جامعة أبو بكر بلقايد
ⵜⴰⵙⴷⴰⵡⵉⵜ ⴰⴱⵓⴱⴽⵔ ⴱⵍⵇⴰⵢⴷ ⵜⵍⵎⵙⵏ
UNIVERSITY OF TLEMCEN

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

UNIVERSITÉ ABOU BEKR BELKAID TLEMCEN

FACULTÉ DES SCIENCES

MÉMOIRE :
pour l'obtention du diplôme de
MASTER
Option : Probabilités-Statistiques
Thème :

Prévision d'une série temporelle par les réseaux de neurones

présenté par :
REFOUFI Mounia [1]

Devant le jury composé de :

Mme BENSABER Fatna, Maître assistant classe A, Université de Tlemcen.	**Présidente.**
Mme Khitri-Kazi-Tani Leila , Maitre de Conférence classe A, Université de Tlemcen.	**Examinatrice.**
Mme KADA KLOUCHA Meryem, Maitre de conférence, ESSAT.	**Rapporteur.**

1. e-mail : mouniarff@gmail.com

Table des matières

Introduction

L'apprentissage automatique ("Machine Learning", en anglais), est une branche de l'intelligence artificielle qui peut être mise à contribution pour appréhender les problèmes de prédictions. On peut classer les méthodes d'apprentissage artificiel en trois grandes familles :

— Apprentissage supervisé : où on dispose d'un ensemble d'objets et pour chaque objet une valeur cible associée.

— Apprentissage non supervisé : où on dispose d'un ensemble d'objets sans aucune valeur cible associée.

— Apprentissage semi supervisé : où on dispose d'un petit ensemble d'objets avec pour chacun une valeur cible associée et d'un plus grand ensemble d'objets sans valeur cible.

Il existe une large variété d'algorithmes de l'apprentissage automatique, certains sont plus couramment utilisés que d'autres.
Dans ce mémoire, nous allons nous intéresser à l'apprentissage artificiel supervisé qui consiste à créer un modèle de prédiction, 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 .

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 pour évaluer ces performances est de calculer l'Erreur Quadratique Moyenne (MSE : Mean Square Error (1)), qui quantifie l'écart moyen entre les prédictions du modèle et les valeurs réelles, une autre mesure couramment utilisée est l'erreur relative (MRE : Mean Relative Error (2)), qui exprime l'écart moyen par rapport aux valeurs réelles.
En outre, pour comparer différentes approches, le critère BIC (Bayesian Information Criterion) peut également être utilisé, prenant en compte à la fois la précision et la complexité du modèle.

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, et quand la valeur cible appartient à un ensemble continu (par exemple $\mathbb{R}$ ou $[0, A]$, $A \in \mathbb{R}^{*+}$), 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 ce mémoire, on s'intéresse surtout à cette dernière tâche, en étudiant la prévision d'une série temporelle.
On observe la trajectoire d'un processus $X = (X(t); t \in \mathbb{R}^+)$ à temps continu, sur des intervalles successifs d'une longueur donnée δ et on s'interesse à prévoir l'évolution globale future du processus sur un intervalle de même longueur. Ceci est le problème de prédiction classique d'un élément fonctionnel $Y = X_{n+1}$ (la variable cible), à partir du vecteur d'entrée $X_1, X_2, ..., X_n$ (morceaux de trajectoire).

Pour approcher Y on cherche à construire un réseau de neurones qui approxime au mieux l'espérance $E[Y/X_1, X_2, ..., X_n]$ par une fonction f

$$E[Y/X_1, X_2, ..., X_n] = f(X_1, X_2, ..., X_n)$$

L'algorithme d'apprentissage est responsable de la mise à jour des poids et des biais du réseau de neurones en fonction des données d'entraînement. Son objectif est de minimiser une fonction de coût qui mesure l'écart entre les prédictions du modèle et les valeurs réelles

Le système pourra être évalué à la fin par les erreurs MSE , MRE :

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\widehat{X_{n+1}(t_i)} - X_{n+1}(t_i)\right)^2 \tag{1}$$

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|\widehat{X_{n+1}(t_i)} - X_{n+1}(t_i)\right|}{|X_{n+1}(t_i)|}. \tag{2}$$

où N représente le nombre d'observations.

Ce mémoire se compose de quatre chapitres distincts :

— Le premier chapitre aborde les modèles d'un réseau de neurones ainsi que la méthode de rétropropagation du gradient.

— Dans le deuxième chapitre, nous passons en revue les principaux résultats liés à l'analyse d'une série temporelle.

— Le troisième chapitre est consacré à l'application des réseaux de neurones à la prévision des séries temporelles, avec une mise en évidence des résultats obtenus dans l'article de Rynkiewicz J. ([18]).

— Enfin, le quatrième chapitre présente diverses applications et exemples illustrant le comportement du prédicteur MLP (Multilayer Perceptron) à travers des exemples numériques. Nous examinons plusieurs cas en variant les paramètres du prédicteur MLP, et les résultats obtenus concordent avec ceux de la convergence presque sûre.
Par la suite, nous appliquons le prédicteur MLP étudié au cas de la série d'El-Niño, tout en comparant ses performances avec celles d'autres méthodes de prédiction existantes dans la littérature statistique ([4], [5], [11]).

Chapitre 1

Les réseaux de neurones

1.1 Introduction

Le machine learning (ML) est une branche de l'intelligence artificielle (IA). Il a la capacité d'apprendre à partir de données d'entraînement en utilisant un algorithme d'apprentissage dont l'objectif est d'effectué des analyses explicatives prédicatives. Ainsi le Deep learning est une sous branche de l'apprentissage automatique qu'on peut utiliser a base de données d'entraînement afin de produire une fonction mathématique .

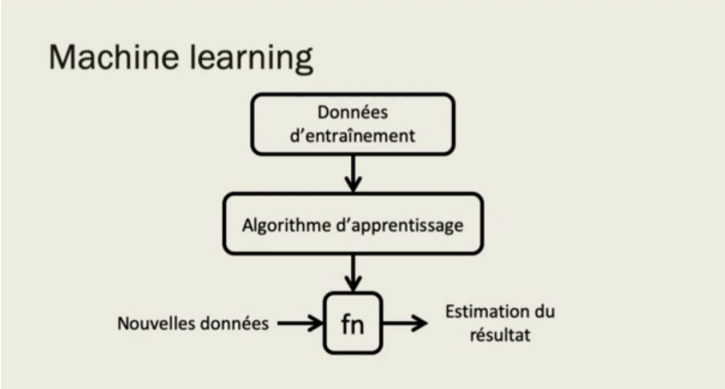


FIGURE 1.1 – Machine learning

Notre cerveau est composé de plus de 100 milliards de neurones, un neurone est une cellule de notre corps qui reçoit des signaux il mélange ces signaux pour produire un signal afin de le propager sur de nouveaux neurones.

En s'inspirant de neurones biologiques pour créer des neurones artificiels, un neurone artificiel prend en entrée des caractéristiques puis va appliquer une fonction à ses caractéristiques pour ressortir une nouvelle caractéristique qui va propager vers d'autres neurones.

1.2 Les modèles de réseaux de neurones

1.2.1 Neurone biologique

Le système nerveux est composé de milliards de cellules, c'est un réseau de neurones biologiques. En effet les neurones ne sont pas indépendants les uns des autres, ils établissent entre eux des liaisons et forment des réseaux plus ou moins complexes.

Schématique, on peut décomposer le neurone biologique en 4 grands entités :

— **Le corps cellulaire (soma) :** il est composé du centre de contrôle traitant les informations reçues par les dendrites.

— **Les dendrites :** sont les principaux fils conducteur par lesquels transitent l'information venue de l'extérieur.

— **L'axone :** c'est un fil conducteur qui conduit le signal de sortie du corps cellulaire vers d'autres neurones .

— **Les synapses :** qui permettent aux neurones de communiquer avec les autres via les axones et les dendrites.

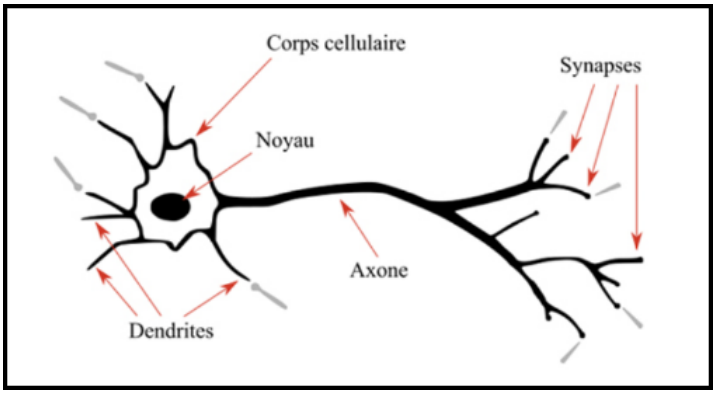


FIGURE 1.2 – Neurone biologique, (**source :** `https://rb.gy/5ckqq`)

1.2.2 Neurone formel (artificiel)

Un neurone formel, encore appelé neurone artificiel est une représentation mathématique et informatique d'un neurone biologique. Il possède plusieurs entrées et une sortie.
On considère le cas général d'un neurone formel à n entrées, auquel on doit donc soumettre les n grandeurs numériques notées x_1 à x_n, un modèle de neurone formel est une formule mathématique qui permet d'associer aux n entrées une sortie, à chaque entrée est associée un poids synaptique c'est à dire une valeur numérique notée de w_1 pour l'entrée 1 jusqu'à w_n pour l'entrée n.

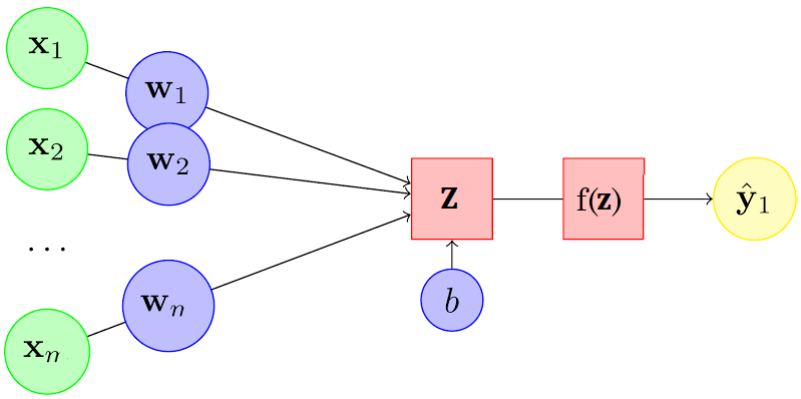


FIGURE 1.3 – Neurone artificiel

avec

— $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$: sont les entrées du neurone (signaux qui lui parviennent).

— $(w_1, w_2, \ldots, w_n)$: les poids associés à chaque connexion.

— $\boldsymbol{b}$: le seuil d'activation.

— Z : la somme pondérée des entrées (potentiel d'activation).

$$\mathbf{Z} = \sum_{i=1}^{n} w_i x_i + b.$$

— f : la fonction de transfert (fonction d'activation)

— $\hat{\mathbf{y}} = f(\mathbf{Z})$: la sortie du neurone (réponse du neurone).

$$\boldsymbol{f}(\mathbf{Z}) = f(\sum_{i=1}^{n} w_i x_i + b). \tag{1.1}$$

1.2.3 Fonctions d'activation

La fonction d'activation (de seuillage, ou de transfert) d'un neurone artificiel définit le rendement de ce neurone donné, à partir d'un ensemble d'entrées.
Plusieur fonctions de transfert pouvant être utilisées , les trois fonctions les plus utilisées, sont les fonctions seuil, sigmoïde et tangente hyperbolique.

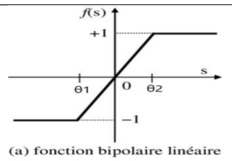
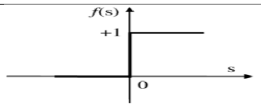
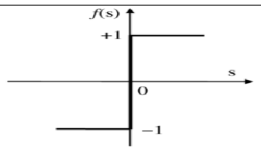
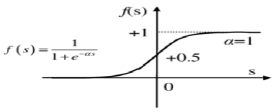
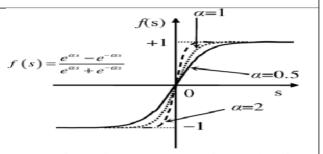
Nom de la fonction	Relation entrée/sortie	Icône
bipolaire linéaire	$f(s) = \begin{cases} -1 & \text{si } s < \theta_1 \\ as + b & \text{si } \theta_1 < s < \theta_2 \\ 1 & \text{si } \theta_2 < s \end{cases}$	 (a) fonction bipolaire linéaire
seuil	$f(s) = \begin{cases} 0 & \text{si } s < 0 \\ 1 & \text{si } s > 0 \end{cases}$	 (b) fonction de seuil
seuil bipolaire	$f(s) = \begin{cases} -1 & \text{si } s < 0 \\ 0 & \text{si } s = 0 \\ 1 & \text{si } s > 0 \end{cases}$	 (c) fonction de seuil bipolaire
sigmoïde	$f(s) = \frac{1}{1 + e^{-as}}$	 (d) fonction sigmoïde
tangente hyperboliq ue	$f(s) = \frac{e^{as} - e^{-as}}{e^{as} + e^{-as}}$	 (e) fonction tangente hyperbolique

FIGURE 1.4 – Fonctions d'activation

1.2.4 La comparaison entre un neurone biologique et un neurone formel

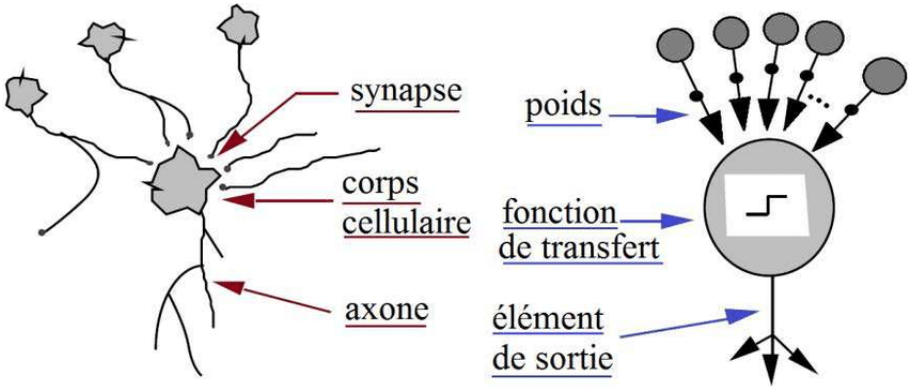


FIGURE 1.5 – La comparaison entre un neurone biologique et un neurone formel [3]

1.2.5 Les différents types de neurones

Perceptron monocouche

Le perceptron monocouche, probablement le plus ancien modèle de calcul neuronal, a été développé par F.Rosenblatt remonte à 1958 et est considéré comme l'un des algorithmes d'apprentissage supervisé les plus simples pour la classification binaire. Ce réseau de neurones contient Une couche d'entrée et un nœud de sortie. Il existe plusieurs types de perceptrons, mais dans sa version la plus simple c'est une conception monocouche composée d'un seul neurone connecté à n entrées.

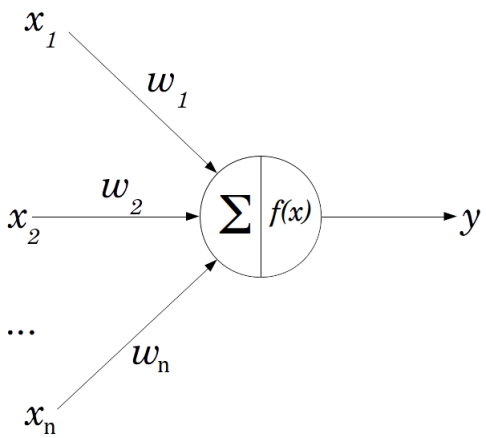


FIGURE 1.6 – Perceptron monocouche [3]

Il existe d'autre version du perceptron monocouche avec plusieurs sorties, le schéma ci-dessous montre un preceptron monocouche avec plusieurs sorties

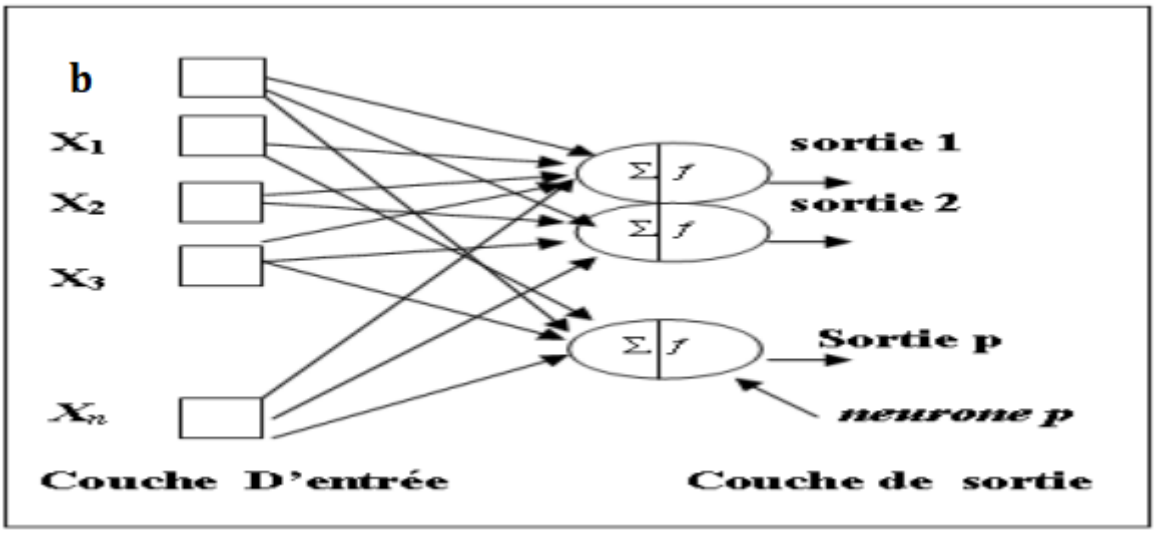


FIGURE 1.7 – Preceptron monocouche avec plusieurs sorties [3]

Perceptron multicouches

Les réseaux de neurones multicouches MLP contiennent plus d'une couche de calcul. Contrairement à un perceptron monocouche, la couche de sortie d'un perceptron multicouche est la seule couche qui effectue des calculs entièrement visibles pour l'utilisateur. Les réseaux multicouches contiennent plusieurs couches de calcul. couches intermédiaires supplémentaires (entrée et sortie) sont appelées couches cachées car le calcul effectué n'est pas visible pour les utilisateurs

— La couche d'entrée est constituée de neurones qui lisent les éléments vecteur et envoyer l'information aux neurones de la première couche cachée.

— Chaque neurone de la couche cachée prend une moyenne pondérée des informations reçues et retransmet ces informations aux neurones de la couche suivante. La valeur moyenne modifiée par la fonction d'activation.

— Perceptron multicouches peut comporter d'une à plusieurs couches cachées.

— Enfin, la dernière couche émet un vecteur ou scalaire de sortie réseau. Sa nature exacte dépend du problème traité (classification ou régression).

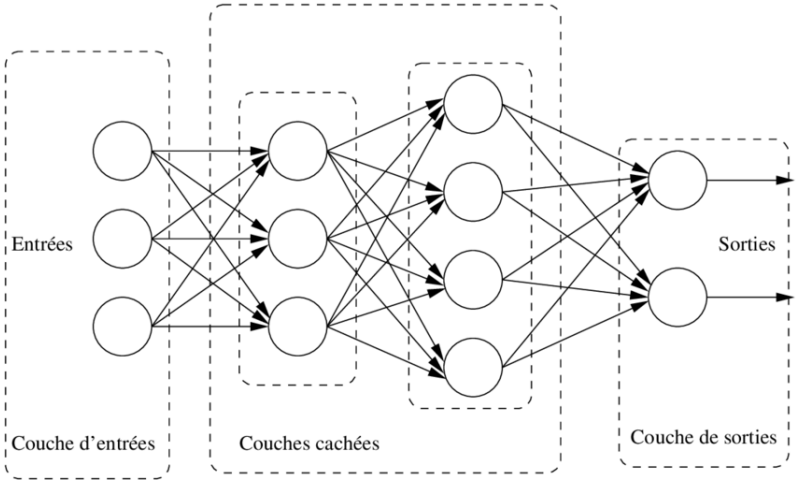


FIGURE 1.8 – Preceptron multicouche `https://rb.gy/jv3ej`

1.3 La différence en termes entre la statistique et les réseaux de neurones

On pourrait synthétiser la différence par :

Les réseaux de Neurones	La statistique
Apprentissage	Estimation
Poids et biais	Paramètres
Apprentissage supervisé	Régression
Apprentissage non supervisé	Estimation de densité
Réseau de neurone	modèle
Ensemble d'apprentissage	Échantillon

TABLE 1.1 – la différence en termes entre la statistique et les réseaux de neurones [3]

1.4 Apprentissage d'un réseau de neurone

L'apprentissage est une étape très importante du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié itérativement jusqu'à obtenir le comportement souhaité en ajustant les poids et les biais des neurones, l'objectif essentiel de l'apprentissage est la résolution du problème par la prévision, la classification ... etc.

Il existe de nombreux types de règles d'apprentissage qui peuvent être regroupées en deux catégories : les règles d'apprentissage supervisé et non supervisé.

1.4.1 Apprentissage non supervisé

L'apprentissage est non supervisé lorsque seules les valeurs d'entrée sont disponibles, le réseau est censé s'organiser par lui même pour produire la réponse correcte, Ce type d'apprentissage est également connu sous le nom d'apprentissage compétitif. L'avantage de ce type d'apprentissage est sa forte adaptabilité reconnue comme une auto-organisation, « "self-organizing» ".
L'apprentissage non supervisé est principalement utilisé dans le traitement du signal et l'analyse factorielle. Voir par exemple les travaux de Jeremie [19]

1.4.2 Apprentissage supervisé

L'apprentissage supervisé est une forme d'apprentissage automatique où l'algorithme est guidé par un ensemble d'observations décrit par un vecteur d'entrée (X) et par un vecteur cible (La réponse Y) .

Dans ce type d'apprentissage, on essaie d'imposer au réseau un fonctionnement donné en forçant la sortie du réseau à prendre une valeur désiré (choisie par l'opérateur) et en modifiant les poids synaptiques.

Le vecteur de sortie $\hat{Y}$ est comparé au vecteur de sortie désiré Y , ce qui permet le calcul de la fonction erreur appelée fonction coût , la fonction d'erreur la plus utilisée est la moyenne des carrées des écarts entre la sortie de neurone et la sortie désirée .

l'algorithme d'apprentissage est la méthode mathématique qui va modifier les poids de tel manière a converger vers une solution qui permettra au réseau d'accomplir la tache désirée . Plusieurs algorithme itératifs peuvent être mis en oeuvre parmi lesquels on note l'algorithme de Rétropropagation.

1.5 Modèle de réseau de neurones

Dans cette partie, nous présentons le modèle de réseau que nous avons exploré, ce modèle est basé sur le réseau "perceptron multicouche" d'architecture "feedforward" où nous avons utilisé l'algorithme rétropropagation "backpropagation".

1.5.1 Méthode rétropropagation du gradient

La rétropropagation du gradient est une méthode qui permet de calculer le gradient de l'erreur, pour chaque neurone du réseau, de la dernière couche vers la première (voir [21] , [17], [8] et [14]).

Le principe de la méthode de rétropropagation du gradient est de calculer les gradients des poids et des biais du réseau de neurones en utilisant la règle de dérivation en chaîne (chaîne de règle du gradient). Cela permet d'ajuster les poids et les biais de manière itérative afin de minimiser l'erreur de prédiction du réseau. La fonction erreur la plus utilisée est la somme des erreurs quadratiques entre les sorties désirées et les sorties calculées . La méthode d'optimisation utilisée est donc la méthode du gradient.

On note par :

— d^l le nombre de neurone du $l^{\text{ème}}$ couche.
— L est le nombre de couches du neurones.
— f est la fonction d'activation, elle est la même pour toutes les couches cachées.
— $l = \overline{1, L}$ $w_{ij}^{(l)}$ est le poids qui relie le ième neurone de la $l^{\text{ième}}$ couche au $j^{\text{ième}}$ neurone de la $l - 1^{\text{ième}}$ couche.
— $w^{(l)}$ est une matrice de taille $d^l \times d^{l-1}$ qui contient tous les poids de la $l^{\text{ième}}$ couche.
— $b_i^{(l)}$ est le biais associé au $i^{\text{ième}}$ neurone de la $l^{\text{ième}}$ couche.
— $b^{(l)}$ est le vecteur qui contient tous les biais de la $l^{\text{ième}}$ couche cachée.
— $z_i^{(l)}$ est la valeur de préactivation

$$z_i^{(l)} = \sum_{j=1}^{d^{l-1}} w_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}. \tag{1.2}$$

— $a_i^{(l)}$ est la valeur d'activation $a_i^{(l)} = f(z_i^{(l)})$.

L'écriture matricielle de l'équation (1.2), nous donne la formule suivante :

$$z^{(l)} = w^{(l)} \cdot a^{(l-1)} + b^{(l)} \tag{1.3}$$

Avec

$$z^{(l)} = \begin{pmatrix} z_1^{(l)} \\ z_2^{(l)} \\ \vdots \\ z_{d^l}^{(l)} \end{pmatrix}, a^{(l)} = \begin{pmatrix} a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{d^l}^{(l)} \end{pmatrix}, b^{(l)} = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{d^l}^{(l)} \end{pmatrix}, w^{(l)} = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & ..... & w_{1,d^{l-1}}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & ..... & ..... \\ ..... & & & \\ w_{d^l,1}^{(l)} & w_{d^l,2}^{(l)} & ..... & w_{d^l,d^{l-1}}^{(l)} \end{pmatrix}.$$

$$a^{(0)} = X \to z^{(1)} \xrightarrow{f} a^{(1)} \to z^{(2)} \xrightarrow{f} a^{(2)} \to ..... \to z^{(L)} \xrightarrow{f} a^{(L)} = \hat{Y} \tag{1.4}$$

$$\text{Entrée} \to 1^{\text{ère}} \text{ couche} \to 2^{\text{ème}} \text{ couche} \to .....L^{\text{ème}} \text{ couche} \to \text{sortie}. \tag{1.5}$$

-$\hat{Y}$ va être comparer à Y (la variable cible) en utilisant la fonction coût, on la notera :

$$C(\hat{Y}, Y) = \frac{1}{2} \left\| Y - \hat{Y} \right\|^2 \tag{1.6}$$

-les valeurs du vecteur des paramètres $\theta^{(l)} = (w^{(l)}, b^{(l)})$ qui minimisent la fonction coût (1.6) est donnée par les formules suivantes :

$$w^{(l)^{new}} = w^{(l)^{old}} - \alpha \cdot \nabla_{w^{(l)}} C \tag{1.7}$$

$$b^{(l)^{new}} = b^{(l)^{old}} - \alpha \cdot \nabla_{b^{(l)}} C. \tag{1.8}$$

où $0 < \alpha < 1$,

avecc $\nabla_{w^{(l)}} C$ et $\nabla_{b^{(l)}} C$, sont données par la proposition suivante :

Proposition 1.5.1. *les valeurs de gradients qui définie la méthode de descente du gradient sont données par les formules suivants :*

$$\nabla_{w^{(l)}} C = \delta^{(l)} (a^{(l-1)})^\top \tag{1.9}$$

$$\nabla_{b^{(l)}} C = \delta^{(l)}. \tag{1.10}$$

et

$$\delta_i^{(l)} = \sum_{j=1}^{d^L} \delta_j^{(l+1)} w_{ji}^{(l+1)} f'(z_i^{(l)}) \qquad \forall l = \overline{1, L-1}$$

$$\delta_i^{(L)} = -\left(y_i - f\left(z_i^{(L)}\right)\right) f'(z_i^{(L)}) \qquad l = L$$

Démonstration. **le calcul de $\nabla_{w^{(l)}} C$**

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = \frac{\partial C}{\partial z_i^{(l)}} \times \frac{\partial z_i^{(l)}}{\partial w_{i,j}^{(l)}} \tag{1.11}$$

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} \times a_j^{(l-1)}. \tag{1.12}$$

le calcul de $\delta_i^{(l)}$

Pour la dernière couche $l = L$

$$\begin{aligned}
\delta_i^{(L)} &= \frac{\partial C}{\partial z_i^{(L)}} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \left\| Y - \hat{Y} \right\|^2 \\
\delta_i^{(L)} &= \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \sum_{j=1}^{d^L} \left(y_j - a_j^{(L)} \right)^2 \\
\delta_i^{(L)} &= \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \sum_{j=1}^{d^L} \left(y_j - f(z_j^{(L)}) \right)^2 \\
\delta_i^{(L)} &= -\left(y_i - f\left(z_i^{(L)} \right) \right) f'(z_i^{(L)}).
\end{aligned}$$

Pour la couche cachée $l = L - 1$

$$\begin{aligned}
\delta_i^{(L-1)} &= \frac{\partial C}{\partial z_i^{(L-1)}} \\
&= \frac{\partial}{\partial z_i^{(L-1)}} \frac{1}{2} \left\| Y - \hat{Y} \right\|^2 \\
&= \frac{\partial}{\partial z_i^{(L-1)}} \frac{1}{2} \sum_{j=1}^{d^L} \left(y_j - a_j^{(L)} \right)^2 \\
&= \frac{\partial}{\partial z_i^{(L-1)}} \frac{1}{2} \sum_{j=1}^{d^L} \left(y_j - f(z_j^{(L)}) \right)^2 \\
&= \frac{1}{2} \sum_{j=1}^{d^L} \frac{\partial}{\partial z_i^{(L-1)}} \left(y_j - f(z_j^{(L)}) \right)^2 \\
&= \frac{1}{2} \sum_{j=1}^{d^L} \frac{\partial}{\partial z_j^{(L)}} \left(y_j - f(z_j^{(L)}) \right)^2 \frac{\partial z_j^{(L)}}{\partial z_i^{(L-1)}} \\
&= 2 \frac{1}{2} \sum_{j=1}^{d^L} -\left(y_j - f\left(z_j^{(L)} \right) \right) f'(z_j^{(L)}) \frac{\partial z_j^{(L)}}{\partial z_i^{(L-1)}} \\
&= \sum_{j=1}^{d^L} \delta_j^{(L)} \frac{\partial}{\partial z_i^{(L-1)}} \sum_{k=1}^{d^{L-1}} w_{jk}^L f(z_k^{(L-1)}) + b_j^L.
\end{aligned}$$

d'où :

$$\delta_i^{(L-1)} = \sum_{j=1}^{d^L} \delta_j^{(L)} w_{ji}^L f'(z_i^{(L-1)})$$

En remplaçant $L - 1$ par l et L par $l + 1$, on trouve

$$\delta_i^{(l)} = \sum_{j=1}^{d^L} \delta_j^{(l+1)} w_{ji}^{(l+1)} f'(z_i^{(l)}) \qquad \forall l = \overline{1, L-1}$$

Donc
de (1.12) :

$$\frac{\partial C}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} \times a_j^{(l-1)}$$

Avec

$$\begin{aligned}\delta_i^{(l)} &= \sum_{j=1}^{d^L} \delta_j^{(l+1)} w_{ji}^{(l+1)} f'(z_i^{(l)}) \qquad \forall l = \overline{1, L-1} \\ \delta_i^{(L)} &= -\left(y_i - f\left(z_i^{(L)}\right)\right) f'(z_i^{(L)}) \qquad l = L\end{aligned}$$

Ainsi

$$\nabla_{w^{(l)}} C = \delta^{(l)} (a^{(l-1)})^\top$$

le calcul de $\nabla_{b^{(l)}} C$

$$\begin{aligned}\frac{\partial C}{\partial b_i^{(l)}} &= \frac{\partial C}{\partial z_i^{(l)}} \times \frac{\partial z_i^{(l)}}{\partial b_i^{(l)}} \\ \frac{\partial C}{\partial b_i^{(l)}} &= \delta_i^{(l)}.\end{aligned}$$

d'où

$$\frac{\partial C}{\partial b_i^{(l)}} = \delta_i^{(l)} = \sum_{j=1}^{d^L} \delta_j^{(l+1)} w_{ji}^{(l+1)} f'(z_i^{(l)})$$

Ainsi

$$\nabla_{b^{(l)}} C = \delta^{(l)}$$

□

1.6 Conclusion

En conclusion, nous avons exploré les réseaux de neurones et l'algorithme de rétropropagation, qui constituent une combinaison puissante pour résoudre des problèmes complexes d'apprentissage automatique. Les réseaux de neurones sont des modèles inspirés du cerveau humain, capables d'apprendre à partir de données et de généraliser des connaissances pour effectuer des prédictions.

L'algorithme de rétropropagation est une méthode d'apprentissage clé pour entraîner les réseaux de neurones. Il permet de calculer les gradients par rapport aux poids et aux biais, ce qui permet d'ajuster ces poids de manière itérative afin de minimiser l'erreur de prédiction. Grâce à l'algorithme de rétropropagation, les réseaux de neurones peuvent apprendre à partir de grandes quantités de données et améliorer leurs performances au fil du temps.

L'un des principaux avantages des réseaux de neurones est leur capacité à modéliser des relations complexes entre les variables d'entrée et de sortie. Ils peuvent être utilisés dans une variété de domaines tels que la vision par ordinateur, le traitement du langage naturel, la prédiction de séries temporelles, etc. Leur flexibilité et leur adaptabilité en font des outils précieux pour résoudre des problèmes de prédiction et de classification.

Chapitre 2

Analyse d'une série temporelle

2.1 Introduction

Une série temporelle (série chronologique) est définie comme une réalisation d'un processus aléatoire ou une suite d'observations sur une variable aléatoire qui sont ordonnées par le temps, souvent notée $(X_t)_{t\in T}$. L'analyse d'une série temporelle est une démarche qui vise à comprendre et à interpréter les caractéristiques, les tendances et les motifs présents dans une séquence de données ordonnées dans le temps. Cela peut être appliqué à différents domaines tels que la finance, l'économie, la météorologie, la santé, etc. [6]

2.2 Processus Stochastique

-**Définition** : On appelle processus stochastique toute famille de variables aléatoires $(X_t)_{t\in T}$ d'un espace probabilisé $(\Omega, \mathcal{A}, P)$ vers un espace probabilisable $(E, \mathcal{E})$,
i.e : $\forall t \in T$: X_t est une v.a. de $(\Omega, \mathcal{A}, P)$ vers $(E, \mathcal{E})$.
L'ensemble $T(\ T = \mathbb{N}, \mathbb{Z})$ est appelé espace des temps
espace des états $E(E = \mathbb{R})$: C'est l'ensemble des valeurs possibles que peut prendre la variable aléatoire à chaque instant de temps. Par exemple, dans le cas de la température quotidienne, l'espace d'états peut être l'ensemble des températures réelles.
Pour $\omega \in \Omega$, on appelle trajectoire du processus la fonction :

$$t \mapsto X_t(\omega).$$

Les processus stochastiques sont caractérisés par leurs propriétés statistiques, telles que la moyenne, la variance, l'autocovariance et l'autocorrélation. Ces propriétés fournissent des informations sur la tendance, la variabilité et la dépendance temporelle des variables aléatoires du processus.

2.3 Indices descriptifs d'une série temporelle

Pour n observations :

— **Indices de tendance centrale** :Nous utilisons comme indicateur de la tendance centrale la moyenne :

$$\bar{X}_n = \frac{1}{n}\sum_{t=1}^{n} X_t$$

— **Indices de dispersion** :Nous utilisons comme indicateur de dispersion la variance empirique (et sa racine carrée, l'écart-type empirique) :

$$\hat{\sigma}_n(0) = \frac{1}{n}\sum_{t=1}^{n}\left(X_t - \bar{X}_n\right)^2.$$

— **Indices de dépendance** : Ces notions, plus spécifiques à l'étude de série temporelle, renseignent sur la dépendance entre les données X_t.

L'auto-covariance empirique d'ordre 1 : Elle renseigne sur la dépendance entre deux données successives

$$\hat{\sigma}_n(1) = \frac{1}{n-1}\sum_{t=1}^{n-1}\left(X_t - \bar{X}_n\right)\left(X_{t+1} - \bar{X}_n\right)$$

L'auto-covariance empirique d'ordre 2 :Elle renseigne sur la dépendance entre deux données écartées de deux pas de temps

$$\hat{\sigma}_n(2) = \frac{1}{n-2}\sum_{t=1}^{n-2}\left(X_t - \bar{X}_n\right)\left(X_{t+2} - \bar{X}_n\right)$$

et ainsi de suite. Pour des raisons de bon sens statistique, nous ne considérerons les covariances empiriques que jusqu'à un ordre h pas trop grand. On appelle fonction d'auto-covariance (empirique) la fonction qui à h associe $\hat{\sigma}_n(h)$.

$$\hat{\sigma}_n(h) = \frac{1}{n-h}\sum_{t=1}^{n-h}\left(X_t - \bar{X}_n\right)\left(X_{t+h} - \bar{X}_n\right)$$

2.3.1 Fonction d'Auto-corrélation

La fonction d'Auto-corrélation est utilisée pour caractériser la dépendance entre les variables :

$$\hat{\rho}_n(h) = \frac{\hat{\sigma}_n(h)}{\hat{\sigma}_n(0)}$$

2.4 Décomposition d'une série temporelle :

Décomposition additive :

$$X_t = T_t + S_t + \varepsilon_t$$

Décomposition multiplicative :

$$X_t = T_t \times S_t \times \varepsilon_t$$

— **Tendance** T_t :elle correspond à l'évolution à long terme de la série. elle n'a pas besoin d'être linéaire.
- IL existe plusieurs choix de T_t (deterministe) :
  — 1. Tendance linéaire : $T_t = b + at$.
  — 2. Tendance polynomiale : $T_t = a_0 + a_1 t + ... + a_p t^p$.
  — 3. Tendance exponentielle : $T_t = C + ae^{bt}$.
  — 4. Tendance logarithmique : $T_t = b + a\log t$.
  — 5. Tendance hyperbolique : $T_t = \frac{1}{b+at}$
  — 6. Tendance logistique : $T_t = \frac{1}{c+be^{-at}}$

— **Saisonnalité** S_t :C'est une propriété d'une série temporelle présentant un comportement périodique de période 'p' (semaine/week-end, été/hiver) qui se répète à une fréquence constante.

— **Bruit** ε_t :Le bruit statistique correspond à des fluctuations irrégulières, aléatoires et inexpliquées, On l'appelle parfois résidu

2.5 Opérateurs sur les séries temporelles

— **L'opérateur Retard (Backward)** : L'opérateur retard est un oprateur linéaire notée B qui fait passer de X_t à X_{t-1} :

$$BX_t = X_{t-1}$$

On a :

$$B^2X_t = B(BX_t) = B(X_{t-1}) = X_{t-2}$$

Par une relation de récurrence, on trouve :

$$B^nX_t = B(B(...(BX_t))) = X_{t-n}$$

— **L'opérateur Avance (Forward)** : L'opérateur d'avance est un oprateur linéaire notée F qui fait passer de X_t à X_{t+1} :

$$FX_t = X_{t+1}$$

Par une relation de récurrence, on trouve :

$$F^nX_t = X_{t+n}$$

*Ces opérateurs sont inversibles tels que : $F^{-1} = B$ et $B^{-1} = F$

— **L'opérateur différence ordinaire**

On note ∇ opérateur de différence ordinaire tel que :

$$\nabla X_t = (1 - B)X_t = X_t - X_{t-1}$$

On définit le d$^{\text{ime}}$ opérateur de différence ordinaire par : $\nabla^d X_t = (1 - B)^d X_t$

Elimination de la tendance :
Cet opérateur permet d'éliminer la composante tendancielle sans la calculer. Par exemple pour un processus de la forme : $X_t = a + bt + \epsilon_t$ on a :
$\nabla X_t = b + \epsilon_t - \epsilon_{t-1}$

— **Opérateur de différence saisonnière** On note ∇_s l'opérateur de différence saisonnière tel que :

$$\nabla_s X_t = \left(1 - B^S\right) X_t = X_t - X_{t-s}$$

On définit le $d^{\text{ème}}$ opérateur de différence saisonnière par : $\nabla_S^d X_t = \left(1 - B^S\right)^d X_t$

2.6 Détermination des composantes T_t, S_t

Moyenne mobile de X_t :
Une moyenne mobile d'ordre 'm' pour X_t est définie par :

— **si** $m = 2k + 1$

$$M_m(X_t) = \frac{1}{2k+1} \sum_{j=-k}^{k} X_{t-j}$$

— **si** $m = 2k$

$$M_m(X_t) = \frac{1}{2k}\left(\sum_{j=-(k-1)}^{(k-1)} X_{t-j} + \frac{X_{t-k} + X_{t+k}}{2}\right)$$

-On obtient donc une nouvelle série $\widetilde{X_t} = M_m(X_t)$, On va montrer que dans cette nouvelle serie $\widetilde{X_t}$ l'effet saisonnière disparaît
En effet, a partir de $X_t = T_t + S_t + \epsilon_t$, on applique la moyenne mobile d'ordre m, M_m

$$M_m(X_t) = M_m(T_t + S_t + \epsilon_t)$$

M_m est un opérateur linéaire, donc :

$$M_m(X_t) = M_m(T_t) + M_m(S_t) + M_m(\epsilon_t).$$

Sous la condition : $\sum_{j=1}^{p} S_j = 0$, on a :

— si p=m=2k+1

$$M_m(S_t) = \frac{1}{m}(S_{t-k} + S_{t-k+1} + ... + S_{t+k})$$

or on a

$$S_{t-k} + S_{t-k+1} + ... + S_{t+k} = S_1 + S_2 + ... + S_p = 0$$

D'où $M_m(S_t) = 0$

— si p=m=2k

$$M_m(S_t) = \frac{1}{m}(\frac{1}{2}S_{t-k} + S_{t-k+1} + ... + S_{t+k-1} + \frac{1}{2}S_{t+k})$$

$$M_m(S_t) = \frac{1}{m}(S_{t-k} + S_{t-k+1} + ... + S_{t+k-1})$$

D'où $M_m(S_t) = 0$

2.7 Bruit blanc fort

-Définition
Un processus de bruit blanc fort est une suite de variables aléatoires$(\epsilon_t)_t$ indépendantes et identiquement distribuées (i.i.d.), d'espérance nulle et de variance constantes.

— $E[\epsilon_t] = 0 \quad \forall t$

— $Var[\epsilon_t] = \sigma^2 \quad \forall t$

2.8 Processus stationnaire

-Définition
-Un processus aléatoire $(X_t)_{t\in T}$ est dit faiblement stationnaire s'il est d'espérance constante et si sa fonction covariance ne dépend que de la différence en temps, i.e :

— $E[X_t] = \mu \quad \forall t$

— $Cor[X_t, X_{t+h}] = \sigma(h) \quad \forall t$

-Un processus aléatoire $(X_t)_{t\in T}$ est dit strictement stationnaire si les lois de dimensions finis sont invariante par translation

— $\mathcal{L}(X_{t_1}, \ldots, X_{t_n}) = \mathcal{L}(X_{t_1+h}, \ldots, X_{t_n+h}), \forall (t_1, \ldots, t_n) \in \mathbb{Z}^n$ et $h \in \mathbb{Z}$

2.9 Processus Autorégressifs

-**Définition 1**

On dit que (X_t) est un processus autorégressif d'ordre p (AR(p)) s'il s'écrit

$$X_t = \epsilon_t + \sum_{j=1}^{p} a_j X_{t-j}$$

où ϵ_t est un bruit blanc fort, et pour $s < t$:

$$E[\epsilon_t X_s] = 0$$

— Dans un AR(p) X_t s'exprime en fonction de son passé (d'ordre p) avec un bruit qui arrive a l'instant t.

— $E[\epsilon_t X_s] = 0$ pour $s < t$ signifie que l'erreur a l'instant t n'a pas d'effet sur les valeurs du passé du processus

- **Définition 2**

Le polynôme caractéristique d'un processus autorégressif est définie par :

$$\pi(z) = z^p - a_1 z^{(p-1)} - ..... - a_{p-1} z - a_p$$

-**Théorème (condition de stationnarité)**

Le processus (X_t) est faiblement stationnaire ssi les racines de $\pi(z)$ sont dans le disque d'unité $D(O(0;0),1)$

Chapitre 3

Prévisions des séries temporelles par les réseaux de neurones

3.1 Introduction

L'analyse des séries temporelles a évolué au fil de l'histoire, passant des méthodes statistiques classiques aux modèles plus avancés basés sur l'intelligence artificielle. Les principales étapes comprennent l'introduction de la méthode des moindres carrés, le développement de la méthode de Box-Jenkins, l'utilisation des modèles ARIMA, l'introduction des modèles ARCH, et l'émergence des réseaux de neurones pour la modélisation et la prévision.Aujourd'hui, l'analyse des séries temporelles continue d'évoluer, avec des approches plus sophistiquées basées sur l'intelligence artificielle, L'utilisation de modèles plus avancés tels que les réseaux de neurones récurrents (RNN) et les réseaux de neurones convolutifs (CNN) . Ces avancées ouvrent de nouvelles possibilités pour comprendre les modèles temporels complexes et réaliser des prévisions précises dans divers domaines d'application.

Nous allons traiter dans ce chapitre la modélisation et la prévision des séries temporelles après avoir détaillé le cas général d'un réseau de neurones nous reviendrons sur une méthodologie complète pour l'estimation des paramètres.

3.2 Réseaux de neurones appliqués aux séries temporelles [18]

le modèle correspond à un modèle de régression non linéaire dont la conception s'inspire des propriétés biologiques neuronales. Plus précisément, nous utilisons un modèle NAR "Neural AutoRegressive" ou "Time Delayed Neural Network" définis par :

$$Y_t = f_w(Y_{t-1}, ..., Y_{t-p}) + \epsilon_t \tag{3.1}$$

— $Y_t \in \mathbb{R}$

— $w = (\beta_{ij}, \alpha_j)$ le vecteur paramètre. Notons que sa dimension est $m = (p + 2)K + 1$

— f_w : la fonction représentée par le perceptron multicouche avec une seule unité de sortie

— Y_{t-i}, $i = 1, ..., p$ sont les retards de la série (Y_t)

— ϵ_t est un bruit blanc fort par i.i.d par exemple une variable $N(0, \sigma^2)$, indépendante du passé de la série.

On considère dans la suite un (p, K)-perceptron multi-couches avec :

— p : nombres d'unités linéaires sur la couche d'entrée

— K : nombres d'unités sur la couche cachée munies d'une fonction d'activation Φ

La forme de la fonction $f_w(.)$ devient :

$$f_w(Y_{t-1}, ..., Y_{t-p})) + \epsilon_t = \Phi\left(\alpha_0 + \sum_{j=1}^{K} \alpha_j * \Phi(\sum_{i=1}^{p} \beta_{ij} Y_{t-i} + \beta_{0j})\right) + \epsilon_t \tag{3.2}$$

où Φ représente la fonction d'activation, le modèle 3.1 s'écrit donc sous la forme suivante :

$$Y_t = \Phi\left(\alpha_0 + \sum_{j=1}^{K} \alpha_j * \Phi(\sum_{i=1}^{p} \beta_{ij} Y_{t-i} + \beta_{0j})\right) + \epsilon_t$$

le schéma ci-dessous montre le modèle (NAR) :

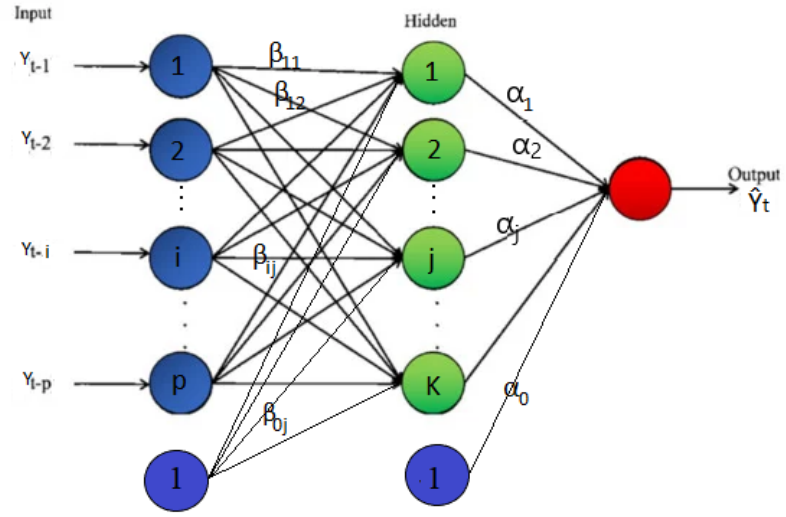


FIGURE 3.1 – Le modèle NAR

— $\hat{Y}_t = f_w(Y_{t-1}, ..., Y_{t-p})$
— $\beta_{ij}, 1 \leqslant i \leqslant p, 1 \leqslant j \leqslant K$:est le paramètre correspondant au poids de la connexion entre l'unité d'entrée i et l'unité caché j.
— $\alpha_j, 1 \leqslant j \leqslant K$:le paramètre correspondant au poids de la connexion entre l'unité caché j et l'unité de sortie.
— $\beta_{0j}, 1 \leqslant j \leqslant K$:le poids de la connexion entre la constante de la couche d'entrée et l'unité caché j
— α_0 :le poids de la connexion entre la constante de la couche caché et la couche de sortie

3.3 Estimation et sélection du modèle

On cherche a estimer les poids α_j et β_{ij} en minimisant $E(w)$ (la somme des carrées résiduels de la fonction d'erreur)

$$E(w) = \frac{1}{2} \sum_{t=1}^{T} (Y_t - f_w(Y_{t-1}, ..., Y_{t-p}))^2 \tag{3.3}$$

$$= \frac{1}{2} \sum_{t=1}^{T} (Y_t - \hat{Y}_t)^2. \tag{3.4}$$

on note $\hat{W} = argminE(w)$ l'estimateur des moindres carrées de w.
Il existe plusieurs méthodes pour minimiser $E(w)$ par exemple l'algorithme de BFGS (Broyden–Fletcher–Goldfarb–Shanno) , l'algorithme de newton et l'algorithme de retropropagation du gradient.

3.3.1 Algorithme de retropropagation du gradient

on applique l'algorithme de rétropropagation pour le modele (NAR) de (1.4) on a

$$a^{(0)} = (Y_{t-1}.....Y_{t-p}) \to z^{(1)} \xrightarrow{\Phi} a^{(1)} \to z^{(2)} \xrightarrow{\Phi} a^{(2)} = \hat{Y}_t$$

avec
$z_j{}^{(1)} = \sum_{i=1}^{p} \beta_{ij} \cdot a_i^{(0)} + \beta_{0j}\,, \qquad 1 \leqslant j \leqslant K.$

$z^{(2)} = \sum_{j=1}^{K} \alpha_j \cdot a_j^{(1)} + \alpha_0$
les valeurs du vecteur $w = (\beta_{ij}, \alpha_j)$ qui minimisent la fonction E(w) sont données par les formules suivantes :

$$\alpha_j^{new} = \alpha_j^{old} - \mu \cdot \frac{\partial E}{\partial \alpha_j} \tag{3.5}$$

$$\beta_{ij}^{new} = \beta_{ij}^{old} - \mu \cdot \frac{\partial E}{\partial \beta_{ij}} \tag{3.6}$$

où $0 < \mu < 1$,
-On répète ce processus jusqu'a la convergence

Proposition 3.3.1. *les valeurs de gradients qui définie la méthode de descente du gradient sont données par les formules suivants :*

$$\frac{\partial E}{\partial \alpha_j} = \delta_1^{(2)} \times a_j^{(1)}$$

$$\frac{\partial E}{\partial \beta_{ij}} = \delta_j^{(1)} \times a_j^{(0)}$$

et :

$$\delta_1^{(2)} = -a \cdot (Y_1 - a z_1^{(2)} - b)$$

$$\delta_j^{(1)} = \delta_1^{(2)} \cdot \alpha_j \cdot \Phi(z_j^{(1)})(1 - \Phi(z_j^{(1)}))$$

Démonstration.

$$\begin{aligned} \frac{\partial E}{\partial \alpha_j} &= \frac{\partial E}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial \alpha_j} \\ &= \delta_1^{(2)} \times a_j^{(1)}. \end{aligned}$$

Ainsi :

$$\begin{aligned} \delta_1^{(2)} &= \frac{\partial}{\partial z_1^{(2)}} \frac{1}{2} (Y_1 - \hat{Y}_1)^2 \\ &= \frac{\partial}{\partial z_1^{(2)}} \frac{1}{2} (Y_1 - (a z_1^{(2)} + b))^2 \quad \forall a, b \in \mathcal{R} \\ \delta_1^{(2)} &= -a \cdot (Y_1 - a z_1^{(2)} - b). \end{aligned}$$

En remplaçant dans 3.6 :

$$\alpha_j^{new} = \alpha_j^{old} + \mu \cdot a \cdot (Y_1 - az_1^{(2)} - b) \times a_j^{(1)} \tag{3.7}$$

D'autre part :

$$\begin{aligned} \frac{\partial E}{\partial \beta_{ij}} &= \frac{\partial E}{\partial z_j^{(1)}} \times \frac{\partial z_j^{(1)}}{\partial \beta_{ij}} \\ &= \delta_j^{(1)} \times a_j^{(0)}. \end{aligned}$$

Ainsi :

$$\begin{aligned} \delta_j^{(1)} &= \frac{\partial}{\partial z_j^{(1)}} \frac{1}{2}(Y_1 - \hat{Y}_1)^2 \\ &= \frac{\partial}{\partial z_j^{(1)}} \frac{1}{2}(Y_1 - \Phi(z_1^{(2)}))^2 \\ &= \frac{1}{2} \frac{\partial}{\partial z_1^{(2)}} (Y_1 - \Phi(z_1^{(2)}))^2 \cdot \frac{\partial z_1^{(2)}}{\partial z_j^{(1)}} \\ &= \delta_1^{(2)} \cdot \frac{\partial}{\partial z_j^{(1)}} \sum_{i=1}^{K} \alpha_i \cdot a_i^{(1)} + C_2 \\ &= \delta_1^{(2)} \cdot \frac{\partial}{\partial z_j^{(1)}} \sum_{i=1}^{K} \alpha_i \cdot \Phi(z_i^{(1)}) + C_2 \\ &= \delta_1^{(2)} \cdot \alpha_j \cdot \Phi'(z_j^{(1)}) \\ \delta_j^{(1)} &= \delta_1^{(2)} \cdot \alpha_j \cdot \Phi(z_j^{(1)})(1 - \Phi(z_j^{(1)})) \end{aligned}$$

où :

$$\begin{aligned} \Phi(x) &= \frac{1}{1 + \exp(-x)} \\ \Phi'(x) &= \Phi(x) \cdot (1 - \Phi(x)). \end{aligned}$$

En remplaçant dans 3.6 :

$$\beta_{ij}^{new} = \beta_{ij}^{old} - \mu \cdot \delta_1^{(2)} \cdot \alpha_j \cdot \Phi(z_j^{(1)})(1 - \Phi(z_j^{(1)})) \times a_j^{(0)} \tag{3.8}$$

□

3.3.2 MLP(Multi Layer Perceptron) réductible

[18] Un MLP réductible est un type de réseau de neurones artificiels dans lequel les connexions entre les neurones sont organisées de manière à permettre une réduction du nombre de neurones et de connexions lors de la formation du réseau. Cela signifie que certaines parties du réseau de neurones n'ont pas d'impact sur la sortie finale, et donc peuvent être supprimées sans affecter la performance du modèle. La réductibilité est un concept important pour la simplification et la compréhension des réseaux de neurones, ainsi que pour leur optimisation en termes de temps de calcul et de capacité de stockage.

-Pour le modèle (3.2), avec $\phi(x) = \tanh(x)$:

Notation 1 : Si $Y = \left(Y_1, \cdots, Y_p\right)^T \in \mathbb{R}^p$ est un vecteur d'entrée, on note $\nu_j(Y)$ l'impulsion de la j-ème unité cachée (la valeur de préactivation) :

$$\nu_j(Y) = z_j^{(1)} = \beta_{0j} + \sum_{i=1}^{p} \beta_{ij} Y_i$$

On dira que deux fonctions affines ν_1, ν_2 sont "signe-équivalentes" si $\nu_1 = \nu_2$ ou $\nu_1 = -\nu_2$.

On note $\sigma(\nu_j(Y))$La sortie de l'impulsion de la j-ème unité cachée :

$$\sigma(\nu_j(Y)) = tanh(\nu_j(Y))$$

on note $\mu(Y)$ l'impulsion de la couche de sortie :

$$\mu(Y) = z_1^{(2)} = \alpha_0 + \sum_{j=1}^{K} \alpha_j \sigma(\nu_j(Y))$$

On dira qu'un MLP est réductible si et seulement si il vérifie au moins une des conditions suivantes :
1. Il existe un poids de sortie nul.
2. Il existe au moins deux indices différents $j_1, j_2 \in \{1, \cdots, K\}$ tels que les fonctions ν_{j_1}, ν_{j_2} soient signe-équivalentes
3. Il existe au moins un indice $j \in \{1, \cdots, K\}$ tel que la fonction ν_j soit constante.
On dira qu'un MLP est irréductible si ce n'est pas un MLP réductible.

Notation 2 : On note $\mathcal{N}_{p,K}$ l'ensemble des MLP avec p entrées et K unités cachées , qui sont irréductible

Notation 3 : Supposons que nous ayons deux réseaux de neurones N_1 et N_2 qui sont I-O équivalents, ce qui signifie qu'ils produisent la même sortie pour chaque entrée donnée. Nous voulons prouver que les réseaux N_1 et N_2 sont essentiellement les mêmes, à l'exception de certaines symétries internes.

Theoreme 1 : [9]
Soit N_1 et N_2 deux réseaux de neurones irréductibles et I-O équivalents, dans $\mathcal{N}_{p,K_1}$ et $\mathcal{N}_{p,K_2}$ Alors :
(i) $K_1 = K_2$
(ii) N_1 et N_2 sont équivalents

Lemme :
Soit J un ensemble fini et soit $\left\{(\varphi_j)\right\}_{j\in J}$ une famille de fonctions affines linéaires non constantes sur $\mathbb{R}^p$, dont aucune deux fonctions sont signe-équivalente. Alors les fonctions $\sigma \circ \varphi_j$, ainsi que la fonction constante 1, sont linéairement indépendantes.

Preuve du Theoreme 1 :
Supposons que nous ayons les réseaux N_1 et N_2 avec K_1 et K_2 nœuds cachés respectivement.. Notre hypothèse est que $\sigma(\mu^1(Y)) = \sigma(\mu^2(Y))$ pour tout x. Étant donné que σ est une fonction injective, cela implique que $\mu^1(Y) = \mu^2(Y)$ pour tout x. Ainsi, l'identité suivante est vérifiée :

$$\alpha_0^1 + \sum_{i=1}^{K_1} \alpha_j^1 \sigma(\nu_j^1(Y)) = \alpha_0^2 + \sum_{i=1}^{K_2} \alpha_j^2 \sigma(\nu_j^2(Y)) \tag{3.9}$$

Soit $J = 1, 2, \ldots, K_1 + K_2$ on definit :

$$\varphi_j(Y) = \begin{cases} \nu_j^1(Y) & \text{si} \quad 1 \leqslant j \leqslant K_1 \\ \nu_{j-K_1}^2(Y) & \text{si} \quad K_1 + 1 \leqslant j \leqslant K_1 + K_2 \end{cases}$$

De plus, posons

$$a_j = \begin{cases} \alpha_0^1 - \alpha_0^2 & \text{si} \quad j = 0 \\ \alpha_j^1 & \text{si} \quad 1 \leqslant j \leqslant K_1 \\ -\alpha_{j-K_1}^2 & \text{si} \quad K_1 + 1 \leqslant j \leqslant K_1 + K_2 \end{cases}$$

L'équation 3.9 devient :

$$a_0 + \sum_{j \in J} a_j \sigma\left(\varphi_j(x)\right) = 0. \tag{3.10}$$

Supposons que tous les a_j ne sont pas nuls. Alors 3.10 implique que :

(i) soit l'une des φ_j est une constante.

(ii) soit deux des φ_j sont signe-équivalentes.

— La première possibilité (i) est exclue car les deux réseaux considérés sont irréductibles , donc (ii) doit être vraie.

— Cependant, étant donné que deux φ_j provenant du même réseau ne peuvent pas être signe-équivalentes (selon la Condition (2) de la définition de l'irréductibilité), il doit exister j_1, j_2 tels que $1 \leq j_1 \leq K_1$ et $K_1 + 1 \leq j_2 \leq K_1 + K_2$, tels que φ_{j_1} et φ_{j_2} sont signe-équivalentes. cela signifie que soit :

$$\nu_{j^1}^1 \equiv \nu_{\hat{j}^1}^2 \quad \text{et} \quad \alpha_{j^1}^1 = \alpha_{\hat{j}^1}^2$$

ou bien

$$\nu_{j^1}^1 \equiv -\nu_{\hat{j}^1}^2 \quad \text{et} \quad \alpha_{j^1}^1 = -\alpha_{\hat{j}^1}^2$$

Avec : $j^1 = j_1$ et $\hat{j}^1 = j_2 - K_1$ En utilisant cela, nous pouvons réécrire 3.10 en supprimant la contribution de j_1 et j_2, puis appliquer à nouveau le lemme à l'identité obtenue. Cela donnera lieu à une deuxième paire de nœuds cachés $j^2, \hat{j}^2$ telle que :

$$\nu_{j^2}^1 \equiv \nu_{\hat{j}^2}^2 \quad \text{et} \quad \alpha_{j^2}^1 = \alpha_{\hat{j}^2}^2$$

ou bien

$$\nu_{j^2}^1 \equiv -\nu_{\hat{j}^2}^2 \quad \text{et} \quad \alpha_{j^2}^1 = -\alpha_{\hat{j}^2}^2$$

En répétant ce processus, nous obtenons une séquence de paires de nœuds cachés $j^1, j^2, \ldots, j^k, \hat{j}^1, \hat{j}^2, \ldots, \hat{j}^k$ provenant des réseaux N_1 et N_2, respectivement. Chaque paire de nœuds $j^l, \hat{j}^l$ est telle que ν_{j^l} est signe-équivalent à $\nu_{\hat{j}^l}$ et à aucun autre ν_j provenant de N_2. De plus, ces paires de nœuds couvrent tous les nœuds de N_1 et N_2.

Cela nous permet de conclure que les deux réseaux ont le même nombre de nœuds cachés, c'est-à-dire $K_1 = K_2$. De plus, en effectuant des permutations sur les nœuds de N_2, nous pouvons supposer que $\hat{j}^l = j^l$ pour chaque l. Par conséquent, nous avons également

montré que les poids correspondants α^1_{jl} et $\alpha^2_{\hat{j}l}$ sont égaux.

En résumé, nous avons montré que les réseaux N_1 et N_2 sont équivalents, car ils ont le même nombre de nœuds cachés et les poids correspondants sont égaux. Cela démontre les deux parties du théorème :
(i) $K_1 = K_2$
(ii) N_1 et N_2 sont équivalents

3.3.3 L'élagage du nombre de neurones SSM : [18]

La méthode d'élagage SSM (State Space Model) est une méthode qui permet de sélectionner le nombre optimal de neurones dans un réseau de neurones pour modéliser une série temporelle. Cette méthode utilise un critère d'information, tel que le BIC (Bayesian Information Criterion) , l'erreur reltive , l'erreur quadratique pour évaluer la qualité du modèle.
L'idée est de commencer avec un réseau de neurones ayant un grand nombre de neurones et de connexions, puis de l'entraîner sur les données de la série temporelle. Ensuite, on réduit le nombre de neurones en éliminant ceux qui ont le moins d'impact sur la performance du modèle, tout en gardant la performance suffisamment élevée

Le critère BIC (Bayesian Information Criterion)

$$\mathrm{BIC} = \ln \frac{S_T(F)}{T} + \frac{\ln T}{T} m(F).$$

$$\mathrm{BIC}^* = \mathrm{BIC}^*(T, F) = \frac{S_T(F)}{T} + \gamma \frac{\ln T}{T} m(F)$$

— **T** : le nombre d'observations

— $S_T(F)$: le minimum de la fonction d'erreur $E(W)$

— **m(F)** : le nombre de paramètres non nul du modèle

— γ : une constante positive

Théorème 2 (Consistance forte et normalité asymptotique) [16] :

Pour le modèle (3.2), avec $\phi(x) = \tanh(x)$, supposons que :

1. $(\varepsilon_t)_{t>0}$ est une suite i.i.d., indépendante des états initiaux $\left(Y_{-p+1}, \ldots, Y_0\right)$, telle que $E(\varepsilon_t^6) < \infty$
2. W appartient à un sous-ensemble compact Θ de l'espace euclidien $\mathbb{R}^m$ de dimension m, tel que
$$W_0 \in \dot{\Theta}(\text{ intérieur de } \Theta).$$
3. (Condition d'identifiabilité) Pour tout W différent de W_0, il existe un $y \in \mathcal{R}$ tel que
$$f(y, W) \neq f(y, W_0)$$

.

4. La matrice de dimension $m \times m$

$$I_0 := 2 \int_{R^p} {}^t Df(\mathbf{y}, W_0)\, Df(\mathbf{y}, W_0)\, \mu_{W_0}(d\mathbf{y}),$$

est inversible.

Alors :

a) L'estimateur $\hat{W}_T$ est fortement consistant, c'est-à-dire qu'il converge presque sûrement vers W_0 quand T tend vers $+\infty$.

b) Le terme $\sqrt{T} I_0[\hat{W}_T - W_0]$ converge en loi vers la distribution gaussienne multidimensionnelle $\mathcal{N}(0, 2\sigma^2 I_0)$.

Preuve :

Nous regroupons ici l'ensemble des hypothèses qui interviendront successivement.
Dans toute la suite : $\xrightarrow{\text{p.s.}}$ (resp. $\xrightarrow{\mathcal{L}}$) désigne la convergence presque sûre (resp. en loi)

Hypothèse de stabilité d'ordre a :
Soit $a \geq 1$. On suppose que la chaîne $Y^{(p)}$ possède une unique loi invariante μ_{W_0} satisfaisant :
- Pour tout $t \geq 0$ et toute loi initiale, $E_{W_0} \left| X_t^{(p)} \right|^a < \infty$.
- $\mu_{W_0}(|\cdot|^a) := \int_{R^p} |\mathbf{x}|^a \mu_{W_0}(d\mathbf{x}) < \infty$.
- Pour toute fonction Ψ de R^p dans R, et pour $\mu_{W_0} - p.$s. continue, on a une loi forte des grands nombres

$$\frac{1}{n} \sum_{t=1}^{n} \Psi\left(X_t^{(p)}\right) \xrightarrow{p.s.} \int_{R^p} \Psi(\mathbf{x}) \mu_{W_0}(d\mathbf{x}).$$

Cadre des modèles étudiés :

1. On considère une famille de modèles (3.2) :
   a) $(\varepsilon_t)_{t>0}$ est un bruit i.i.d. à valeurs dans R^d, centré, de matrice de covariance Γ, indépendant de l'état initial $Y_0^{(p)}$.
   b) La famille de modèles considérés est identifiée par la famille de fonctions de régression $f(\cdot; W)\}$, toutes de R^p dans R, où le paramètre W appartient à un compact Θ de R^m avec $W_0 \in \dot{\Theta}$

2. Le vrai modèle $f(\cdot; W_0)$ et le bruit (ε_t) satisfont à l'hypothèse de stabilité avec $a \geq 2$.

3. a) Pour tout $W, \mathbf{y} \mapsto f(\mathbf{y}; W)$ est $\mu_{W_0} - p.s.$ continue ;
   b) il existe un module de continuité G tel que, $\forall x \in R^p, \forall (\alpha, \beta) \in \Theta^2$:

$$\|f(\mathbf{y}; \alpha) - f(\mathbf{y}; \beta)\| \leq G(\|\alpha - \beta\|) \left(1 + |\mathbf{y}|^{a/2}\right)$$

4. Pour tout $W \in \Theta$, $f(\cdot; W) = f(\cdot; W_0)\, \mu_{W_0}$-p.s. implique que $W = W_0$.

Nous verrons que dans ce cadre , $\hat{W}_T$ est fortement consistant. Sa normalité asymptotique nécessite des conditions habituelles de régularité sur les dérivées premières et secondes du contraste $E(W)$. Nous notons D_i, D_{ii}^2 les dérivations partielles premières et secondes.

On suppose que les conditions précédentes sont satisfaites , On suppose de plus qu'il existe un voisinage $\mathcal{V}$ de W_0 sur lequel pour tout $y \in \mathcal{R}^d$ la fonction f de $\mathbf{W} \mapsto f(\mathbf{y}; W)$ est deux fois continûment dérivable , pour tout $i, j = 1, \ldots, m$, on ait :

1. Pour tout $W \in V, y \mapsto D_i f(\mathbf{y}; W)$ et $y \mapsto D_{ij}^2 f(\mathbf{y}; W)$ sont $\mu_{W_n} - p.s.$ continues.

2. Pour tout y $\in R^p$.

$$|D_i f(\mathbf{y}; W_0)| \leq \text{cte}\left(1 + |\mathbf{y}|^{a/2}\right) : \quad \left|D^2_{ij} f(\mathbf{y}; W_0)\right| \leq \text{cte}\left(1 + |\mathbf{y}|^{a/2}\right).$$

3. Il existe un module de continuité σ_{ij} tel que

$$\left|D^2_{ij} f(\mathbf{y}; W) - D^2_{ij} f_k(\mathbf{y}; W_0)\right| \leq \sigma_{ij}(\|W - W_0\|)\left(1 + |\mathbf{y}|^{a/2}\right), W \in V, \in R^p.$$

4. Enfin, si $Df(\mathbf{y}; W)$ désigne la matrice $\left[D_j f(\mathbf{x}; W)\right], 1 \leq j \leq m$, nous posons la matrice

$$I_0 := 2\int_{R^p} {}^t Df(\mathbf{y}, W_0)\, Df(\mathbf{y}, W_0)\, \mu_{W_0}(d\mathbf{y}),$$

les Principaux résultats :

— $\hat{W}_T$ est fortement consistant.

— (Normalité asymptotique) : $\sqrt{T} I_0\left[\hat{W}_T - W_0\right] \xrightarrow{\mathcal{L}} \mathcal{N}\left(0, 2\sigma^2 I_0\right)$.

Ces deux résultats s'obtiennent en suivant la théorie classique d'estimation par minimum de contraste. Nous pouvons préciser la vitesse de convergence de $\left(\widehat{W}_T\right)$ en utilisant la loi du logarithme itéré pour les martingales .

Théoreme 3 : (Loi du logarithme itéré) [16]

- On se place dans le cadre du théorème 2. On suppose de plus que :

(i) L'exposant a des hypothèses du théoreme 2 est > 2.

(ii) Les matrices I_0 et J_0 avec $J_0 = 2\sigma^2 I_0$ sont inversibles. Alors, pour tout $u \in R^m, u \neq 0$, on a p.s.

$$\begin{aligned}
&\limsup_T \sqrt{\frac{T}{2\ln\ln T}}\,\langle D\,E(W_0), u\rangle = \sqrt{{}^t u J_0 u} \\
&= -\liminf_T \sqrt{\frac{n}{2\ln\ln T}}\,\langle D\,E(W_0), u\rangle, \\
&\limsup_T \sqrt{\frac{T}{2\ln\ln T}}\left\langle \widehat{W}_T - W_0, u\right\rangle = \sqrt{{}^t u I_0^{-1} J_0 I_0^{-1} u} \\
&= -\liminf_T \sqrt{\frac{T}{2\ln\ln T}}\left\langle \widehat{W}_T - W_0, u\right\rangle.
\end{aligned}$$

Proposition 3.3.2. *On suppose que les conditions du* ***Théorème 2*** *sont vérifiées. On suppose aussi le taux de pénalisation $c(T)$ est tel que*

$$\lim_T \frac{c(T)}{T} = 0, \quad \textit{et} \quad \liminf_T \frac{c(T)}{2\ln\ln T} > \sigma^2 \frac{\Lambda}{\lambda}$$

où Λ (resp. λ) est la plus grande (resp. la plus petite) valeur propre de la matrice I_0. Alors, le couple $\left(\hat{F}_T, \hat{W}_{T,\hat{F}_T}\right)$ converge presque sûrement vers la vraie valeur (F_0, W_0), quand T tend vers ∞.

3.4 Conclusion

Nous avons exploré le modèle neuronal autoregressif et les théorèmes associés à son estimation, en mettant en évidence l'utilisation du MLP réductible. Ce modèle présente des avantages significatifs dans l'analyse et la prédiction des séries temporelles. En utilisant le MLP réductible, nous avons pu réduire le nombre de connexions et de paramètres du modèle, ce qui améliore l'efficacitéet la capacité de généralisation du modèle. En conclusion, le modèle neuronal autoregressif avec MLP réductible offre une approche prometteuse pour l'analyse et la prédiction des séries temporelles. Son utilisation nous permet d'obtenir des résultats fiables et précis.

Chapitre 4

Applications et résultats

4.1 Données de Cornell

Dans cette partie, nous allons appliquer la méthode de rétropropagation du gradient à l'exemple des données de Cornell telles qu'introduites dans l'étude de Kettaneh-Wold (1992) [15] et reprises par Tenenhaus (1998) [20]. Nous procéderons ensuite à une comparaison des différentes méthodes discrètes mentionnées dans la littérature [13]. Les données utilisées comprennent douze échantillons d'essence, avec leur composition respective en proportions, comme présenté dans le tableau 4.1. Notre objectif est de déterminer l'influence du mélange sur l'indice d'octane y, en utilisant l'algorithme du perceptron multicouche (MLP) et le package **neuralnet** du langage **R**.

Description des variables :

y = Indice d'octane
x_1 = Distillation directe (compris entre 0 et .21)
x_2 = Réformat (compris entre 0 et 0.62)
x_3 = Naphta de craquage thermique (compris entre 0 et 0.12)
x_4 = Naphta de craquage catalytique (compris entre 0 et 0.62)
x_5 = Polymére (compris entre 0 et 0.12)
x_6 = Alkylat (compris entre 0 et 0.74)
x_7 = Essence naturelle (compris entre 0 et 0.08)

Les douze mélange ont été choisis selon un plan d'expériences D-optimal, en selectionnant une partie des sommets, les données representent des proportions et la somme des variables de X_1 à X_7 est égale à 1 :

X_1	X_2	X_3	X_4	X_5	X_6	X_7	Y
0.00	0.23	0.00	0.00	0.00	0.74	0.03	98.7
0.00	0.10	0.00	0.00	0.12	0.74	0.04	97.8
0.00	0.00	0.00	0.10	0.12	0.74	0.04	96.6
0.00	0.49	0.00	0.00	0.12	0.37	0.02	92
0.00	0.00	0.00	0.62	0.12	0.18	0.08	86.6
0.00	0.62	0.00	0.00	0.00	0.37	0.01	91.2
0.17	0.27	0.10	0.38	0.00	0.00	0.08	81.9
0.17	0.19	0.10	0.38	0.02	0.06	0.08	83.1
0.17	0.21	0.10	0.38	0.00	0.06	0.08	82.4
0.17	0.15	0.10	0.38	0.02	0.10	0.08	83.2
0.21	0.36	0.12	0.25	0.00	0.00	0.06	81.4
0.00	0.00	0.00	0.55	0.00	0.37	0.08	88.1

TABLE 4.1 – Données de Cornell

Le modèle s'écrit sous la forme suivante :

$$\widehat{Y} = f_w(X_1, X2, ..., X_7),$$

où f_x est la fonction représentée par le perceptron multicouche avec une seule unité de sortie. **La Figure**4.1 illustre la structure du réseau neuronal, montrant les connexions entre les différentes couches de neurones et les poids associés.

Les poids entre les nœuds du graphe représentent les connexions entre les neurones du réseau neuronal, où chaque poids indique l'influence d'un neurone sur les neurones suivants.

Les résultats obtenus indiquent que le réseau neuronal a été entraîné pendant 21 itérations. À la fin de la 21ème itération, l'erreur quadratique moyenne obtenue était de 0, 02875. Ce résultat suggère que le modèle a convergé vers une solution acceptée.

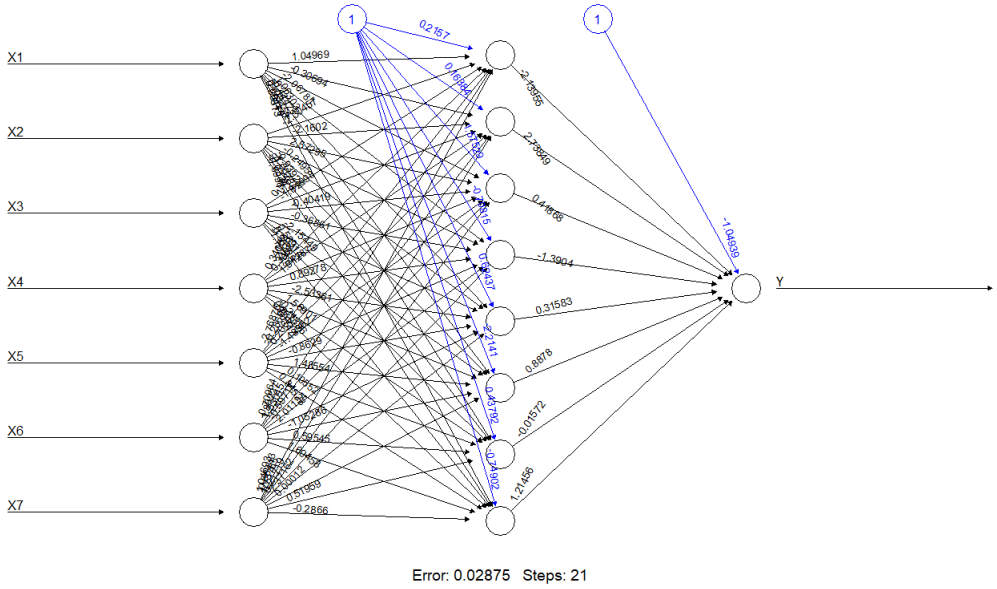


FIGURE 4.1 – Réseaux de neurones pour l'exemple de cornell

De plus, les résultats montrent que, pour le mélange spécifique utilisé, l'indice d'octane prédit est représenté par le vecteur suivant :

$$\hat{Y} = \begin{pmatrix} 95.06931 \\ 97.01476 \\ 96.87128 \\ 96.46847 \\ 94.26758 \\ 91.40246 \\ 82.27269 \\ 82.59689 \\ 82.35070 \\ 82.68692 \\ 82.10243 \\ 90.08795 \end{pmatrix}$$

4.2 Exemple de la température d'el Niño

Nous considérons la série chronologique décrivant le phénomène climatologique, "ENSO", qui résulte des interactions entre l'atmosphére et l'océan au dessus de l'Océan Pacifique tropical. Le phénomène EL Niño ("EN") est la composante océan dans ("ENSO") tandis que Southern Oscillation ("SO") est la composante atmosphère .
Un index mesurant la variabilité d'EL Niño, est fourni par les températures à la surface de l'océan ramenées à une moyenne observée dans le domaine Niño-1 ($5^oS - 5^oN, 150^oW - 90^oW$). Des valeurs moyennes mensuelles sont enregistrées depuis le mois de Janvier 1950 à février 2023 par des centres nationaux de la prévision environnementale aux États-Unis.
La série chronologique de cet index montre des variations interannuelles marquées et superposées à une composante saisonnière forte, cette série a été analysée par beaucoup d'auteurs (voir par exemple [12], ([4], [5], [11]).

Nous considérons les données historiques de la température d'el Niño(site web //www.cpc. ncep. noaa. gov / products / analysis_monitoring /sacoches
/ ensoyears.shtml), et pour comparer la performance de notre prédicteur avec les autres méthodes de prédiction existant dans la littérature (voir ([4], [5], [11]), [12]), on considère la prédiction de l'année de référence 2006 , en utilisant l'algorithme du perceptron multicouche (multilayer perceptron MLP en anglais) et le package **nnfor** du langage **R**.
On note par "**Adata**", les données de la température (EL Niño) entre 1950 et 2006 , on importe les données et on commence par représenter cette série temporelle, comme le montre la Fig. 4.2
la Fig. 4.3 représente les variations de la température d'EL Niño au cours de l'année 2006.

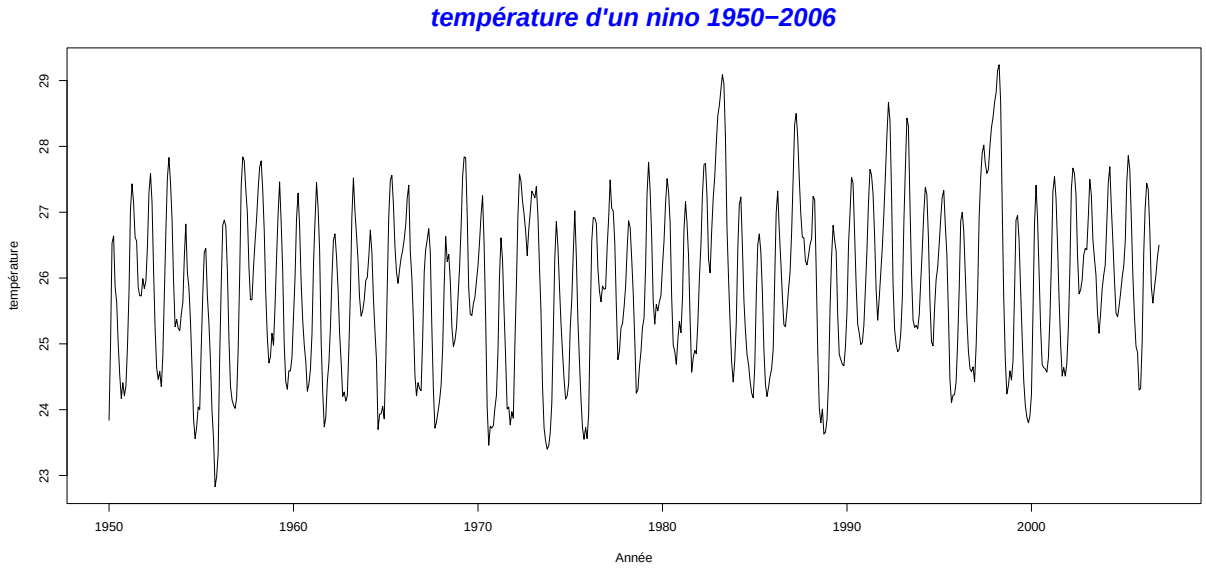


FIGURE 4.2 – Température d'el Nino (1950 – 2006)

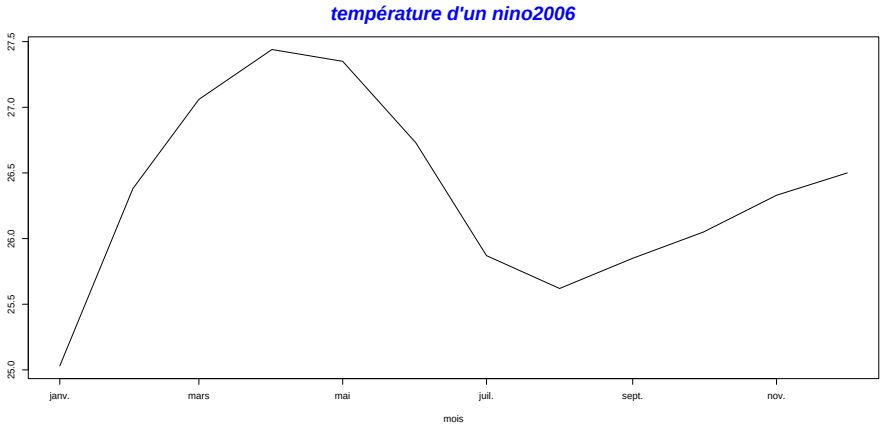


FIGURE 4.3 – Température d'el Nino (2006)

4.2.1 Analyse de la série temporelle

Tendance et Saisonnalité

```
>plot(decompose(Adata))
```

le résultat est affiché sous la forme suivante

La Fig 4.4 montre les quatre éléments suivants :

— La série temporelle originale, qui est tracée sur le premier graphique.

— La tendance, qui est tracée sur le deuxième graphique, (La tendance représente la direction générale de la série temporelle sur une longue période).

— La saisonnalité, qui est tracée sur le troixième graphique, (La saisonnalité représente les variations régulières qui se produisent dans la série temporelle en fonction du temps).

— Les résidus, qui sont tracés sur le dernier graphique, (Les résidus représentent les variations aléatoires de la série temporelle qui ne peuvent pas être expliquées par la tendance et la saisonnalité).

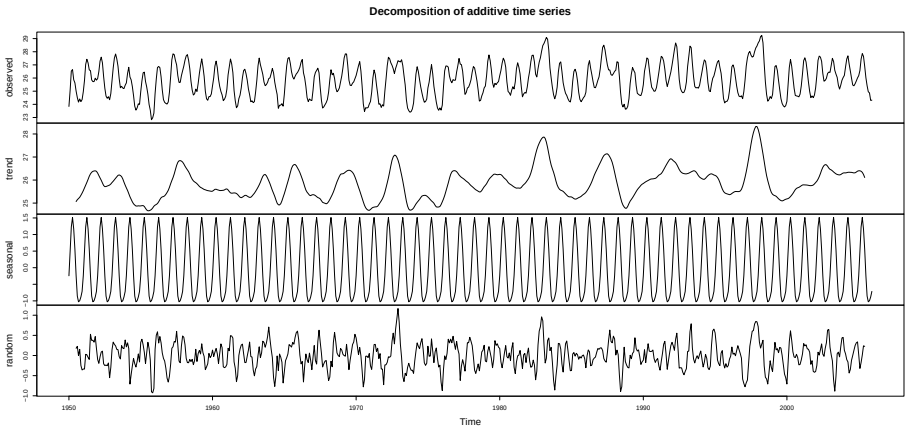


FIGURE 4.4 – Décomposition de la série

Pour confirmer les résultats graphique, on utilise :

— **le "trendtest"** : la méthode statistique qui permet de tester l'existence d'une tendance dans une série chronologique, en fournissant une estimation de la probabilité que la tendance observée soit due au hasard.

— **seasplot** : une fonction du package R "seasonal" qui permet de tracer un graphique des saisons d'une série chronologique. Cette fonction est souvent utilisée pour visualiser les variations saisonnières dans les données et pour aider à identifier les modèles saisonniers dans la série chronologique.

```
>trendtest(Adata, extract=TRUE)
 TRUE
```

le résultat est "TRUE" donc la tendance existe.
pour la saisonalité on utilise la commande suivante

```
>seasplot(Adata)
 Results of statistical testing
Evidence of trend: TRUE  (pval: 0)
Evidence of seasonality: TRUE  (pval: 0)
```

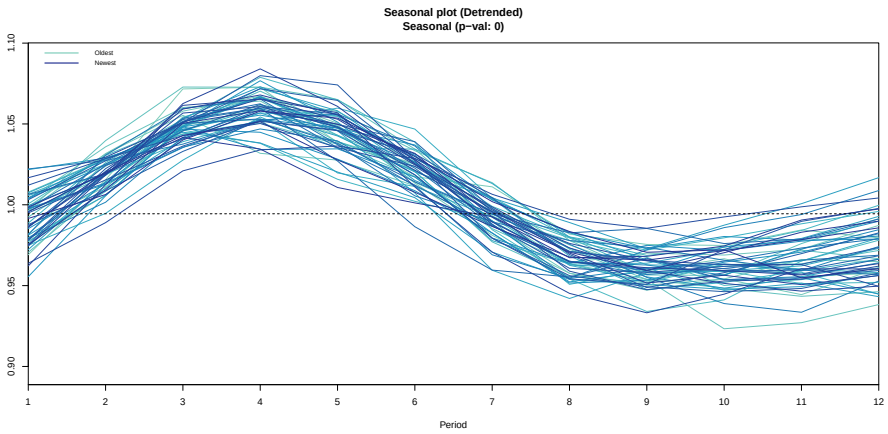


FIGURE 4.5 – es variations saisonnières pour l'année 2006

Le graphique résultant , Fig. 4.5, montre les variations saisonnières dans les données, avec des valeurs plus élevées pour le mois d'avril et des valeurs plus faibles pour le moi de septembre.

Cela peut aider à identifier les modèles saisonniers dans les données et à comprendre la dynamique de la série chronologique.

```
>seasplot(Adata, outplot=2)
 Results of statistical testing
Evidence of trend: TRUE  (pval: 0)
Evidence of seasonality: TRUE  (pval: 0)
```

Le graphique résultant Fig. 4.6 montre les boîtes pour chaque période saisonnière de l'année, avec les valeurs médianes, les quartiles et les valeurs aberrantes pour chaque période. Cela permet de visualiser les variations saisonnières dans les données après avoir enlevé la tendance et les effets saisonniers, et peut aider à identifier les anomalies ou les points atypiques dans les données.

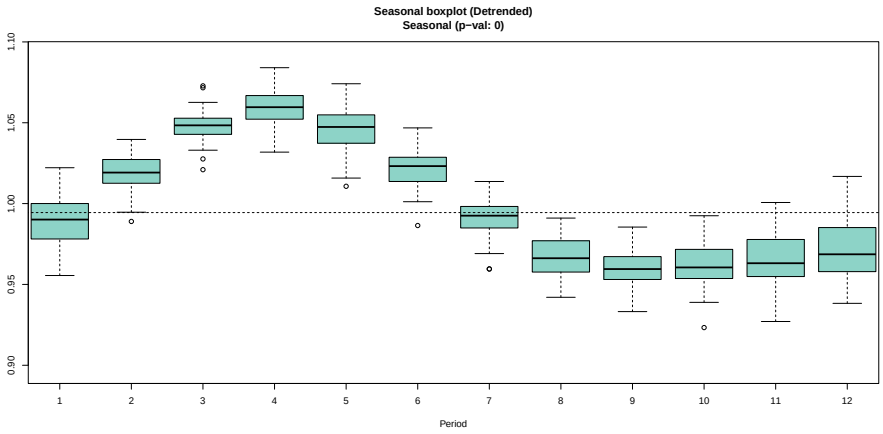


FIGURE 4.6 – Les boîtes pour chaque période saisonnière de l'année 2006

4.2.2 Prévision avec MLP

La base de données de la température d'El Ninio, a été divisée en deux sous ensembles, le premier sert à effectuer l'apprentissage "**Train Region**" et le deuxième sous ensemble est utilisé pour tester les performances du Prédicteur, c'est ce qu'on appelle "**Test Region**".
Notons que l'ensemble d'apprentissage contient 672 mois, ce qui fais une période de 56 ans allant du mois de janvier 1950 au mois de décembre 2006 et l'ensemble test contient 12 mois ce qui donne une période d'une année allant du mois de janvier 2006 au mois de décembre 2006, comme le montre la Fig.4.7 :

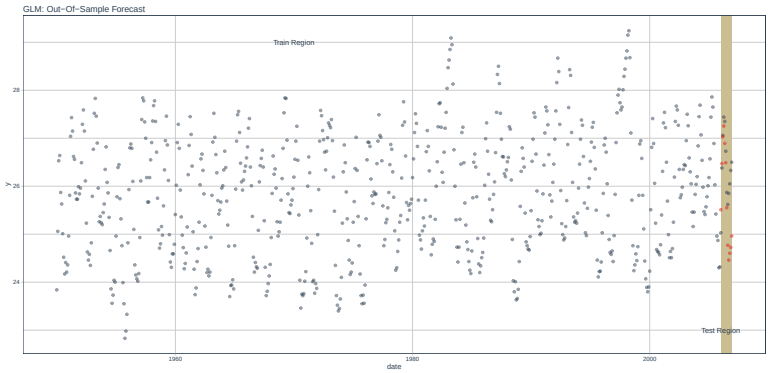


FIGURE 4.7 – L'ensemble d'apprentissage

pour appliquer la methode MLP , il suffit d'entrer la série temporelle à modéliser.mlp()

```
> a<-mlp(Adata)
 > print(a)
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1, 2, 4, 7, 9, 10, 12)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 0.0381.
```

La sortie indique que le réseau résultant a 5 nœuds cachés, il a été formé 20 fois, et les différentes prévisions ont été combinées en utilisant l'opérateur médian.
La fonction génère automatiquement des ensembles de réseaux, dont l'entrainement commence par différents poids initiaux aléatoires, de plus, il fournit les entrées qui ont été incluses dans le réseau, l'erreur obtenu est 3.81%.
On peut obtenir un résumé visuel en utilisant la fonction **plot()** :

```
 > plot(a)
```

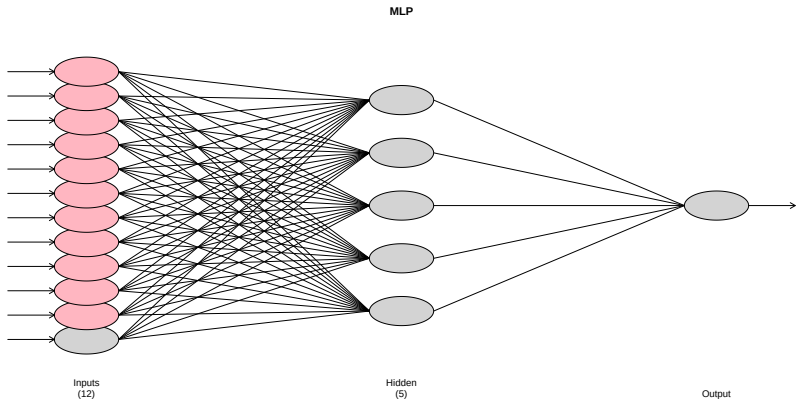


FIGURE 4.8 – Réseaux de neurones avec une couche cachée

Les nœuds gris en entrée sont des autorégressions, tandis que les nœuds roses sont des entrées déterministes (saisonnalité dans ce cas). Si d'autres régresseurs étaient inclus : ils seraient affichés en bleu clair.
pour effectuer une prévision de la trajectoire de la température d'EL NINIO de l'année 2006, nous avons utilisé la fonction "forecast()" et nous avons affiché les résultats de la prévision à l'aide de la fonction "plot()".

```
 > frc<-forecast(a)
 > print(frc)
          Jan      Feb      Mar      Apr      May
2006 25.08098 25.95263 26.66820 26.99089 26.74060
          Jun      Jul      Aug      Sep      Oct
2006 26.09377 25.33319 24.75524 24.69801 24.86875
          Nov      Dec
2006 24.94809 25.22820

 >plot(frc)
```

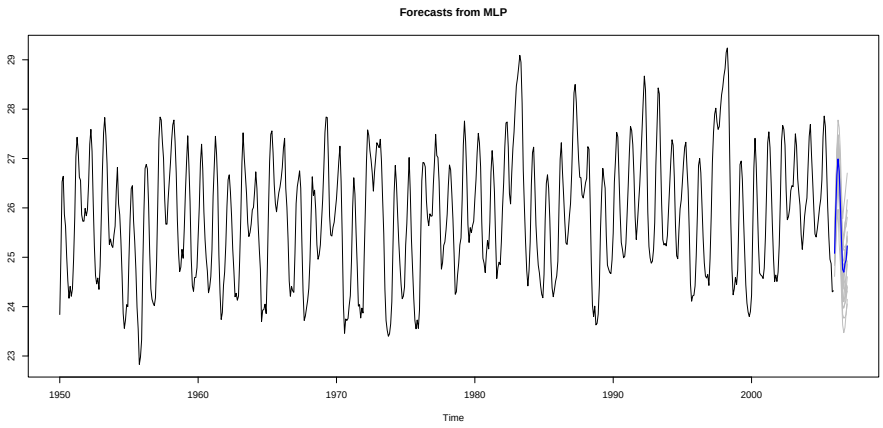


FIGURE 4.9 – Prédicteur de la température d'el niño année 2006

La figure suivante Fig. 4.10 montre les résultats de la prévision, en rouge les sorties désirées (la trajectoire X_{57}), et en noir les sorties prédites (MLP) calculées en fonction des 56 observations de la température d'el Niño3 pendant la période 1950 – 2005 (où la période est l'année)
Nous obtenons pour l'erreur quadratique moyenne

$$MSE = 0.0381$$

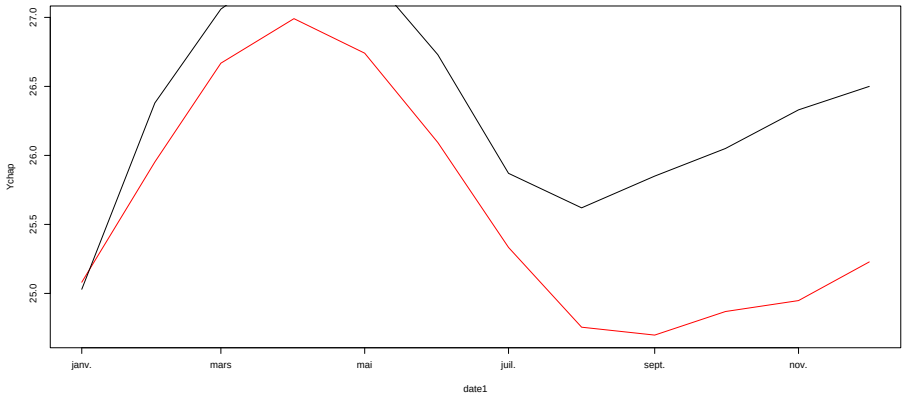


FIGURE 4.10 – Prédicteur de la température d'el niño année 2006

Méthode MLP avec deux couches cachées

La fonction MLP accepte plusieurs arguments pour affiner le réseau résultant, L'argument **"hd"** définit le nombre fixe de noeuds masqués. S'il s'agit d'un seul nombre, les neurones sont disposés en une seule couche cachée , S'il s'agit d'un vecteur (chaque nombre représente le nombre de noeuds d'une couche cachée), ceux-ci sont disposés en plusieurs couches.

```
> a<-mlp(Adata, hd=c(10, 5))
 > print(a)
MLP fit with (10, 5) hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1, 2, 4, 7, 9, 10, 12)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 0.011.
```

La sortie , dans ce cas, indique que le réseau résultant a 2 couches cachées, la première contienne 10 nœuds cachés, la deuxième contienne 5 nœuds cachés, il a été formé 20 fois, et les différentes prévisions ont été combinées en utilisant l'opérateur médian, l'erreur obtenu est de 1.1%.
On peut obtenir un résumé visuel en utilisant la fonction **plot()** :

```
> plot(a)
```

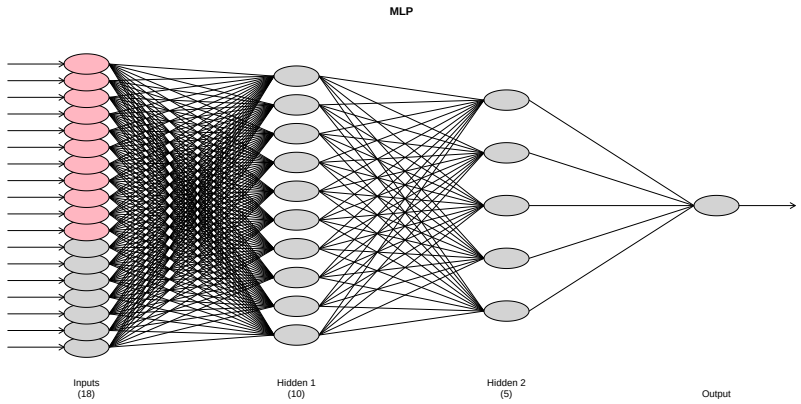


FIGURE 4.11 – Réseaux de neurones avec 2 couche cachée

Prévision

```
> frc<-forecast(a)
> print(frc)
         Jan      Feb      Mar      Apr      May
2006 24.88110 25.90753 26.43322 26.86810 26.46523
         Jun      Jul      Aug      Sep      Oct
2006 25.88010 24.95994 23.99351 24.08695 23.94439
         Nov      Dec
2006 24.32718 24.64001

>plot(frc)
```

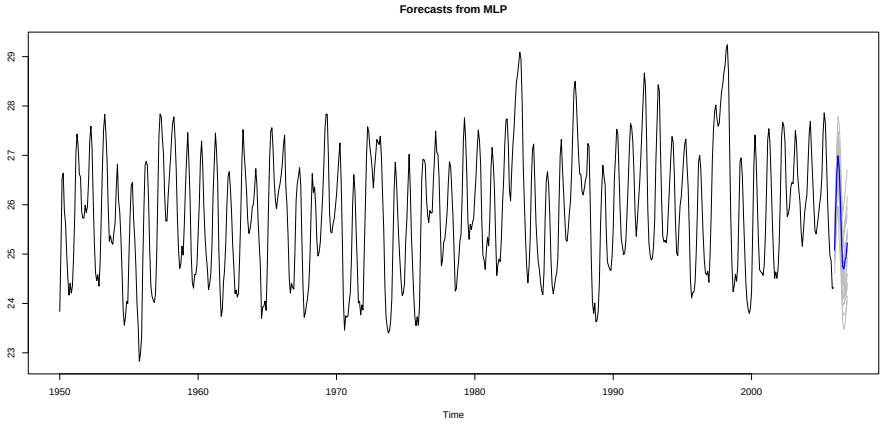


FIGURE 4.12 – Prédicteur de la température d'el niño année 2006

La figure suivante Fig. 4.13 montre les résultats de la prévision, en rouge les sorties désirées (la trajectoire X_{57}), et en noir les sorties prédites (MLP) calculées en fonction des 56 observations de la température d'el Niño3 pendant la période 1950 – 2005 (où la période est l'année)
Nous obtenons pour l'erreur quadratique moyenne

$$MSE = 0.011$$

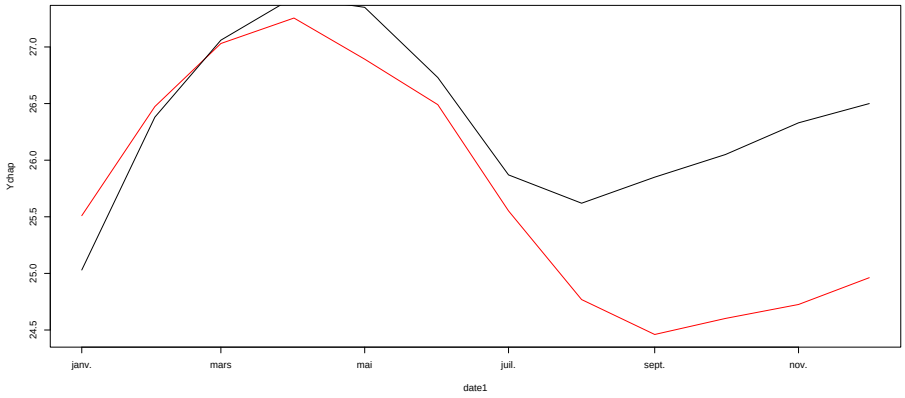


FIGURE 4.13 – Prédicteur de la température d'el niño année 2006

Remarque

— Dans notre exemple , l'ajout d'une couche cachée a réduit l'erreur de prédiction du modèle de réseau de neurones.

— En général, l'ajout d'une couche cachée permet à un réseau de neurones d'apprendre des représentations plus complexes et abstraites des données en entrée, cela peut être, particulièrement, utile lorsque les relations entre les caractéristiques en entrée sont non linéaires.

— L'ajout d'une couche cachée peut également rendre le modèle plus complexe et plus difficile à entraîner, ce qui peut conduire à un surajustement "**overfitting**" si la taille de l'ensemble de données d'entraînement est trop petite par rapport à la complexité du modèle, par conséquent, l'ajout d'une couche cachée n'est pas toujours la meilleure solution pour réduire l'erreur de prédiction d'un modèle de réseau de neurones. Il est important de considérer la complexité du problème et la taille de l'ensemble de données d'entraînement pour déterminer la meilleure architecture du modèle.

— La fonction d'activation utilisée dans la fonction "mlp" pour construire un réseau de neurones multicouche est la fonction d'activation sigmoide.

Optimisation du nombre de couches cachées dans un modèle MLP à l'aide du critère BIC

Le critère BIC peut être utilisé pour sélectionner le nombre optimal de neurones dans un modèle de réseaux de neurones pour la prévision d'une série temporelle par MLP.
Nous commençons par diviser les données en ensembles d'apprentissage et de validation.

```
Adata1<-ts(z, start=c(1950, 1), end=c(2006, 12), frequency=12)
train_data <- window(Adata1, end = c(2005, 12))
validation_data <- window(Adata1, start = c(2006, 1))
```

Ensuite, nous procédons à la division des données en ensembles d'apprentissage et de test.

```
train_size <- length(train_data) / length(Adata1)
train_index <- 1:length(train_data)
train_data <- window(Adata1, end = c(2005, 12))
test_data <- window(Adata1, start = c(2006, 1))
```

Nous décidons de fixer le nombre maximal de couches cachées à six pour notre modèle

```
max_hidden_layers <- 6
```

Pour chaque nombre de couches cachées, nous calculons le BIC (Bayesian Information Criterion) en utilisant la formule suivante :

$$BIC = n * log(\frac{RSS}{n}) + k * log(n)$$

où RSS (Residual Sum of Squares) est la somme des carrés des résidus, c'est-à-dire la mesure de l'erreur du modèle, k est le nombre de couches cachées, qui varie de 1 à 6 dans ce cas et n représente le nombre d'observations dans l'ensemble de validation

```
bic_values <- numeric(max_hidden_layers)
rss_values<- numeric(max_hidden_layers)
for (i in 1:max_hidden_layers) {
model <- nnet(x = train_data, y = train_data, size = i,
linout = TRUE)
validation_pred <- predict(model, newdata = validation_data)
RSS <- sqrt(mean((validation_data - validation_pred)^2))
bic <- length(validation_data)
*log(rmse^2/length(validation_data))+
i* log(length(validation_data))
bic_values[i] <- bic }
```

La sortie fournit deux vecteurs : un contenant les valeurs du BIC et l'autre contenant les résidus

```
bic_values
[1] -31.12914  -28.64423  -171.51825  -160.59067
    -169.65478 -18.70461
rss_values
[1] 0.853737116 0.853737116 0.001999709 0.002842772
    0.001756935 0.853737116
```

Pour déterminer le nombre optimal de couches cachées, nous utilisons les commandes suivantes

```
optimal_num_layers <- which.min(bic_values)
cat("Meilleur nombre de couches cachees selon le critere
BIC:", optimal_num_layers, "\n")
Meilleur nombre de couches cachees selon le critere BIC: 3
```

Les figures ci-dessous Fig. 4.14et Fig.4.15 , illustrent les variations des valeurs du critère BIC , respectivement, de l'erreur RSS en fonction du nombre de couches cachées.

Remarque :

l'erreur quadratique moyenne (MSE, Mean Squared Error) est calculée en divisant le RSS (Residual Sum of Squares) par la taille de l'échantillon.

Ainsi, pour $k = 3$, l'erreur quadratique moyenne est égale à

$$MSE = \frac{0.001999709}{12} = 0.0001666424$$

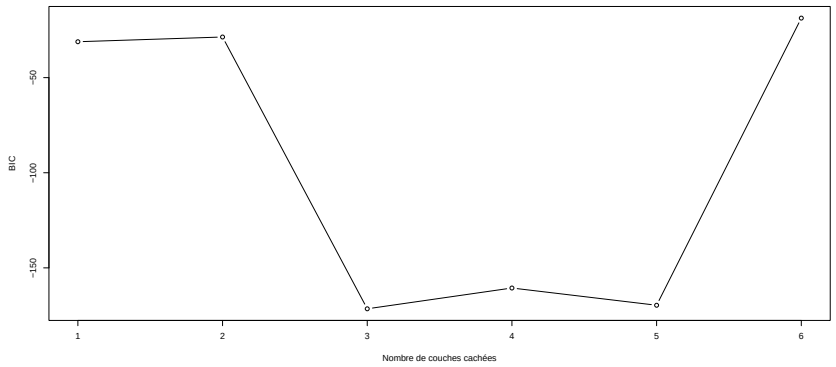


FIGURE 4.14 – BIC en fonction du nombre des couches cachées

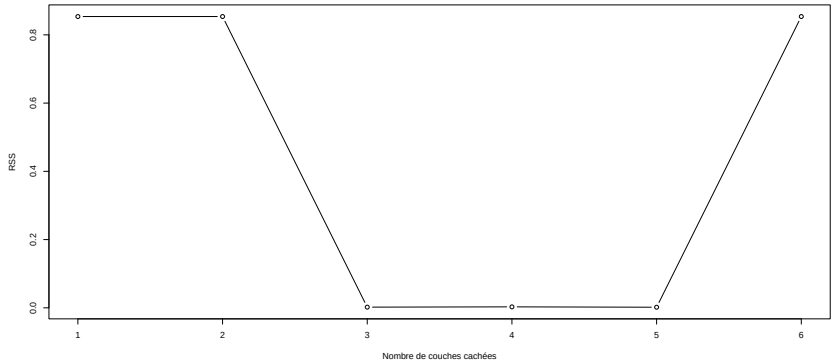


FIGURE 4.15 – RSS en fonction du nombre des couches cachées

La figure suivante Fig. 4.16 illustre les résultats de la prévision, les sorties désirées (trajectoire X_{57}) sont représentées en bleu, tandis que les sorties prédites (MLP) calculées à l'aide de la méthode MLP avec 3 couches cachées sont représentées en rouge. Ces prédictions sont basées sur les 56 observations de la température d'El Niño3 pendant la période 1950-2005.

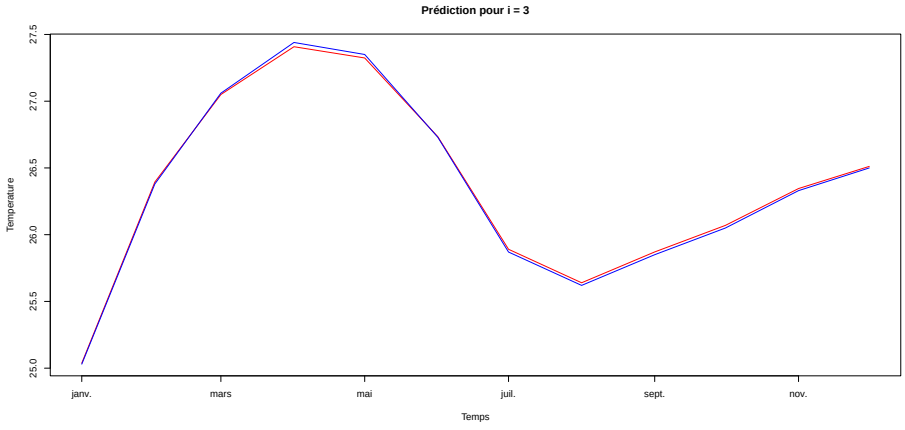


FIGURE 4.16 – Prédicteur de la température d'el niño année 2006

Le tableau 4.2 donne les valeurs de l'erreur MSE associées aux différentes méthodes de prévision ([4], [5], [11]).

Nous remarquons que le prédicteur obtenu par la méthode MLP donne une erreur assez faible.

Méthodes de prévision	MSE
Climatologie	2, 5%
Sarima	3, 7%
Kernel	2, 3%
Functional Kernel	2, 2%
Smooth FAR(1)	2, 3%
Smooth FAR(1) with q=p=12	2, 4%
Local FAR(1)	2, 2%
BLUP discret	1, 4%
BLUP continu	2, 25%
Predicteur sieves	2, 46%
Predicteur sieves 2017	2, 07%
BLP predicteur	2, 06%
Predicteur (PLS)	0, 5%
Predicteur (MLP)	0.01666424%

TABLE 4.2 – Erreur MSE de prévision de différentes méthodes de la température d'el Niño 2006

4.3 Prévision de la température à Nottingham

Nous appliquons aussi la méthode MLP sur la série historique des relevés de la température à Nottingham (température moyenne mensuelle au château de Nottingham de Janvier 1920 et arrêtée en 1939) disponible dans la série Nottem de la bibliothèque de MASSE de S-PLUS. Nous utilisons les observations de la période 1920 – 1939 pour calculer le prédicteur MLP de la température de l'année 1939 et nous présentons les erreurs MSE des différentes méthodes de prévision disponibles (voir [12]).
en utilisant l'algorithme du perceptron multicouche (multilayer perceptron MLP en anglais) et le package **nnfor** du langage **R**.
On note par "**nottem**", les données la température à Nottingham entre 1920 et 1939 , on importe les données et on commence par représenter cette série temporelle, comme le montre la Fig 4.17
.

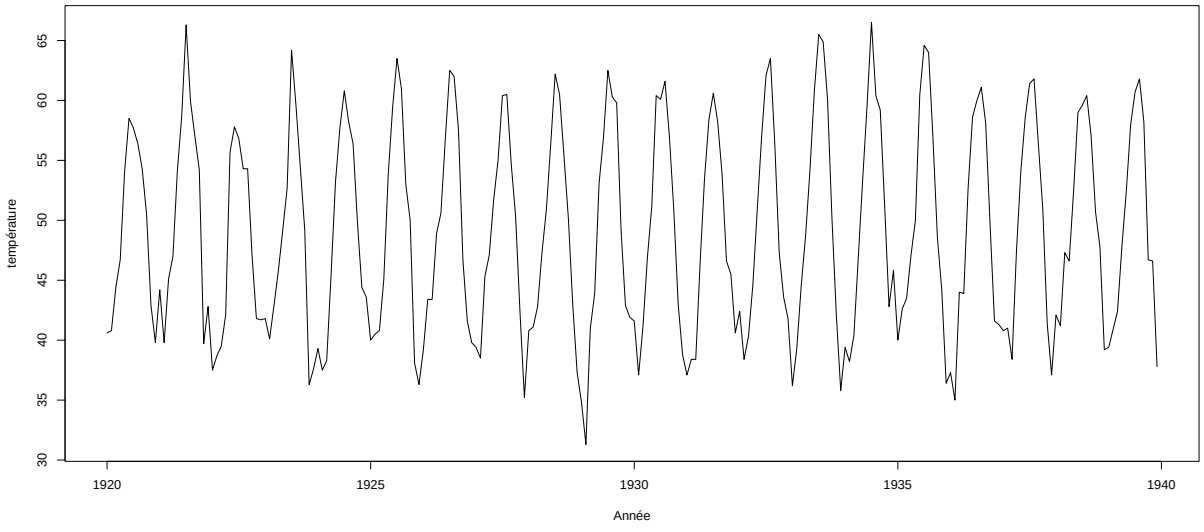


FIGURE 4.17 – Température à Nottingham (1920-1939)

4.3.1 Analyse de la série temporelle

Tendance et Saisonnalité

```
>plot(decompose(nottem))
```

le résultat est affiché sous la forme suivante :

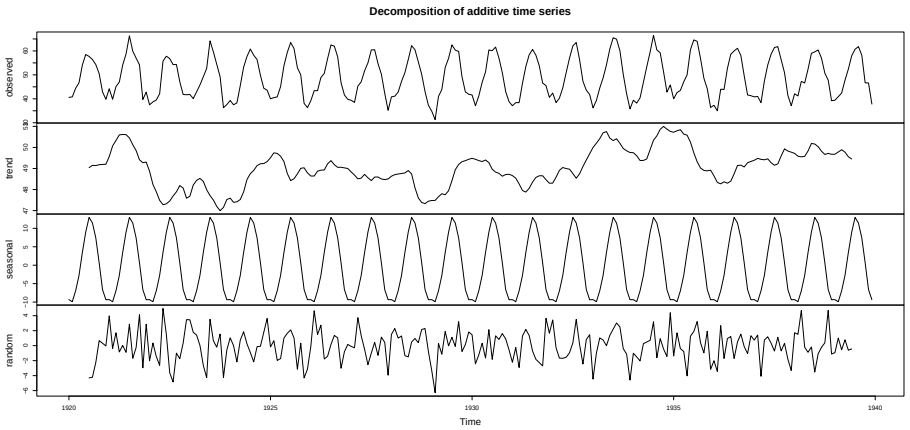


FIGURE 4.18 – Décomposition de la série temporelle

La Fig 4.18 montre les quatre éléments suivants :

- La serie temporelle originale, qui est tracée sur le premier graphique.
- La tendance, qui est tracee sur le deuxième graphique.
- La saisonnalite, qui est tracée sur le troixième graphique.
- Les residus, qui sont tracés sur le dernier graphique.

Pour confirmer les resultats graphique, on utilise :

```
>trendtest(nottem, extract=TRUE)
 TRUE
```

le résultat est ”TRUE” donc la tendance existe.
pour la saisonalité on utilise la commande suivante

```
>seasplot(nottem)
 Results of statistical testing
Evidence of trend: TRUE  (pval: 0)
Evidence of seasonality: TRUE  (pval: 0)
```

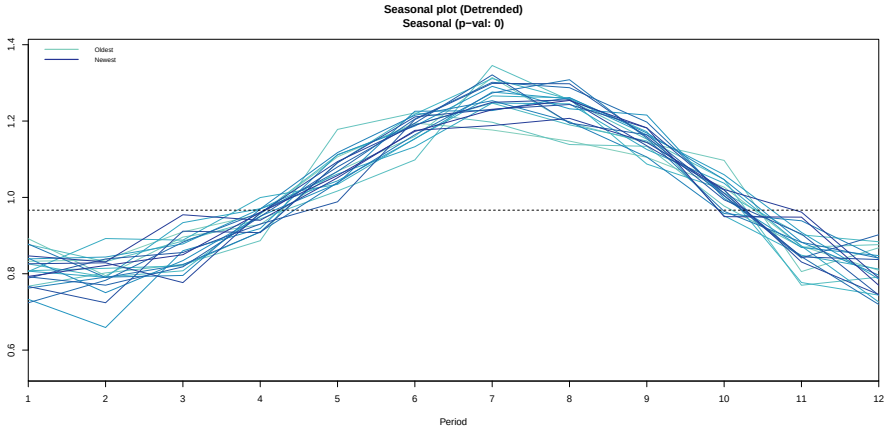


FIGURE 4.19 – les variations saisonnières pour l'année 1939

Le graphique résultant , Fig4.19, montre les variations saisonnières dans les données, ´ avec des valeurs plus elevées pour le mois de juillet et des valeurs plus faibles pour le mois de février.

on applique la methode MLP, mlp().

```
> a<−mlp(nottem)
 > print(a)
MLP fit with 5 hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1, 2, 4, 7, 9, 10, 12)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 0.0857.
```

La sortie indique que le réseau résultant a 5 nœuds cachés, il a été formé 20 fois, et les différentes prévisions ont été combinées en utilisant l'opérateur médian.L'erreur obtenu est 8.57%.
On peut obtenir un résumé visuel en utilisant la fonction **plot()** :

```
> plot(a)
```

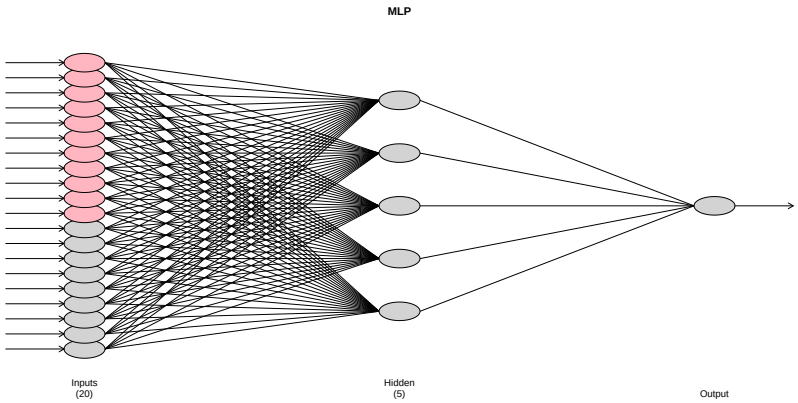


FIGURE 4.20 – Réseaux de neurones avec une couche cachée

La figure suivante Fig4.21 montre les résultats de la prévision, en rouge les sorties desirées (la trajectoire X_{20}), et en noir les sorties predites (MLP) calculées en fonction des 19 observations de la temperature à nottingham pendant la période 1920 – 1939 Nous obtenons pour l'erreur quadratique moyenne $MSE = 0.0857$

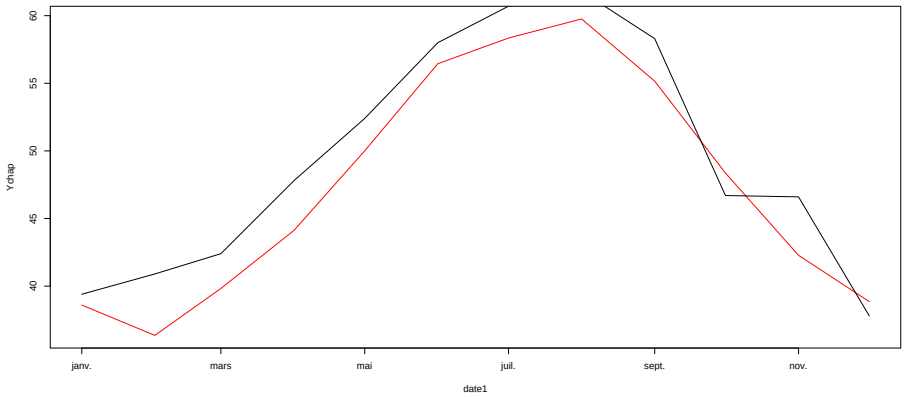


FIGURE 4.21 – Prédicteur (MLP1) année 1939

Methode MLP avec deux couches cachées

```
> a<-mlp(nottem, hd=c(10, 5))
 > print(a)
MLP fit with (10, 5) hidden nodes and 20 repetitions.
Series modelled in differences: D1.
Univariate lags: (1, 2, 4, 7, 9, 10, 12)
Deterministic seasonal dummies included.
Forecast combined using the median operator.
MSE: 0.0174.
```

La sortie , dans ce cas, indique que le réseau résultant a 2 couches cachées, la première contienne 10 nœuds cachés, la deuxième contienne 5 nœuds cachés, il a été formé 20 fois, et les différentes prévisions ont été combinées en utilisant l'opérateur médian, l'erreur obtenu est de 1.74%.

On peut obtenir un résumé visuel en utilisant la fonction **plot()** :

```
> plot(a)
```

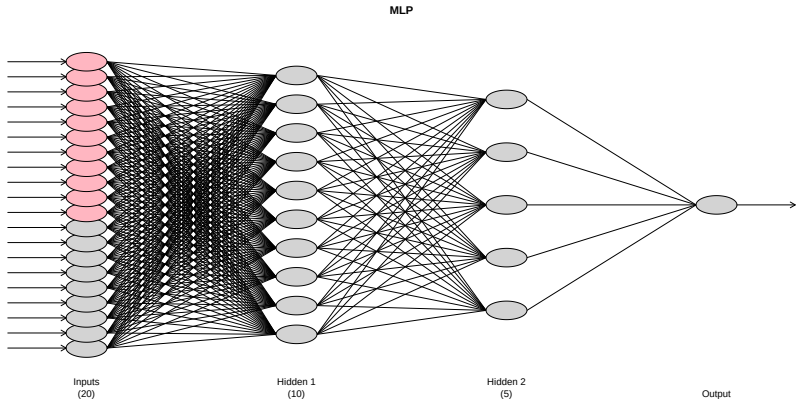


FIGURE 4.22 – Réseaux de neurones avec 2 couches cachées

La figure 4.23 représente la température à Nottingham de 1920 au 1939 et sa prévision MLP2.Nous obtenons pour l'erreur quadratique moyenne $MSE = 0.0174$

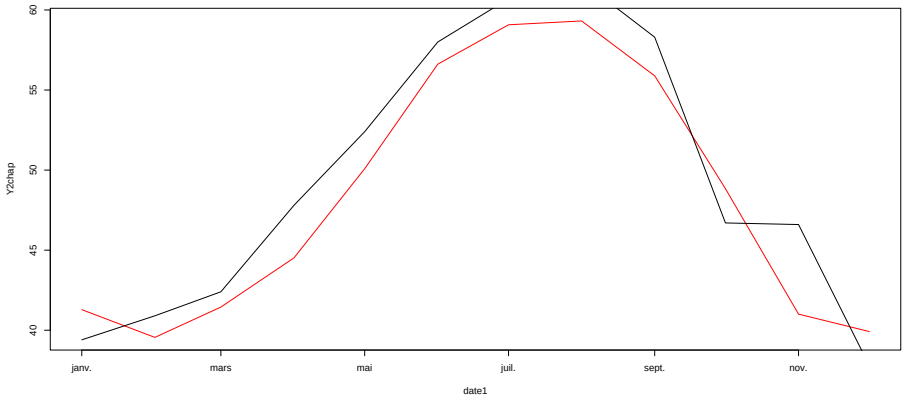


FIGURE 4.23 – Prédicteur (MLP2) année 1939

Le tableau 4.3 présente les erreurs MSE de prévision pour différentes méthodes de la température de l'année 1939 à Nottingham.

Méthodes de prévision	**MSE**
Wavelet-Kernel	30%
Sarima	31%
SS(Spline Smoothing)	28%
BLUP discret	2, 95%
BLUP continu	2, 96%
Predicteur sieves	2, 95%
Predicteur sieves 2017	6, 09%
BLP predicteur	3, 20%
Predicteur (PLS)	2.5%
Predicteur (MLP1)	8.57%
Predicteur (MLP2)	1.74%

TABLE 4.3 – Erreur MSE de prévision de différentes méthodes de la température de l'année 1939 à Nottingham

Nous remarquons que le prédicteur obtenu par la méthode MLP donne une MSE assez faible à celles données par les autres méthodes.

Nombre optimal de couches cachées

Les résultats obtenus après l'exécution, indiquent les performances du modèle MLP (Multi-Layer Perceptron) avec différents nombres de couches cachées (1, 2, 3, 4 et 5) :

```
train_data<-window(nottem, end = c(1938, 12))
validation_data <- window(nottem, start = c(1939, 1))
num_hidden_layers<-c(1, 2, 3, 4, 5)
mse_scores<-numeric(length(num_hidden_layers))
for(i in seq_along(num_hidden_layers)) {
num_hidden <- num_hidden_layers[i]
create_model <- function(hidden_units) {
formula <- as.formula(paste("train_data ~ ."))
model <- nnet(formula, data = as.data.frame(train_data),
size = hidden_units, linout = TRUE)
return(model)
```

```
  }
model <- create_model(num_hidden)
predicted <- predict(model, newdata = as.data.frame
(validation_data))
mse_scores[i] <- mean((validation_data - predicted)^2)
}
```

Pour déterminer le nombre optimal de couches cachées, on a utilisé les commandes suivantes :

```
optimal_num_hidden <- num_hidden_layers[which.min(mse_scores)]
print(mse_scores)
[1] 6.808312e+01 6.808312e+01 5.562938e-04 8.073101e-04
    6.808312e+01
```

Les résultats contient les valeurs d'erreur MSE pour chaque nombre de couches cachées testé, le nombre optimal de couches cachées est déterminé en utilisant la fonction suivante :

```
cat("Le nombre optimal de couches cachees est", \\
 optimal_num_hidden, "\n")
Le nombre optimal de couches cachees est 3
```

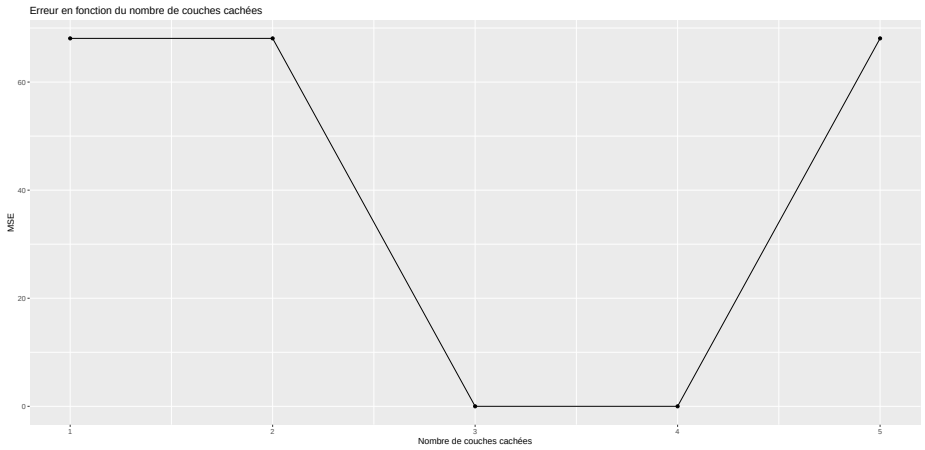


FIGURE 4.24 – MSE en fonction nombre des couches cachées

Conclusion générale :

En conclusion, la prévision des séries temporelles par les réseaux de neurones représente une approche puissante et prometteuse pour analyser et prédire les tendances et les modèles dans les données temporelles. Les réseaux de neurones, tels que les réseaux MLP, ont démontré leur capacité à capturer les dépendances temporelles et à fournir des prévisions précises.

L'utilisation des réseaux de neurones permet de bénéficier de leur capacité à apprendre à partir des données, à modéliser les relations complexes entre les observations temporelles et à s'adapter à différents types de séries temporelles. Ils offrent également la possibilité de traiter des séquences de longueur variable et de prendre en compte les caractéristiques spécifiques des données temporelles, telles que les saisons, les tendances et les variations irrégulières.

Cependant l'entraînement des réseaux de neurones pour la prévision des séries temporelles implique l'optimisation des paramètres du modèle en utilisant des techniques d'optimisation telles que la rétropropagation du gradient. La sélection des hyperparamètres, tels que le nombre de couches cachées, le nombre de neurones, le taux d'apprentissage, etc., est cruciale pour obtenir de bonnes performances prédictives.

La performance des modèles de réseaux de neurones pour la prévision des séries temporelles peut être évaluée en utilisant des métriques telles que l'erreur quadratique moyenne (MSE), le critère BIC pour évaluer la robustesse et la généralisation des modèles.

En résumé, l'utilisation des réseaux de neurones offre une approche mathématiquement rigoureuse pour la prévision des séries temporelles. Ces modèles peuvent capturer les structures temporelles complexes et fournir des prédictions précises. Cependant, il est important de prendre en compte les aspects pratiques de la modélisation, tels que la sélection des paramètres et l'élagage du nombre de couches cachées , pour garantir des résultats fiables et valides.

Bibliographie

[1] Allam, A. and T, Mourid. Geometric Absolute regularity of Banach space autoregressive processes. *Statistics and Probability Letters 60 (2002) p. 241-252*.

[2] Antoniadis, A and E. Paparoditis and T. Sapatinas. A functional wavelet-kernel approach for time series prediction, *journal of the royal Statistical Society. Series B, (2006), vol.68, no. 5, pp. 837-857*.

[3] Benbouteldja.M, *Series temporelles par les réseaux de neurones et architecture optimale* , mémoire de Magistère , Université des Sciences et de la Technologie Houari Boumediene, *N°* d'ordre : 13 / 2010-M / M T

[4] Bensmain, N. Prédiction des processus AR Hilbertien via la méthode des seives. Simulations et exemples , *Ann. ISUP, (2013), 57, 3, 103-115.*

[5] Berhoune, K. and N, Bensmain, . (2018). Sieves estimator of functional autoregressive process. *Statistics and Probability Letters 135, pp 60-69*.

[6] Bourbonnais R., virginie TERRAZA, *Analyse des séries temporelles* - 5e édition avril 2022, Éco Sup, Dunod.

[7] Dauxois J-Y, *"Introduction à l'Étude des Séries Temporelles"*, (2016/2017).

[8] Demuth H., M. Beale, *Neural networks Toolbox for use wiht MATLAB,* (2004).

[9] Héctor J. Sussmann, Uniqueness of the weights for minimal feedforward nets with a given input-output map, *Neural Networks, Volume 5, Issue 4, Pages 589-593, ISSN 0893-6080* 1992.

[10] JACQUES J.*Introduction aux séries temporelles*, Cours dispensé à l'Université de Lyon, 2020.

[11] Kada Kloucha M and Tahar Mourid Best linear predictor of a C[0, 1]-valued functional autoregressive process.*Statistics and Probability Letters 150 (2019) 114–120.*

[12] Kada Kloucha M. *Prévision d'un autoregressif fonctionnel via les sous espace clos*. . Thèse de Doctorat Sciences.Université Abou Bekr Belkaid. 2019.

[13] Kada Kloucha, M. *Méthodes de régression Ridge Lasso et PLS théorie et exemples*. mémoire de Magistère. Université Abou Bekr Belkaid. 2009.

[14] Kalakh M., *"Modélisation avec les réseaux de neurones d'un canal UWB dans un environnement minier souterrain"*, Mars, 2013.

[15] Kettaneh-Wold, N. (1992). PLS-regression : a basic tool of chemometrics. *Chemometrics and intelligent laboratory systems, 18(1), 3-10*.

[16] MANGEAS M. et Jian-Feng YAO Sur l'estimateur des moindres carrés d'un module autor gressif fonctionnel *SAMOS, Universit Paris I, 90 rue de Tolbiae, 75013 Paris, France* .

[17] Principe J. C., N. R. Euliano, W. C. Lefebvre, *"Neural and Adaptive Systems. Fundamentals through Simulations"*, Wiley, 2000.

[18] Rynkiewicz J., M. Cottrell, M. Mangeas, J.F. Yao Modèles de réseaux de neurones pour l'analyse des séries temporelles ou la régression : Estimation, Identification, Méthode d'élagage SSM, *SAMOS, Université de Paris I, S MM : RIA. Volume X-n° X/2001.*

[19] Sublime.J, L'apprentissage non-supervisé et ses contradictions. *1024 : Bulletin de la Société In-formatique de France, Société Informatique de France, 2022, pp.145-156. 10.48556/SIF.1024.19.145. hal-03648943*

[20] Tenenhaus, M.(1998), *La régréssion PLS Théorie et pratique, Technip*, Paris.

[21] T.W.S Chow, S.Y. Cho, *"Neural Networks and Computing, Learning Algorithms and Applications, Imperial College Press"*, (2007).