumineuses, on remarque une difficulté à améliorer les performances. Dans le cas de la série Laser, une série qui ne possède pas un aspect saisonnier constant, on peut observer clairement que les prédicteurs globaux sont plus performants. La série MG-17 est l'exemple parfait de l'apport positif de l'approche locale, car elle est assez volumineuse pour effectuer l'apprentissage et possède d'autre part une caractéristique saisonnière répartie sur toutes les périodes.

Nous avons pu également observer l'adéquation entre l'algorithme de clustering TKM et les prédicteurs récurrents, cette adéquation n'est pas valide pour les autres types de prédicteurs testés (à fenêtre temporelle) où l'algorithme classique était plus efficace.

Bien que l'approche locale permette d'améliorer les performances du RNN, il est important de remarquer que les algorithmes de clustering testés étaient plus efficaces, quelque soit les données, avec les prédicteurs à fenêtre temporelle (MLP et SVM). Pour cette raison, nous nous concentrerons dans les expérimentations futures sur ces deux types de prédicteurs.

Nous pouvons dire que dans les trois cas expérimentés, l'approche locale possède un apport positif sur la prévision à long terme. Nous effectuerons ce test pour les méthodes

de clustering que nous avons proposé dans le chapitre précédent.

Enfin, nous pouvons remarquer l'importance du choix du nombre de clusters. Nous avons vu que les meilleures performances ont tendance à se répéter autour d'un optimal qui varie en fonction de la série utilisée. C'est l'idée principale qui motive les méthodes testées dans les sections suivantes. Nous vérifierons si dans les deux cas nous aboutirons vers des résultats qui seront similaires ou proches des résultats trouvés ici empiriquement.

## 5.4 Clustering par Auto-SOM

### 5.4.1 Étude des performances

Les expérimentations qui vont suivre concerne l'algorithme proposé dans le chapitre précédent et nommé Auto-SOM. Comme nous l'avons dit précédemment, nous utiliserons dans cette méthode les prédicteurs à base de fenêtres temporelles (MLP et SVM).

#### Problèmes de convergences

La convergence de l'algorithme Auto-SOM est un critère important à vérifier avant de commencer notre étude. Ce que nous entendons ici par convergence est le fait d'obtenir des classes bien séparées sur la carte et contenant assez de données pour effectuer l'apprentissage. En effet, en termes d'exécution, l'algorithme finit par s'arrêter avec au pire des cas des clusters élémentaires de très petites tailles séparées d'un nombre de clusters vides et une carte de grande taille. Nous cherchons à éviter ce cas de figure extrême. Nous verrons que ce critère dépend principalement des données traitées.

Le cas de la série des taches solaires est le cas le plus simple. La figure 5.17 nous montre comment l'algorithme Auto-SOM converge assez rapidement vers la carte avec des classes bien séparées. Le nombre d'itérations est assez limité, avec environ une dizaine d'itérations. La courbe en bleu représente la taille moyenne des classes obtenues dynamiquement au cours des itérations (notée TdC), la courbe horizontale en rouge représente un taux fixe choisi arbitrairement d'environ 10% de l'ensemble d'apprentissage, référant à une taille moyenne des clusters (notée TmC). On remarque que l'algorithme finit par converger avant que la courbe dépasse la taille limite fixée.

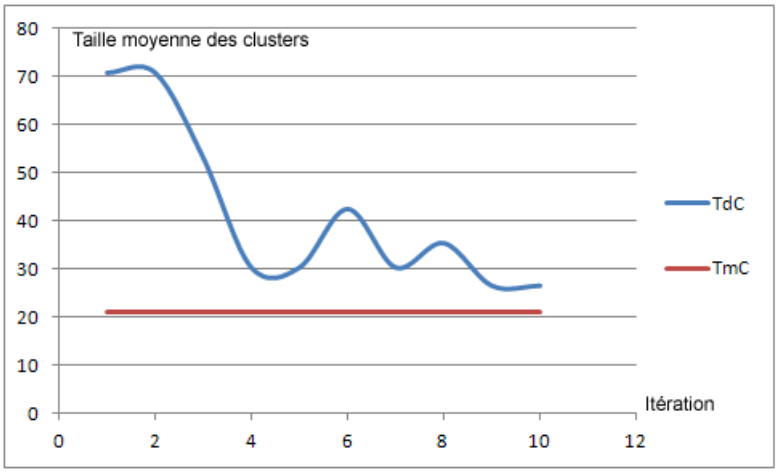


FIGURE 5.17 – Taille moyenne des classes obtenues dynamiquement par l'algorithme Auto-SOM au cours des itérations (notée TdC) et taille moyenne des clusters 10% de l'ensemble d'apprentissage (notée TmC), cas de la série des taches solaires.

Ceci n'est pas le cas des séries Laser et MG-17 comme le montrent les figures 5.18 et 5.19. On observe que la taille moyenne des classes obtenues est en chute constante en progressant dans les itérations de l'algorithme. Il en résulte des cartes de très grandes tailles avec des clusters contenant de petits volumes de données et séparés par des clusters vides. Cette situation est problématique et constitue un des inconvénients majeurs de cette méthode. Afin d'éviter ces configurations, nous rajoutons une condition d'arrêt quand la taille moyenne des classes obtenues devient plus petite qu'un taux fixé au départ. Dans le cas de nos expérimentation, nous avons choisi 10% de l'ensemble d'apprentissage.

Le point commun entre la série Laser et MG-17 c'est qu'elles sont des séries considérées comme chaotiques, contrairement à la série des taches solaires. Dans le reste des expérimentations, nous utiliserons ce dernier critère d'arrêt afin d'éviter que la base d'apprentissage ne soir trop dispersée dans la carte. Bien que ce critère semble inutile pour les taches solaires, il parait important pour des autres séries utilisées.

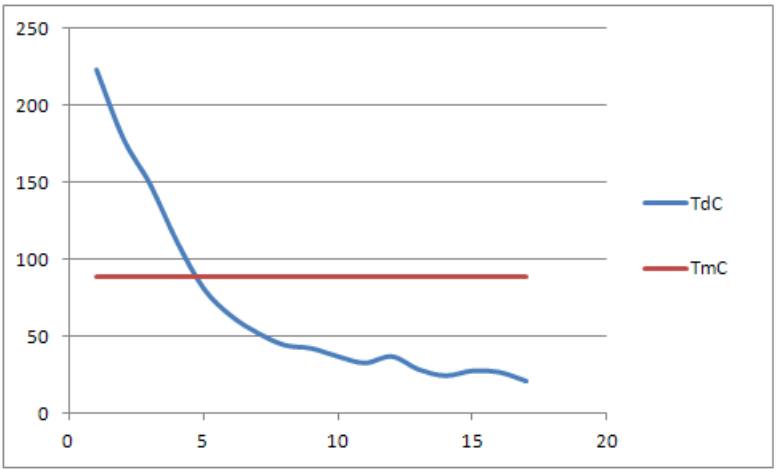


FIGURE 5.18 – Taille moyenne des classes obtenue par l'algorithme Auto-SOM obtenues dynamiquement au cours des itérations (notée TdC) et taille moyenne des clusters 10% de l'ensemble d'apprentissage (notée TmC), cas de la série Laser.

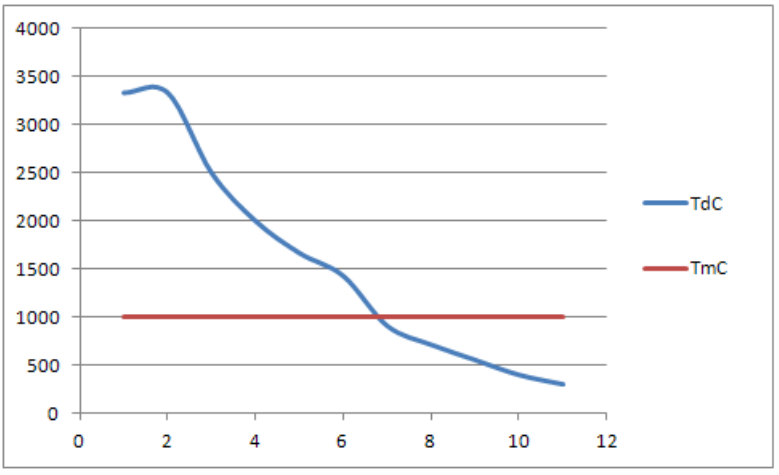


FIGURE 5.19 – Taille moyenne des classes obtenues dynamiquement par l'algorithme Auto-SOM obtenues au cours des itérations (notée TdC) et taille moyenne des clusters 10% de l'ensemble d'apprentissage (notée TmC), cas de la série MG-17.

### Composition de la carte

Dans une première étape nous choisissons d'étudier la composition des clusters obtenus par l'algorithme Auto-SOM et les performances associées à chacun des clusters. Ceci nous permettra d'établir une première comparaison avec l'algorithme de clustering classique. Nous comparerons également entre deux modes différents de clustering vus dans le chapitre

précédent. Le premier consiste à considérer la carte obtenue telle qu'elle. Le second consiste à considérer les grandes classes constituées par les groupements obtenus sur la carte.

Les figures 5.20, 5.21 et 5.22 montrent, pour respectivement les séries des tâches solaires, Laser et MG-17, la composition obtenue sur une carte linéaire avec l'algorithme Auto-SOM. Le schéma en haut montre l'état final de la carte obtenue. Tant disque dans le schéma en bas montre le regroupement des clusters en classes.

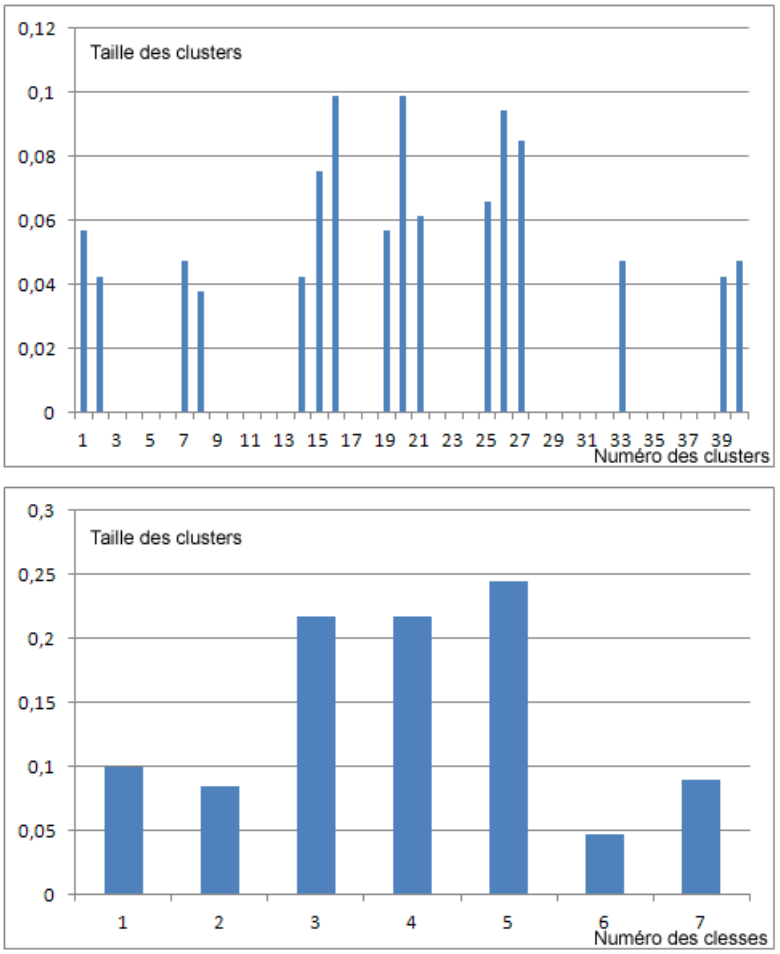


FIGURE 5.20 – Répartition des clusters pour les taches solaires

Dans la figure 5.20, il s'agit des résultats d'une exécution de l'approche locale par Auto-SOM sur les données des taches solaires. Dans cet exemple, le SVM a été utilisé en tant que prédicteur. Dans l'histogramme en haut, nous pouvons observer clairement la décomposition de la carte, la NMSE obtenue sur l'ensemble de test après le clustering est de $0,0048$ et dans l'histogramme en bas, les groupes de clusters ont été rassemblés en 7 classes donnant une NMSE sur l'ensemble de test de $0,0047$. On observe donc une petite amélioration.

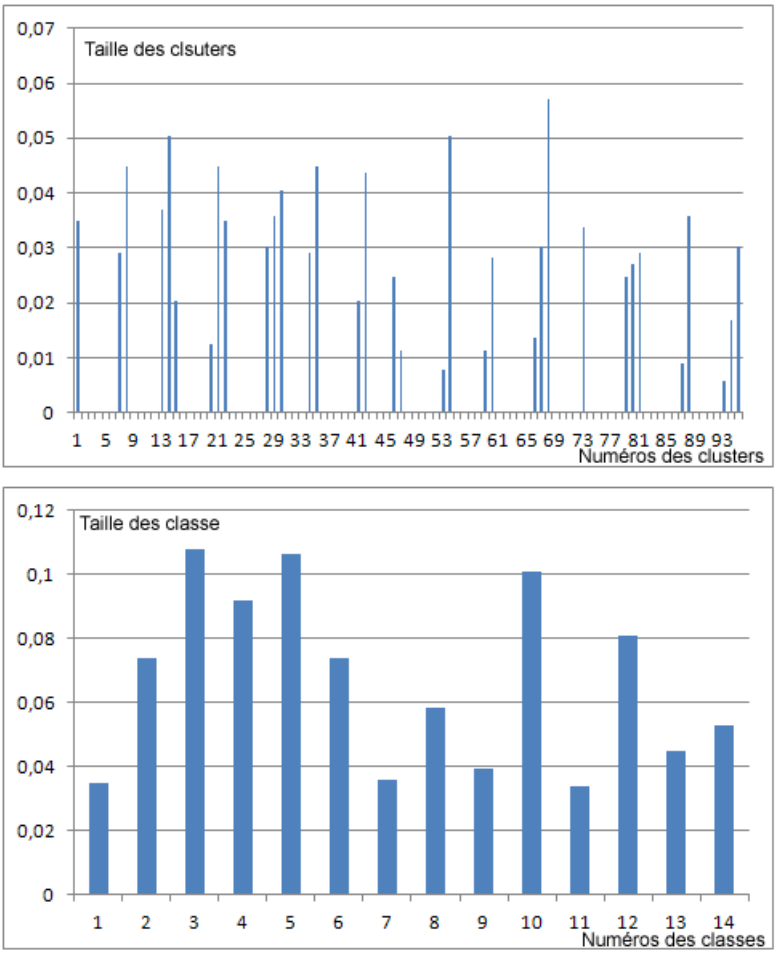


FIGURE 5.21 – Répartition des clusters pour la série Laser

Dans la figure 5.21, nous retrouvons les résultats obtenus sur la série Laser. Dans le schéma en haut, nous pouvons observer la répartition des clusters dans la carte. La NMSE correspondante sur l'ensemble test est de $0,0062$. Dans le schéma en bas, nous pouvons retrouver les 14 classes correspondantes, ce mode de clustering permet d'obtenir une meilleure NMSE sur l'ensemble de test $(0,0055)$.

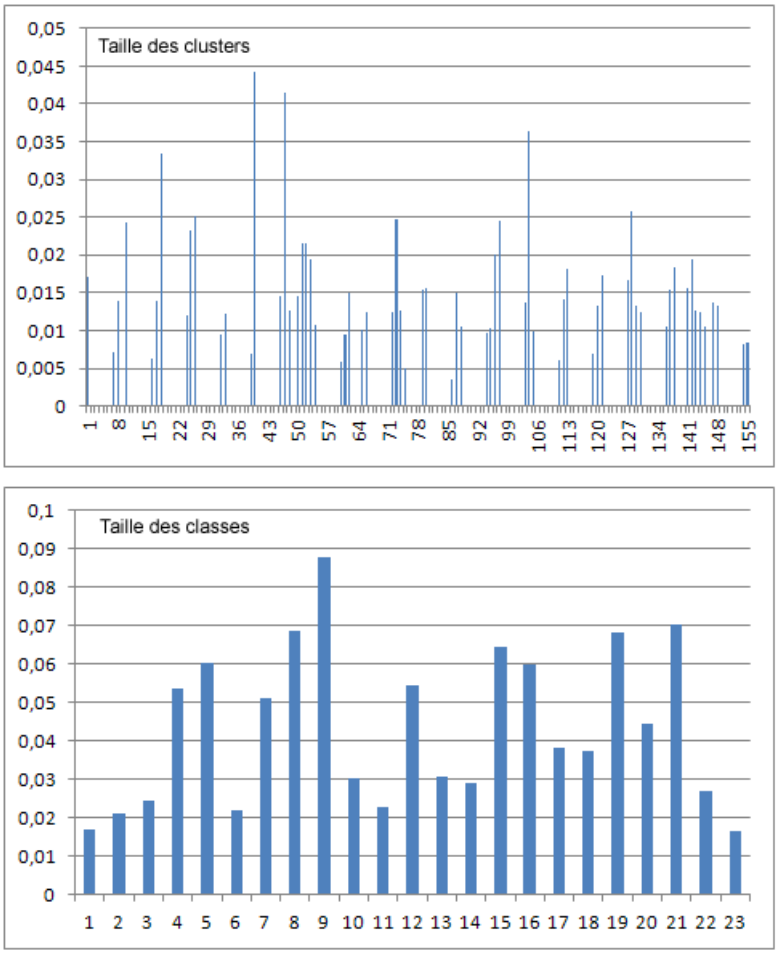


FIGURE 5.22 – Répartition des clusters pour la série MG-17

La figure 5.22 nous montre les deux modes de clustering obtenus sur la série MG-17. Il est naturel que la carte obtenue pour cette série soit la plus grande (environ 150 clusters). La NMSE réalisée sur la carte est de $2,3 \times 10^{-7}$ par rapport à $1,5 \times 10^{-7}$ lorsqu'on considère uniquement les 23 classes obtenues.

Dans les trois cas testés, nous avons remarqué une amélioration de l'erreur en utilisant les classes issues de la carte finale, au lieu d'utiliser directement les clusters de la carte finale. Ce résultat est généralement validé avec 9 expérimentations sur 10 (soit dans 90% des cas). Ceci peut être expliqué par l'obtention d'un meilleur compromis entre homogénéité et richesse (en terme de nombre) des données dans un même cluster grâce à l'algorithme Auto-SOM.

### Les prototypes obtenus

Nous pouvons évaluer la qualité du clustering obtenu en visualisant les prototypes des différents clusters. Cela nous permettra d'évaluer également l'intérêt de rassembler les

unités de la cartes qui sont proches en une classe, en comparant les différents prototypes obtenus. La figure 5.23 reprend les résultats de l'algorithme auto-SOM dans une de ces exécutions choisie arbitrairement. Les prototypes sont représentés par des portions de courbes dont l'abscisse représente la position dans la fenêtre temporelle et l'ordonnée représente la valeur associée dans le prototype. Les prototypes sont rassemblés selon la classe dont ils font parti.

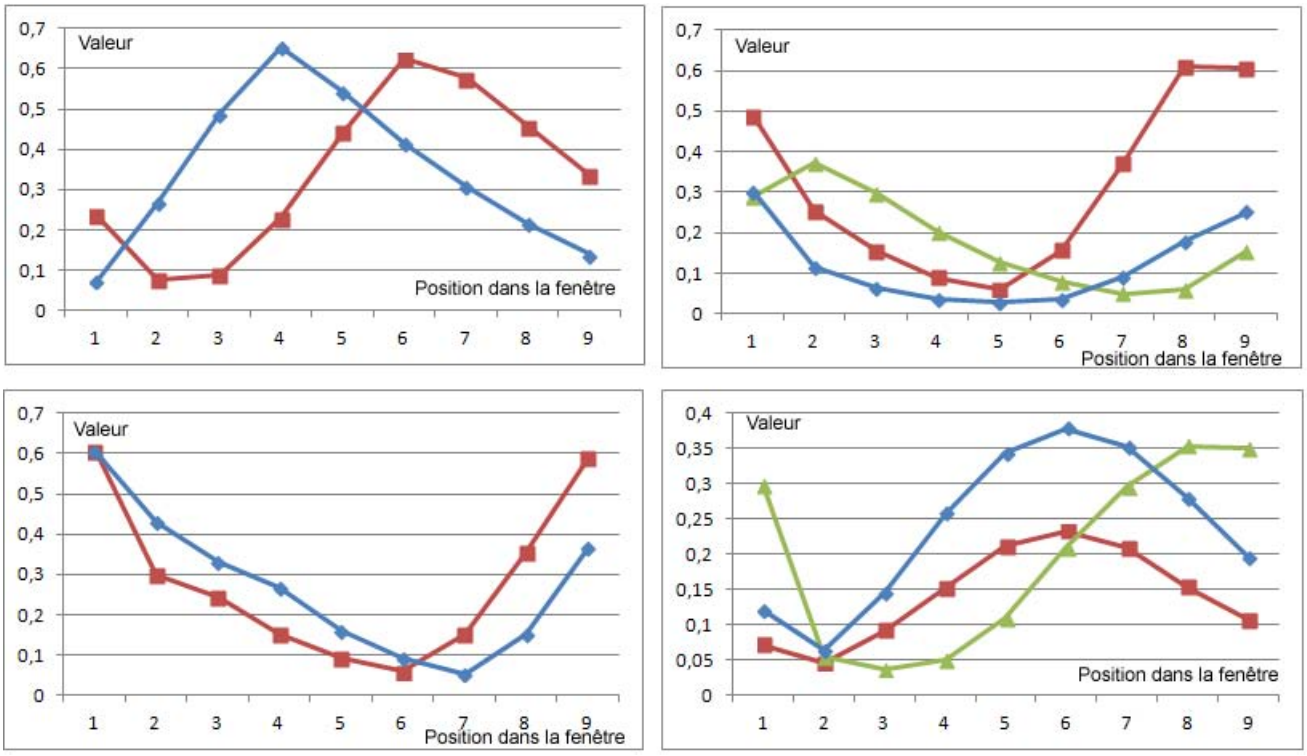


FIGURE 5.23 – Les prototypes obtenus dans les différentes classes pour les taches solaires

Dans chaque schéma de la figure 5.23 on trouve pour le cas des taches solaires deux ou trois prototypes qui ont été regroupés dans une même classe. On peut remarquer une ressemblance dans les prototypes dans une même classe. On peut aussi voir une certaine ressemblance entre des classes différentes. Dans l'exemple actuel, nous avons obtenu 4 classes. Nous rappelons que nous avons trouvé précédemment de manière empirique que deux clusters étaient le nombre de clusters optimal pour les taches solaires. Ceci peut représenter une décomposition excédentaire qui pourrait limiter les performances. Le problème est que dans le cas des Auto-SOM est que le résultat final dépend de l'initialisation de la carte. Lorsque deux unités éloignées dans la carte se trouvent avec deux prototypes similaires, cette situation provoquerait la composition d'une nouvelle classe. Ce défaut est directement rattaché à la nature des cartes de Kohonen. On pourrait alors imaginer une adaptation des *Neural gas* qui nous permettrait de mieux cerner ce point.

### L'évolution de l'erreur de quantification

Il est normal que la convergence de l'algorithme auto-SOM prenne plus de temps que celle de l'algorithme classique SOM puisqu'il s'agit d'un procédé itératif de SOM. Cependant, il est important de remarquer que plus on avance dans le temps, plus la convergence devient rapide, car les exemples deviennent de plus en plus rangés et donc les variations dans la carte deviennent de plus en plus locales.

La figure 5.24 représente la variation de l'erreur de quantification en fonction du temps pour l'algorithme Auto-SOM. Nous représentons ici le graphique de la série des taches solaires, le plus lisible. Le comportement de la courbe est le même pour les autres séries.

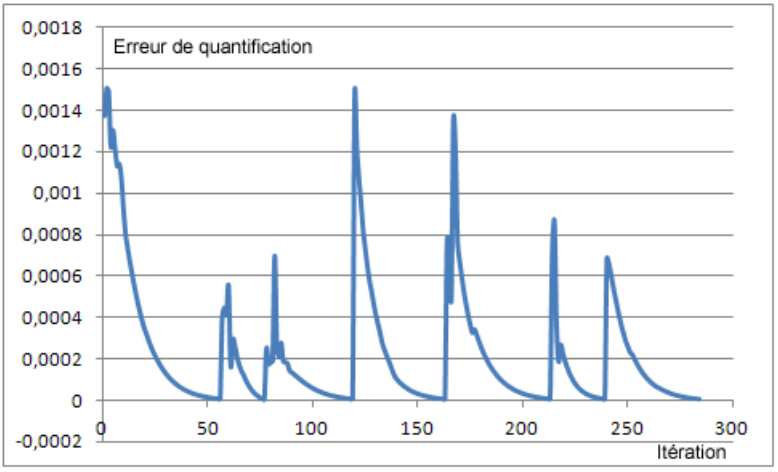


FIGURE 5.24 – Evolution de l'erreur de quantification en fonction des itération de l'algorithme auto-SOM

Les pics dans la figure 5.24 représentent les moments où de nouvelles unités ont été rajoutées à la carte. Au fur et à mesure de la répétition du processus, la convergence devient plus simple et plus rapide. Ceci peut être traduit par trois éléments dans la courbe : la largeur de la descente qui devient plus faible, la hauteur des pics qui a tendance à devenir plus basse et enfin la courbe de la descente qui devient de plus en plus lisse.

## 5.4.2 Impact de la taille de la fenêtre

Comme pour les algorithmes classiques, nous étudions l'impact de la taille de la fenêtre temporelle sur les résultats finaux de l'algorithme Auto-SOM. La figure 5.25 nous présente ce rapport pour les séries temporelles des taches solaires, Laser et MG-17 dans respectivement les schémas (a), (b) et (c). Dans chacun de ces schémas on trouve une courbe pour la performance du prédicteur MLP (en bleu) et une autre pour le prédicteur SVM (en rouge).

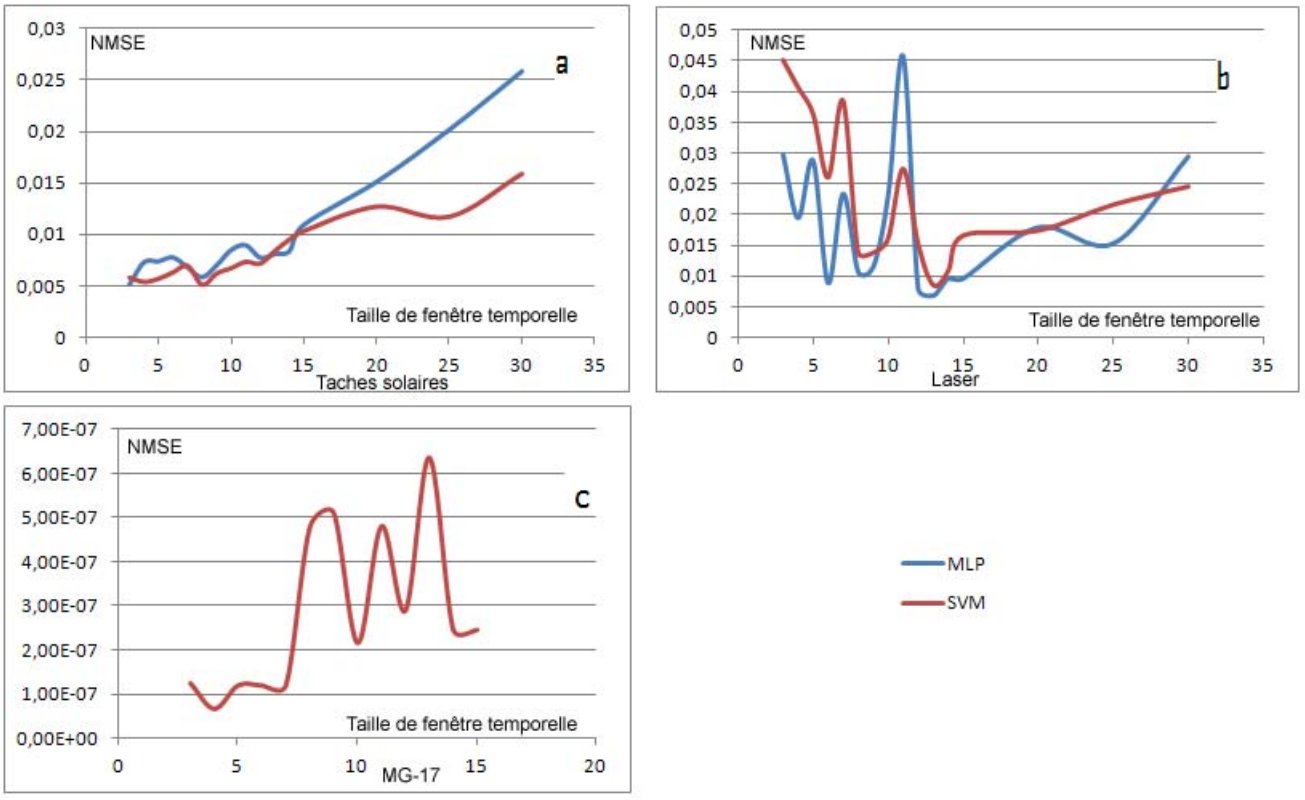


FIGURE 5.25 – Evolution de l'erreur de prévision en fonction de la taille de la fenêtre temporelle

D'après la figure 5.25, on peut observer que dans les trois séries il existe un impact négatif sur la performance NMSE lorsqu'on augmente la taille de la fenêtre temporelle. Bien évidement, la taille optimale varie en fonction de la série en question tout en ayant des comportements semblables, que ce soit pour le MLP ou le SVM. Cependant, les prédicteurs locaux SVM sont plus robustes que les MLP Locaux à la malédiction de la surdimensionnalité. Dans le cas de la série Laser on peut dire qu'il y a une similarité entre les deux prédicteurs. Pour la série MG-17, nous n'avons représenté que les performances des SVM locaux, leurs performances étaient nettement supérieures aux MLP.

### 5.4.3 Impact de l'horizon de prévision

Nous évaluons dans ce paragraphe l'impact de l'horizon de prévision sur les performances du système, et cela en le comparant avec les résultats obtenus avec des modèles globaux. La figure 5.26 nous présente cette comparaison pour les séries temporelles des taches solaires, Laser et MG-17 dans respectivement les schémas (a), (b) et (c). Dans chacun de ces schémas, on trouve une courbe pour la performance du prédicteur MLP (en bleu) et une autre pour le prédicteur SVM (en rouge). La courbe en vert représente les performances du modèle global SVM.

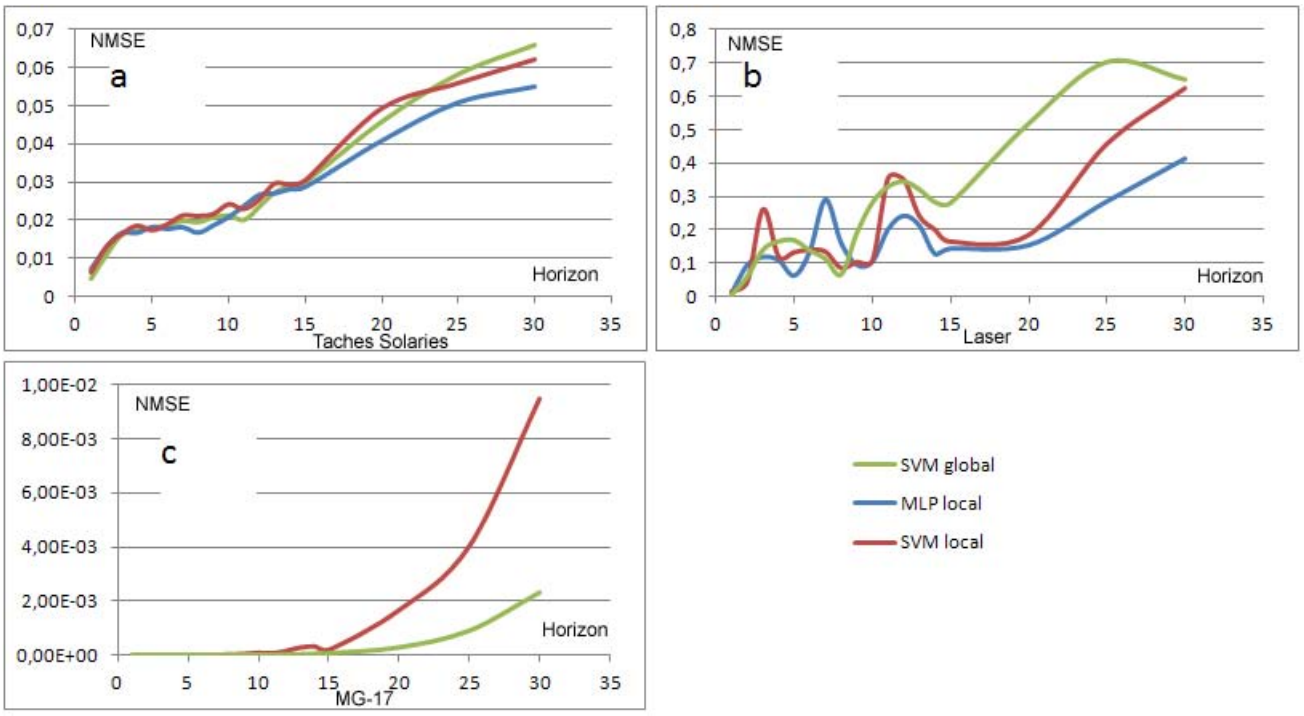


FIGURE 5.26 – Evolution de l'erreur de prévision en fonction de la taille de la fenêtre temporelle (en rouge SVM, en bleu MLP)

L'effet négatif sur l'erreur de prévision au fur et à mesure qu'on augmente l'horizon de prévision apparaît clairement dans la figure 5.26 pour les trois schémas. Dans le cas de la série des taches solaires, les performances entre les MLP et les SVM locaux et le SVM global peuvent être considérées comme proches, mais à partir de l'horizon $h = 15$, on peut ressentir que le prédicteur MLP local commence à se distinguer par rapport aux autres. Il n'en est pas moins vrai pour la série Laser ; on peut dire qu'après un horizon d'environ $h = 12$, on observe une supériorité du MLP local à prédire à long terme, suivit du SVM local et avec un écart plus large pour le prédicteur SVM global. Cependant, pour le cas de MG-17, le SVM global semble être plus performant pour prédire à long terme que le modèle local. Nous avons représenté uniquement les performances du prédicteur SVM car ses performances restent globalement supérieures à celles du MLP.

### 5.4.4 Comparaison avec l'état de l'art

Dans cette section, nous effectuerons une comparaison des résultats obtenus en utilisant l'algorithme de clustering auto-SOM et les prédicteurs à fenêtre temporelle (MLP et SVM) avec les résultats collectionnés dans l'état de l'art. Le tableau 5.10 prend en compte le cas de la série des taches solaires. Les résultats ddans ce tableau sont présentés dans l'ordre décroissant de la NMSE. Une description des méthodes peut être trouvée dans la section précédente. Les modèles proposés utilisant l'algorithme Auto-SOM et les prédicteurs

MLP et SVM dans l'approche locale donnent les meilleurs résultats. L'erreur NMSE est quasiment divisée par 10 comparée à [Jiang *et al.*, 2007], avec une supériorité du MLP par rapport au SVM.

| Modèle | NMSE Test |
|---|---|
| [Cao, 2003] | 0, 1544 |
| CPA-NN [Oh *et al.*, 2005] | 0.99 |
| MLP [Weigend *et al.*, 1992] | 0, 086 |
| RNN [Boné, 2000] | 0, 084 |
| MLP [Kouam et Fogelman, 1993] | 0, 082 |
| MLP [Czernichow, 1993] | 0, 078 |
| Boosting [Assaad *et al.*, 2008] | 0, 078 |
| SVM [Jiang *et al.*, 2007] | 0, 039 |
| | |
| auto-SOM + MLP | 0, 0037 |
| auto-SOM + SVM | 0, 0047 |

TABLE 5.10 – Tableau comparatif des meilleurs résultats sur la série des taches solaires

Le tableau 5.11 présente un comparatif avec l'état de l'art sur la série Laser. Les méthodes sont présentées dans l'ordre décroissant de la NMSE. Une description des méthodes peut être trouvée dans la section précédente. Les meilleurs résultats obtenus dans l'état de l'art sont liées à l'utilisation des réseaux de neurones récurrents, d'abord en tant que prédicteur global [Boné, 2000] $(0, 008)$ et ensuite en l'intégrant avec l'algorithme de Boosting [Assaad *et al.*, 2008] $(0, 005)$. Les approches locales par Auto-SOM donnent des résultats aussi proches avec $0, 0062$ pour le MLP local et $0, 0052$ le SVM local.

En conclusion, l'approche locale améliore aussi bien le prédicteur que le boosting dans le cas de la série Laser.

| Modèle | NMSE Test |
|---|---|
| RSOM [Koskela *et al.*, 1997] | 0, 084 |
| FIR MLP [Wan, 1993] | 0, 023 |
| [Noelia Sánchez-Maroño *et al.*, 2003] | 0, 011 |
| RNN [Boné, 2000] | 0, 008 |
| Boosting [Assaad *et al.*, 2008] | 0, 005 |
| | |
| Auto-SOM+MLP | 0, 0062 |
| Auto-SOM+SVM | 0, 0052 |

TABLE 5.11 – Tableau comparatif des meilleurs résultats sur la série Laser

Le tableau 5.17 présente un comparatif avec l'état de l'art sur Mackey-Glass. Les méthodes sont présentées dans l'ordre décroissant des l'erreur NMSE. Un réseau RBFN est utilisé dans [Zemouri *et al.*, 2003], par une approche globale. Une description des méthodes peut être trouvée dans la section précédente. La meilleure performance récoltée dans l'état de l'art est celle de [Noelia Sánchez-Maroño *et al.*, 2003] dépassant de loin toute les autres.

Auto-SOM permet d'améliorer aussi bien les performances des prédicteurs avec $0,44 \times 10^{-3}$ pour le MLP local et $1,54 \times 10^{-7}$ pour le prédicteur SVM, dépassant ainsi les résultats de [Noelia Sánchez-Maroño *et al.*, 2003].

| Modèle | NMSE Test ($10^{-3}$) |
|---|---|
| [Zemouri *et al.*, 2003] | 38 |
| [Koskela *et al.*, 1997] | 3,11 |
| [Barreto et Araujo, 2004] | 0,75 |
| [Vesanto, 1997] | 0,17 |
| Boosting [Assaad *et al.*, 2008] | 0,11 |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $5,75 \times 10^{-4}$ |
| Auto-SOM+MLP | 0,44 |
| Auto-SOM+SVM | $1,54 \times 10^{-4}$ |

TABLE 5.12 – Tableau comparatif des meilleurs résultats sur la série MG-17

Les expérimentations réalisées dans cette partie nous ont permis d'analyser l'apport du clustering par Auto-SOM sur la prévision avec des prédicteurs à fenêtre temporelle (c'est à dire SVM et MLP). On remarque une amélioration des performances dans les trois séries de référence. Cependant, nous avons signalé certains défauts et manque d'Auto-SOM. D'abord, les résultats qui sont dépendant de l'initialisation de la carte. Puis, le problème de convergence rencontré avec les séries chaotiques. Et enfin, l'indépendance entre la phase de clustering et la phase de prévision. Pour cette raison, nous supposons que l'approche locale à clustering hiérarchique sera plus à même de résoudre certains défauts d'Auto-SOM.

## 5.5 Expérimentation de l'approche hiérarchique

Nous présentons dans cette section, les différents résultats obtenus par l'approche locale en utilisant le clustering hiérarchique. Nous avons défini dans le chapitre précédent trois stratégies associées à cette l'approche. Nous proposons ici de les utiliser avec les prédicteurs à fenêtre temporelle MLP et SVM. Nous suivons à peu près la même organisation que les sections précédentes : une étude des performances générales suivie d'une étude des variations des paramètres tels que la taille de la fenêtre temporelle et l'horizon de prévision et enfin une comparaison avec l'état de l'art.

### 5.5.1 Étude des performances

Le tableau 5.13 résume les meilleures performances obtenues (sur les ensembles de tests) par toutes les combinaisons possibles des stratégies de clustering et des prédicteurs utilisés et ce pour les trois séries de références. Pour qu'une comparaison soit la plus efficace possible , nous avons apporté de légères modifications dans le code de manière que l'arbre obtenu soit utilisé par les différentes stratégies, sans qu'une telle modification n'impacte dans le déroulement des différents algorithmes. Les résultats obtenus sont donc équivalent à des tests séparés effectués sur les mêmes données ou le résultat du clustering est le même.

Pour la série Sunspot, on trouve que la stratégie de mapping nous donne le meilleur résultat avec le MLP comme prédicteur $(0,0034)$ et avec le SVM $(0,0045)$. Toutefois les résultats sont assez proches dans les deux cas entre les trois différentes stratégies.

Dans le cas de la série Laser, la meilleure performance est obtenue avec la stratégie pré-pruning pour les deux prédicteurs MLP et SVM (respectivement $0,0039$ et $0,0043$).

Dans le cas de MG-17, on peut voir clairement que la supériorité des SVM est encore confirmée avec les deux stratégies pré-pruning et post-pruning au même niveau pour les deux prédicteurs avec des NMSE égales ($1,37 \times 10^{-8}$ pour SVM et $9,82 \times 10^{-6}$ pour le MLP). Dans le tableau 5.14, on peut remarquer que parfois nous obtenons les même résultats avec deux stratégies différentes. Ceci est tout à fait normal car, comme nous l'avons déjà expliqué, les stratégies s'exécutent sur le même arbre. L'avantage est que cette de cette manière nous pouvons, comparer l'efficacité des méthodes.

| | SunSpots | | Laser | | MG-17 | |
|---|---|---|---|---|---|---|
| | MLP | SVM | MLP | SVM | MLP | SVM |
| Pre Prunning | 0,004 | 0,0049 | 0,0039 | 0,0043 | 9,82E-06 | 1,37E-08 |
| Post Pruning | 0,0051 | 0,0046 | 0,0039 | 0,0050 | 9,82E-06 | 1,37E-08 |
| Mapping | 0,0034 | 0,0045 | 0,0040 | 0,0043 | 1,50E-05 | 1,94E-08 |

TABLE 5.13 – Meilleures performances obtenues avec l'approche hiérarchique

Le tableau 5.14 est organisé de la façon suivante : les lignes représentent les stratégies de pruning utilisées, les colonnes représentent l'ordre des performances (1 pour les meilleures performances parmi 3, 2 pour deuxième performance parmi 3, etc.). Les éléments du tableau représentent le nombre de fois où cet ordre a été obtenu. Le tableau a été calculé sur une quarantaine d'expérimentations (avec les deux prédicteurs). Il est clair, d'après le tableau 5.14, que le pré-pruning est la méthode la plus efficace avec 69% des cas ayant obtenu les meilleures performances, 15% la deuxième et 15% la troisième. On peut dire que le mapping est le deuxième plus efficace puisqu'on a environ 51% d'instances en première position, 48% pour la deuxième et 0 pour la troisième.

| | 1 | 2 | 3 |
|---|---|---|---|
| Pre Prinning | 0,69 | 0,15 | 0,15 |
| Post Pruning | 0,43 | 0,46 | 0,10 |
| Mapping | 0,51 | 0,48 | 0 |

TABLE 5.14 – Comparaison des performances des stratégies de pruning pour la série des taches solaires

Nous proposons par la suite d'étudier les résultats de chaque stratégie de pruning en rentrant dans leurs mécanismes internes. Nous examinerons en premier lieu les arbres obtenus. Chacun des arbres présentés dans les figures 5.27, 5.30 et 5.33 (pour respectivement le pré-pruning, post-pruning et le mapping) représente un cas de figure où cette stratégie a été la plus efficace.

Nous nous sommes restreint au cas de figure des taches solaires car c'est la série qui permet d'avoir des arbres de taille raisonnable afin de les représenter clairement dans un manuscrit.

Chaque nœud de l'arbre est représenté par un rectangle, on y trouve comme information : l'identification du nœud, la taille des données qui y sont attribuées en base d'apprentissage, l'erreur d'apprentissage et la valeur de la contribution du nœud entre parenthèses. Les nœuds qui ont été sélectionnés par l'algorithme comme étant les nœuds finaux du clustering sont représentés en gris avec en plus, dans la dernière ligne, l'erreur de prévision de l'ensemble test et la contribution associée entre parenthèses.

**pré-pruning**

La figure 5.27 correspond à l'arbre obtenu par la stratégie pre-pruning. Nous pouvons observer que la coupe optimale détectée est le niveau 1 de l'arbre. Nous obtenons donc deux clusters (ce qui représente la valeur optimale pour la série taches solaires vu précédemment. Nous avons remarqué, selon les différentes instances lancées, que la stratégie pre-pruning offrait le meilleur clustring lorsque l'arbre obtenu était relativement bien équilibré aussi bien pour les nœuds que pour la répartitions des données. Cette stratégie permet d'arrêter la décomposition des données lorsque elle détecte que la contribution des feuilles est devenue moins bonne que les nœuds supérieurs, ce qui rend cette méthode la plus efficace.

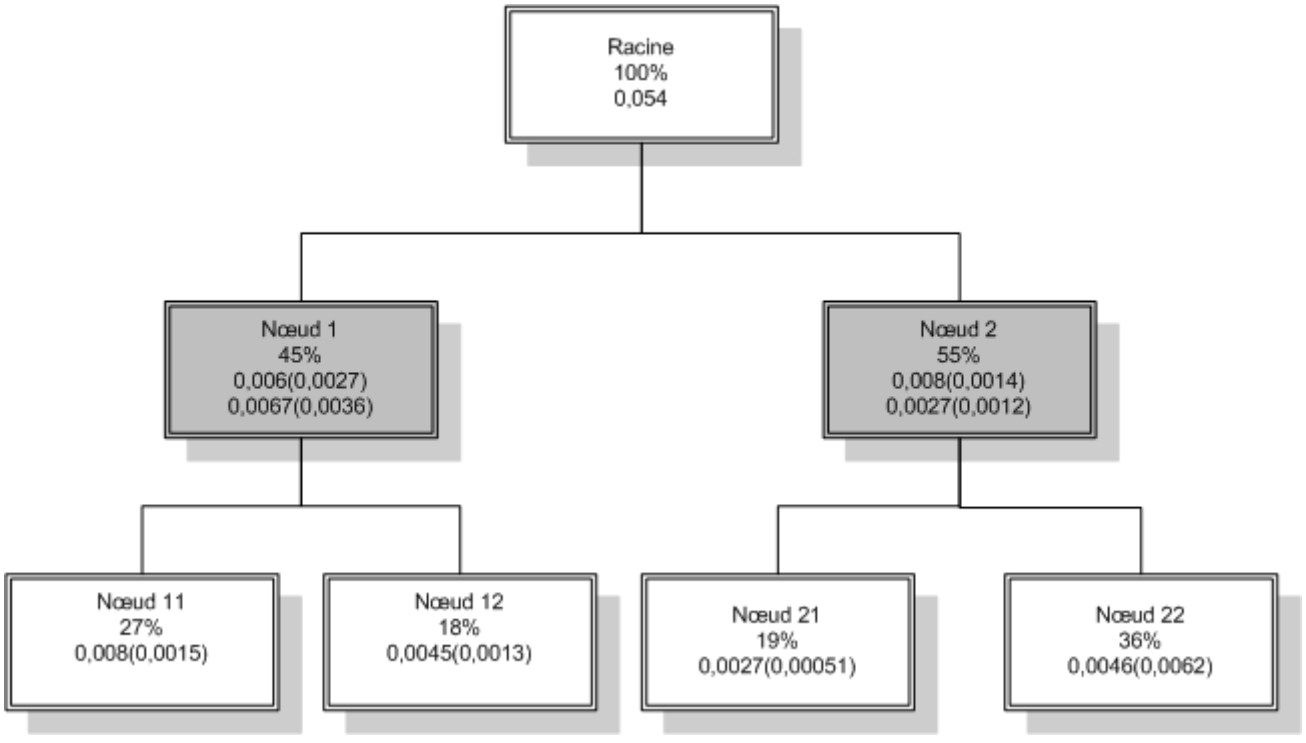


FIGURE 5.27 – Décomposition en arborescence avec la stratégie pre-pruning, cas des taches solaires

La figure 5.28 représente dans un diagramme la répartition des données de la base d'apprentissage dans les deux classes. Cette répartition est assez équilibrée (45% pour l'un et 55% pour l'autre). Les résultats obtenus en terme de NMSE sont $0,006$ pour le

premier cluster avec une contribution de $0,0036$, $0,008$ pour le deuxième cluster avec une contribution de $0,0012$. Clairement, lorsque calcule l'erreur totale sur les données, on obtient une performance meilleure que l'approche globale.

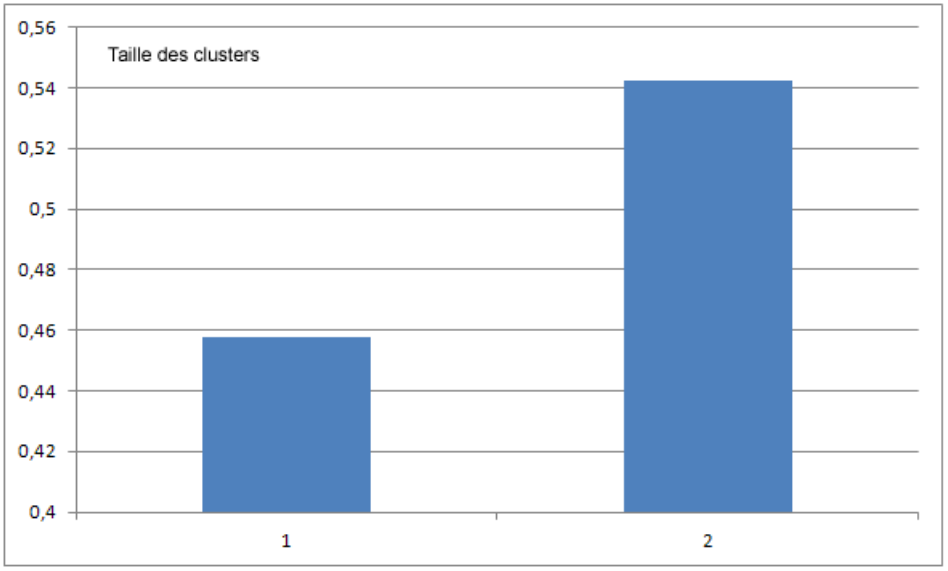


FIGURE 5.28 – Décomposition des clusters avec la stratégie de pre-prunning, cas des taches solaires

Nous pouvons observer dans la figure 5.29 les deux prototypes représentant les données des deux classes. Graphiquement nous observons deux régimes séparés tout en restant non-linéaires. Le fait de séparer ces deux régimes permet d'obtenir un bon compromis entre une base de données assez riche et des exemples assez proches.

### Stratégie post-pruning

La figure 5.30 représente l'arbre obtenu par la stratégie post-pruning. On remarque que l'arbre obtenu est moins équilibré que le précédent, ceci diminue la probabilité qu'une stratégie pré-pruning soit efficace (constat expérimental). Nous pouvons trouver une explication à partir de la taille des données par nœud. En effet, la structure de l'arbre et la répartition des données sont deux facteurs liés. Plus un nœud possède d'éléments, plus il va être développé en profondeur. La structure augmentera donc la chance qu'une décomposition favorable puisse exister dans un niveau encore plus bas. A titre d'exemple, dans ce cas de figure, la stratégie de pré-pruning aurait été arrêtée au niveau 1.

La figure 5.31 représente les clusters obtenus à partir de cet arbre. On y observe clairement un déséquilibre dans la répartition avec le quatrième cluster (correspondant au nœud 2). La NMSE sur ce cluster est égale à $0,007$ mais en considérant le reste des données, la contribution est de $0,0034$, en faveur de la décomposition. Pour les autres clusters le fait que le nombre de données soit petit voire même nul, fait que leurs impacts devient mineur sur la performance globale. Le modèle joue donc le rôle d'une sorte filtre anti-bruit. Nous

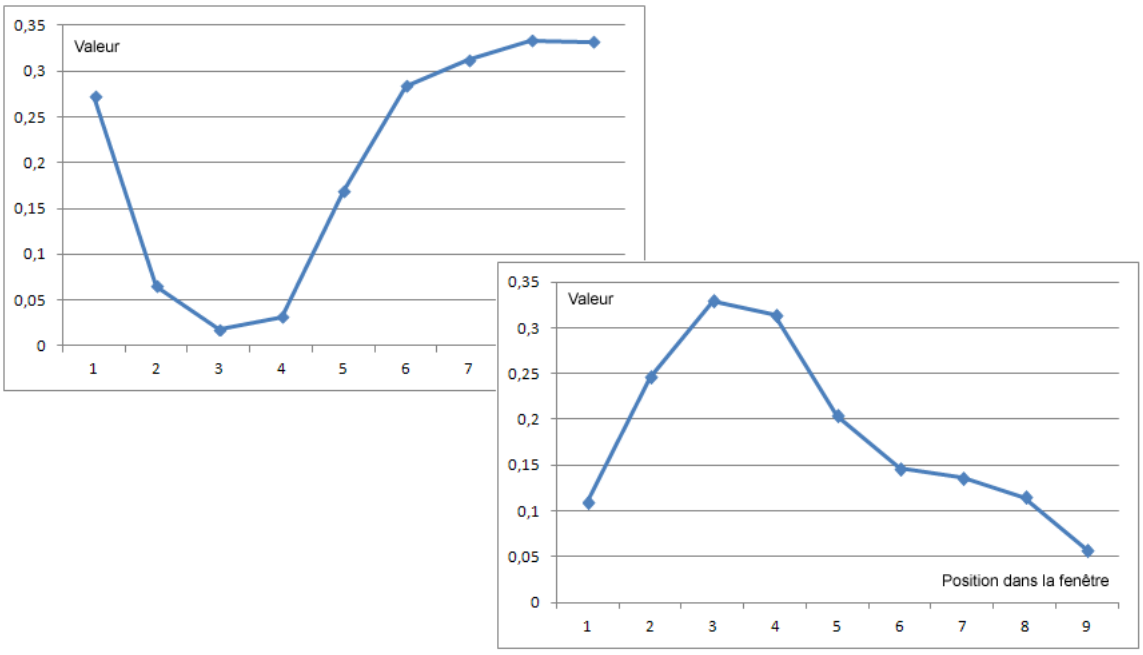


FIGURE 5.29 – Illustration des prototypes des deux classes obtenues par pre-prunning

pouvons imaginer que cette information pourra aussi dans le futur être exploitée afin de donner des valeurs de confiances aux clusters.

Nous pouvons observer dans la figure 5.32 les prototypes associés aux quatre clusters. On y remarque une similitude entre eux, encore une fois favorisant le résultat du nombre de clusters optimal égal à deux.

### La stratégie de mapping des feuilles

Dans la figure 5.33, on trouve l'arbre obtenu par la stratégie de mapping. Selon la réflexion effectuée précédemment, le risque d'incompatibilité de la stratégie pré-pruning est encore plus élevé. De plus, nous pouvons expliquer que ce genre de structure d'arbre favorise plus la stratégie de mapping. Quand l'arbre est extrêmement déséquilibré, nous obtenons fréquemment des nœuds frères avec une répartition déséquilibrée des données. Par conséquent, la chance d'avoir une disparité entre les deux nœuds frères en termes de contribution à la NMSE total augmente. Dans ce cas de figure, la stratégie de mapping devient la plus adaptée, en profitant des nœuds qui donnent les meilleures contributions et en affectant les vecteurs associés aux nœuds faibles aux parents les mieux entrainés pour que l'erreur globale ne s'affecte pas.

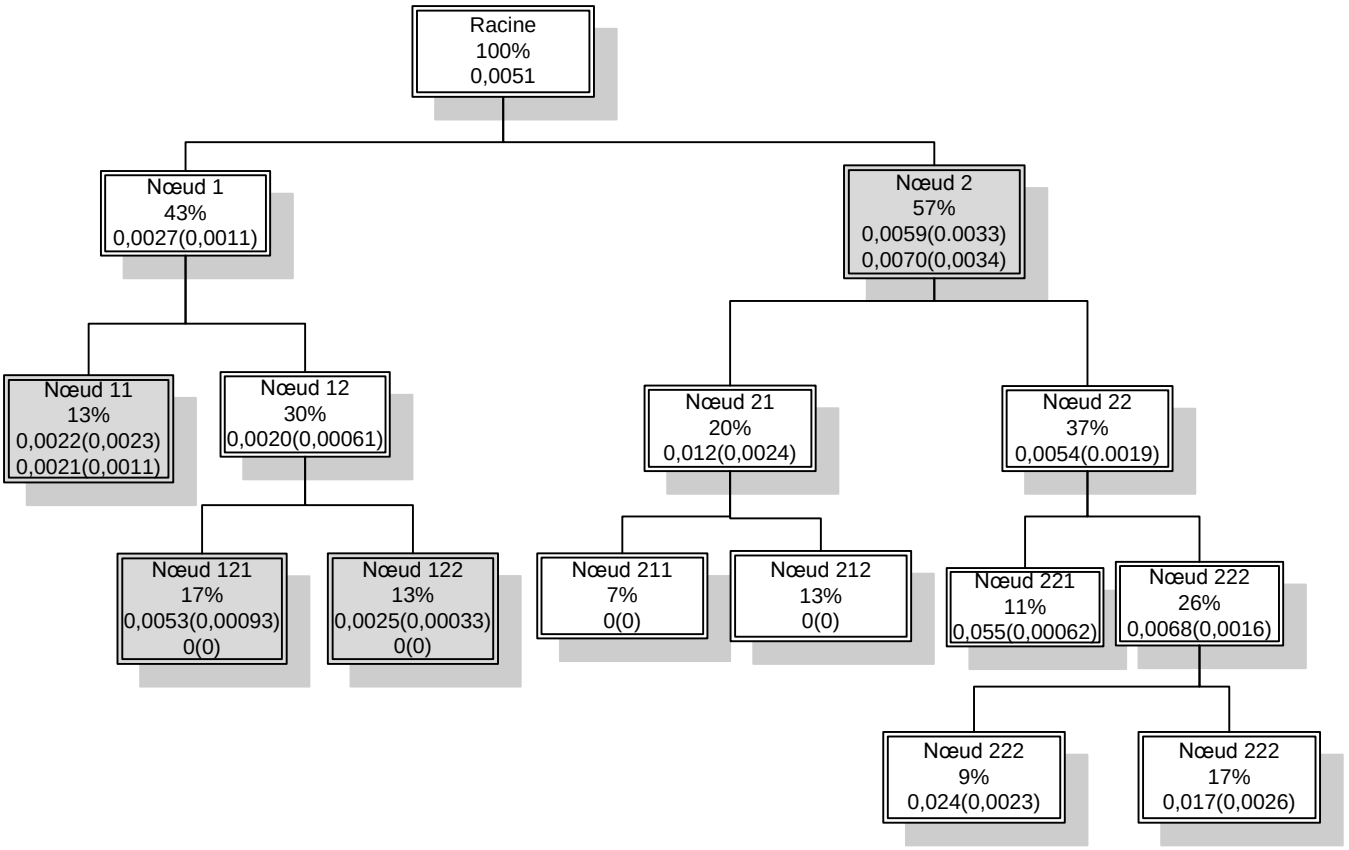


FIGURE 5.30 – Décomposition en arborescence avec la stratégie post-pruning, cas des taches solaires

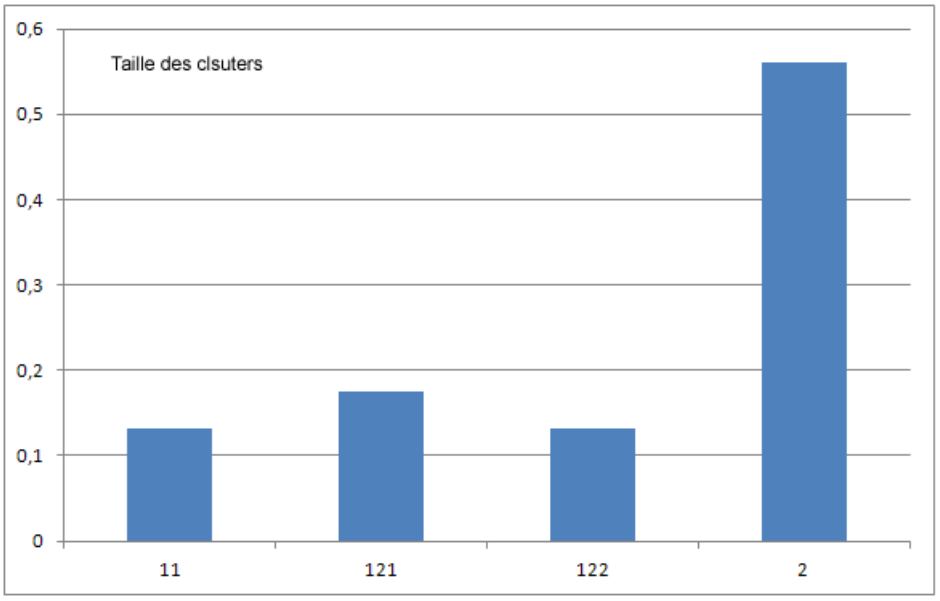


FIGURE 5.31 – Décomposition des clusters avec la stratégie de post-prunning, cas des taches solaires

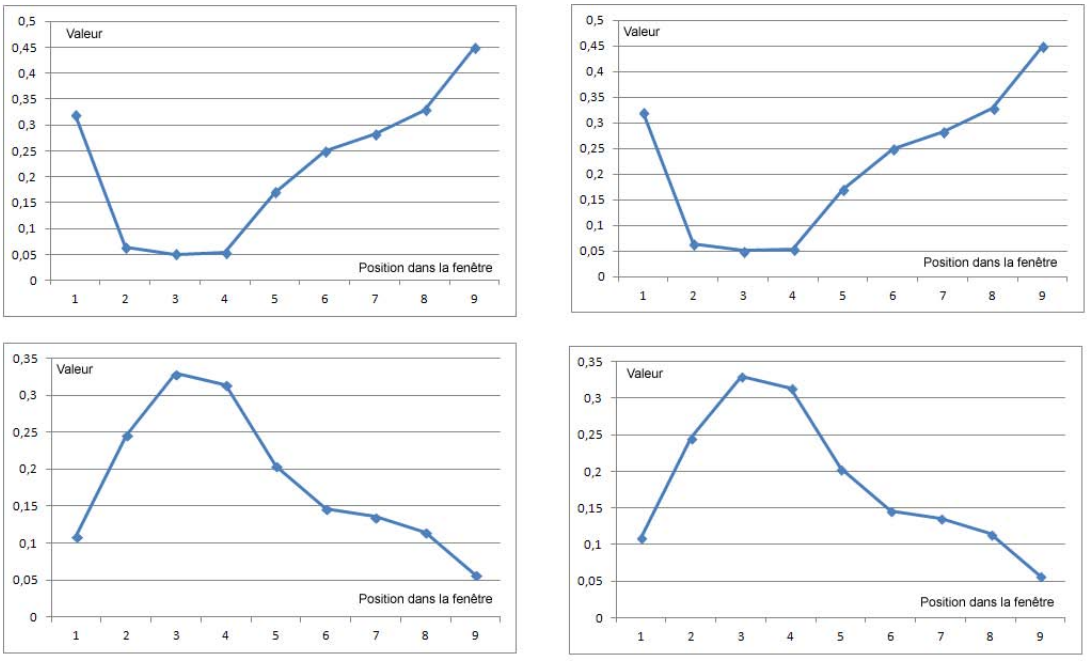


FIGURE 5.32 – Illustration des prototypes des quatre classes obtenues par post-prunning de gauche à droite et du haut en bas (11 ; 121 ; 122 ; 2)

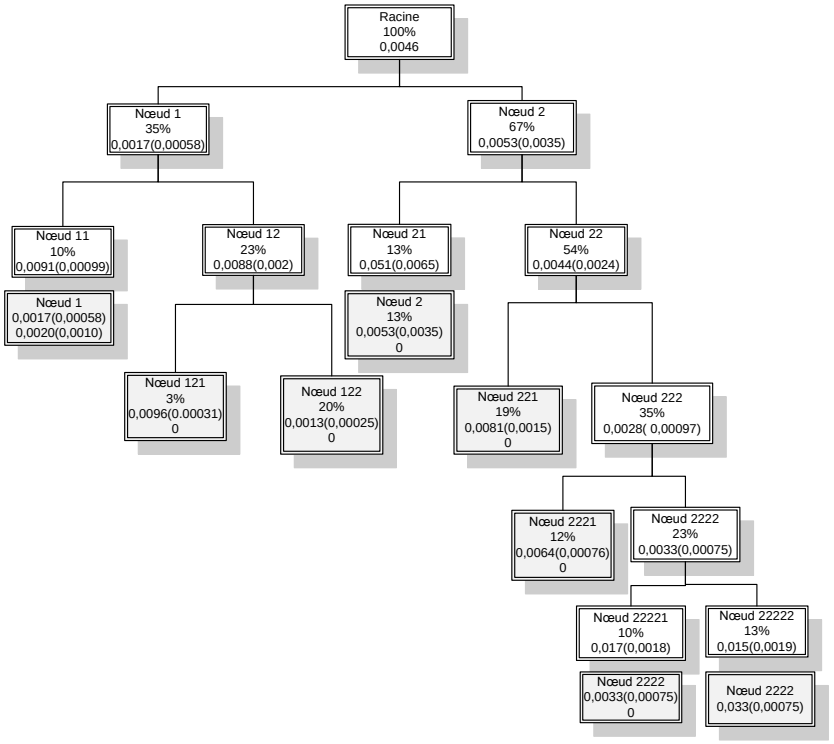


FIGURE 5.33 – Décomposition en arborescence avec la stratégie de mappage des feuilles, cas des taches solaires

### 5.5.2 Effet de la taille de la fenêtre

A présent on s'intéresse à l'effet de la taille de la fenêtre sur la performance totale du modèle. Nous effectuerons en même temps la comparaison entre les trois stratégies de pruning. La figure 5.34 nous montre l'évolution de la NMSE finale en fonction de la taille de la fenêtre. Nous trouvons en rouge la courbe pour le post-pruning, en bleu la courbe du pré-pruning et en vert celle du mapping des feuilles.

On peut déduire de cette figure que le post-pruning semble être le moins performant (la courbe rouge au dessus des deux autres dans la plus part des valeurs.). On peut aussi remarquer que le pré-pruning et le mapping sont assez proches en termes de performances et ont des réactions similaires, surtout qu'elles s'alignent sur l'optimum (pour la taille de la fenêtre) entre 8 et 9.

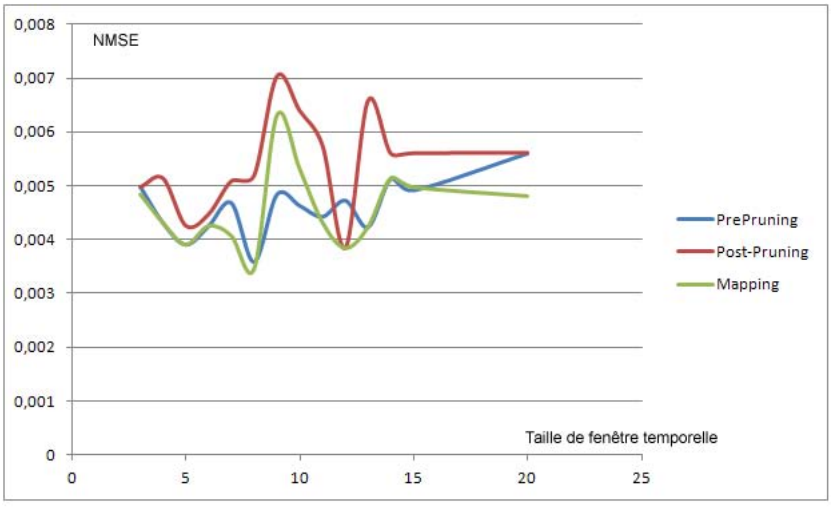


FIGURE 5.34 – comparaison des performances des trois stratégies hiérarchiques en fonction de la taille de la fenêtre temporelle

### 5.5.3 Effet de l'horizon de prévision

En ce qui concerne l'impact de l'horizon de prévision sur la performance globale de l'approche hiérarchique, nous retrouvons les mêmes constats que pour les méthodes présentées précédemment. La figure 5.35 illustre ces résultats. Nous y trouvons les trois courbes associées aux stratégies de pruning et un prédicteur MLP global.

La courbe du prédicteur global est toujours au dessus des autres, donc toutes les stratégies de pruning dépassent le modèle global pour tous les horizons de prévisions. En ce qui concerne le pré-pruning, on observe deux régimes. Pour les horizons de prévision moyens ($h < 15$) il apparait le moins performant. Par la suite, il s'aligne avec les deux autres. Dans le sens général, toutes les courbes ont tendance à augmenter, avec des vitesses différentes, ce qui traduit que la prévision devient de plus en plus difficile au fur et à mesure qu'on augmente l'horizon, avec un avantage sensible de l'approche hiérarchique en général sur

l'approche globale.

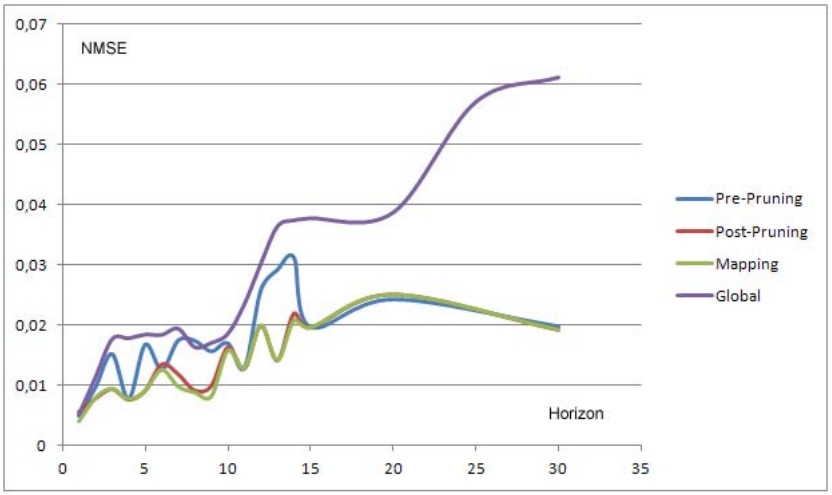


FIGURE 5.35 – Comparaison des performances des trois stratégies hiérarchiques en fonction de l'horizon de prévision

### 5.5.4 Comparaison avec l'état de l'art

Dans cette section nous effectuons une comparaison des résultats obtenus en utilisant l'approche hiérarchique et les prédicteurs à fenêtre temporelle (MLP et SVM) avec les résultats obtenus dans l'état de l'art. Le tableau 5.15 prend en compte le cas de la séries des taches solaires.

Les résultats de l'état de l'art présentés dans ce tableau sont classés dans l'ordre décroissant de la NMSE. Une description des méthodes peut être trouvée dans la section 5.3.4. Les modèles proposés intégrant l'algorithme auto-SOM et les prédicteurs à fenêtre temporelle dans l'approche locale donnent les meilleurs résultats. L'erreur NMSE est proche de celle de l'Auto-SOM. L'avantage de l'approche hiérarchique réside dans la minoration de la variance de la NMSE dans le cadre de l'approche hiérarchique.

| Modèle | NMSE Test |
|---|---|
| [Cao, 2003] | $0,1544$ |
| CPA-NN [Oh *et al.*, 2005] | 0.99 |
| MLP [Weigend *et al.*, 1992] | $0,086$ |
| RNN [Boné, 2000] | $0,084$ |
| MLP [Kouam et Fogelman, 1993] | $0,082$ |
| MLP [Czernichow, 1993] | $0,078$ |
| Boosting [Assaad *et al.*, 2008] | $0,078$ |
| SVM [Jiang *et al.*, 2007] | $0,039$ |
| auto-SOM + MLP | $0,0037$ |
| Mapping + MLP | $0,0034$ |

TABLE 5.15 – Tableau comparatif des meilleurs résultats sur la série taches solaires

Le tableau 5.16 présente un comparatif avec l'état de l'art sur la série Laser. Les méthodes sont présentées par ordre décroissant de la NMSE. Les meilleurs résultats obtenus dans l'état de l'art sont liées à l'utilisation des réseaux de neurones récurrents, d'abord en tant que prédicteur global [Boné, 2000] $(0,008)$ et ensuite en l'intégrant avec l'algorithme de Boosting [Assaad *et al.*, 2008] $(0,005)$. Les approches locales par clustering hiérarchique donnent des meilleurs résultats avec $0,0039$.

| Modèle | NMSE Test |
|---|---|
| RSOM [Koskela *et al.*, 1997] | $0,084$ |
| FIR MLP [Wan, 1993] | $0,023$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $0,011$ |
| RNN [Boné, 2000] | $0,008$ |
| Boosting [Assaad *et al.*, 2008] | $0,005$ |
| Auto-SOM+SVM | $0,0052$ |
| Pre-Pruning + MLP | $0,0039$ |

TABLE 5.16 – Tableau comparatif des meilleurs résultats sur la série Laser

Le tableau 5.17 présente un comparatif avec l'état de l'art sur la série de Mackey-Glass. Les méthodes sont présentées dans l'ordre décroissant de l'erreur NMSE. La meilleure performance récoltée dans l'état de l'art est celle de [Noelia Sánchez-Maroño *et al.*, 2003] dépassant de loin toutes les autres. L'approche hiérarchique permet d'améliorer les performances des prédicteurs avec $1,37 \times 10^{-7}$ pour le SVM local, dépassant ainsi les résultats qui se trouve dans le tableau, tout en étant proche de Auto-SOM $(1,54 \times 10^{-7})$.

L'approche que nous venons de présenter dans cette section, dépassent les résultats de l'état de l'art mais elle dépasse également les autres méthodes que nous avons présentées, en terme d'erreur de prévision NMSE.

En plus des meilleures NMSE obtenue, le tableau 5.5.4 montre aussi que l'écart type

| Modèle | NMSE Test ($10^{-3}$) |
|---|---|
| [Zemouri *et al.*, 2003] | 38 |
| [Koskela *et al.*, 1997] | $3,11$ |
| [Barreto et Araujo, 2004] | $0,75$ |
| [Vesanto, 1997] | $0,17$ |
| Boosting [Assaad *et al.*, 2008] | $0,11$ |
| [Noelia Sánchez-Maroño *et al.*, 2003] | $5,75 \times 10^{-4}$ |
| Auto-SOM+SVM | $1,54 \times 10^{-4}$ |
| Pre-Pruning + SVM | $1,37 \times 10^{-5}$ |

TABLE 5.17 – Tableau comparatif des meilleurs résultats sur la série MG-17

de la NMSE est plus faible que les autres méthodes en particulier Auto-SOM. Le succès de l'approche hiérarchique est dû au lien que nous avons établi entre le clustering et les performances des prédicteurs locaux. De plus, on remarque que l'utilisation des SVM donne des écarts types plus petits. Ce qui est logique car les paramètres du SVM sont prédéfini au préalable via un *grid search*. Les variations viennent du fait qu'on obtient des clusters différents dans les différentes instances des tests contrairement à MLP.

| Modèle | Écart type |
|---|---|
| Auto-SOM+MLP | 0,00295 |
| Auto-SOM+SVM | 0,000765 |
| Pre-Pruning+MLP | 0,0016 |
| Pre-Pruning+SVM | 0,00043 |
| post-Pruning+MLP | 0,000575 |
| post-Pruning+SVM | 0,0002255 |
| Mapping+MLP | 0,000655 |
| Mapping+SVM | 0,00065 |

TABLE 5.18 – Les écart types des différentes méthodes proposées

Nous avons établi un lien entre les structures des arbres obtenus et la stratégie de pruning adéquate. Ceci peut représenter un des axes d'amélioration de cette méthode en essayant de trouver des techniques automatiques pour dépasser les effets indésirables de l'aléatoire lorsque l'algorithme de clustering est exécuté.

## Conclusion du chapitre

Dans ce chapitre nous avons effectué des expérimentations détaillées avec les méthodes qui ont été proposées dans cette thèse. Ces expérimentations ont été menées sur des séries temporelles standard (taches solaires, Laser et MG-17). Nous avons pu établir certaines conclusions intéressantes à la fin de ces expérimentations, dont l'adéquation des prédicteurs à la fenêtre temporelle pour les approches locales par rapport aux RNN. Ainsi que la

supériorité des approches hiérarchiques sur les autres méthodes locales à clustering partitionnel. Nous dresserons, dans la conclusion, le bilan des ces conclusions et nous exposerons quelques perspectives.

# Chapitre 6

# Conclusion

Dans cette thèse nous avons présenté un panorama des méthodes de prévision des séries temporelles. Nous sommes partis des méthodes classiques de l'apprentissage automatique, pour arriver aux approches locales basées sur le clustering des données. Nous avons montré l'intérêt de l'approche locale générale par des résultats expérimentaux et nous avons apporté des explications théoriques de ces résultats. Dans un premier temps, nous avons proposé l'intégration des réseaux de neurones récurrents dans l'approche locale. Nous avons ensuite proposé deux nouvelles techniques de clustering qui visent à optimiser la structure et le nombre des clusters à obtenir. La première est une variante des cartes de Kohonen qui permet, en effectuant des itérations de l'algorithme SOM tout en apportant des modifications sur la carte, de guider l'évolution des distributions de données dans le but d'obtenir des groupements à la fois compacts et répartis dans la carte. La deuxième est un clustering hiérarchique sous forme d'arbre binaire guidé par les performances des prédicteurs eux-mêmes.

L'étude de l'état de l'art nous a permis de constater l'importance des RNN dans les problématiques de prévision. En partant de ce constat, nous souhaitons évaluer la contribution des méta-méthodes. Ce fut le cas pour la méthode du boosting avec les RNN. L'approche locale a été testée et a montré son impact positif sur la prévision en utilisant des modèles linéaires. Qu'en est-t-il pour les RNN locaux ?

Nos expérimentations avec les RNN ont été menées avec trois types d'algorithmes de clustering présents dans l'état de l'art : le SOM classique, le TKM et le RSOM. Les expérimentations ont montré que la combinaison TKM/RNN est la plus efficace pour l'approche locale. Ceci montre l'importance de la disposition des classes dans la carte. En effet, TKM se distingue par rapport aux deux autres algorithmes par ses résultats donnant un nombre de classes plus réduit en permettant l'accumulation du plus de trames de données successives dans un cluster. Ceci explique l'importance de la phase de clustering sur les performances des prédicteurs locaux. Malheureusement, et même s'il y a une amélioration sur les RNN en passant par l'approche locale, ces améliorations restent moins importantes comparées à celles des autres prédicteurs à fenêtre temporelle (SVM et MLP).

Les expérimentations qui ont été menées nous ont permis de déduire l'importance de la structure des clusters pour les résultats finaux. C'est la raison qui nous a poussés à orienter nos deux propositions vers cette problématique. L'élément qui impacte le plus la structure des clusters est leur nombre. La difficulté réside principalement dans le fait qu'il est difficile d'avoir une idée préalable sur ce nombre de clusters.

La variante Auto-SOM que nous avons proposée permet de rajouter de nouvelles unités à la carte à des endroits bien particulier, dans le but de favoriser la composition de ces groupements de données. Théoriquement, si la convergence à cet état de la carte pouvait être garantie, on pourrait espérer une plus grande efficacité de cette méthode. Malheureusement, ce ne fut pas le cas avec les séries chaotiques (Laser et MG-17). Malgré cela, nous avons remarqué que cette méthode permet d'obtenir de meilleurs résultats que ceux de l'approche locale par l'algorithme de clustering classique. Toutefois, l'écart type plus large pour Auto-SOM nous indique que la méthode Auto-SOM est plus influencée par l'effet aléatoire du clustering effectué, ce qui nous permet d'avancer l'hypothèse que lorsque le clustering est guidé par les performances des prédicteurs, les résultats sont meilleurs.

Plusieurs faiblesses ont été remarquées dans Auto-SOM, ce qui donne un point de départ pour l'amélioration de ces méthodes. Nous avons déjà cité dans le paragraphe précédent l'effet aléatoire de l'initialisation de la carte qui peut mener vers des résultats différents et parfois de mauvaises performances. Une autre limitation que nous avons décelée est la duplication des clusters. En effet, les expérimentations ont parfois montré que deux unités éloignées dans la carte et possédant deux prototypes similaires peuvent influencer négativement le résultat final. Ce problème, qui est lié à l'existence d'une topologie, est déjà présent pour l'algorithme des cartes de Kohonen. Auto-SOM dans sa version actuelle ne le traite pas. Mais la plus grande faiblesse que nous avons détectée est que les propositions faites sur l'approche locale ne prennent pas en compte les performances des prédicteurs pour générer les clusters. Nous pensons donc qu'avoir une sorte de feedback sur le clustering serait d'un impact positif sur les approches locales.

C'est sous cette dernière hypothèse que nous avons proposé la troisième approche locale nommée l'approche hiérarchique. Comme son nom l'indique, l'idée est de partir d'un nœud racine contenant la totalité des données et de les décomposer sous la forme d'un arbre binaire en prenant en compte les résultats des prédicteurs locaux sur les données de ces nœuds. L'hypothèse étant que trouver le clustering qui améliore le plus l'erreur de validation pourrait donner une indication sur les meilleurs clusters possible améliorant l'erreur de test. Pour ce faire, trois stratégies ont été proposées :

- Une stratégie de pré-pruning qui consiste à établir un critère d'arrêt au fur et à mesure que la décomposition de l'arbre est faite.
- Une stratégie de post-pruning qui permet de réaliser l'élagage de l'arbre une fois que la décomposition hiérarchique est faite.
- La troisième stratégie permet de faire un mapping entre les feuilles et le nœud de la branche le plus efficace en termes de contribution à l'erreur de prévision finale.

Les expérimentations ont montré que le pré-pruning et le mapping étaient les plus efficaces, ce qui est logique puisque que le post-pruning repose sur une méthode linéaire qui ne teste pas toutes les combinaisons possibles.

En général, l'hypothèse de la supériorité de l'approche hiérarchique a bien été validée par les expérimentations effectuées. Tout au long de cette thèse, nous avons pu explorer un axe peu développé dans la thématique des prévisions de séries temporelles. Nous avons pu prouver l'intérêt de l'approche locale qui permet aux prédicteurs simples (MLP et SVM) d'obtenir des performances dépassant même d'autres modèles plus complexes en termes de conception et de mise en œuvre.

Dans les perspectives de notre travail, plusieurs axes de recherche nous semblent prometteurs :

- **L'Amélioration de la méthode Auto-SOM** : La principale difficulté que rencontre Auto-SOM est la non convergence dans le cas des séries chaotiques. La manière avec laquelle les nouvelles unités sont insérées dans la carte peut influencer le comportement de l'algorithme et donc améliorer la convergence. Nous pouvons imaginer d'autres critères qui peuvent entrer en vigueur, telle que la similarité entre prototypes ce qui permettrait d'apporter une réponse au cas où des nœuds avec des prototypes très similaires se trouvent dans des régions éloignées de la carte. Nous pensons que d'autres idées permettraient d'étendre cette méthode à de nouvelles topologies ou encore vers d'autres domaines d'applications.
- **L'approche hiérarchique** : L'approche hiérarchique est, parmi les méthodes que nous avons présentées, celle qui a donné les meilleurs résultats. Cependant nous pensons que cette approche peut être encore améliorée. Par exemple, nous imaginons d'exploiter la structure de l'arbre obtenu pour prévoir la stratégie de pruning (ou retrouver l'ensemble des clusters les plus efficaces).
- **Neural Gas** : C'est un algorithme que nous avons présenté dans l'état de l'art et les conclusions faites sur cet algorithme étaient positives. Cela nous permettrait d'envisager dans une première phase d'utiliser cet algorithme dans la phase de clustering. Mais en plus, Neural Gas possède la particularité d'avoir des unités (ou neurones) libres de toutes topologies cela permettrait peut être d'éviter les contraintes de la topologie rencontrée dans la cadre de l'algorithme Auto-SOM (discutée dans le chapitre précédent) en adaptant la même idée ou presque pour neural gas.
- **La fenêtre temporelle** : Les tests effectués tout au long de cette thèse ont permis de conclure que le choix de la fenêtre temporelle permet d'avoir un impact positif ou négatif sur les résultats des prédicteurs. Dans nos tests, nous avons pris la même taille de la fenêtre temporelle que pour l'algorithme de clustering et de prédicteur local. Si on est obligé, à priori, d'avoir la même fenêtre temporelle pour les prototypes des clusters (vu la nature des algorithmes de quantification vectorielle utilisés), qu'en est il pour les prédicteurs ? Nous pouvons de manière anticipée envisager la possibilité de sélectionner les éléments de la fenêtre temporelle qui seront effectifs à la prévision par clusters, on peut même étendre ce raisonnement à des valeurs en dehors de la fenêtre comme on peut aussi inclure des variables exogènes.

- **Des applications en situation réelle** de nos algorithmes sont en cours notamment dans le domaine de la finance et du transport. Plusieurs nouvelles difficultés peuvent apparaitre. Par exemple, la compréhension des phénomènes qui génèrent les séries temporelles nécessite des avis d'experts. Ou encore la difficulté d'effectuer des benchmarks car les données utilisées sont souvent sous formes brutes mesurées de différentes façon ou se voient appliquer des traitements différents d'un spécialiste à un autre. Toutefois, les expériences effectuées au long de cette thèse et les connaissances et expertises accumulées et surtout les bons résultats obtenus nous encouragent à passer vers une phase d'application réelle.
- **Généralisation vers un système d'apprentissage** : Nous avons déjà évoqué la possibilité de la généralisation du système de prévision qui s'est constitué tout au long de cette thèse. Nous nous sommes orientés vers la problématique de prévision de séries temporelles. Mais le même esprit de travail est toujours possible dans des problématiques de classification. Comment ce système peut être modifié pour attaquer des problématiques de classification (qu'elles soient supervisées ou non) ? Et quelles sont les autres informations qu'on peut générer avec ce système ? Par exemple nous pouvons penser à un taux de confiance. L'approche locale semble être adéquate dans un monde où aujourd'hui tout est informatisé, tout se connecte et tout se partage. L'information est devenue riche et abondante et l'enjeu réside sur comment exploiter cette information. De nombreuses applications peuvent être exploitées comme par exemple dans les réseaux sociaux.

# Bibliographie


[Aburto et Weber, 2007] Aburto, L. et Weber, R. (2007). Improved supply chain management based on hybrid demand forecasts. *Applied Soft Computing*, 7(1):136–144.

[Akaike, 1978] Akaike, H. (1978). On the Likelihood of a Time Series Model. *The Statistician*, 27(3):217–235.

[Anand, 2009] Anand, C. N. (2009). *Internet traffic modeling and forecasting using nonlinear time series model GARCH*. Thèse de doctorat.

[Ao et Palade, 2011] Ao, S. et Palade, V. (2011). Ensemble of Elman neural networks and support vector machines for reverse engineering of gene regulatory networks. *Applied Soft Computing*, 11(2):1718–1726.

[Assaad *et al.*, 2008] Assaad, M., Bone, R. et Cardot, H. (2008). A new boosting algorithm for improved time-series forecasting with recurrent neural networks. *Information Fusion*, 9(1):41–55.

[Atiya *et al.*, 1999] Atiya, A. F., El-Shoura, S. M., Shaheen, S. I. et El-Sherif, M. S. (1999). A comparison between neural-network forecasting techniques–case study : river flow forecasting. *IEEE Transactions on Neural Networks*, 10(2):402–409.

[Aussem *et al.*, 1995] Aussem, A., Murtagh, F. et Sarazin, M. (1995). Dynamical Recurrent Neural Networks : Towards Environmental Time Series Prediction.

[Back *et al.*, 1995] Back, A., Wan, E. et Lawrence, S. (1995). A unifying view of some training algorithms for multilayer perceptrons with FIR filter synapses. *In Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 146–154. IEEE.

[Back et Tsoi, 1991] Back, A. D. et Tsoi, A. C. (1991). FIR and IIR Synapses, a New Neural Network Architecture for Time Series Modeling. *Neural Computation*, 3(3):375–385.

[Back et Tsoi, 1994] Back, A. D. et Tsoi, A. C. (1994). Alternative discrete-time operators in neural networks for nonlinear prediction. *Proceedings of ANZIIS 94 Australian New Zealnd Intelligent Information Systems Conference*, pages 14–17.

[Barreto et Araujo, 2004] Barreto, G. a. et Araujo, a. R. (2004). Identification and control of dynamical systems using the self-organizing map. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 15(5):1244–59.

[Barreto *et al.*, 2004] Barreto, G. A., Mota, J. a. C. M., Souza, L. G. M. et Frota, R. A. (2004). Nonstationary Time Series Prediction Using Local Models Based on Competitive Neural Networks. *Lecture Notes in Computer Science*, 3029.

[Bellman et Kalaba, 1959] Bellman, R. et Kalaba, R. (1959). On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9.

[Bengio *et al.*, 1993] Bengio, Y., Frasconi, P. et Simard, P. (1993). The problem of learning long-term dependencies in recurrent networks. *In IEEE International Conference on Neural Networks*, pages 1183–1188. IEEE.

[Benoudjit et Verleysen, 2003] Benoudjit, N. et Verleysen, M. (2003). On the Kernel Widths in Radial-Basis Function Networks. *Neural Process. Lett.*, 18(2):139–154.

[Bertschinger et Natschläger, 2004] Bertschinger, N. et Natschläger, T. (2004). Real-time computation at the edge of chaos in recurrent neural networks. *Neural computation*, 16(7):1413–36.

[Bishop, 1995] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*, volume 4 de *Information science and statistics*. Springer.

[Bollerslev, 1986] Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327.

[Boné, 2000] Boné, R. (2000). *Recurrent Neural Networks for Time Series Prediction*. Thèse de doctorat.

[Bone *et al.*, 2000] Bone, R., Crucianu, M. et Asselin de Beauville, J.-P. (2000). Two constructive algorithms for improved time series processing with recurrent neural networks. *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*, pages 55–64.

[Boné *et al.*, 2002] Boné, R., Crucianu, M. et Asselin de Beauville, J.-P. (2002). Learning long-term dependencies by the selective addition of time-delayed connections to recurrent neural networks. *Neurocomputing*, 48(1-4):251–266.

[Boser *et al.*, 1992] Boser, B. E., Guyon, I. M. et Vapnik, V. N. (1992). *A training algorithm for optimal margin classifiers*. ACM Press, New York, New York, USA.

[Box *et al.*, 1994] Box, G. E. P., Jenkins, G. M. et Reinsel, G. C. (1994). *Time Series Analysis : Forecasting and Control (Wiley Series in Probability and Statistics)*. Wiley.

[Breiman, 1996a] Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Breiman, 1996b] Breiman, L. (1996b). Stacked regressions. *Machine Learning*, 24(1):49–64.

[Burges, 1998] Burges, C. J. C. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167.

[Cao, 2003] Cao, L. (2003). Support vector machines experts for time series forecasting. *Neurocomputing*, 51:321–339.

[Cao *et al.*, 2000] Cao, L., Tay, F., Leung, K., Chan, L.-W. et Meng, H. (2000). $\epsilon$-Descending Support Vector Machines for Financial Time Series Forecasting. 1983(Intelligent Data Engineering and Automated Learning - IDEAL 2000. Data Mining, Financial Engineering, and Intelligent Agents):405–424.

[Cao et Tay, 2003] Cao, L. J. et Tay, F. H. (2003). Support vector machine with adaptive parameters in financial time series forecasting. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 14(6):1506–18.

[Casdagli, 1989] Casdagli, M. (1989). Nonlinear prediction of chaotic time series. *Physica D : Nonlinear Phenomena*, 35(3):335–356.

[Casey, 1996] Casey, M. (1996). The Dynamics of Discrete-Time Computation, with Application to Recurrent Neural Networks and Finite State Machine Extraction. *Neural Computation*, 8(6):1135–1178.

[Castillo *et al.*, 1998] Castillo, E., Cobo, A., Gutierrez, J. A. et Pruneda, R. E. (1998). *Functional Networks With Applications : A Neural-Based Paradigm*. Kluwer Academic Publishers.

[Chakrabarti *et al.*, 2006] Chakrabarti, D., Kumar, R. et Tomkins, A. (2006). Evolutionary clustering. *Proceedings of the 12th ACM*.

[Chakraborty *et al.*, 1992] Chakraborty, K., Mehrotra, K., Mohan, C. K. et Ranka, S. (1992). Forecasting the behavior of multivariate time series using neural networks. *Neural Networks*, 5(6):961–970.

[Chang *et al.*, 2001] Chang, M.-W., Chen, B.-J. et Lin, C.-J. (2001). EUNITE Network Competition : Electricity Load Forecasting.

[Chappell et Taylor, 1993] Chappell, G. J. et Taylor, J. G. (1993). The temporal Kohonen map. *Neural Networks*, 6(3):441–445.

[Chen *et al.*, 2001] Chen, H., Grant-Muller, S., Mussone, L. et Montgomery, F. (2001). A study of hybrid neural network approaches and the effects of missing data on traffic forecasting. *Neural Computing \& Applications*, 10(3):277–286.

[Cheong, 2009] Cheong, C. W. (2009). Modeling and forecasting crude oil markets using ARCH-type models. *Energy Policy*, 37(6):2346–2355.

[Cigizoglu, 2004] Cigizoglu, H. (2004). Estimation and forecasting of daily suspended sediment data by multi-layer perceptrons. *Advances in Water Resources*, 27(2):185–195.

[Connor et Atlas, 1991] Connor, J. et Atlas, L. (1991). Recurrent neural networks and time series prediction. *Neural Networks, IJCNN-91-Seattle International Joint Conference on*, pages 301–306.

[Connor *et al.*, 1994] Connor, J. T., Martin, R. D. et Atlas, L. E. (1994). Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(2):240–54.

[Cottrell, 1998] Cottrell, M. (1998). Theoretical aspects of the SOM algorithm. *Neurocomputing*, 21(1-3):119–138.

[Cottrell *et al.*, 1995] Cottrell, M., Girard, B., Girard, Y., Mangeas, M. et Muller, C. (1995). Neural modeling for time series : A statistical stepwise method for weight elimination. *IEEE Transactions on Neural Networks*, 6(6):1355–1364.

[Crucianu, 1997] Crucianu, M. (1997). Algorithmes d'évolution pour les réseaux de neurones. Rapport technique.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics Of Control Signals And Systems*, 2(4):303–314.

[Czernichow, 1993] Czernichow, T. (1993). *Apport des réseaux récurrents à la prévision de séries temporelles, applications à la prévision de consommation d'électricité.* Thèse de doctorat, Université Paris 6, Paris.

[De Almeida et Fishwick, 1991] De Almeida, C. et Fishwick, P. A. (1991). Time series forecasting using neural networks vs. Box- Jenkins methodology. *Simulation*, 57(5):303–310.

[Dittenbach *et al.*, 2000] Dittenbach, M., Merkl, D. et Rauber, A. (2000). The growing hierarchical self-organizing map. *Neural Networks 2000 IJCNN 2000 Proceedings of the IEEEINNSENNS International Joint Conference on*, 6(6):15–19.

[Drucker, 1999] Drucker, H. (1999). Boosting using neural networks. *In* Sharkey, A. J. C., éditeur : *Combining Artificial Neural Nets*, pages 51–78. Springer-Verlag.

[Drucker et Branch, 1997] Drucker, H. et Branch, W. L. (1997). Improving Regressors using Boosting Techniques. *Proc 14th International Conference on Machine Learning*, pages 107–115.

[Drucker *et al.*, 1997] Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. et Vapnik, V. (1997). Support vector regression machines. *Advances in Neural Information Processing Systems 9*, 1(June):155–161.

[Dudul, 2005] Dudul, S. (2005). Prediction of a Lorenz chaotic attractor using two-layer perceptron neural network. *Applied Soft Computing*, 5(4):333–355.

[Duro et Reyes, 1999] Duro, R. J. et Reyes, J. S. (1999). Discrete-time backpropagation for training synaptic delay-based artificial neural networks. *IEEE Transactions on Neural Networks*, 10(4):779–789.

[Şeker *et al.*, 2003] Şeker, S., Ayaz, E. et Türkcan, E. (2003). Elman's recurrent neural network applications to condition monitoring in nuclear power plant and rotating machinery. *Engineering Applications of Artificial Intelligence*, 16(7-8):647–656.

[Ekici *et al.*, 2009] Ekici, S., Yildirim, S. et Poyraz, M. (2009). A transmission line fault locator based on Elman recurrent networks. *Applied Soft Computing*, 9(1):341–347.

[Elman, 1989] Elman, J. (1989). Representation and Structure in Connectionist Models. Rapport technique 8903, University of California, San Diego.

[Engle, 1982] Engle, R. F. (1982). Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica*, 50(4):987 – 1007.

[Euliano *et al.*, 1996] Euliano, N., Principe, J. et Kulzer, P. (1996). A self-organizing temporal pattern recognizer with application to robot landmark recognition. *Sintra Spatiotemporal Models in Biological and Artificial Systems Workshop.*

[Eulianoa et Principe, 1999] Eulianoa, N. et Principe, J. (1999). A spatio-temporal memory based on SOMs with activity diffusion. *Kohonen maps*, page 253.

[Farmer et Sidorowich, 1988] Farmer, J. et Sidorowich, J. (1988). Exploiting chaos to predict the future and reduce noise.

[Flanagan, 1998] Flanagan, J. A. (1998). Sufficient conditions for self-organization in the one-dimensional SOM with a reduced width neighbourhood. *Neurocomputing*, 21(1–3):51–60.

[Frasconi *et al.*, 1992] Frasconi, P., Gori, M. et Soda, G. (1992). Local Feedback Multilayered Networks. *Neural Computation*, 4(1):120–130.

[Freund et Schapire, 1996] Freund, Y. et Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. *Update*, pages:148–156.

[Fu *et al.*, 2001] Fu, T.-c., Chung, F.-l., Ng, V. et Luk, R. (2001). Pattern Discovery from Stock Time Series Using Self-Organizing Maps. *Workshop Notes of KDD2001 Workshop on Temporal Data Mining*, pages 26–29.

[Funahashi et Nakamura, 1993] Funahashi, K.-i. et Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806.

[Gama, 1998] Gama, J. (1998). Combining Classifiers by Constructive Induction. *ICML '97 Proceedings of the Fourteenth International Conference on Machine Learning*, pages 178–189.

[Geman et Bienenstock, 1992] Geman, S. et Bienenstock, E. (1992). Neural networks and the bias/variance dilemma. *Neural computation*.

[Gencay, 1994] Gencay, R. (1994). Nonlinear prediction of noisy time series with feedforward networks. *Physics Letters A*, 187(5-6):397–403.

[Gersho et Gray, 1992] Gersho, A. et Gray, R. M. (1992). *Vector Quantization and Signal Compression*, volume 0 de *The Kluwer international series in engineering and computer science : Communications and information theory : Kluwer international series in engineering and computer science : Kluwer international series in engineering and computer science*. Kluwer.

[Gonçalves *et al.*, 2010] Gonçalves, L. F., Schneider, E. L., Henriques, R. V. B., Lubaszewski, M., Bosa, J. L. et Engel, P. M. (2010). Fault prediction in electrical valves using temporal Kohonen maps. *(LATW), 2010 11th*, pages 1–6.

[Gonzalez Mromera *et al.*, 2008] Gonzalez Mromera, E., Jaramillo Moran, M. et Carmona Fernadez, D. (2008). Monthly electric energy demand forecasting with neural networks and Fourier series. *Energy Conversion and Management*, 49(11):3135–3142.

[Grossberg, 1987] Grossberg, S. (1987). Competitive learning : From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23–63.

[Guler *et al.*, 2005] Guler, N., Ubeyli, E. et Guler, I. (2005). Recurrent neural networks employing Lyapunov exponents for EEG signals classification. *Expert Systems with Applications*, 29(3):506–514.

[Guyon *et al.*, 2006] Guyon, I., Gunn, S., Nikravesh, M. et Zadeh, L. (2006). *Feature Extraction, Foundations and Applications*, volume 207 de *Studies in Fuzziness and Soft Computing*. Springer.

[Guyon *et al.*, 1992] Guyon, I., Vapnik, V., Boser, B., Bottou, L. et Solla, S. A. (1992). Structural risk minimization for character recognition. *In* Moody, J. E., Hanson, S. J. et Lippmann, R., éditeurs : *Advances in Neural Information Processing Systems 4*, volume 4, pages 471–479. Morgan Kaufmann.

[Hagenbuchner, 2003] Hagenbuchner, M. (2003). A self-organizing map for adaptive processing of structured data.

[Hakim *et al.*, 1991] Hakim, N., Kaufmann, J., Cerf, G. et Meadows, H. (1991). Nonlinear time series prediction with a discrete-time recurrent neural network model. *In IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, page 900. IEEE.

[Hamilton, 1994] Hamilton, J. D. (1994). *Time Series Analysis*, volume 3. Princeton Univ Pr.

[Han *et al.*, 2004] Han, M., Xi, J., Xu, S. et Yin, F.-L. (2004). Prediction of Chaotic Time Series Based on the Recurrent Predictor Neural Network. *IEEE Transactions on Signal Processing*, 52(12):3409–3416.

[Harpham et Dawson, 2006] Harpham, C. et Dawson, C. (2006). The effect of different basis functions on a radial basis function network for time series prediction : A comparative study. *Neurocomputing*, 69(16-18):2161–2170.

[Haykin, 1999] Haykin, S. (1999). *Neural Networks : A Comprehensive Foundation*. Prentice Hall.

[Herrera *et al.*, 2007] Herrera, L., Pomares, H., Rojas, I., Guillén, A., Prieto, A. et Valenzuela, O. (2007). Recursive prediction for long term time series forecasting using advanced models. *Neurocomputing*, 70(16-18):2870–2880.

[Hihi, 1996] Hihi, S. E. (1996). Hierarchical recurrent neural networks for long-term dependencies. *Neural Computation*.

[Hill *et al.*, 1996] Hill, T., O'Connor, M. et Remus, W. (1996). Neural Network Models for Time Series Forecasts. *Management Science*, 42(7):1082–1092.

[Ho *et al.*, 2002] Ho, S., Xie, M. et Goh, T. (2002). A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction. *Computers & Industrial Engineering*, 42(2-4):371–375.

[Hoptroff, 1993] Hoptroff, R. G. (1993). The principles and practice of time series forecasting and business modelling using neural nets. *Neural Computing & Applications*, 1(1):59–66.

[Horowitz et Alvarez, 1995] Horowitz, R. et Alvarez, L. (1995). Convergence properties of -organizing neural networks. *In Proceedings of the 1995 American Control Conference*, volume 2, pages 1339–1344. American Autom Control Council.

[Huerta, 2003] Huerta, G. (2003). Time series modeling via hierarchical mixtures. *Statistica Sinica*, 13:1097–1118.

[Hyndman *et al.*, 2011] Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G. et Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics & Data Analysis*, 55(9):2579–2589.

[Jacobs *et al.*, 1991] Jacobs, R. a., Jordan, M. I., Nowlan, S. J. et Hinton, G. E. (1991). Adaptive Mixtures of Local Experts.

[Jaeger, 2001] Jaeger, H. (2001). The "echo state" approach to analysing and training recurrent neural networks. *submitted for publication*, 148(GMD Report 148):43.

[James, 1995] James, D. (1995). SARDNET : A self-organizing feature map for sequences. *Advances in neural information processing*, pages 1–9.

[Jiang *et al.*, 2008] Jiang, F., Berry, H. et Schoenauer, M. (2008). *Supervised and Evolutionary Learning of Echo State Networks*, volume 5199 de *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Jiang *et al.*, 2007] Jiang, T., Wang, S. et Wei, R. (2007). Support vector machine with composite kernels for time series prediction. *Advances in Neural Networks–ISNN 2007*, pages 350–356.

[Jordan et Jacobs, 1994] Jordan, M. et Jacobs, R. (1994). Hierarchical mixtures of experts and the EM algorithm. *Neural computation*, 6(2):181–214.

[Jordan, 1986] Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. *In Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546. IEEE Press.

[Kaastra et Boyd, 1996] Kaastra, I. et Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, 10(3):215–236.

[Karunasinghe et Liong, 2006] Karunasinghe, D. S. et Liong, S.-Y. (2006). Chaotic time series prediction with a global model : Artificial neural network. *Journal of Hydrology*, 323(1-4):92–105.

[Kassam, 1993] Kassam, S. (1993). Radial basis function networks in nonlinear signal processing applications. *In Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 1021–1025. IEEE Comput. Soc. Press.

[Kaufman et Rousseeuw, 1990] Kaufman, L. et Rousseeuw, P. J. (1990). *Finding Groups in Data : An Introduction to Cluster Analysis*, volume 39 de *Wiley Series in Probability and Mathematical Statistics*. John Wiley & Sons.

[Kelo et Dudul, 2012] Kelo, S. et Dudul, S. (2012). A wavelet Elman neural network for short-term electrical load prediction under the influence of temperature. *International Journal of Electrical Power & Energy Systems*, 43(1):1063–1071.

[Khashei et Bijari, 2012] Khashei, M. et Bijari, M. (2012). A new class of hybrid models for time series forecasting. *Expert Systems with Applications*, 39(4):4344–4357.

[Khashei *et al.*, 2009] Khashei, M., Bijari, M. et Raissi Ardali, G. A. (2009). Improvement of Auto-Regressive Integrated Moving Average models using Fuzzy logic and Artificial Neural Networks (ANNs). *Neurocomputing*, 72(4-6):956–967.

[Koenig et Youn, 2011] Koenig, L. et Youn, E. (2011). Hierarchical signature clustering for time series microarray data. *Advances in experimental medicine and biology*, 696:57–65.

[Kohonen, 1982] Kohonen, T. (1982). Analysis of a simple self-organizing process. *Biological cybernetics*, 44(2):135–140.

[Köker, 2005] Köker, R. (2005). Reliability-based approach to the inverse kinematics solution of robots using Elman's networks. *Engineering Applications of Artificial Intelligence*, 18(6):685–693.

[Koskela *et al.*, 1997] Koskela, T., Varsta, M., Heikkonen, J. et Kaski, K. (1997). Time Series Prediction Using Recurrent SOM with Local Linear Models. *INT. J. OF KNOWLEDGE-BASED INTELLIGENT ENGINEERING SYSTEMS*, 2:60 – 68.

[Kouam et Fogelman, 1993] Kouam, A. et Fogelman, F. (1993). *Connectionist approches for time series forecasting*. Thèse de doctorat.

[Kulzer, 1996] Kulzer, P. (1996). NAVBOT – Autonomous robotic agent with neural network learning of autonomous mapping and navigation strategies. Rapport technique, University of Aveiro.

[Lapedes et Farber, 1987] Lapedes, A. et Farber, R. (1987). Nonlinear signal processing using neural networks : Prediction and system modelling.

[Leung et Yuen, 2002] Leung, A. et Yuen, R. (2002). Air pollutant parameter forecasting using support vector machines. *In Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02 (Cat. No.02CH37290)*, pages 630–635. IEEE.

[Liang et Sun, 2003] Liang, Y. et Sun, Y. (2003). An improved method of support vector machine and its applications to financial time series forecasting. *Progress in Natural Science*, 13(9):696–700.

[Lin et Chen, 2005] Lin, G.-F. et Chen, L.-H. (2005). Time series forecasting by combining the radial basis function network and the self-organizing map. *Hydrological Processes*, 19(10):1925–1937.

[Lin *et al.*, 1998] Lin, T., Horne, B., Tino, P. et Giles, C. (1998). Learning long-term dependencies is not as difficult with NARX recurrent neural networks. *Time*, pages 1–23.

[Lin *et al.*, 1996] Lin, T., Horne, B. G., Tino, P. et Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 7(6):1329–38.

[Liou *et al.*, 2008] Liou, C.-Y., Huang, J.-C. et Yang, W.-C. (2008). Modeling word perception using the Elman network. *Neurocomputing*, 71(16-18):3150–3157.

[Mackey et Glass, 1977] Mackey, M. C. et Glass, L. (1977). Oscillation and chaos in physiological control systems. *Science (New York, N.Y.)*, 197(4300):287–9.

[Mangasarian, 2002] Mangasarian, O. (2002). Large scale kernel regression via linear programming. *Machine Learning*.

[Martin-Merino et Roman, 2006] Martin-Merino, M. et Roman, J. (2006). A New SOM Algorithm for Electricity Load Forecasting. *In Neural Information Processing*, pages 995–1003. Springer.

[Martinetz *et al.*, 1993] Martinetz, T. M., Berkovich, S. G. et Schulten, K. J. (1993). Neural-gas' network for vector quantization and its application to time-series prediction. *IEEE Transactions on Neural Networks*, 4(4):558–569.

[McDonnell et Waagen, 1994] McDonnell, J. R. et Waagen, D. (1994). Evolving recurrent perceptrons for time-series modeling. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(1):24–38.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*, volume 4 de *McGraw-Hill Series in Computer Science*. McGraw-Hill.

[Mizushima, 2006] Mizushima, F. (2006). Language learnability by feedback self-organizing maps. *Neural Information Processing*, pages 228–236.

[Moody, 1995] Moody, J. (1995). Economic forecasting : Challenges and neural network solutions. *In Proceedings of the International Symposium on Artificial Neural Networks*, numéro December, pages 1–8.

[Mori et Urano, 1996] Mori, H. et Urano, S. (1996). Short-term load forecasting with chaos time series analysis. *In Proceedings of International Conference on Intelligent System Application to Power Systems*, pages Intelligent Systems Applications to Power Systems,. IEEE.

[Müller *et al.*, 1997] Müller, K., Smola, A. J., Rätsch, G., Schölkopf, B., Kohlmorgen, J. et Vapnik, V. (1997). Predicting Time Series with Support Vector Machines. *Artificial Neural Networks ICANN97*, 1327(1):999–1004.

[Nilsson, 2004] Nilsson, N. J. (2004). *Introduction to Machine Learning*, volume 56. The MIT Press.

[Niska, 2004] Niska, H. (2004). Evolving the neural network model for forecasting air pollution time series. *Engineering Applications of Artificial Intelligence*, 17(2):159–167.

[Niska *et al.*, 2005] Niska, H., Rantamäki, M., Hiltunen, T., Karppinen, A., Kukkonen, J., Ruuskanen, J. et Kolehmainen, M. (2005). Evaluation of an integrated modelling system containing a multi-layer perceptron model and the numerical weather prediction model HIRLAM for the forecasting of urban airborne pollutant concentrations. *Atmospheric Environment*, 39(35):6524–6536.

[Noelia Sánchez-Maroño *et al.*, 2003] Noelia Sánchez-Maroño, Fontenla-Romero, O., Alonso-Betanzos, A. et Guijarro-Berdiñas, B. (2003). Self-organizing maps and functional networks for local dynamic modeling. *In Proceedings of the European Symposium on Artificial Neural Networks (ESANN'95)*, pages 39 – 44.

[Oh *et al.*, 2005] Oh, S., Kim, M., Eom, T. et Lee, J. (2005). Heterogeneous local model networks for time series prediction. *Applied Mathematics and Computation*, 168(1):164–177.

[Ohta et Gunji, 2006] Ohta, H. et Gunji, Y. P. (2006). Recurrent neural network architecture with pre-synaptic inhibition for incremental learning. *Neural networks : the official journal of the International Neural Network Society*, 19(8):1106–19.

[Orr, 1998] Orr, M. (1998). Optimising the Widths of Radial Basis Functions. *Proceedings of the Fifth Brazilian Symposium on Neural Networks*.

[Osuna *et al.*, 1997] Osuna, E., Freund, R. et Girosi, F. (1997). *An improved training algorithm for support vector machines*. IEEE.

[Pai et Hong, 2005] Pai, P.-F. et Hong, W.-C. (2005). Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms. *Electric Power Systems Research*, 74(3):417–425.

[Palmer *et al.*, 2006] Palmer, A., Josemontano, J. et Sese, A. (2006). Designing an artificial neural network for forecasting tourism time series. *Tourism Management*, 27(5): 781–790.

[Pampalk *et al.*, 2003] Pampalk, E., Widmer, G. et Chan, A. (2003). A New Approach to Hierarchical Clustering and Structuring of Data with Self-Organizing Maps Self-Organizing Maps. *Intelligent Data Analysis*, 8(2):1–23.

[Parker et Chua, 1989] Parker, T. S. et Chua, L. O. (1989). Practical numerical algorithms for chaotic systems.

[Pham et Karaboga, 1999] Pham, D. et Karaboga, D. (1999). Training Elman and Jordan networks for system identification using genetic algorithms. *Artificial Intelligence in Engineering*, 13(2):107–117.

[Pi et Peterson, 1994] Pi, H. et Peterson, C. (1994). Finding the embedding dimension and variable dependencies in time series. *Neural Computation*, 6(3):509–520.

[Poddar et Unnikrishnan, 1991] Poddar, P. et Unnikrishnan, K. (1991). Nonlinear prediction of speech signals using memory neuron networks. *In Neural Networks for Signal Processing Proceedings of the 1991 IEEE Workshop*, pages 395–404. IEEE.

[Prado *et al.*, 2006] Prado, R., Molina, F. et Huerta, G. (2006). Multivariate time series modeling and classification via hierarchical VAR mixtures. *Computational Statistics & Data Analysis*, 51(3):1445–1462.

[Press *et al.*, 1992] Press, W. H., Flannery, B. P., Teukolsky, S. A. et Vetterling, W. T. (1992). *Numerical Recipes in C : The Art of Scientific Computing, Second Edition.* Cambridge University Press.

[Rand et Young, 1981] Rand, D. et Young, L.-S., éditeurs (1981). *Detecting strange attractors in turbulence*, volume 898 de *Lecture Notes in Mathematics*. Springer Berlin Heidelberg.

[Rao et Gabr, 1984] Rao, T. et Gabr, M. (1984). *An Introduction to Bispectral Analysis and Bilinear Time Series Models (Lecture Notes in Statistics)*. Springer.

[Rivas *et al.*, 2004] Rivas, V., Merelo, J., Castillo, P., Arenas, M. et Castellano, J. (2004). Evolving RBF neural networks for time-series forecasting with EvRBF. *Information Sciences*, 165(3-4):207–220.

[Rubio *et al.*, 2011] Rubio, G., Pomares, H., Rojas, I. et Herrera, L. J. (2011). A heuristic method for parameter selection in LS-SVM : Application to time series prediction. *International Journal of Forecasting*, 27(3):725–739.

[Rumelhart *et al.*, 1986] Rumelhart, D. E., Hinton, G. E. et Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel distributed processing : explorations in the microstructure of cognition*, 1:318–362.

[Ruwisch *et al.*, 1997] Ruwisch, D., Dobrzewski, B. et Bode, M. (1997). Wave propagation as a neural coupling mechanism : hardware for self-organizing feature maps and the representation of temporal sequences. *In Neural Networks for Signal Processing VII. Proceedings of the 1997 IEEE Signal Processing Society Workshop*, pages 306–315. IEEE.

[Saad *et al.*, 1998] Saad, E. W., Prokhorov, D. V. et Wunsch, D. C. (1998). Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 9(6):1456–70.

[Sansom *et al.*, 1998] Sansom, D. C., Downs, T. et Saha, T. K. (1998). Evaluation of support vector machine based forecasting tool in electricity price forecasting for Australian national electricity market participants. *Power Engineering*, 22(3):227–233.

[Saxen, 1997] Saxen (1997). on the equivalence between ARMA models and simple recurrent neural networks.

[Schapire, 1990] Schapire, R. E. (1990). The strength of weak learnability.

[Schmidhuber, 1992] Schmidhuber, J. (1992). A fixed size storage O(n 3 ) time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, 4(2):243–248.

[Schmidt et Gish, 1996] Schmidt, M. et Gish, H. (1996). Speaker Identification via Support Vector Classifiers. *In ProcICASSP 96*, volume 1, pages 105–108. IEEE.

[Scholkopf *et al.*, 1997] Scholkopf, B., Burges, C., Girosi, F., Niyogi, P., Poggio, T. et Vapnik, V. (1997). Comparing support vector machines with Gaussian kernels to radial basis function classifiers. *IEEE Transactions on Signal Processing*, 45(11):2758–2765.

[Schölkopf *et al.*, 2000] Schölkopf, B., Smola, A. J., Williamson, R. C. et Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, 12(5):1207–1245.

[Schuster et Paliwal, 1997] Schuster, M. et Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[Sharda et Patil, 1992] Sharda, R. et Patil, R. B. (1992). Connectionist approach to time series prediction : an empirical test. *Computer*, 3(5):317–323.

[Sheta et De Jong, 2001] Sheta, A. F. et De Jong, K. (2001). Time-series forecasting using GA-tuned radial basis functions. *Information Sciences*, 133(3-4):221–228.

[Simon *et al.*, 2007] Simon, G., Lee, J., Cottrell, M. et Verleysen, M. (2007). Forecasting the CATS benchmark with the Double Vector Quantization method. *Neurocomputing*, 70(13-15):2400–2409.

[Simon *et al.*, 2004] Simon, G., Lendasse, A., Cottrell, M., Fort, J.-C. et Verleysen, M. (2004). Double quantization of the regressor space for long-term time series prediction : method and proof of stability. *Neural networks : the official journal of the International Neural Network Society*, 17(8-9):1169–81.

[Smola *et al.*, 1999] Smola, A., Scholkopf, B. et Ratsch, G. (1999). Linear programs for automatic accuracy control in regression. *Artificial Neural Networks, 1999. ICANN 99. Ninth*, 470:pp575–580.

[Smola *et al.*, 2000] Smola, A. J., Elisseeff, A., Scholkopf, B. et Williamson, R. C. (2000). Entropy Numbers for Convex Combinations and MLPs. *In* Smola, A. J., Bartlett, P. L., Schölkopf, B. et Schuurmans, D., éditeurs : *Advances in Large Margin Classifiers*, pages 369–387. {MIT} Press.

[Smola et Schölkopf, 2004] Smola, A. J. et Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

[Song, 2011] Song, Q. (2011). Robust initialization of a Jordan network with recurrent constrained learning. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 22(12):2460–73.

[Srinivasan *et al.*, 1994] Srinivasan, B., Prasad, U. R. et Rao, N. J. (1994). Back propagation through adjoints for the identification of nonlinear dynamic systems using recurrent neural models. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 5(2):213–28.

[Strickert et Hammer, 2003] Strickert, M. et Hammer, B. (2003). Unsupervised Recursive Sequence Processing. *NEUROCOMPUTING*, 63:433 – 439.

[Strickert et Hammer, 2005] Strickert, M. et Hammer, B. (2005). Merge SOM for temporal data. *Neurocomputing*, 64:39–71.

[Sun *et al.*, 1992] Sun, G.-Z., Chen, H.-H. et Lee, Y.-C. (1992). Green's Function Method for Fast On-Line Learning Algorithm of Recurrent Neural Networks. *In* Moody, J. E., Hanson, S. J. et Lippmann, R. P., éditeurs : *Advances in Neural Information Processing Systems*, volume 4, pages 333–340. Morgan Kaufmann Publishers, Inc.

[Sun *et al.*, 2004] Sun, Y. F., Liang, Y. C., Zhang, W. L., Lee, H. P., Lin, W. Z. et Cao, L. J. (2004). Optimal partition algorithm of the RBF neural network and its application to financial time series forecasting. *Neural Computing and Applications*, 14(1):36–44.

[Suykens *et al.*, 2000] Suykens, J., Lukas, L. et Vandewalle, J. (2000). Sparse approximation using least squares support vector machines. *2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353)*, pages 757–760.

[Suykens, 2002] Suykens, J. A. K. (2002). Weighted least squares support vector machines : robustness and sparse approximation.

[Suykens *et al.*, 2002] Suykens, J. A. K., Van Gestel, T., De Brabanter, J., De Moor, B. et Vandewalle, J. (2002). *Least Squares Support Vector Machines*. Numéro July. World Scientific.

[Svarer *et al.*, 1993] Svarer, C., Hansen, L. et Larsen, J. (1993). On design and evaluation of tapped-delay neural network architectures. *Neural Networks, 1993., IEEE ...*, (2).

[Tang et Fishwick, 1993] Tang, Z. et Fishwick, P. A. (1993). Feedforward Neural Nets as Models for Time Series Forecasting. *INFORMS Journal on Computing*, 5(4):374–385.

[Tay et Cao, 2002] Tay, F. E. et Cao, L. (2002). Modified support vector machines in financial time series forecasting. *Neurocomputing*, 48(1-4):847–861.

[Tenti, 1996] Tenti, P. (1996). Forecasting foreign exchange rates using recurrent neural networks. *Applied Artificial Intelligence*, 10(6):567–582.

[Thissen *et al.*, 2003] Thissen, U., van Brakel, R., de Weijer, A., Melssen, W. et Buydens, L. (2003). Using support vector machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems*, 69(1-2):35–49.

[Tong et Lim, 1980] Tong, H. et Lim, K. S. (1980). Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society Series B Methodological*, 42(3): 245–292.

[Toomarian et Barhen, 1992] Toomarian, N. B. et Barhen, J. (1992). Learning a trajectory using adjoint functions and teacher forcing. *Neural Networks*, 5(3):484–473.

[Trafalis et Ince, 2000] Trafalis, T. et Ince, H. (2000). Support vector machine for regression and applications to financial forecasting. *In Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing : New Challenges and Perspectives for the New Millennium*, pages 348–353 vol.6. IEEE.

[Tung-Bo et Von-Wun, 1996] Tung-Bo, C. et Von-Wun, S. (1996). A comparative study of recurrent neural network architectures on learning temporal sequences. *In Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 4, pages 1945–1950. IEEE.

[Tuv, 2006] Tuv, E. (2006). Ensemble Learning. *Feature Extraction*, 207:187–204.

[Ulbricht, 1994] Ulbricht, C. (1994). Multi-Recurrent Networks for Traffic Forecasting. *In National Conference on Artificial Intelligence (12th : 1994 : Seattle Wash.). Proceedings ... Vol. 2.*

[Vapnik, 2000] Vapnik, V. N. (2000). The nature of statistical learning theory.

[Varsta *et al.*, 2001] Varsta, M., Heikkonen, J., Lampinen, J. et Millán, J. (2001). Temporal Kohonen map and the recurrent self-organizing map : analytical and experimental comparison. *Neural processing letters*, 13(3):237–251.

[Varsta *et al.*, 1997] Varsta, M., Heikkonen, J. et Millan, J. d. R. (1997). Context Learning with the Self-Organizing Map.

[Verleysen et François, 2005] Verleysen, M. et François, D. (2005). The curse of dimensionality in data mining and time series prediction. *Analysis*, 3512:758 – 770.

[Vesanto, 1997] Vesanto, J. (1997). Using the SOM and local models in time-series prediction. *In Proc. Workshop on Self-Organizing Maps*, pages 209–214.

[Villmann et Erzsébet, 2010] Villmann, T. et Erzsébet, M. (2010). Extensions and Modifcations of the Kohonen-SOM and Applications in Remote Sensing Image Analysis. *Springer Verlag.*

[Vlachos *et al.*, 2003] Vlachos, M., Lin, J., Keogh, E. et Gunopulos, D. (2003). A Wavelet-based Anytime Algorithm for K-means Clustering of Time Series. *In Proc Workshop on Clustering High Dimensionality Data and Its Applications*, volume 52, pages 23–30. Citeseer.

[Voegtlin, 2002] Voegtlin, T. (2002). Recursive self-organizing maps. *Neural networks : the official journal of the International Neural Network Society*, 15(8-9):979–91.

[Walter *et al.*, 1990] Walter, J., Riter, H. et Schulten, K. (1990). Nonlinear prediction with self-organizing maps. *In Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 589–594.

[Wan, 1990] Wan, E. (1990). Temporal backpropagation for FIR neural networks. *In International Joint Conference on Neural Networks*, pages 575–580. IEEE.

[Wan, 1993] Wan, E. (1993). Finite impulse response neural networks for autoregressive time series prediction. *Time Series Prediction : Forecasting the Future and Understanding the Past*, (2):195–218.

[Wang et Lu, 2006] Wang, D. et Lu, W. (2006). Forecasting of ozone level in time series using MLP model with a novel hybrid training algorithm. *Atmospheric Environment*, 40(5):913–924.

[Wedding et Cios, 1996] Wedding, D. et Cios, K. (1996). Time series forecasting by combining RBF networks, certainty factors, and the Box-Jenkins model. *Neurocomputing*, 10(2):149–168.

[Weigend, 1993] Weigend, A. S. (1993). *Time Series Prediction : Forecasting The Future And Understanding The Past (Santa Fe Institute Series)*. Westview Press.

[Weigend *et al.*, 1990] Weigend, A. S., Huberman, B. A. et Rumelhart, D. E. (1990). Predicting the Future : A Connectionist Approach. *International Journal of Neural Systems*, 1(3):193–209.

[Weigend *et al.*, 1992] Weigend, A. S., Huberman, B. A. et Rumelhart, D. E. (1992). Predicting Sunspots and Exchange Rates with Connectionist Networks. *In* Casdagli, M. et Eubank, S., éditeurs : *Nonlinear modeling and forecasting*, volume 12, pages 395–432. Addison-Wesley.

[Weiss *et al.*, 1995] Weiss, C.-O., Hübner, U., Abraham, N. B. et Tang, D. (1995). Lorenz-like chaos in NH3-FIR lasers. *Infrared Physics & Technology*, 36(1):489–512.

[Weston *et al.*, 1999] Weston, J., Gammerman, A., Stitson, M., Vapnik, V., Vovk, V. et Watkins, C. (1999). Support Vector Density Estimation. *In* Schölkopf, B., Burges, C. J. C. et Smola, A. J., éditeurs : *Advances in Kernel Methods Support Vector Learning*, pages 293–306. MIT Press.

[Wolpert, 1992] Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2): 241–259.

[Yu-Kun *et al.*, 2005] Yu-Kun, B., Zhi-Tao, L., Lei, G. et Wen, W. (2005). Forecasting stock composite index by fuzzy support vector machines regression. *In 2005 International Conference on Machine Learning and Cybernetics*, pages 3535–3540 Vol. 6. IEEE.

[Yudong et Lenan, 2009] Yudong, Z. et Lenan, W. (2009). Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network. *Expert Systems with Applications*, 36(5):8849–8854.

[Yule, 1927] Yule, G. U. (1927). On a Method of Investigating Periodicities in Disturbed Series. *Philosophical Transactions of the Royal Society of London*, 226(Series A):167–298.

[Zemouri *et al.*, 2010] Zemouri, R., Gouriveau, R. et Zerhouni, N. (2010). Defining and applying prediction performance metrics on a recurrent NARX time series model. *Neurocomputing*, 73(13-15):2506–2521.

[Zemouri *et al.*, 2003] Zemouri, R., Racoceanu, D. et Zerhouni, N. (2003). Recurrent radial basis function network for time-series prediction. *Engineering Applications of Artificial Intelligence*, 16(5-6):453–463.

[Zhang, 2003] Zhang, G. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175.

[Zhang, 2005] Zhang, G. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514.

[Zhang *et al.*, 2001] Zhang, G., Patuwo, B. et Hu, M. Y. (2001). A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers & Operations Research*, 28(4):381–396.

[Zhang *et al.*, 2004] Zhang, H. Z. H., Ho, T.-B. H. T.-B. et Lin, M.-S. L. M.-S. (2004). An evolutionary K-means algorithm for clustering time series data.

[Zhou *et al.*, 2011] Zhou, H., Su, G. et Li, G. (2011). Forecasting Daily Gas Load with OIHF-Elman Neural Network. *Procedia Computer Science*, 5(null):754–758.

[Zuo *et al.*, 2010] Zuo, Z., Yang, C. et Wang, Y. (2010). A new method for stability analysis of recurrent neural networks with interval time-varying delay. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 21(2):339–44.

# Liste des publications

## 2013

Cherif, A., Cardot, H., & Boné, R. (2011). Hierarchical clustering for local time series forecasting. International Conference on Neural Information (article soumis)

## 2011

Cherif, A., Cardot, H., & Boné, R. (2011). SOM time series clustering and prediction with recurrent neural networks. Neurocomputing, 74(11), 1936–1944.

## 2009

Cherif, A., Cardot, H., & Bone, R. (2009). Recurrent Neural Networks as Local Models for Time Series Prediction. In C. S. L. M. C. J. H. Leung (Ed.), Neural Information Processing Pt 2 Proceedings (Vol. 5864, pp. 786–793).

Cherif, A., Cardot, H., & Boné, R. (2009). Recurrent Neural Networks for Local Models Prediction. International Conference on Cognitive Neurodynamics, Hangzhou China, 8.

# Glossaire

- $AGNES$ : Agglomerative NESting
- $ANN$ : Artificial Neural Network
- $AR$ : Auto Regressive
- $BP$ : Back Propagation
- $BPTT$ : Back Propagation Through Time
- $DIANA$ : Divisive ANAlysis
- $FFNN$ : Feed Forward Neural Networks
- $FIR$ : Finite Impluse Response
- $FSOM$ : Feedback Self Organizing Maps
- $IIR$ : Infinite Impluse Response
- $LRGF$ : Locally Recurrent Globally Feed Forword
- $MA$ : Moving Average
- $ML$ : Machine Learning
- $MLP$ : Multi Layer Perceptron
- $MSE$ : Mean Square Error
- $MSOM$ : Merge Self Organizing Maps
- $NG$ : Neural Gas
- $NMSE$ : Normalized Mean Square Error

- $RBFN$ : Radial Basis Function Network
- $RecSOM$ : Recursive Self Organizing Maps
- $RNN$ : Recurrent Neural Network
- $RSOM$ : Recurrent Self Organizing Maps
- $RTRL$ : Real Time Recurrent Learning
- $SARDNET$ : Sequential Activation Retention and
- $SOM$ : Self Organizing Maps
- $SOMSD$ : Self Organizing Maps for Structured Data
- $SOMTAD$ : Self Organizing Maps with Temporal
- $SVM$ : Support Vector Machine
- $SVR$ : Support Vector Regression
- $SARDNET$ : Sequential Activation Retention and Decay NETwork
- $TKM$ : Temporal Kohonen Maps

## Résumé

La prévision des séries temporelles est un problème qui est traité depuis de nombreuses années. On y trouve des applications dans différents domaines tels que : la finance, la médecine, le transport, etc. Dans cette thèse, on s'est intéressé aux méthodes issues de l'apprentissage artificiel : les réseaux de neurones et les SVM. On s'est également intéressé à l'intérêt des méta-méthodes pour améliorer les performances des prédicteurs, notamment l'approche locale. Dans une optique de diviser pour régner, les approches locales effectuent le clustering des données avant d'affecter les prédicteurs aux sous ensembles obtenus. Nous présentons une modification dans l'algorithme d'apprentissage des réseaux de neurones récurrents afin de les adapter à cette approche. Nous proposons également deux nouvelles techniques de clustering, la première basée sur les cartes de Kohonen et la seconde sur les arbres binaires.

## Mots clés

Réseaux de neurones, Perceptron multi-couche, réseaux de neurones récurrents, SVM (Support Vector Machines), prédiction des séries temporelles, régression, apprentissage artificiel supervisé et non supervisé.

## Abstract

Time series forecasting is a widely discussed issue for many years. Researchers from various disciplines have addressed it in several application areas : finance, medical, transportation, etc. In this thesis, we focused on machine learning methods : neural networks and SVM. We have also been interested in the meta-methods to push up the predictor performances, and more specifically the local models. In a divide and conquer strategy, the local models perform a clustering over the data sets before different predictors are affected into each obtained subset. We present in this thesis a new algorithm for recurrent neural networks to use them as local predictors. We also propose two novel clustering techniques suitable for local models. The first is based on Kohonen maps, and the second is based on binary trees.

## Keywords

Neural networks, multi layer perceptron, recurrent neural networks, SVM (Support Vector Machines), time series forecasting, regression, machine learning, supervised learning, unsupervised learning.