

Département Informatique et Mathématiques
3^{ème} Année Cycle d'Ingénieur Génie Informatique

Rapport de Stage de Fin d'Études (PFE)
Pour l'obtention du Diplôme d'Ingénieur d'État

**Développement d'un outil de prévision multi-horizons
de la demande électrique nationale basé sur
l'intelligence artificielle**



Office National d'Électricité et d'Eau Potable - Branche d'Électricité

Réalisé par :
Îmad HARIR

Encadré par :
- **Adil HADDI** (Encadrant ENSA)
- **Khalid BOUIHAT** (Encadrant ENSA)
- **Abdessamad LAMALLAM** (Encadrant d'entreprise)

Soutenue le 27/06/2024, devant le jury composé de :

- **Abdeljalil SAKAT**
- **Adil HADDI**
- **Khalid BOUIHAT**

Année universitaire : 2023/2024
Période de stage : 05/02/2024 - 05/06/2024

Dédicace

A ma chère mère

Aucune dédicace ne peut exprimer l'amour, l'estime et le respect que j'ai toujours eu pour toi. Ce travail est le fruit de tes sacrifices pour mon éducation, ma formation et mon bien-être. Je te souhaite une vie pleine de santé et de bonheur.

A mon cher père

Tu nous as quitté trop tôt mais je ne t'ai jamais oublié, ton souvenir m'inspire à faire de mon mieux face aux difficultés, cette œuvre est la tienne. Que Dieu t'accueille dans son paradis.

A mes chers amis

En témoignage de l'amitié qui nous unit et des souvenirs de tous les moments passés ensemble, je vous dédie ce travail et je vous souhaite le meilleur pour la suite.

Remerciements

En préambule de ce rapport, je tiens à exprimer mes sincères remerciements et ma gratitude à tous ceux qui ont contribué à la mise en œuvre de ce projet et rendu possible sa réalisation.

Tout d'abord, j'adresse mes remerciements à notre coordinateur du cycle d'ingénieur Génie Informatique à l'École Nationale des Sciences Appliquées à Berrechid Monsieur Lahcen MOMOUN pour ses efforts, sa disponibilité et son dévouement à la gestion de ce cycle d'ingénieur.

J'ai l'honneur, en marge de ces travaux, d'exprimer ma profonde gratitude et toute ma reconnaissance à Messieurs HADDI et BOUIHAT pour le temps qu'ils m'ont accordé, pour avoir toujours veillé aux intérêts de ses étudiants, pour ses conseils avisés et pour la qualité de leurs enseignement.

Mes remerciements vont aussi particulièrement à mes encadrants de stage, le chef de division système télé conduite Monsieur Abdessamad LAMALLAM et le responsable dans le service de prévision Monsieur Walid BENABED qui n'ont pas cessé de m'encourager, conseiller et motiver tant durant la période de formation que dans la phase de développement de ce projet.

Avec un grand respect, j'adresse mes sincères remerciements à tout le corps professoral de l'École Nationale des Sciences Appliquées à Berrechid pour leurs qualités pédagogiques.

Résumé

Le présent rapport résume le travail réalisé dans le cadre du projet de prévision de consommation électrique au sein de l'Office National de l'Électricité et de l'Eau Potable (ONEE). L'objectif principal de ce projet est de développer un modèle de prévision précis et fiable de la consommation électrique, permettant à l'ONEE d'anticiper et de planifier efficacement la production et la distribution d'électricité.

Dans un contexte où la demande en électricité varie en fonction de nombreux facteurs, tels que les saisons, les jours de la semaine, les événements spéciaux et les conditions météorologiques, il devient essentiel pour l'ONEE de disposer d'outils de prévision avancés pour garantir un approvisionnement stable et fiable en électricité.

Ce projet intervient donc pour répondre à ce besoin crucial en exploitant des techniques avancées d'apprentissage automatique, telles que les réseaux de neurones récurrents (RNN), pour analyser et prévoir la consommation électrique à différentes échelles temporelles (horaire, quotidienne, hebdomadaire et mensuelle). En utilisant des données historiques de consommation électrique ainsi que d'autres variables pertinentes telles que la température, le jour de la semaine, le moment de la journée et les événements spéciaux, le modèle de prévision cherche à capturer les schémas et les tendances dans les données pour améliorer la précision des prévisions.

Les résultats de ce projet visent à fournir à l'ONEE des informations précieuses pour optimiser la gestion de la production et de la distribution d'électricité, réduire les coûts opérationnels et garantir une alimentation électrique stable et fiable pour les citoyens et les entreprises.

Mots-Clés : Prévision de Consommation Électrique, Office National de l'Électricité et de l'Eau Potable (ONEE), Réseaux de Neurones Récurrents (RNN), Analyse de Données, Apprentissage Automatique, Gestion de la Production et de la Distribution d'Électricité.

Abstract

This report summarizes the work carried out within the framework of the electricity consumption forecasting project at the National Office of Electricity and Drinking Water (ONEE). The main objective of this project is to develop an accurate and reliable model for predicting electricity consumption, enabling ONEE to anticipate and efficiently plan electricity production and distribution.

In a context where electricity demand varies due to numerous factors such as seasons, days of the week, special events, and weather conditions, it becomes crucial for ONEE to have advanced forecasting tools to ensure stable and reliable electricity supply.

This project aims to address this crucial need by leveraging advanced machine learning techniques such as Long Short-Term Memory (LSTM) neural networks to analyze and forecast electricity consumption at different time scales (hourly, daily, weekly, and monthly). By using historical electricity consumption data as well as other relevant variables such as temperature, day of the week, time of day, and special events, the forecasting model seeks to capture patterns and trends in the data to improve prediction accuracy.

The results of this project aim to provide ONEE with valuable insights to optimize electricity production and distribution management, reduce operational costs, and ensure stable and reliable power supply for citizens and businesses.

Keywords : Electricity Consumption Forecasting, National Office of Electricity and Drinking Water (ONEE), Long Short-Term Memory (LSTM) Neural Networks, Data Analysis, Machine Learning, Electricity Production and Distribution Management.

Table des figures

1.1	Activités de l'ONE	3
1.2	Structure générale de l'Office National d'Électricité	5
1.3	Pôle industriel de l'Office National d'Électricité	6
1.4	Direction Centrale Transport	6
1.5	Direction Opérateur Système	7
1.6	Division Systèmes de Téléconduite	7
3.1	Rendement de 70 batches consécutifs d'un procédé chimique [14]	14
3.2	Séries temporelles typiques apparaissant dans les problèmes de prévision et de contrôle [14]	15
3.3	Exemple d'une série temporelle univariée	16
3.4	Exemple d'une série temporelle multivariée	16
3.5	Tendance linéaire	17
3.6	Variation saisonnière	18
3.7	Variation aléatoire	18
3.8	Valeurs aberrantes	18
3.9	Bruit Blanc	19
3.10	Marche aléatoire	20
3.11	ACF	22
3.12	PACF	23
3.13	Commandes d'équipements électriques : la composante tendance-cycle (rouge) et les données brutes (gris)	25
3.14	Les commandes d'équipements électriques (en haut) et ses trois composantes additives	26
3.15	Décomposition multiplicative classique de l'indice des nouvelles commandes pour l'équipement électrique	27
3.16	Une décomposition X11 de l'indice des nouvelles commandes pour l'équipement électrique.	28
4.1	Réseau de neurones artificiels simplifié	32
4.2	Perceptron	33
4.3	Réseau de neurones à action directe	33
4.4	Un perceptron multicouche	34
4.5	Architecture standard d'un réseau à convolutions	34
4.6	Réseau de neurones récurrents	35
4.7	Schéma d'un réseau de neurones récurrents à une unité reliant l'entrée et la sortie du réseau	36
4.8	Schéma d'un RNN	36
4.9	Réseau de neurones récurrents Un à Un	37
4.10	Réseau de neurones récurrents Un à Plusieurs	37
4.11	Réseau de neurones récurrents Plusieurs à Un	38
4.12	Réseau de neurones récurrents Plusieurs à Un à taille égale des unités	38
4.13	Réseau de neurones récurrents Plusieurs à Un à taille inégale des unités	39
5.1	Schémas de la disparition et de l'explosion du gradient	43
5.2	Réseau RNN	44
5.3	Réseau LSTM	44
5.4	Composants du LSTM	45
5.5	Cellule de mémoire	45
5.6	Porte d'oubli	46
5.7	Porte d'entrée	46

5.8	Nouvelle cellule de mémoire	46
5.9	Porte de sortie	47
5.10	Données transformées en 3D pour un réseau LSTM	47
5.11	Diagramme de flux d'un réseau autoencodeur LSTM	48
5.12	Déférence entre <code>return_sequences=True</code> et <code>return_sequences=False</code>	48
5.13	LSTM empilée	50
5.14	Peephole LSTM	50
5.15	LSTM Bidirectionnel	51
5.16	Architecture du Bi-LSTM	52
5.17	Niveaux d'itération pour l'entraînement des LSTM	53
6.1	Cellule GRU et ses portes	55
6.2	Unité récurrente fermée, version entièrement fermée	57
6.3	GRU Type 1	57
6.4	GRU Type 2	58
6.5	GRU Type 3	58
7.6	Logo du Flask	62
7.7	Logo du Chart.js	62
8.1	Données initiales de consommation électrique	66
8.2	Matrice de corrélation entre différentes villes du Maroc	67
8.3	Regroupement des stations météorologiques	67
8.4	Températures collectées de l'API des 5 stations représentatives	69
8.5	Statistiques descriptives des variables	70
8.6	Représentation de l'heure de la journée et de ses composantes sinusoïdales (sinus et cosinus) sur une semaine	71
8.7	Diagramme en boîte (boxplot) des valeurs horaires de la consommation d'électricité	72
8.8	Évolution temporelle des données pendant la période donnée	73
8.9	Évolution temporelle des données pendant une année	73
8.10	Évolution temporelle des données pendant deux semaines	74
8.11	Évolution temporelle de la consommation électrique pendant une journée pour différents types de jour	74
8.12	Décomposition STL de la série temporelle de la consommation électrique	75
8.13	Test ADF de la série du consommation électrique	76
8.14	ACF / PACF	77
8.15	Extrait de jeu de données encodé	78
8.16	Divion des données (Entraînement / Validation / Test)	78
8.17	Code Python de mise à l'échelle	79
8.18	Extrait des données d'entraînement mises à l'échelle	79
8.19	Exemple d'un petit ensemble de données de séries temporelles fictif	80
8.20	Exemple d'un ensemble de données de séries temporelles comme apprentissage supervisé	80
8.21	Exemple d'un petit ensemble de données fictif de séries temporelles multivariées	81
8.22	Exemple d'ensemble de données de séries temporelles multivariées transformé en un problème d'apprentissage supervisé	81
8.23	Exemple d'ensemble de données de séries temporelles multivariées formulé comme un problème d'apprentissage supervisé de prédiction multi-étape ou de séquence	81
8.24	Exemple d'ensemble de données de séries temporelles univariées formulé comme un problème d'apprentissage supervisé de prédiction multi-étapes ou de séquence	82
8.25	Code de conversion des données en un problème d'apprentissage supervisé	83
8.26	Les données transformées sous forme de séquences	84
8.27	Architecture du modèle LSTM	85
8.28	Courbe d'apprentissage du modèle LSTM	86
8.29	Architecture du modèle GRU	88
8.30	Courbe d'apprentissage du modèle	89
8.31	Évaluation des performances du modèle LSTM	90
8.32	Évaluation des performances du modèle LSTM	90
8.33	Téléversement des données historiques de consommation électrique	92
8.34	Code du prétraitement des données	92

8.35	Code de génération des prévisions de la consommation électrique	93
8.36	Résultats de prévision	93

Acronymes

ACF	Autocorrelation Function.
ADF	Augmented Dickey-Fuller Test.
AED	Analyse Exploratoire des Données.
ANN	Artificial Neural Network.
API	Application Programming Interface.
AR	AutoRégressif.
ARIMA	AutoRegressive Integrated Moving Average.
Bi-LSTM	Bidirectional LSTM.
BPTT	BackPropagation Through Time.
CNN	Convolutional Neural Network.
ConvNet	Convolutional Network.
CRBP	Causal Recursive BackPropagation.
FC	Fully Connected.
GRU	Gated Recurrent Unit.
HTTP	HyperText Transfer Protocol.
KPSS	Kwiatkowski–Phillips–Schmidt–Shin Test.
LSTM	Long Short-Term Memory.
MA	Moving Average.
MAE	Mean Absolute Error.
MLP	MultiLayer Perceptron.
NN	Neural Network.
ONEE	Office National d'Électricité et Eau Potable.
ONEE-BE	Office National d'Électricité et Eau Potable - Branche Électricité.
PACF	Partial Autocorrelation Function.
PIB	Produit Intérieur Brut.
ReLU	Rectified Linear Unit.
RMSE	Root Mean Square Error.
RNN	Recurrent Neural Network.
RTRL	Real-Time Recurrent Learning.
STL	Seasonal-Trend decomposition using LOESS.

TS	Time Series.
VAR	Vector AutoRegression.
VECM	Vector Error Correction Model.

Table des matières

Table des figures

Table des matières

Introduction Générale	1
1 Cadre général du projet	2
1.1 Présentation de l'organisme d'accueil	3
1.2 Présentation de l'ONEE - Branche Électricité	3
1.2.1 Domaines d'activités	3
1.2.2 Missions	4
1.2.3 Organisation de l'ONEE-BE	5
1.3 Présentation du Division Système Télé Conduite (DI/DOS)	8
1.4 Vision globale du projet	8
1.4.1 Problématique	8
1.4.2 L'objectif du projet et la solution proposée	8
1.4.3 Analyse des besoins	9
2 Revue de la littérature	11
2.1 Méthodes Traditionnelles	12
2.2 Modèles Semi-Paramétriques et Additifs	12
2.3 Apprentissage Automatique et Réseaux de Neurones	12
2.4 Modèles Hybrides	12
2.5 Approches de Régression	12
2.6 Critique	12
3 Les séries temporelles	13
3.1 Définition d'une série temporelle	14
3.2 Types des séries temporelles	15
3.2.1 Série Temporelle Univariée	15
3.2.2 Série Temporelle Multivariée	16
3.3 Caractéristiques d'une série temporelle	17
3.4 Stationnarité d'une série temporelle	19
3.5 Analyse des séries temporelles	20
3.5.1 Tests de stationnarité	20
3.5.2 Fonctions d'Autocorrélation (ACF) et d'Autocorrélation Partielle (PACF)	22
3.5.3 Décomposition des séries temporelles	24
4 Réseaux de neurones récurrents (RNN)	30
4.1 Réseau de neurones	31
4.1.1 Définition	31
4.1.2 Types des réseaux de neurones	32
4.2 Réseau de neurones récurrents	35
4.2.1 Fonctionnement	36
4.2.2 Types de réseaux neuronaux récurrents	36
4.2.3 Comment les réseaux neuronaux récurrents se comparent-ils aux autres réseaux de deep learning ?	39

4.2.4	Entraînement	39
4.2.5	Limitations	40
5	Réseaux de neurones à mémoire à long terme (LSTM)	42
5.1	Disparition du gradient	43
5.2	Définition	43
5.3	Structure	44
5.3.1	Cellule d'État (Cellule de Mémoire)	45
5.3.2	Porte d'oubli	45
5.3.3	Porte d'entrée	46
5.3.4	Porte de sortie	46
5.4	Architectures	47
5.4.1	Réseau Autoencodeur LSTM	47
5.4.2	LSTM empilée	49
5.5	Variantes	50
5.5.1	Peephole LSTM	50
5.5.2	Bi-LSTM (LSTM bidirectionnel)	51
5.6	Étapes	52
5.7	Niveaux d'itération	52
5.8	Avantages et Inconvénients	53
5.8.1	Avantages	53
5.8.2	Inconvénients	53
6	Unités Récurrentes à Portes (GRU)	54
6.1	Définition	55
6.2	Structure	55
6.3	Pourquoi GRU ?	55
6.4	Fonctionnement des GRU	55
6.5	Architectures	56
6.5.1	Unité entièrement fermée	57
6.5.2	Unité fermée minimale	58
7	Languages et outils utilisés	60
7.1	Langages de Programmation	61
7.2	Bibliothèques de Deep Learning	62
7.3	Environnement de Développement	63
7.4	Outils de Gestion de Versions	63
8	Réalisation	65
8.1	Collection des données	66
8.1.1	Données de consommation	66
8.1.2	Données de température	67
8.2	Prétraitement des données	69
8.2.1	Feature Engineering	69
8.2.2	Nettoyage des données	71
8.2.3	Traitement des valeurs aberrantes	72
8.3	Analyse Exploratoire des Données	73
8.3.1	Description de la série temporelle	73
8.3.2	Décomposition de la série temporelle	75
8.3.3	Test de stationnarité (Test de Dickey-Fuller Augmenté)	76
8.3.4	Fonction d'autocorrélation (ACF) et d'autocorrélation partielle (PACF)	77
8.4	Implémentation des modèles	78
8.4.1	Encodage des données	78
8.4.2	Division des données	78
8.4.3	Normalisation des données	79
8.4.4	Modélisation	79
8.4.5	LSTM	84

8.4.6	GRU	88
8.5	Évaluation des performances	90
8.5.1	Performance du Modèle LSTM	90
8.5.2	Performance du Modèle GRU	90
8.5.3	Comparaison des Modèles	91
8.6	Application	91
8.6.1	Fonctionnalités	91
8.6.2	Processus de prévision	91

Introduction générale

L'Office National de l'Électricité et de l'Eau Potable (ONEE) joue un rôle crucial dans le développement et la gestion des infrastructures d'électricité et d'eau potable au Maroc. Ce rapport présente une analyse détaillée d'un projet réalisé au sein de l'ONEE, plus particulièrement dans sa branche Électricité.

Le projet en question s'inscrit dans une vision globale visant à améliorer l'efficacité et la performance des prévisions de la consommation électrique. L'objectif principal est de proposer des solutions innovantes pour répondre à des problématiques spécifiques identifiées par l'ONEE.

Ce rapport est structuré en plusieurs chapitres, chacun abordant des aspects clés du projet, des concepts théoriques aux implémentations pratiques :

- **Chapitre 1** : Cadre général du projet, incluant la présentation de l'ONEE et la vision globale du projet.
- **Chapitre 2** : Revue de la littérature sur les méthodes de prévision de la consommation électrique.
- **Chapitre 3** : Analyse des séries temporelles, définissant les concepts clés et les techniques d'analyse.
- **Chapitre 4** : Présentation des réseaux de neurones récurrents (RNN) et leur application dans la prévision.
- **Chapitre 5** : Focus sur les réseaux de neurones à mémoire à long terme (LSTM), leurs structures et variantes.
- **Chapitre 6** : Exploration des unités récurrentes à portes (GRU), une alternative aux LSTM.
- **Chapitre 7** : Description des langages et outils utilisés pour le développement des modèles.
- **Chapitre 8** : Détails de la réalisation, incluant la collection et le prétraitement des données, l'implémentation et l'évaluation des modèles.

Chapitre 1

Cadre général du projet

Ce chapitre introduira le cadre général du projet dont l'objectif est de présenter l'organisme d'accueil et de donner une vision globale du projet en décrivant la problématique et la solution proposée, la méthodologie de travail adoptée et les différentes phases de développement du projet.

1.1 Présentation de l'organisme d'accueil

Au lendemain de l'indépendance, le Maroc a dû prendre lui-même en main le secteur de l'électricité afin de l'organiser, le soutenir et garantir le service public. L'Office National de l'Electricité a été créé en 1963 par Dahir pour substituer la société Energie Electrique du Maroc, à qui était confiée depuis 1924, la concession d'une organisation de production, de transport et de distribution et qui assurait 90% de la production nationale. Le 22 avril 2012, l'Office National de l'Electricité et l'Office National de l'Eau Potable connaissent une fusion pour devenir l'Office National de l'Electricité et de l'Eau Potable. Ce regroupement qui selon la loi n°40-09 relative à l'Office National de l'Electricité et de l'Eau potable permet d'harmoniser les stratégies nationales dans ces deux secteurs clés qui sont liés par des champs de synergie.

1.2 Présentation de l'ONEE - Branche Électricité

L'Office National d'Électricité et Eau Potable - Branche Électricité (ONEE-BE) est un établissement semi-public à caractère industriel et commercial, doté de la personnalité civile et de l'autonomie financière, et placé sous la tutelle du Ministère de l'énergie et des mines. Il est chargé du service public de production, de transport et de distribution de l'énergie électrique. Les droits et obligations de l'ONEE-BE sont définis dans un cahier de charge approuvé par décret en 1974, lequel définit les conditions techniques, administratives et financières relatives à l'exploitation des ouvrages de production, transport et distribution de l'électricité. En tant que producteur, l'ONEE-Branche Electricité a la responsabilité de fournir sur tout le territoire national et à tout instant une énergie de qualité dans les meilleures conditions économiques. Le prix de cette énergie est fixé par Décret de Premier Ministre excepté en cas de gestion déléguée où elle est définie contractuellement.

1.2.1 Domaines d'activités

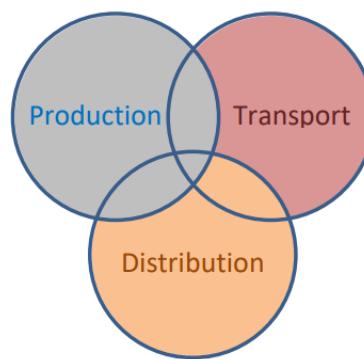


FIGURE 1.1 – Activités de l'ONE

- **La Production de l'énergie électrique :**

En tant que producteur, l'ONEE-BE a la responsabilité de fournir sur tout le territoire national et à tout instant une énergie électrique de qualité dans les meilleures conditions économiques. L'ONEE-BE assure cette fourniture par l'exploitation directe d'unités de production ainsi que par les ouvrages qu'il a confiés à des opérateurs privés dans le cadre de contrats de production concessionnels. Au delà de la gestion technique et de l'amélioration des ouvrages de son parc de production, l'ONEE - BE développe de nouveaux moyens de production et de nouvelles technologies en conciliant performance économique, expertise technique et préservation de l'environnement.

Dans ce cadre la Direction Production de l'ONEE-BE a pour mission de :

- Assurer une gestion optimale du parc de production ;
- Veiller à la satisfaction de la demande en énergie électrique exprimée par le Dispatching National et ceci dans les meilleures conditions de sécurité, de rendement, de disponibilité et du coût. Le parc de production dont dispose l'ONEE-BE est composé de moyens de production thermique, hydraulique et éolienne.

- **Le transport de l'énergie électrique :**

L'électricité n'étant pas stockable, il est donc nécessaire de gérer en continu le flux de cette énergie entre les lieux de production et les points de livraison à la clientèle. Les réseaux haute et très haute tension couvrant le territoire marocain et les interconnections régionales avec les réseaux Algérien et Espagnol sont au cœur du métier de transport

de l'électricité. Un métier hautement spécialisé que l'ONEE-BE assure pour une bonne maîtrise de la gestion des besoins et de la fluidité des échanges en temps réel. En poursuivant le développement de ces réseaux, l'ONEE-BE vise à accroître la capacité et les économies d'échanges avec les pays voisins et à renforcer la sécurité d'alimentation en énergie électrique des grands centres de consommation.

- **La distribution de l'énergie électrique :**

A travers le métier de distributeur, l'ONEE couvre toutes les activités nécessaires à la gestion et au développement des réseaux de distribution moyenne et à basse tension. Son champ d'action couvre tout le territoire national à l'exception des agglomérations urbaines gérées par des régies de distribution publiques ou par des distributeurs privées, à savoir : Casablanca, Rabat-Salé, Marrakech, Fès, Meknès, Tanger, Tétouan, Kenitra, Safi, El Jadida-Azemmour et Larache-Ksar El Kabr.

L'office assure également l'ensemble des prestations liées à la gestion de la relation clientèle dans le but d'offrir une alimentation électrique fiable et sécurisée et des services de qualité adaptés aux attentes de ses clients avec le souci d'améliorer sans cesse leur niveau de satisfaction.

La distribution de l'ONEE-BE dessert une clientèle multiple, diversifiée et très dispersée à travers tout le territoire marocain. Elle s'est fixée pour objectif d'étendre son réseau à toutes les agglomérations de façon à couvrir l'ensemble des Régions du royaume du Maroc.

1.2.2 Missions

- ✓ Répondre aux besoins du pays en énergie électrique
- ✓ Gérer et développer le réseau du transport
- ✓ Planifier, intensifier et généraliser l'extension de l'électrification rurale
- ✓ œuvrer pour la promotion et le développement des énergies renouvelables
- ✓ Assurer le service public à moindre coût

1.2.3 Organisation de l'ONEE-BE

Pour répondre au besoin du pays en électricité L'ONEE-BE maintient une activité continue en matière d'études et de réalisations d'ouvrages de production, de transport et de distribution d'énergie électrique. Pour cela, l'ONEE-BE est organisé en plusieurs pôles et directions :

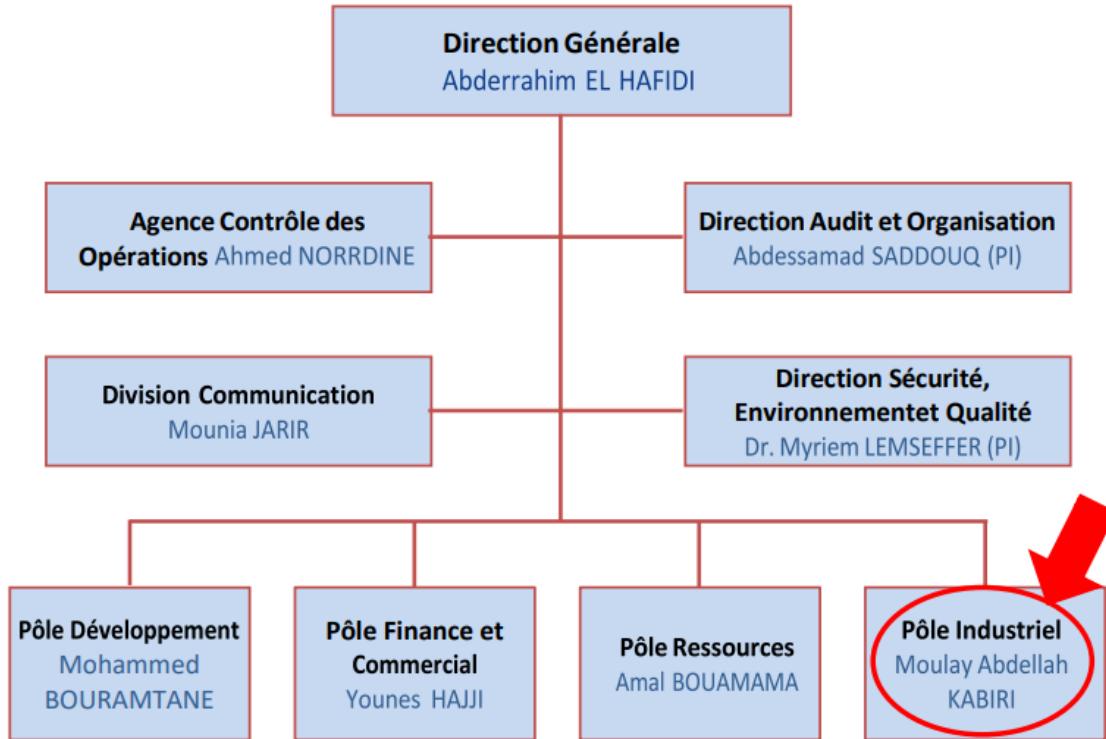


FIGURE 1.2 – Structure générale de l'Office National d'Électricité

J'ai été accueilli au sein de l'Office à Roches Noires, précisément dans la Direction d'Opération Système (DI/CTR/DOS/TS), dont l'organigramme suivant montre sa localisation dans L'O.N.E

Pôle Industriel

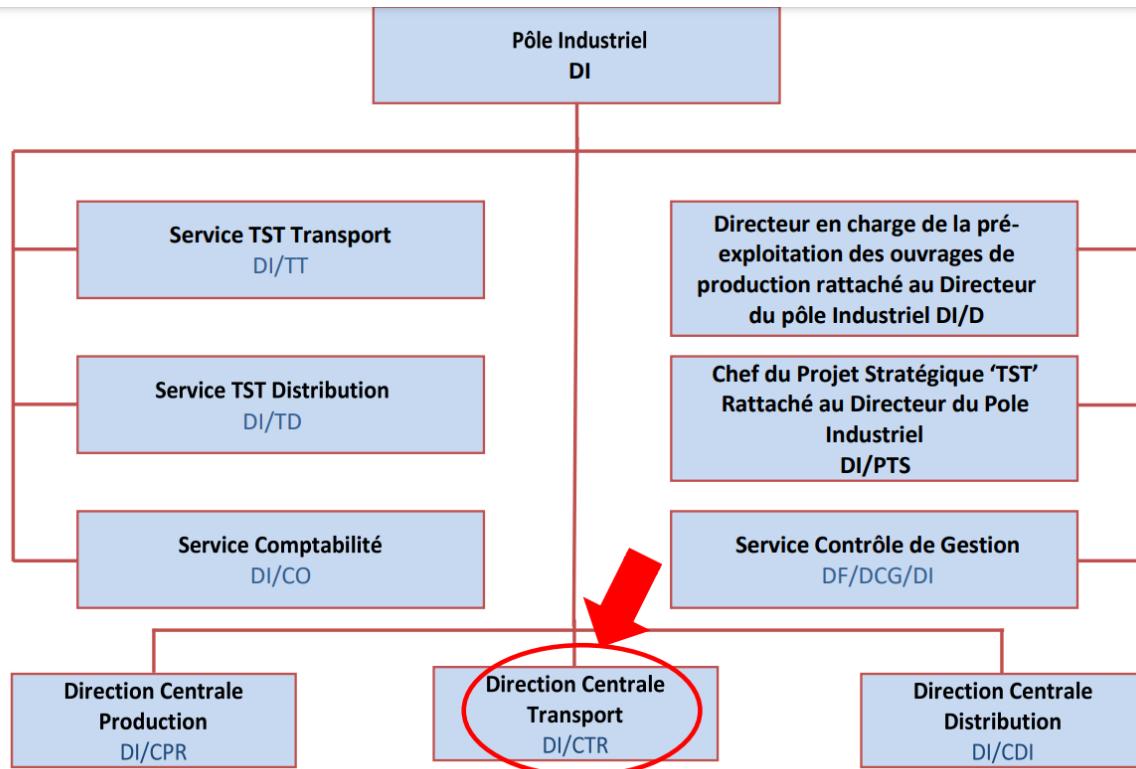


FIGURE 1.3 – Pôle industriel de l'Office National d'Électricité

Structure organisationnelle de la Direction Centrale Transport

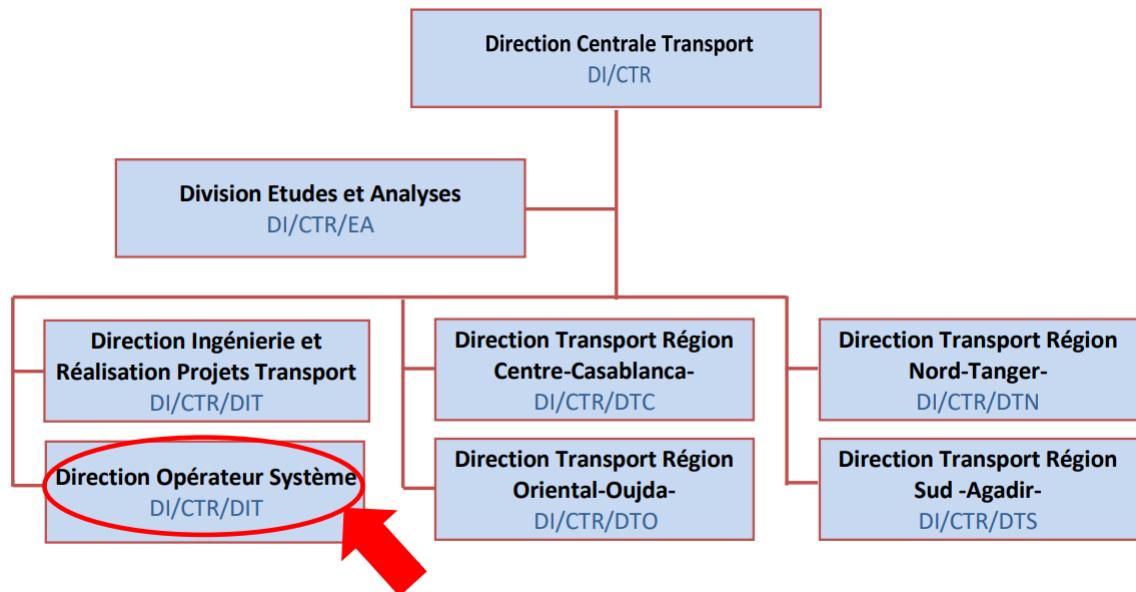


FIGURE 1.4 – Direction Centrale Transport

Structure organisationnelle de la Direction Opération Système

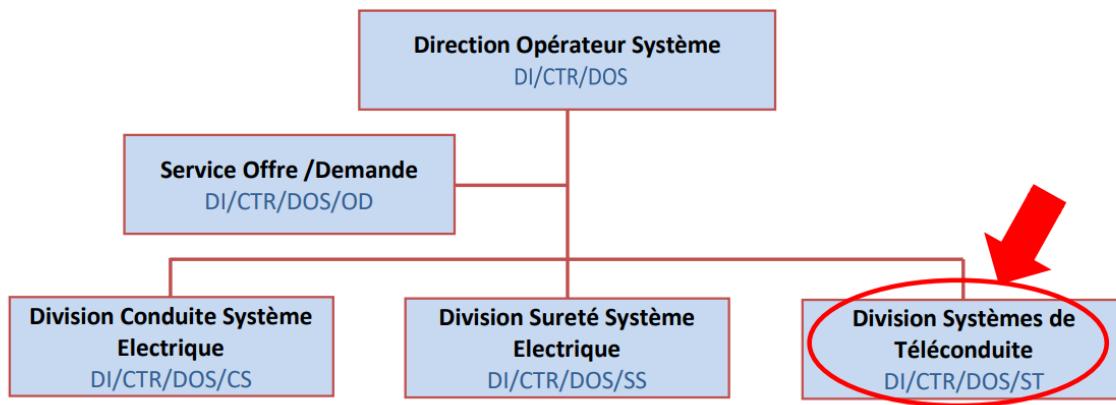


FIGURE 1.5 – Direction Opérateur Système

Structure organisationnelle de la Division de Téléconduite

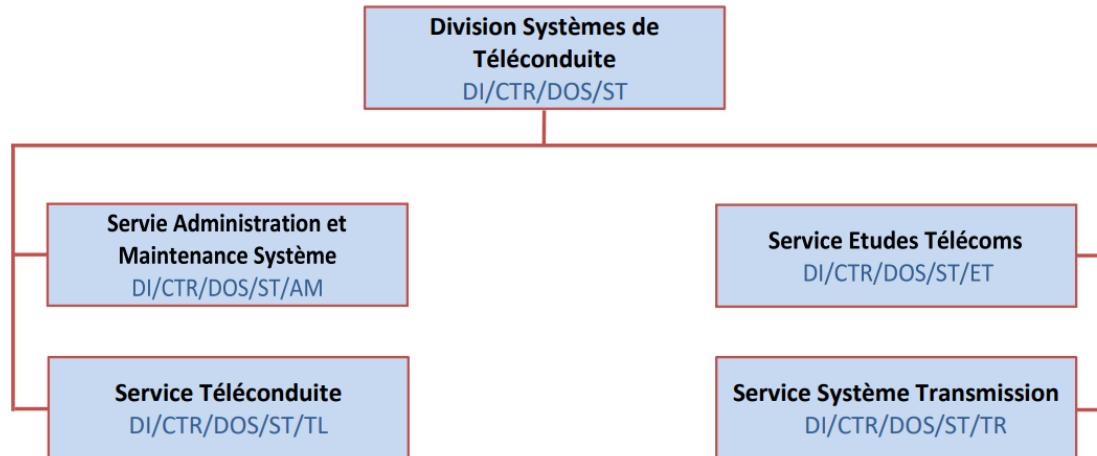


FIGURE 1.6 – Division Systèmes de Téléconduite

1.3 Présentation du Division Système Télé Conduite (DI/DOS)

La Division Système Télé Conduite (DI/DOS) au sein de l'Office National de l'Électricité et d'Eau Potable (ONE-BE) est une unité chargée de la gestion et de l'exploitation des systèmes de téléconduite utilisés dans le réseau électrique. La téléconduite est un système de contrôle à distance qui permet de surveiller et de commander les équipements électriques à distance, généralement à partir d'un centre de contrôle centralisé.

Les responsabilités de la DI/DOS peuvent inclure :

- **Gestion des infrastructures de téléconduite** : Cela comprend la conception, l'installation, la mise en service et la maintenance des équipements de téléconduite, tels que les automates programmables, les systèmes de communication, les équipements de surveillance et de contrôle, etc.
- **Surveillance du réseau électrique** : La DI/DOS surveille en permanence le réseau électrique à l'aide des systèmes de téléconduite pour détecter les pannes, les surcharges, les fluctuations de tension, etc. Elle prend également des mesures préventives pour minimiser les interruptions de service.
- **Commande à distance** : La DI/DOS peut intervenir à distance pour effectuer des opérations de commande sur le réseau électrique, telles que l'ouverture et la fermeture des disjoncteurs, le rétablissement du service après une panne, etc.
- **Optimisation des performances du réseau** : En utilisant les données recueillies par les systèmes de téléconduite, la DI/DOS peut analyser les performances du réseau électrique et proposer des améliorations pour optimiser son fonctionnement.
- **Sécurité du réseau** : La DI/DOS veille à ce que les systèmes de téléconduite soient sécurisés contre les cyberattaques et autres menaces potentielles, afin de garantir la fiabilité et la sécurité du réseau électrique.

1.4 Vision globale du projet

1.4.1 Problématique

L'énergie électrique doit être consommée dès qu'elle est produite car elle ne peut pas être stockée. La demande et l'offre doivent donc être soigneusement équilibrées grâce à des prévisions de charge précises. En outre, le succès des activités du domaine de l'énergie telles que la programmation de la charge, la planification du système électrique et l'exploitation économique des centrales électriques dépend de façon cruciale de la précision des prévisions de charge à court et à moyen terme. L'analyse des séries temporelles est l'un des moyens d'obtenir des prévisions de charge précises. L'objectif d'une telle technique est d'examiner attentivement les observations passées afin d'identifier les modèles de données qui décrivent le mieux la structure inhérente à la série et de capturer le processus de génération de données sous-jacent. L'algorithme de la moyenne mobile intégrée autorégressive (ARIMA) est une approche statistique bien connue, couramment utilisée pour la modélisation des séries temporelles afin de prédire avec précision les structures temporelles à court terme. De même, le modèle autorégressif (AR) fournit une représentation adéquate du mécanisme de génération de données basé sur les séries temporelles. Ces techniques reposent sur l'hypothèse que les données des séries temporelles ne dépendent que linéairement de certains pas de temps antérieurs de la même série temporelle. Cependant, pour les séries temporelles longues et complexes, les performances de ces techniques en matière de prévision diminuent, car les modèles qui en découlent supposent une dépendance linéaire et des propriétés stationnaires des séries temporelles.

Le défi se relève à comment concevoir et mettre en œuvre un système de prévision de la demande électrique multi-horizons, capable de fournir des estimations précises et fiables pour différents intervalles de temps, afin d'optimiser la planification et la gestion des ressources énergétiques, tout en garantissant la fiabilité et la sécurité du système électrique, notamment face à la variabilité croissante de la demande et des sources d'énergie renouvelable ?

1.4.2 L'objectif du projet et la solution proposée

Le profil de la charge électrique du Maroc suit des modèles cycliques et saisonniers complexes qui sont liés au calendrier de la production industrielle, aux impacts météorologiques, aux activités humaines et aux événements spéciaux. Les modèles non linéaires utilisant l'historique donnent généralement de meilleurs résultats que les modèles linéaires simples tels que la moyenne mobile (MA) et la moyenne mobile intégrée autorégressive (ARIMA) [14]. En général, plusieurs approches de la prévision de la charge électrique utilisant des techniques statistiques traditionnelles et des techniques d'apprentissage automatique ont été proposées pour améliorer la précision des prévisions, mais le besoin de modèles de prévision de la charge plus robustes reste une priorité [27].

Dans cette direction, les approches d'apprentissage profond ont récemment gagné une attention significative de la part de nombreux chercheurs et ont montré des progrès considérables dans de nombreux domaines tels que le traitement du langage naturel et la reconnaissance d'images [11, 12, 13].

Ces structures en couches profondes augmentent la capacité d'abstraction des caractéristiques et permettent de modéliser des modèles non linéaires complexes. En particulier, les réseaux neuronaux récurrents (RNN), qui est une architecture d'apprentissage profond conçue spécifiquement pour fonctionner sur des données de séries temporelles séquentielles, et le LSTM, qui est une variante de RNN développée à l'origine par Hochreiter et al [28].

LSTM est un schéma attrayant pour la modélisation de données séquentielles car il encode les informations contextuelles des entrées passées grâce à sa capacité à apprendre des modèles non linéaires complexes et à extraire automatiquement les caractéristiques pertinentes. Malgré son attrait, les applications de ce modèle d'apprentissage profond sont relativement rares et concernent principalement des domaines liés à l'informatique. Bien qu'il ait obtenu de meilleurs résultats que d'autres approches d'apprentissage automatique, le modèle de prévision LSTM proposé a révélé un manque de validité et n'a donc pas été en mesure de maintenir une performance élevée hors de l'échantillon. En effet, il ne tient pas compte des caractéristiques complexes de la charge électrique présentées par les séries temporelles, telles que la périodicité, la fréquence des données, les tendances, les niveaux, les ruptures structurelles et les effets de calendrier. En particulier, les modèles complexes quotidiens, hebdomadiers, mensuels et annuels de la charge électrique ne sont pas pris en compte dans les modèles de prévision précédents basés sur la méthode LSTM. Ce modèle ne considère qu'une seule séquence de charges passées comme entrée pour prédire les charges électriques à court et moyen terme. En conséquence, le modèle proposé précédemment ignore d'importantes connaissances du domaine qui pourraient avoir un impact sur la validité et la robustesse des prévisions.

Notre objectif est de développer un système de prévision avancé pour anticiper la demande électrique à différents horizons temporels, allant des prévisions à court terme aux prévisions à plus long terme, afin d'optimiser la planification et la gestion des ressources énergétiques. Pour atteindre cet objectif, nous proposons une approche basée sur les réseaux de neurones récurrents à mémoire longue et courte (LSTM). Ces modèles LSTM sont particulièrement adaptés pour capturer les dépendances temporelles complexes présentes dans les données de séries temporelles, ce qui en fait un choix approprié pour la prévision de la demande électrique. Dans notre approche, nous utiliserons des LSTM pour modéliser la relation entre les variables d'entrée, telles que la consommation d'électricité historique, les données météorologiques et les calendriers saisonniers, et les variables de sortie, qui sont les prévisions de demande électrique à différents horizons temporels. Nous prévoyons également d'optimiser les hyperparamètres des modèles LSTM, tels que le nombre de couches LSTM, le nombre de neurones par couche et la taille de la fenêtre temporelle, pour améliorer les performances des prévisions. En évaluant les performances de nos modèles LSTM à l'aide de métriques telles que l'erreur quadratique moyenne (RMSE) et l'erreur absolue moyenne (MAE), nous serons en mesure de déterminer l'efficacité de notre approche dans la prévision précise de la demande électrique à différents horizons temporels.

1.4.3 Analyse des besoins

Dans le cadre de l'analyse des besoins pour notre projet de prévision de la demande électrique multi-horizons, nous identifions plusieurs aspects clés à prendre en compte pour répondre aux attentes des utilisateurs et aux exigences du système. Ces besoins peuvent être regroupés comme suit :

1. Collecte et Intégration des Données :

Il est essentiel de collecter des données de qualité provenant de différentes sources telles que les compteurs intelligents, les données météorologiques, les données économiques et démographiques. Ces données doivent être intégrées de manière transparente dans le système pour assurer une analyse complète et précise.

2. Précision et Fiabilité :

Les utilisateurs exigent des prévisions précises et fiables pour prendre des décisions éclairées en matière de planification énergétique. Par conséquent, le modèle de prévision doit être capable de capturer les tendances et les variations saisonnières de manière efficace, tout en minimisant les erreurs de prédiction.

3. Adaptabilité :

Le système doit être capable de s'adapter aux fluctuations de la demande électrique, aux changements dans les conditions météorologiques et aux événements tels que le Ramadan et les événements spéciaux.

4. Facilité d'Utilisation :

Les utilisateurs, qu'ils soient des planificateurs énergétiques, des gestionnaires de réseau, doivent pouvoir accéder

facilement aux prévisions et aux analyses pertinentes. Par conséquent, l'interface utilisateur doit être conviviale et intuitive.

5. Interopérabilité

Le système doit être capable de s'intégrer avec d'autres systèmes d'information et outils de gestion de l'énergie déjà en place dans l'organisation, afin de faciliter le partage des données et la collaboration entre les différentes équipes.

En prenant en compte ces besoins regroupés, nous pourrons concevoir et développer un système de prévision de la demande électrique robuste et efficace, qui répond aux attentes des utilisateurs et contribue à une gestion énergétique plus intelligente et durable.

Conclusion

Ce chapitre a présenté le cadre général du projet en introduisant l'organisme d'accueil, l'Office National de l'Électricité et de l'Eau Potable (ONEE), et en donnant une vue d'ensemble de son organisation et de ses activités.

Nous avons ensuite exposé la problématique centrale du projet, qui concerne la nécessité d'un système de prévision de la demande électrique multi-horizons précis et fiable. La solution proposée s'appuie sur des techniques avancées d'apprentissage profond, notamment les réseaux de neurones récurrents à mémoire longue et courte (LSTM), pour améliorer la précision des prévisions de la demande électrique en tenant compte des dépendances temporelles complexes et des caractéristiques spécifiques de la consommation électrique au Maroc.

L'objectif du projet est de développer un modèle robuste et efficace capable de fournir des prévisions de la demande électrique à court et à moyen terme, en intégrant des données historiques, météorologiques et contextuelles.

Ainsi, ce chapitre a posé les bases pour le développement du projet en fournissant une compréhension claire de l'environnement organisationnel, des défis à relever et des approches méthodologiques envisagées. Les chapitres suivants détailleront les aspects techniques et les étapes de développement du modèle de prévision, ainsi que l'évaluation de ses performances.

Chapitre 2

Revue de la littérature

La prévision de la demande électrique est une tâche cruciale pour la gestion des réseaux de distribution et de production d'énergie. Elle permet d'assurer un équilibre entre l'offre et la demande, d'optimiser les opérations et de réduire les coûts. Plusieurs méthodes ont été développées et améliorées au fil des ans pour adresser ce problème complexe. Dans cette revue, nous explorerons les principales approches utilisées pour la prévision de la demande électrique, en mettant l'accent sur les techniques modernes qui intègrent des méthodes traditionnelles et des algorithmes d'apprentissage automatique.

2.1 Méthodes Traditionnelles

Les méthodes traditionnelles de prévision incluent les modèles basés sur les séries temporelles tels que les modèles ARIMA (AutoRegressive Integrated Moving Average). Zhang (2003) propose un modèle hybride combinant ARIMA et les réseaux de neurones pour améliorer les performances de prévision [8]. Cette approche permet de capturer à la fois les relations linéaires et non linéaires dans les données.

Une autre méthode couramment utilisée est le lissage exponentiel. Smyl (2020) présente une méthode hybride intégrant le lissage exponentiel avec les réseaux de neurones récurrents, démontrant une amélioration significative des prévisions [19].

2.2 Modèles Semi-Paramétriques et Additifs

Les modèles semi-paramétriques et additifs offrent une flexibilité accrue en combinant des composants paramétriques et non paramétriques. Fan et Hyndman (2012) proposent un modèle additif semi-paramétrique pour la prévision de la charge à court terme, intégrant des variables exogènes telles que les données météorologiques [9].

2.3 Apprentissage Automatique et Réseaux de Neurones

Avec l'avènement de l'apprentissage automatique, les réseaux de neurones, en particulier les réseaux de neurones profonds (DNN) et les réseaux de neurones récurrents (RNN), ont gagné en popularité pour la prévision de la demande électrique. Hochreiter et Schmidhuber (1997) introduisent les réseaux de neurones à mémoire à long terme (LSTM), une variante des RNN capable de capturer les dépendances à long terme dans les séries temporelles [4].

Marino et al. (2016) appliquent les réseaux de neurones profonds pour la prévision de la charge énergétique des bâtiments, démontrant leur efficacité par rapport aux méthodes traditionnelles [15]. De même, Ryu et al. (2017) explorent l'utilisation des DNN pour la prévision de la charge à court terme, obtenant des résultats prometteurs [16].

2.4 Modèles Hybrides

Les modèles hybrides combinent les avantages de différentes techniques pour améliorer la précision des prévisions. Zhang (2003) et Smyl (2020) illustrent l'efficacité des modèles hybrides en combinant ARIMA avec des réseaux de neurones et le lissage exponentiel avec des RNN, respectivement [8, 19].

2.5 Approches de Régression

Les méthodes de régression restent également populaires pour la prévision de la demande électrique. Haben et al. (2019) utilisent une régression linéaire multiple pour la prévision de la charge à un jour d'avance, comparant les performances de ce modèle avec d'autres approches plus complexes [18].

2.6 Critique

La majorité des études sur la prévision de la demande électrique se concentrent principalement sur la consommation électrique historique comme principal facteur d'entrée. Par exemple, Zhang (2003) et Smyl (2020) utilisent des modèles basés sur les séries temporelles qui reposent essentiellement sur les données de consommation passées [8, 19].

Cependant, cette approche présente des limitations significatives. La demande en électricité est influencée par une variété de facteurs externes tels que les conditions météorologiques, le type de jour (semaine vs week-end) et les jours fériés. Ignorer ces variables peut réduire la précision des prévisions. Fan et Hyndman (2012) démontrent l'importance d'intégrer des données météorologiques pour améliorer la précision des prévisions à court terme [9].

Haben et al. (2019) notent également que les modèles qui intègrent des variables exogènes, telles que la température et les données calendaires, tendent à fournir des prévisions plus robustes [18]. En conséquence, il est crucial de développer des modèles hybrides ou d'apprentissage automatique qui prennent en compte une gamme plus large de facteurs influençant la demande électrique.

Chapitre 3

Les séries temporelles

Dans ce chapitre, nous entrons dans le cœur de notre étude en explorant de manière approfondie les données disponibles. L'objectif principal de cette phase est de découvrir des tendances, des motifs et des relations cachées dans nos données, ce qui nous permettra de mieux comprendre le comportement de la demande électrique et de préparer le terrain pour la modélisation ultérieure.

Introduction

L'analyse exploratoire des données (AED) est une étape cruciale dans toute étude de séries temporelles, particulièrement lorsqu'il s'agit de prévisions de la demande électrique. Cette phase permet d'acquérir une compréhension approfondie des caractéristiques et des comportements intrinsèques des données. L'AED est une approche analytique qui combine des techniques statistiques et des visualisations graphiques pour identifier les tendances, les motifs saisonniers, les anomalies, et les relations entre les variables.

Dans ce chapitre, nous allons procéder à une exploration détaillée de nos données de séries temporelles. Nous commencerons par une présentation des données, suivie de l'analyse des statistiques descriptives pour chaque variable. Ensuite, nous examinerons les distributions des variables individuelles à l'aide de visualisations univariées, et nous explorerons les relations entre les différentes variables par le biais d'analyses de corrélation et de visualisations bivariées. Nous poursuivrons par une analyse temporelle pour identifier les tendances à court et à long terme, ainsi que les motifs saisonniers et cycliques. Enfin, nous aborderons la gestion des valeurs manquantes et des valeurs aberrantes pour garantir la qualité et l'intégrité de nos données avant de passer à la modélisation.

L'objectif de cette analyse exploratoire est de fournir une base solide pour les étapes de modélisation ultérieures en identifiant les caractéristiques clés et en comprenant les dynamiques de la demande électrique. Cette compréhension nous permettra de développer des modèles de prévision plus robustes et précis, adaptés aux particularités de nos données.

3.1 Définition d'une série temporelle

Une série temporelle (TS), également connue sous le nom série chronologique, désigne une séquence d'observations (x_1, x_2, \dots, x_n) indexées en fonction du temps, qu'il soit discrète ou continu. Cette indexation temporelle peut être établie selon diverses unités telles que la minute, l'heure, le jour, l'année, etc. La longueur de cette suite, notée "n", représente le nombre total d'observations.

En d'autres termes, une série chronologique est une séquence d'observations prises de manière séquentielle dans le temps. De nombreux ensembles de données apparaissent comme des séries temporelles : une séquence mensuelle de la quantité de marchandises expédiées par une usine, une série hebdomadaire du nombre d'accidents de la route, des précipitations quotidiennes, des observations horaires sur le rendement d'un processus chimique, etc. Les exemples de séries temporelles abondent dans des domaines tels que l'économie, les affaires, l'ingénierie, les sciences naturelles (en particulier la géophysique et la météorologie) et les sciences sociales. Les figures 3.1 et 3.2 présentent des exemples de données du type de celles qui nous intéressent sous la forme de séries chronologiques. Une caractéristique intrinsèque d'une série temporelle est que, généralement, les observations adjacentes sont dépendantes. La nature de cette dépendance entre les observations d'une série temporelle présente un intérêt pratique considérable. L'analyse des séries temporelles s'intéresse aux techniques d'analyse de cette dépendance. Cela nécessite le développement de modèles stochastiques et dynamiques pour les données de séries temporelles et l'utilisation de ces modèles de tels modèles dans des domaines d'application importants.

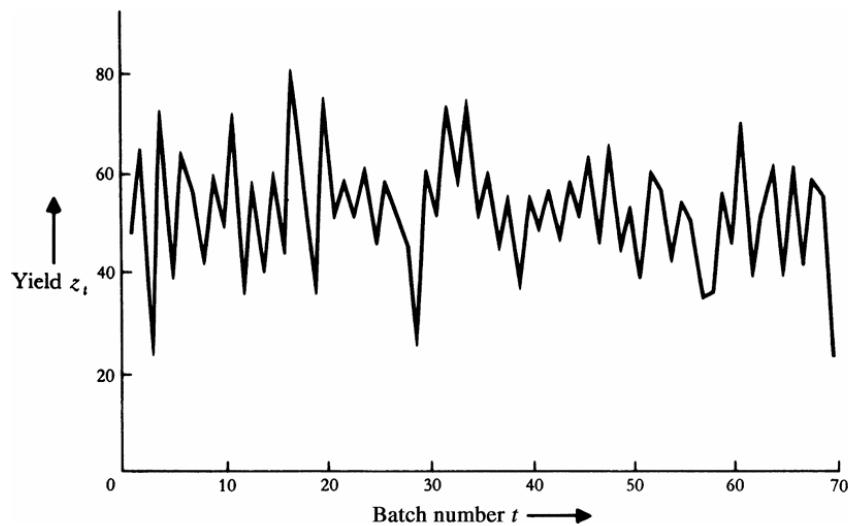


FIGURE 3.1 – Rendement de 70 batches consécutifs d'un procédé chimique [14]

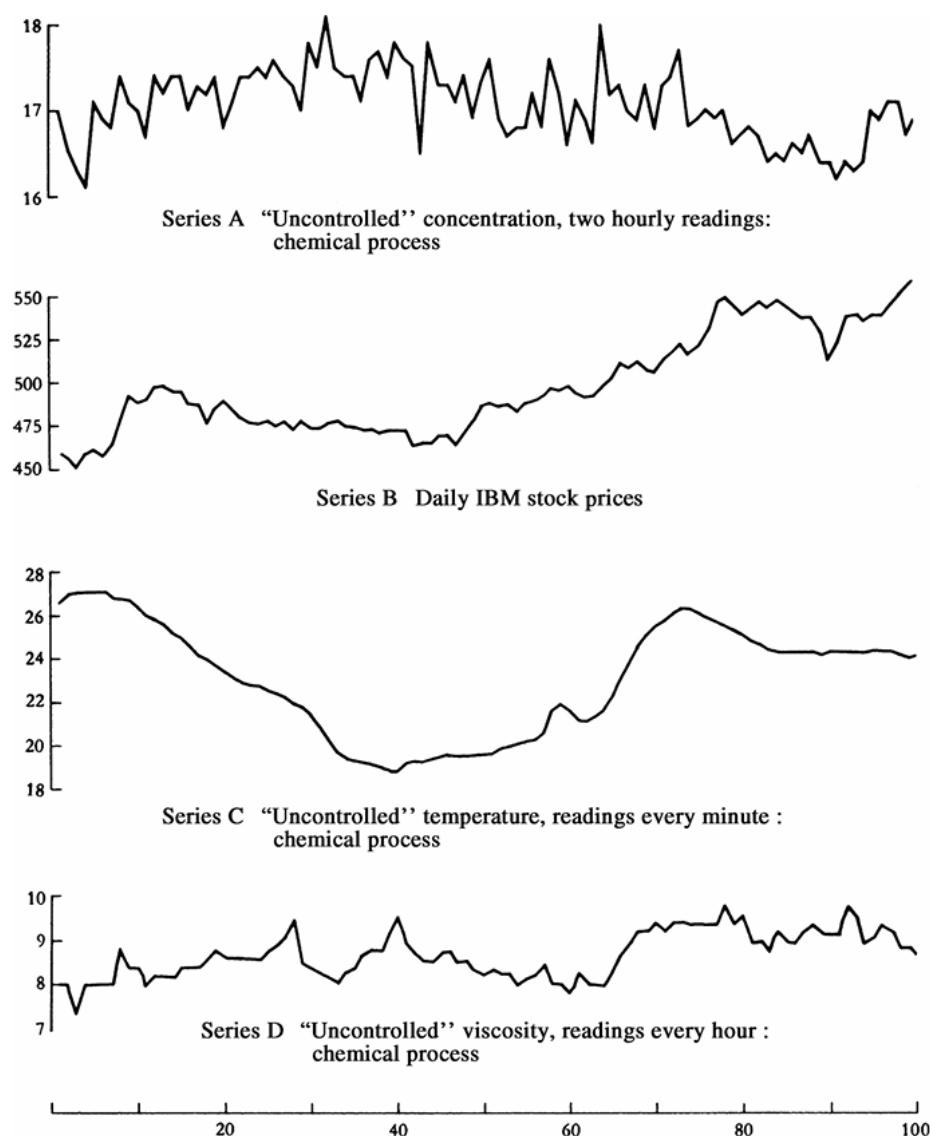


FIGURE 3.2 – Séries temporelles typiques apparaissant dans les problèmes de prévision et de contrôle [14]

3.2 Types des séries temporelles

Les séries temporelles univariées et multivariées sont deux catégories fondamentales dans l'analyse des données chronologiques.

3.2.1 Série Temporelle Univariée

Une série temporelle univariée se compose d'une seule variable mesurée à des intervalles de temps successifs. C'est le type de série temporelle le plus simple et le plus couramment étudié.

Analyser une série temporelle univariée est souvent plus simple que d'analyser une série multivariée, car il n'y a qu'une seule variable à considérer.

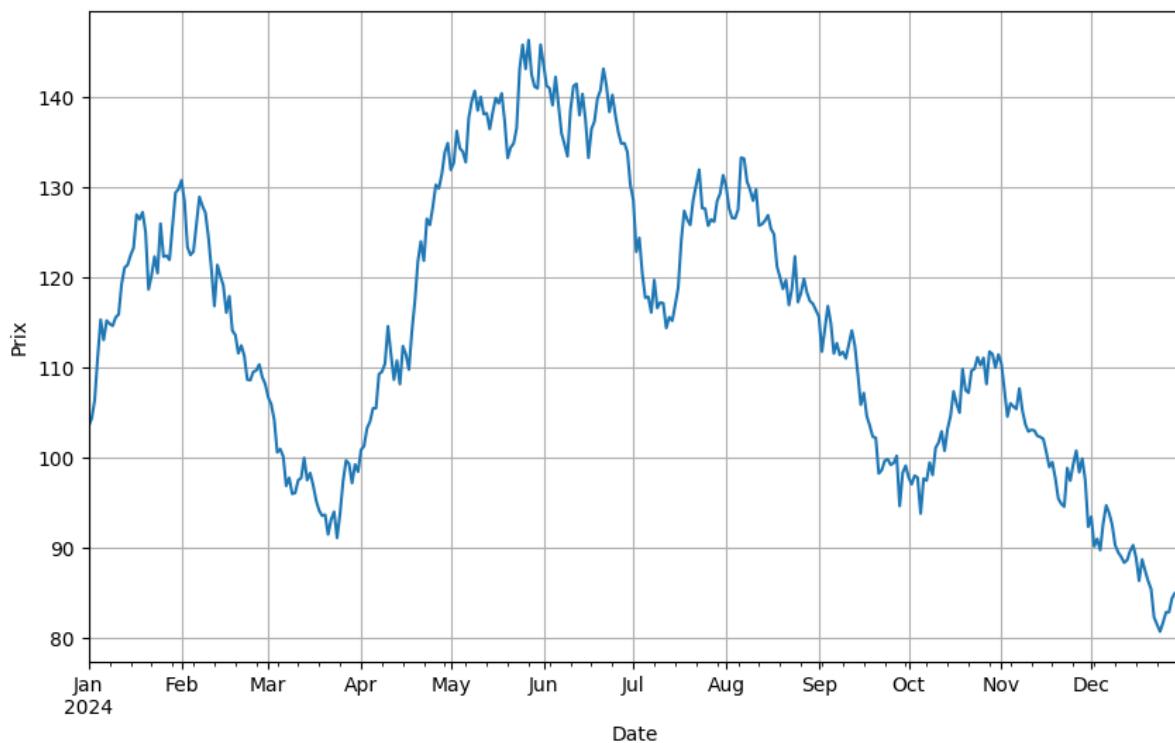


FIGURE 3.3 – Exemple d'une série temporelle univariée

Méthodes d'analyse :

- *Moyenne mobile* : Utilisée pour lisser les fluctuations et identifier les tendances à court terme.
- *Décomposition* : Sépare la série en composantes de tendance, saisonnalité et bruit.
- *Modèles ARIMA (AutoRegressive Integrated Moving Average)* : Utilisés pour modéliser et prédire les valeurs futures de la série.
- *Lissage exponentiel* : Utilisé pour lisser les données et faire des prévisions à court terme.

Applications :

- *Finance* : Prévision des prix des actions, des taux d'intérêt, etc.
- *Économie* : Analyse des indices économiques comme le PIB, le taux de chômage.
- *Météorologie* : Prévision des températures, des précipitations.

3.2.2 Série Temporelle Multivariée

Une série temporelle multivariée contient plusieurs variables mesurées simultanément à des intervalles de temps successifs. Ces variables peuvent être interdépendantes et leur analyse peut fournir une compréhension plus riche des phénomènes étudiés. L'analyse des séries temporelles multivariées est plus complexe en raison de l'interdépendance des variables.

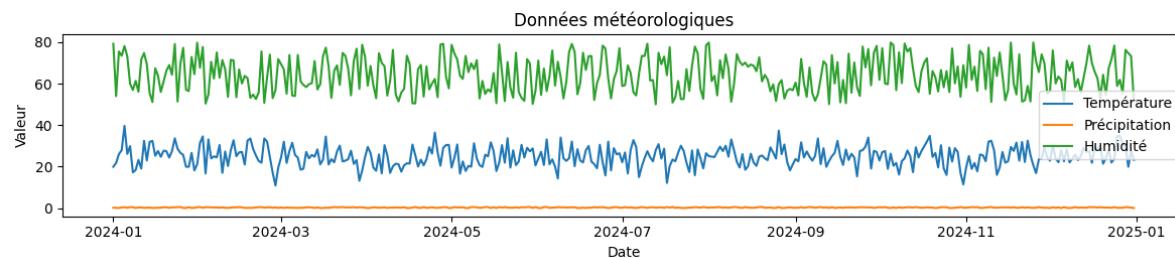


FIGURE 3.4 – Exemple d'une série temporelle multivariée

Méthodes d'analyse :

- **VAR (Vector AutoRegression)** : Utilisé pour modéliser les relations entre plusieurs séries temporelles et prévoir les valeurs futures.
- **VECM (Vector Error Correction Model)** : Une extension du VAR pour les séries temporelles non stationnaires et co-intégrées.
- **Modèles à composantes principales** : Réduisent la dimensionnalité des données tout en conservant la variation principale.
- **Analyse de cohérence spectrale** : Utilisée pour identifier les relations de fréquence entre les séries.

Applications :

- **Finance** : Analyse des relations entre différentes actions, taux de change, taux d'intérêt.
- **Économie** : Étude des interactions entre différents indicateurs économiques.
- **Météorologie** : Modélisation des interactions entre diverses variables météorologiques comme la température, l'humidité, la pression atmosphérique.

3.3 Caractéristiques d'une série temporelle

Lors du traitement d'une série temporelle pour une application donnée, il est nécessaire de tenir compte de certains éléments (ou caractéristiques) qui définissent cette série. Ces caractéristiques comprennent :

- **Tendance** : correspond à l'évolution à long terme d'une série de données, représentant son changement fondamental au fil du temps, qu'il soit croissant ou décroissant. La figure 3.5 montre un exemple de ce à quoi ressemble une tendance, où la ligne en rouge indique une tendance linéaire à long terme.

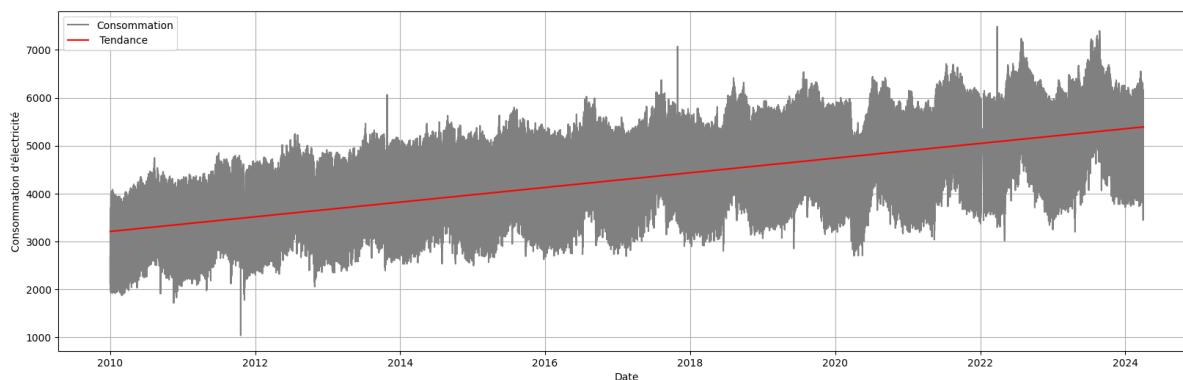


FIGURE 3.5 – Tendance linéaire

- **Saisonnalité** : une fluctuation régulière des valeurs d'une série chronologique sur des périodes fixes et connues. Le mouvement saisonnier peut être observé sur une base trimestrielle, mensuelle, hebdomadaire ou même quotidienne, et la saisonnalité est soit déterministe, soit stochastique. L'apparence de la composante saisonnière est illustrée par la figure 3.6.

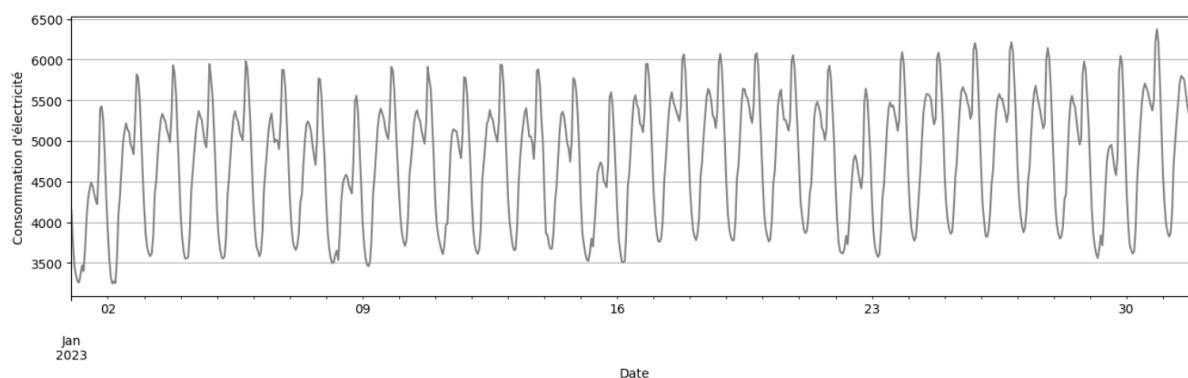


FIGURE 3.6 – Variation saisonnière

- **Variations aléatoires** : aussi connus sous le nom d'erreurs résiduelles, sont des fluctuations imprévisibles dans une série chronologique qui ne suivent pas de schéma régulier ou récurrent. Elles sont provoquées par des événements imprévus tels que des crises sanitaires, des révoltes, des conflits armés ou des grèves. La figure 3.7 illustre un exemple de série (Consommation d'électricité) à variations aléatoires. Les variations aléatoires ne sont pas prévisibles.

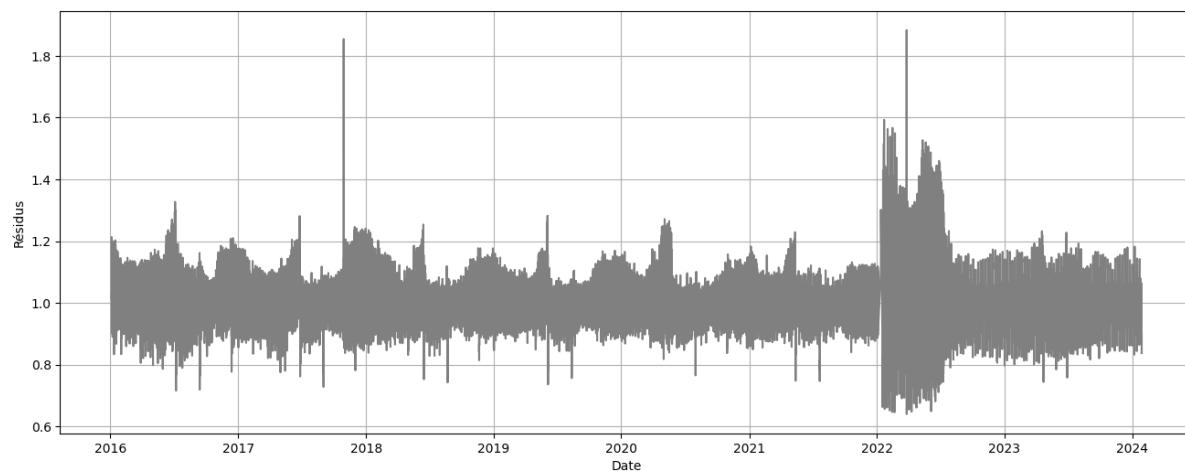


FIGURE 3.7 – Variation aléatoire

- **Valeurs aberrantes** : sont des observations qui se démarquent nettement de la distribution générale des données. Elles peuvent causer des erreurs importantes dans les modèles construits à partir de ces données. Souvent, les valeurs aberrantes sont éliminées à l'aide des techniques statistiques lors de la préparation des données pour construire le modèle. Cependant, cette approche peut entraîner la perte d'autres informations importantes.

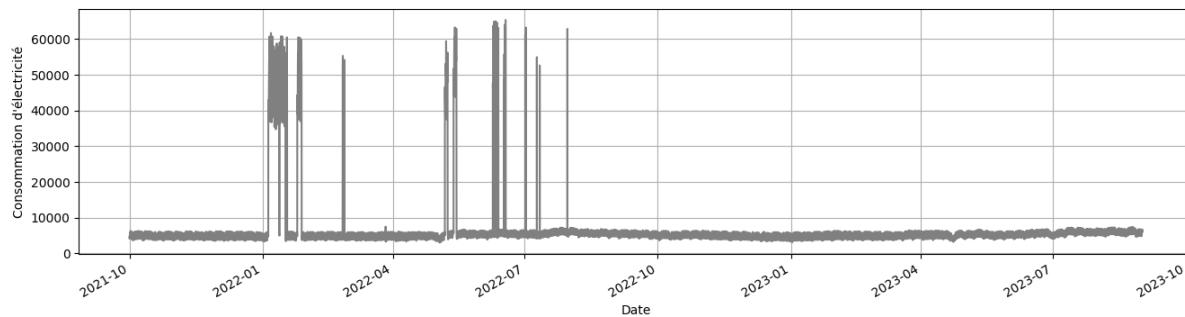


FIGURE 3.8 – Valeurs aberrantes

3.4 Stationnarité d'une série temporelle

Une des grandes questions dans l'étude de séries temporelles (ou chronologiques) est de savoir si celles-ci suivent un Processus stationnaire. On entend par là le fait que la structure du processus sous-jacent supposé évolue ou non avec le temps. Si la structure reste la même, le processus est dit alors **stationnaire**.

Une série temporelle $Y_t (t = 1, 2, \dots)$ est dite stationnaire (au sens faible) si ses propriétés statistiques ne varient pas dans le temps (Espérance, Variance, Auto-corrélation). Un exemple de série temporelle stationnaire est le *Bruit blanc*[20], par exemple une série où la loi de Y_t est une loi normale $\mathcal{N}(\mu, \sigma^2)$ indépendante de t . Un exemple de série non-stationnaire est la marche aléatoire. La marche aléatoire est un exemple de processus présentant une **racine unitaire** car elle peut être décrite par un processus auto-régressif d'ordre 1 dont le coefficient est égal à 1.

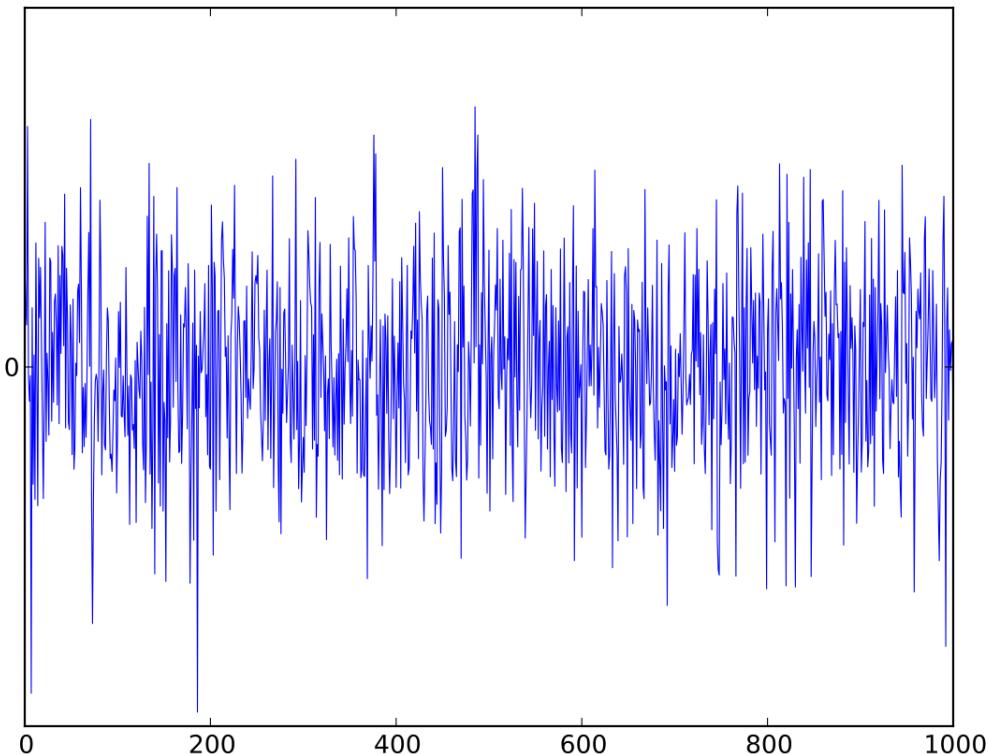


FIGURE 3.9 – Bruit Blanc

La figure 3.9 ci-dessus montre une série temporelle typique de bruit blanc. Le bruit blanc est un type de processus stochastique caractérisé par les propriétés suivantes :

- **Moyenne Constante** : La moyenne du bruit blanc est constante et égale à zéro. Cela signifie qu'il y a autant de valeurs positives que de valeurs négatives, se compensant mutuellement.
- **Variance Constante** : La variance du bruit blanc est constante au fil du temps. Dans l'image, l'amplitude des fluctuations semble rester relativement stable sur toute la série.
- **Aucune Autocorrélation** : Les observations du bruit blanc sont indépendantes et identiquement distribuées. Cela signifie qu'il n'y a pas de corrélation entre les valeurs passées, présentes et futures.

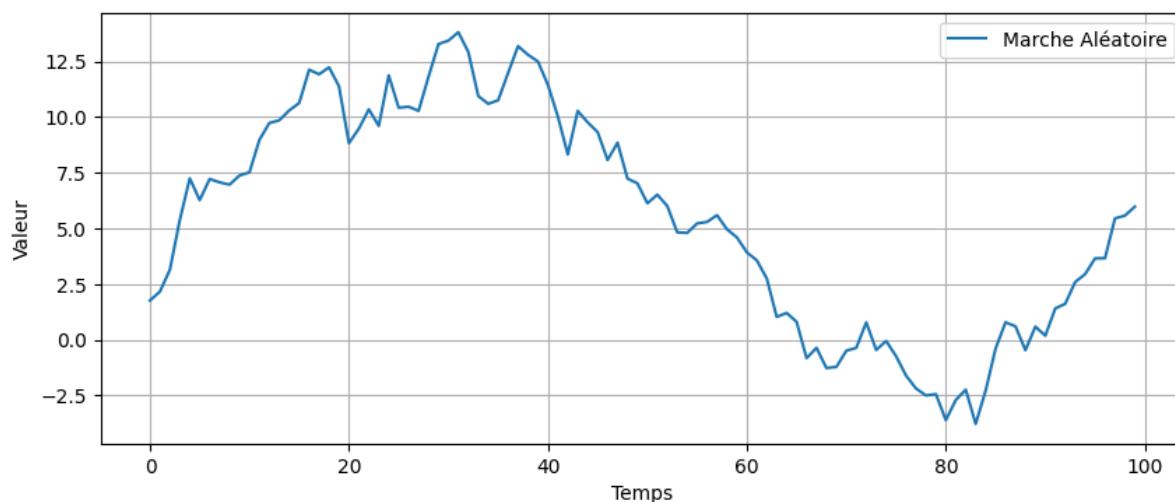


FIGURE 3.10 – Marche aléatoire

La figure 3.10 montre une série temporelle représentant une marche aléatoire. La marche aléatoire est un processus stochastique où chaque valeur de la série est la somme de la valeur précédente et d'un terme d'erreur aléatoire. Cela conduit à un comportement caractéristique que l'on peut observer dans le graphique :

- **Tendance Aléatoire** : Contrairement au bruit blanc dans la figure 3.9, la marche aléatoire présente une tendance qui peut varier de manière imprévisible. La série commence à une certaine valeur et fluctue de manière irrégulière, parfois montant, parfois descendant, sans revenir à une moyenne fixe.
- **Non-Stationnarité** : La série n'est pas stationnaire car ses propriétés statistiques (comme la moyenne et la variance) changent avec le temps. Par exemple, la variance augmente généralement avec le temps, et il n'y a pas de moyenne constante autour de laquelle les valeurs oscillent.
- **Dépendance Temporelle** : Les valeurs successives de la série sont fortement corrélées. Chaque valeur dépend de la valeur précédente plus un terme aléatoire.

3.5 Analyse des séries temporelles

L'analyse de ces séries peut fournir des informations cruciales pour la prise de décision, la prévision et la compréhension des phénomènes sous-jacents. Dans cette section, nous explorerons plusieurs outils et techniques essentiels pour analyser une série temporelle.

3.5.1 Tests de stationnarité

Il est important de pouvoir déterminer si une série temporelle est stationnaire. Plutôt que de décider entre deux options strictes, il s'agit généralement d'être en mesure d'établir, avec une forte probabilité, qu'une série est générée par un processus stationnaire.

Les méthodes les plus élémentaires de détection de la stationnarité consistent à représenter graphiquement les données, ou des fonctions de celles-ci, et à déterminer visuellement si elles présentent certaines propriétés connues des données stationnaires (ou non stationnaires).

Une autre approche, plus rigoureuse, de la détection de la stationnarité dans les données de séries temporelles consiste à utiliser des tests statistiques développés pour détecter des types spécifiques de stationnarité, à savoir ceux induits par des modèles paramétriques simples du processus stochastique générateur. [21]

Je présenterai ici les tests les plus importants.

Test de Dickey-Fuller

En statistique, le test de Dickey-Fuller teste l'hypothèse nulle selon laquelle une racine unitaire est présente dans un modèle de série temporelle autorégressive (AR). L'hypothèse alternative diffère en fonction de la version du test utilisée, mais est généralement la stationnarité ou la stationnarité de tendance. Le test est nommé d'après les statisticiens David Dickey et

Wayne Fuller, qui l'ont développé en 1979[1].

Dans un modèle AR simple,

$$y_t = \rho y_{t-1} + u_t$$

où y_t est la variable d'intérêt, t est l'indice de temps, ρ est un coefficient, et u_t est le terme d'erreur (supposé être un bruit blanc). Une racine unitaire est présente si $\rho = 1$. Le modèle serait non stationnaire dans ce cas. Le modèle de régression peut être écrit comme :

$$\Delta y_t = (\rho - 1)y_{t-1} + u_t = \delta y_{t-1} + u_t$$

où Δ est l'opérateur de première différence et $\delta \equiv \rho - 1$. Ce modèle peut être estimé, et tester la présence d'une racine unitaire est équivalent à tester $\delta = 0$. Comme le test est effectué sur le terme résiduel plutôt que sur les données brutes, il n'est pas possible d'utiliser une distribution t standard pour fournir des valeurs critiques. Par conséquent, cette statistique t a une distribution spécifique simplement appelée la table Dickey-Fuller. Il existe trois versions principales du test :

1. *Test de racine unitaire :*

$$\Delta y_t = \delta y_{t-1} + u_t$$

2. *Test de racine unitaire avec constante :*

$$\Delta y_t = a_0 + \delta y_{t-1} + u_t$$

3. *Test de racine unitaire avec constante et tendance temporelle déterministe :*

$$\Delta y_t = a_0 + a_1 t + \delta y_{t-1} + u_t$$

Chaque version du test a sa propre valeur critique qui dépend de la taille de l'échantillon. Dans chaque cas, l'hypothèse nulle est qu'il y a une racine unitaire, $\delta = 0$. Les tests ont une faible puissance statistique en ce sens qu'ils ne peuvent souvent pas distinguer entre les processus de racine unitaire vrais ($\delta = 0$) et les processus de racine unitaire proches (δ est proche de zéro). C'est ce qu'on appelle le problème d'équivalence d'observation proche. L'intuition derrière le test est la suivante. Si la série y est stationnaire (ou stationnaire avec tendance), alors elle a tendance à revenir à une moyenne constante (ou tendance de manière déterministe). Par conséquent, de grandes valeurs auront tendance à être suivies de valeurs plus petites (changements négatifs), et de petites valeurs par des valeurs plus grandes (changements positifs). En conséquence, le niveau de la série sera un prédicteur significatif du changement de la période suivante, et aura un coefficient négatif. Si, en revanche, la série est intégrée, alors les changements positifs et négatifs se produiront avec des probabilités qui ne dépendent pas du niveau actuel de la série ; dans une marche aléatoire, où vous êtes maintenant n'affecte pas la direction que vous prendrez ensuite. Il est à noter que

$$\Delta y_t = a_0 + u_t$$

peut être réécrit comme

$$y_t = y_0 + \sum_{i=1}^t u_i + a_0 t$$

avec une tendance déterministe provenant de $a_0 t$ et un terme d'interception stochastique provenant de $y_0 + \sum_{i=1}^t u_i$, ce qui donne ce qu'on appelle une tendance stochastique.

Test Augmenté de Dickey-Fuller

L'ADF (Test Augmenté de Dickey-Fuller) teste l'hypothèse nulle selon laquelle une racine unitaire est présente dans un échantillon de série temporelle. L'hypothèse alternative diffère en fonction de la version du test utilisée, mais est généralement la stationnarité ou la stationnarité de tendance. Il s'agit d'une version augmentée du test de Dickey-Fuller pour un ensemble de modèles de séries chronologiques plus grand et plus compliqué. La statistique ADF (test augmenté de Dickey-Fuller) utilisée dans le test est un nombre négatif. Plus il est négatif, plus le rejet de l'hypothèse qu'il existe une racine unitaire est fort à un certain niveau de confiance.

La procédure de test pour le test ADF est la même que pour le test de Dickey-Fuller, mais elle est appliquée au modèle

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \delta_1 \Delta y_{t-1} + \cdots + \delta_{p-1} \Delta y_{t-p+1} + \varepsilon_t,$$

où α est une constante, β est le coefficient d'une tendance temporelle et p est l'ordre de retard du processus autorégressif. L'imposition des contraintes $\alpha = 0$ et $\beta = 0$ correspond à modéliser une marche aléatoire, et l'utilisation de la contrainte $\beta = 0$ correspond à modéliser une marche aléatoire avec une dérivé. Par conséquent, il existe trois versions principales du test, analogues à celles discutées dans le test de Dickey-Fuller.

En incluant des retards de l'ordre p , la formulation ADF permet de prendre en compte des processus autorégressifs d'ordre supérieur. Cela signifie que la longueur du retard p doit être déterminée lors de l'application du test. Une approche possible est de tester en descendant à partir d'ordres élevés et d'examiner les valeurs t sur les coefficients.

Test KPSS

Les tests Kwiatkowski–Phillips–Schmidt–Shin (KPSS) sont utilisés pour tester l'hypothèse nulle selon laquelle une série temporelle observable est stationnaire autour d'une tendance déterministe (c'est-à-dire stationnaire de tendance) par rapport à l'alternative d'une racine unitaire.

Contrairement à la plupart des tests de racine unitaire, la présence d'une racine unitaire n'est pas l'hypothèse nulle mais l'alternative. De plus, dans le test KPSS, l'absence de racine unitaire n'est pas une preuve de stationnarité mais, par conception, de stationnarité de tendance. Cette distinction est importante car il est possible qu'une série temporelle soit non stationnaire, n'ait pas de racine unitaire et soit néanmoins stationnaire de tendance. Dans les processus de racine unitaire et stationnaires de tendance, la moyenne peut croître ou décroître au fil du temps ; cependant, en présence d'un choc, les processus stationnaires de tendance ont tendance à revenir à leur moyenne (c'est-à-dire transitoire, la série temporelle convergera à nouveau vers la moyenne croissante, qui n'a pas été affectée par le choc), tandis que les processus à racine unitaire ont un impact permanent sur la moyenne (c'est-à-dire aucune convergence au fil du temps).

3.5.2 Fonctions d'Autocorrélation (ACF) et d'Autocorrélation Partielle (PACF)

Les fonctions d'autocorrélation (ACF) et d'autocorrélation partielle (PACF) sont des outils graphiques utilisés pour identifier les modèles de dépendance dans une série temporelle. L'ACF mesure la corrélation entre une observation et les observations décalées à différents intervalles de temps. La PACF, en revanche, mesure la corrélation entre une observation et une observation décalée, tout en éliminant les contributions des intervalles de temps intermédiaires.

Fonction d'Autocorrélation

L'autocorrélation est un concept fondamental dans l'analyse de séries temporelles. L'autocorrélation est un concept statistique qui évalue le degré de corrélation entre les valeurs d'une variable à différents moments dans le temps.

L'autocorrélation mesure le degré de similitude entre une série temporelle donnée et sa version retardée sur des périodes de temps successives. C'est similaire au calcul de la corrélation entre deux variables différentes, sauf qu'en autocorrélation, nous calculons la corrélation entre deux versions différentes X_t et X_{t-k} de la même série temporelle.

Mathématiquement, le coefficient d'autocorrélation est noté par le symbole ρ (rho) et est exprimé comme $\rho(k)$, où k représente le décalage temporel ou le nombre d'intervalle entre les observations. Le coefficient d'autocorrélation est calculé en utilisant la corrélation de Pearson ou la covariance.[22]

Pour une série temporelle, l'autocorrélation au décalage k ($\rho(k)$) est déterminée en comparant les valeurs de la variable au temps t avec les valeurs au temps $t - k$.

$$\rho(k) = \frac{\text{Cov}(X_t, X_{t-k})}{\sigma(X_t) \cdot \sigma(X_{t-k})}$$

Ici,

- Cov est la covariance
- σ est l'écart-type
- X_t représente la variable au temps t

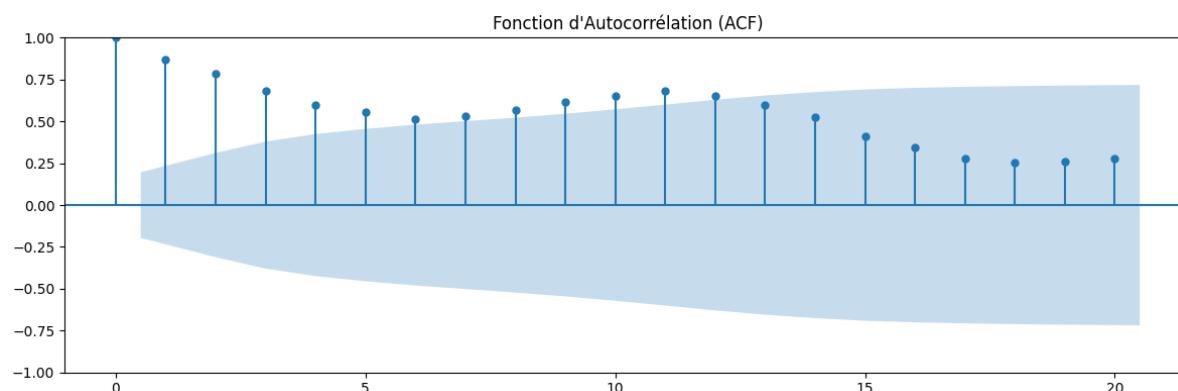


FIGURE 3.11 – ACF

La figure 3.14a de la fonction d'autocorrélation (ACF) montre une décroissance progressive des valeurs d'autocorrélation à mesure que les lags augmentent. Initialement, l'autocorrélation est très élevée à lag 1, ce qui indique une forte relation entre les

observations adjacentes de la série temporelle. À partir du lag 2, les valeurs d'autocorrélation diminuent graduellement, tout en restant significatives jusqu'à environ le lag 10. Cette tendance suggère que la série possède une dépendance à long terme, où les valeurs passées continuent d'influencer les valeurs futures au-delà des premiers lags. Au-delà du lag 10, les valeurs d'autocorrélation deviennent de moins en moins significatives, bien que certains lags (comme ceux autour de 15-20) montrent encore des pics d'autocorrélation. La présence de ces pics pourrait indiquer des cycles ou des structures sous-jacentes plus complexes dans la série temporelle. Dans l'ensemble, ce schéma d'autocorrélation indique une série temporelle avec une forte persistance, où les observations passées ont une influence notable sur les observations futures.

Interprétation de l'autocorrélation :

- Une autocorrélation positive ($\rho > 0$) indique une tendance pour les valeurs à un moment donné à être positivement corrélées avec les valeurs à un moment ultérieur. Une autocorrélation élevée à un décalage spécifique suggère une forte relation linéaire entre les valeurs actuelles et ses valeurs passées à ce décalage.
- Une autocorrélation négative ($\rho < 0$) suggère une relation inverse entre les valeurs à différents intervalles de temps. Une autocorrélation faible ou nulle indique un manque de dépendance linéaire entre les valeurs actuelles et passées de la variable à ce décalage.

Utilisations de l'autocorrélation :

- L'autocorrélation détecte les motifs et les tendances répétitives dans les données de séries temporelles. Une autocorrélation positive à des décalages spécifiques peut indiquer la présence de saisonnalité.
- L'autocorrélation guide la détermination de l'ordre des modèles ARIMA et MA en fournissant des informations sur le nombre de termes de décalage à inclure.
- L'autocorrélation aide à vérifier si une série temporelle est stationnaire ou présente des tendances et un comportement non stationnaire.
- Des pics ou des baisses soudaines de l'autocorrélation à certains décalages peuvent indiquer la présence d'anomalies et de valeurs aberrantes.

Fonction d'Autocorrélation Partielle

Dans l'analyse des séries temporelles, la fonction de corrélation partielle (PACF) donne la corrélation partielle d'une série temporelle stationnaire avec ses propres valeurs retardées, en régressant les valeurs de la série temporelle à tous les retards plus courts. Elle est différente de la fonction d'autocorrélation, qui ne contrôle pas les autres retards.

La corrélation partielle quantifie la relation entre une observation spécifique et ses valeurs retardées. Cela nous aide à examiner l'influence directe d'un point temporel passé sur le point temporel actuel, en excluant l'influence indirecte à travers les autres valeurs retardées. Elle cherche à déterminer la corrélation unique entre un point temporel spécifique et un autre point temporel, en tenant compte de l'influence des points temporels intermédiaires.

$$PACF(T_i, k) = \frac{Cov[(T_i | T_{i-1}, T_{i-2}, \dots, T_{i-k+1}), (T_{i-k} | T_{i-1}, T_{i-2}, \dots, T_{i-k+1})]}{\sigma_{T_i | T_{i-1}, T_{i-2}, \dots, T_{i-k+1}} \cdot \sigma_{T_{i-k} | T_{i-1}, T_{i-2}, \dots, T_{i-k+1}}}$$

Voici,

- $T_i | T_{i-1}, T_{i-2}, \dots, T_{i-k+1}$ est la série temporelle des résidus obtenue en ajustant un modèle linéaire multivarié à $T_{i-1}, T_{i-2}, \dots, T_{i-k+1}$ pour prédire T_i .
- $T_k | T_{i-1}, T_{i-2}, \dots, T_{i-k+1}$ est la série temporelle des résidus obtenue en ajustant un modèle linéaire multivarié à $T_{i-1}, T_{i-2}, \dots, T_{i-k+1}$ pour prédire T_{i-k} .

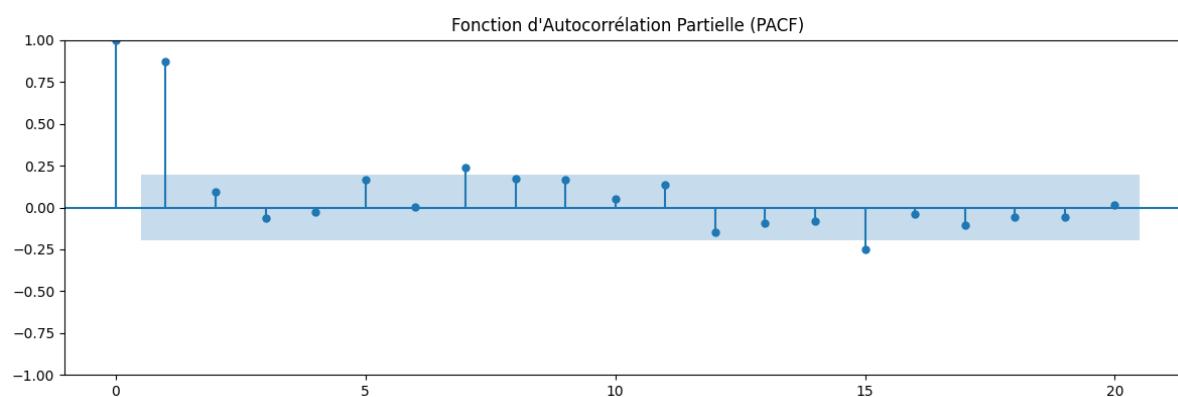


FIGURE 3.12 – PACF

La figure 8.14b de la fonction d'autocorrélation partielle (PACF) montre des valeurs significatives à quelques lags spécifiques, en particulier au lag 1 où la valeur est très élevée, proche de 1. Cela indique une forte autocorrélation directe avec la première observation précédente. Aux lags 2 et 3, les valeurs restent significatives mais sont nettement plus faibles, ce qui suggère que l'influence directe diminue rapidement après le premier lag. Au-delà du lag 3, la plupart des valeurs d'autocorrélation partielle deviennent non significatives, à l'exception de quelques lags sporadiques, ce qui indique qu'il n'y a pas de dépendance directe substantielle après le troisième lag. Cette coupure nette après le lag 1 suggère qu'un modèle autorégressif d'ordre 1 (AR(1)) pourrait bien capturer la structure principale de la série temporelle. Cependant, les quelques valeurs significatives à des lags plus élevés peuvent suggérer la présence de dynamiques plus complexes ou de structures supplémentaires dans la série.

L'autocorrélation est importante dans les séries temporelles car :

1. L'autocorrélation aide à révéler les motifs ou tendances répétitifs au sein d'une série temporelle. En analysant comment une variable est corrélée avec ses valeurs passées à différents décalages, les analystes peuvent identifier la présence de motifs cycliques ou saisonniers dans les données. Par exemple, dans les données économiques, l'autocorrélation peut révéler si certains indicateurs économiques présentent des motifs réguliers sur des intervalles de temps spécifiques, tels que des cycles mensuels ou trimestriels.
2. Les analystes financiers et les traders utilisent souvent l'autocorrélation pour analyser les mouvements de prix historiques sur les marchés financiers. En identifiant les motifs d'autocorrélation dans les changements de prix passés, ils peuvent tenter de prédire les mouvements de prix futurs. Par exemple, s'il y a une autocorrélation positive à un décalage spécifique, indiquant une tendance dans les mouvements de prix, les traders peuvent utiliser cette information pour éclairer leurs prédictions et leurs stratégies de trading.
3. La Fonction d'Autocorrélation (ACF) est un outil crucial pour modéliser les données de séries temporelles. L'ACF aide à identifier quels décalages présentent des corrélations significatives avec l'observation actuelle. Dans la modélisation des séries temporelles, comprendre la structure d'autocorrélation est essentiel pour sélectionner des modèles appropriés. Par exemple, s'il y a une autocorrélation significative à un décalage particulier, cela peut suggérer la présence d'un composant autorégressif (AR) dans le modèle, influençant la valeur actuelle en fonction des valeurs passées. Le graphique ACF permet aux analystes d'observer la décroissance de l'autocorrélation sur les décalages, guidant le choix des valeurs de décalage à inclure dans les modèles autorégressifs.

3.5.3 Décomposition des séries temporelles

Les données de séries temporelles peuvent présenter une variété de motifs, et il est souvent utile de diviser une série temporelle en plusieurs composantes, chacune représentant une catégorie de motif sous-jacent.

Lorsque nous décomposons une série temporelle en composantes, nous combinons généralement la tendance et le cycle en une seule composante tendance-cycle (parfois appelée tendance pour simplifier). Ainsi, nous considérons une série temporelle comme composée de trois composantes : une composante tendance-cycle, une composante saisonnière et une composante de reste (contenant tout ce qui reste dans la série temporelle).

Dans cette partie, nous examinons quelques méthodes courantes pour extraire ces composantes d'une série temporelle. Souvent, cela est fait pour aider à améliorer la compréhension de la série temporelle, mais cela peut également être utilisé pour améliorer la précision des prévisions.

Si nous supposons une décomposition additive, alors nous pouvons écrire

$$y_t = S_t + T_t + R_t,$$

où y_t est la donnée, S_t est la composante saisonnière, T_t est la composante tendance-cycle, et R_t est la composante de reste, tous à la période t . Alternative, une décomposition multiplicative serait écrite comme

$$y_t = S_t \times T_t \times R_t.$$

La décomposition additive est la plus appropriée si l'amplitude des fluctuations saisonnières, ou la variation autour de la tendance-cycle, ne varie pas avec le niveau de la série temporelle. Lorsque la variation dans le motif saisonnier, ou la variation autour de la tendance-cycle, semble être proportionnelle au niveau de la série temporelle, alors une décomposition multiplicative est plus appropriée. Les décompositions multiplicatives sont courantes avec les séries temporelles économiques.

Une alternative à l'utilisation d'une décomposition multiplicative est de d'abord transformer les données jusqu'à ce que la variation dans la série semble stable au fil du temps, puis utiliser une décomposition additive. Lorsqu'une transformation logarithmique a été utilisée, cela équivaut à utiliser une décomposition multiplicative car

$$y_t = S_t \times T_t \times R_t \text{ est équivalent à } \log y_t = \log S_t + \log T_t + \log R_t.$$

Nous examinerons plusieurs méthodes pour obtenir les composantes S_t , T_t et R_t plus tard dans cette partie, mais d'abord, il est utile de voir un exemple. Nous allons décomposer l'indice des nouvelles commandes d'équipements électriques illustré dans la figure 3.13. Les données montrent le nombre de nouvelles commandes pour les équipements électriques (produits informatiques, électroniques et optiques) dans la zone euro (16 pays). Les données ont été ajustées en fonction des jours ouvrables et normalisées de telle sorte qu'une valeur de 100 correspond à 2005.[23]

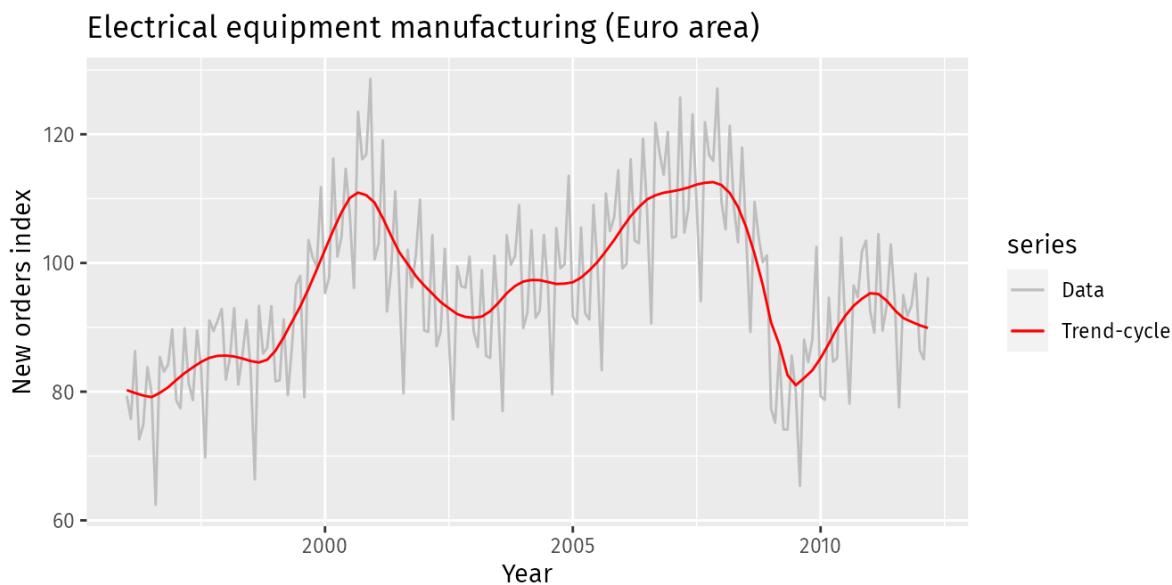


FIGURE 3.13 – Commandes d'équipements électriques : la composante tendance-cycle (rouge) et les données brutes (gris)

La figure 3.13 montre la composante tendance-cycle, T_t , en rouge et les données originales, y_t , en gris. La composante tendance-cycle montre le mouvement global de la série, en ignorant la saisonnalité et toutes les petites fluctuations aléatoires. La figure 3.14 montre une décomposition additive de ces données. La méthode utilisée pour estimer les composantes dans cet exemple est STL, qui est discutée dans la Section du STL.

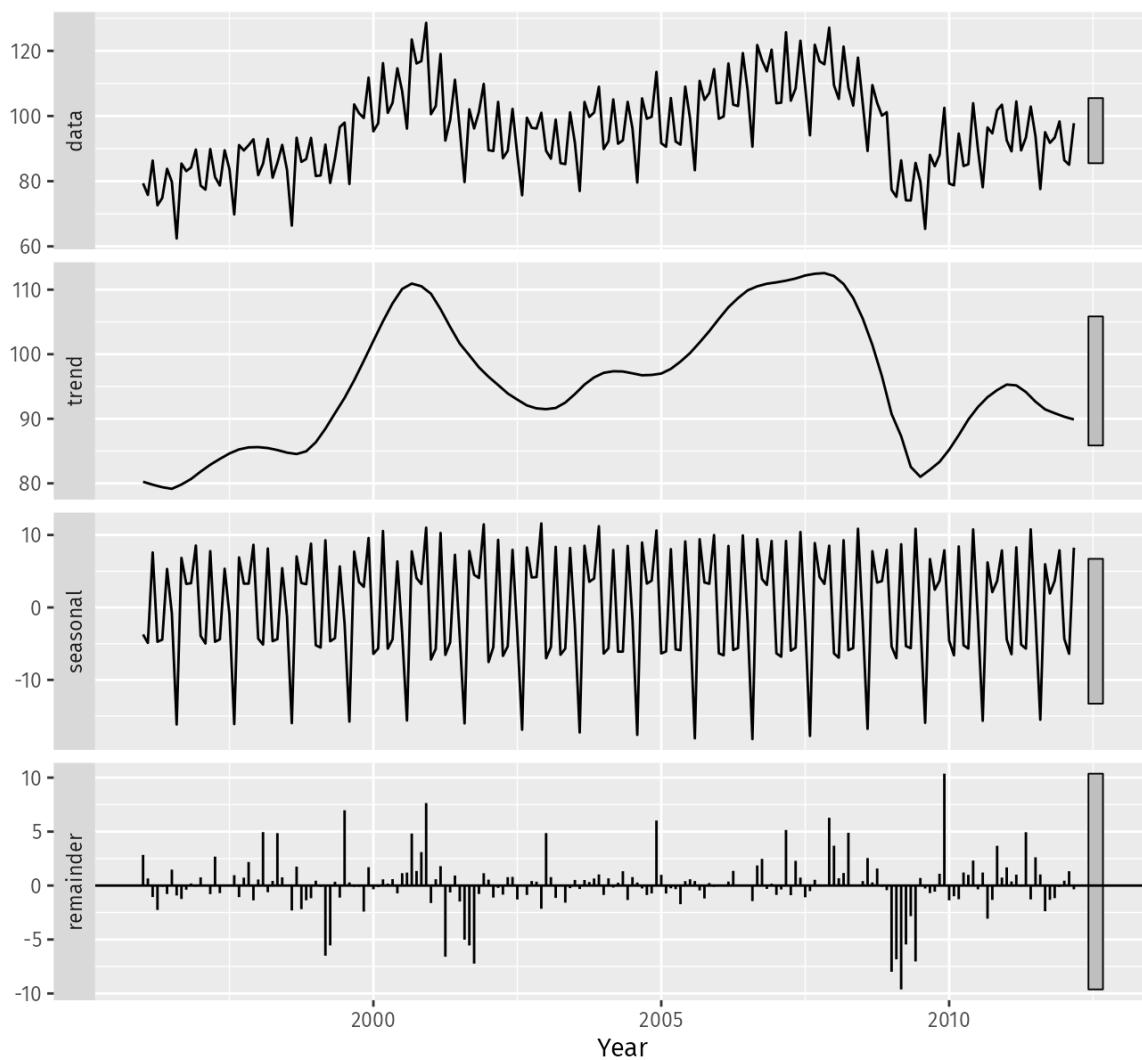


FIGURE 3.14 – Les commandes d'équipements électriques (en haut) et ses trois composantes additives

Les trois composantes sont montrées séparément dans les trois panneaux inférieurs de la figure 3.14. Ces composantes peuvent être ajoutées ensemble pour reconstruire les données montrées dans le panneau supérieur. On remarque que la composante saisonnière change lentement avec le temps, de sorte que deux années consécutives ont des modèles similaires, mais des années éloignées peuvent avoir des modèles saisonniers différents. La composante reste montrée dans le panneau inférieur est ce qui reste lorsque les composantes saisonnière et de tendance-cycle ont été soustraite des données.

Les barres grises à droite de chaque panneau montrent les échelles relatives des composantes. Chaque barre grise représente la même longueur mais parce que les graphiques sont à des échelles différentes, les barres varient en taille. La grande barre grise dans le panneau inférieur montre que la variation dans la composante reste est petite par rapport à la variation dans les données, qui a une barre d'environ un quart de la taille. Si nous réduisions les trois panneaux inférieurs jusqu'à ce que leurs barres deviennent de la même taille que celle dans le panneau de données, alors tous les panneaux seraient à la même échelle.

Décomposition classique

L'une des méthodes de décomposition temporelle les plus classiques, apparue dans les années 1920, est la méthode classique de décomposition. Elle constitue une procédure relativement simple et sert de point de départ pour la plupart des autres méthodes de décomposition des séries temporelles. La méthode classique de décomposition existe sous deux formes : une décomposition additive et une décomposition multiplicative. Voici leur description pour une série temporelle avec une période saisonnière m .

Décomposition additive :• *Étape 1 :*

Calculer la composante tendance-cycle, \hat{T}_t , en utilisant une $2 \times m$ -MA si m est pair, ou une MA- m si m est impair.

• *Étape 2 :*

Calculer la série détrendue : $y_t - \hat{T}_t$.

• *Étape 3 :*

Estimer la composante saisonnière, \hat{S}_t , en faisant la moyenne des valeurs détrendées pour chaque saison. Assurez-vous que ces valeurs de composante saisonnière s'additionnent à zéro.

• *Étape 4 :*

Obtenir la composante reste, \hat{R}_t , en soustrayant les composantes saisonnière et tendance-cycle estimées :

$$\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t.$$

Décomposition multiplicative :• *Étape 1 :*

Calculer la composante tendance-cycle, \hat{T}_t , en utilisant la même méthode de moyenne mobile (MA) que dans la décomposition additive.

• *Étape 2 :*

Calculer la série détrendue : y_t / \hat{T}_t .

• *Étape 3 :*

Estimer la composante saisonnière, \hat{S}_t , en faisant la moyenne des valeurs détrendées pour chaque saison. Ajustez ces indices saisonniers pour vous assurer qu'ils s'additionnent à m .

• *Étape 4 :*

Obtenir la composante reste, \hat{R}_t , en divisant les composantes saisonnière et tendance-cycle estimées :

$$\hat{R}_t = y_t / (\hat{T}_t \hat{S}_t).$$

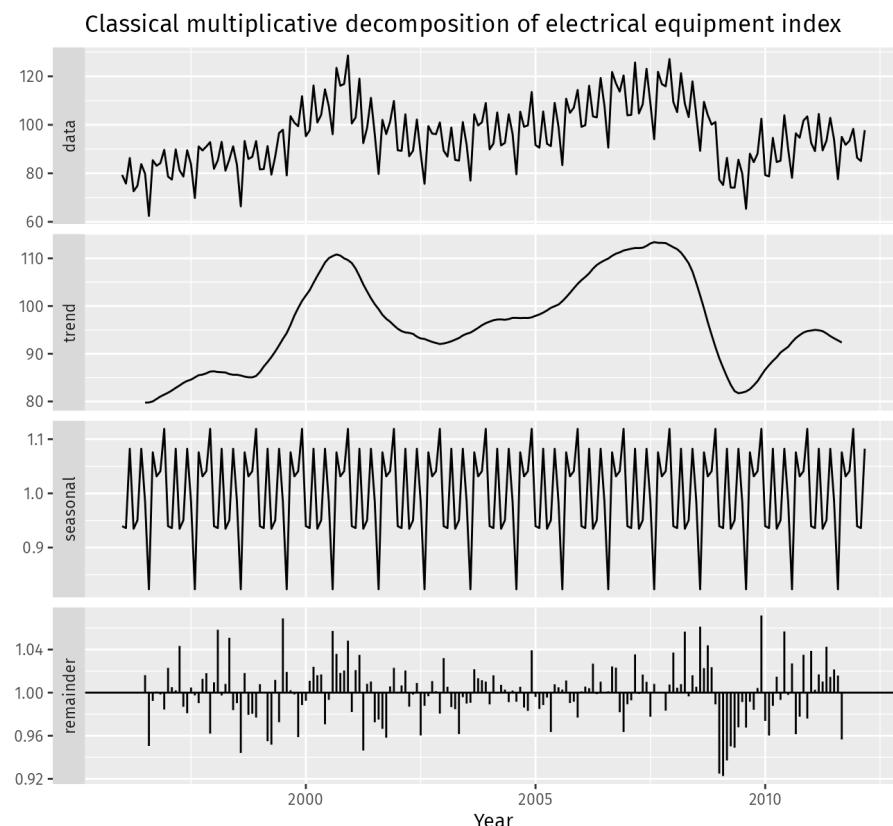


FIGURE 3.15 – Décomposition multiplicative classique de l'indice des nouvelles commandes pour l'équipement électrique

La figure 3.15 montre une décomposition classique de l'indice de l'équipement électrique. Comparez cette décomposition avec celle présentée dans la figure 6.1. La série de valeurs de reste inférieures à 1 en 2009 suggère qu'il y a une certaine "fuite" du composant tendance-cycle dans le composant de reste. L'estimation du tendance-cycle a trop lissé la baisse des données, et les valeurs de reste correspondantes ont été affectées par la mauvaise estimation du tendance-cycle.

Cependant, malgré sa simplicité et son utilité, la méthode classique présente des limitations dans la modélisation de certains types de données. Par conséquent, des méthodes plus avancées ont été développées, telles que la méthode X11 et la méthode de décomposition par STL (Seasonal and Trend decomposition using Loess). Ces méthodes offrent des techniques plus sophistiquées pour séparer les différentes composantes d'une série temporelle, ce qui permet une analyse plus approfondie et des prévisions plus précises.

Décomposition X11

Cette méthode est basée sur la décomposition classique, mais inclut de nombreuses étapes et fonctionnalités supplémentaires afin de surmonter les limitations de la décomposition classique qui ont été discutées dans la section précédente. En particulier, des estimations de tendance-cycle sont disponibles pour toutes les observations, y compris les points finaux, et la composante saisonnière est autorisée à varier lentement dans le temps. X11 gère à la fois la décomposition additive et la décomposition multiplicative. Le processus est entièrement automatique et tend à être très robuste aux valeurs aberrantes et aux changements de niveau dans la série temporelle.

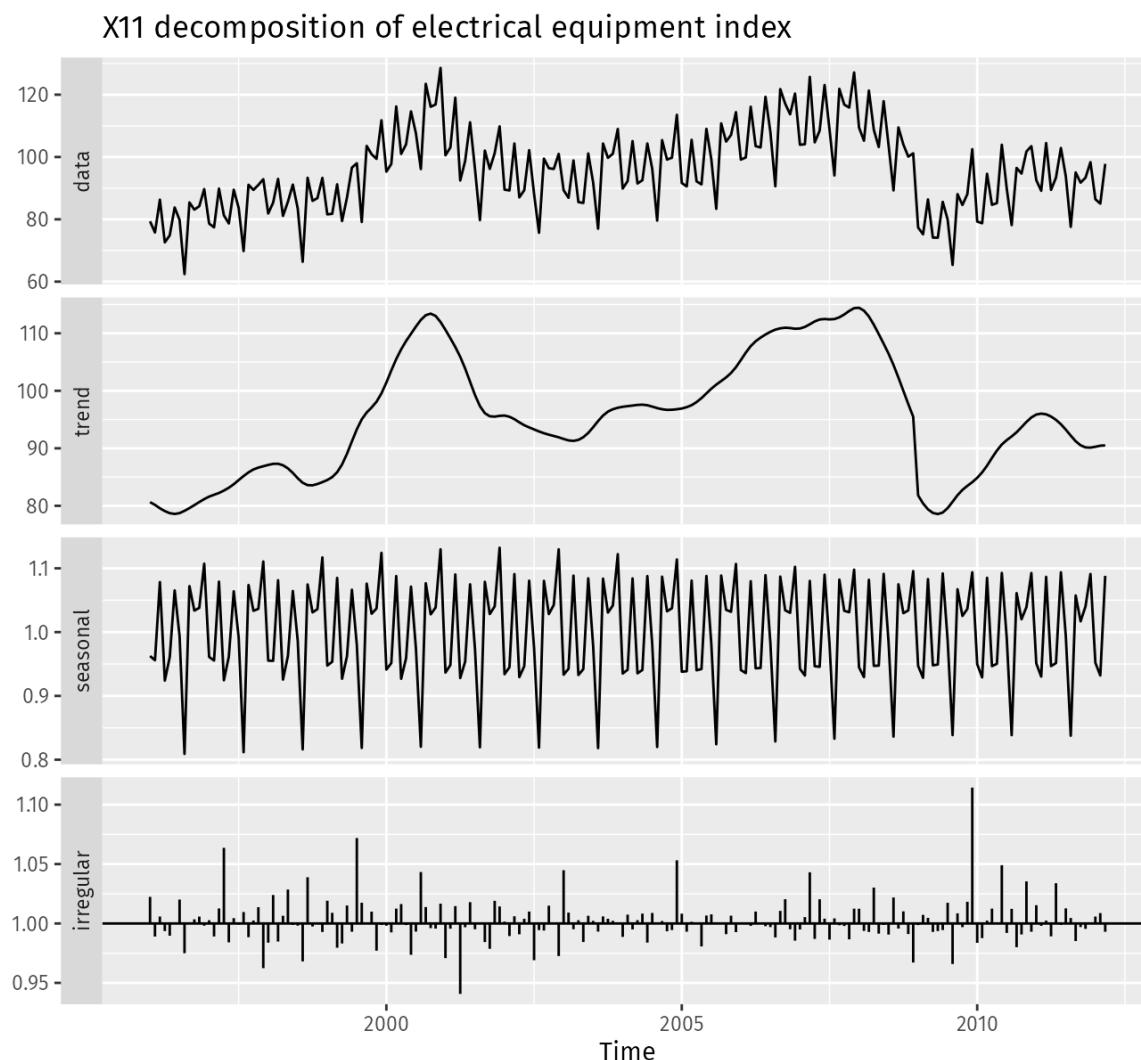


FIGURE 3.16 – Une décomposition X11 de l'indice des nouvelles commandes pour l'équipement électrique.

Comparée avec la décomposition STL présentée dans la Figure 3.13 et la décomposition classique présentée dans la figure 3.14. La tendance-cycle X11 a mieux capturé la chute soudaine des données au début de 2009 que les deux autres méthodes, et l'observation inhabituelle à la fin de 2009 est maintenant plus clairement visible dans le composant de reste.

Décomposition STL

STL est une méthode polyvalente et robuste pour la décomposition des séries chronologiques. STL est un acronyme pour "Seasonal and Trend decomposition using Loess", tandis que Loess est une méthode d'estimation des relations non linéaires. La méthode STL a été développée par R. B. Cleveland, Cleveland, McRae, & Terpenning (1990)[2].

STL présente plusieurs avantages par rapport aux méthodes de décomposition classique, SEATS et X11 :

- Contrairement à SEATS et X11, STL peut gérer n'importe quel type de saisonnalité, pas seulement des données mensuelles et trimestrielles.
- La composante saisonnière peut changer avec le temps, et le taux de changement peut être contrôlé par l'utilisateur.
- Le lissage du cycle de tendance peut également être contrôlé par l'utilisateur.
- Il peut être robuste aux valeurs aberrantes (c'est-à-dire que l'utilisateur peut spécifier une décomposition robuste), de sorte que les observations occasionnelles inhabituelles n'affecteront pas les estimations des composantes du cycle de tendance et saisonnières. Cependant, elles affecteront la composante reste.

En revanche, STL présente certains inconvénients. En particulier, il ne gère pas automatiquement les variations des jours ouvrables ou du calendrier, et il ne fournit que des facilités pour les décompositions additives.

Il est possible d'obtenir une décomposition multiplicative en prenant d'abord les logarithmes des données, puis en retransformant les composantes. Les décompositions entre additive et multiplicative peuvent être obtenues en utilisant une transformation de Box-Cox des données avec $0 < \lambda < 1$. Une valeur de $\lambda = 0$ correspond à la décomposition multiplicative tandis que $\lambda = 1$ est équivalente à une décomposition additive.

Conclusion

Au terme de ce chapitre sur les séries temporelles, nous avons exploré en profondeur les fondements et les techniques essentielles pour comprendre et analyser ces données temporelles complexes. Nous avons abordé divers aspects, notamment la définition des séries temporelles, les différents types de séries (univariées et multivariées), les caractéristiques clés telles que la tendance, la saisonnalité et la stationnarité, ainsi que les méthodes d'analyse telles que les tests de stationnarité et les fonctions d'autocorrélation.

Nous avons également étudié les techniques de décomposition des séries temporelles, permettant de séparer les différentes composantes des données, notamment la tendance, la saisonnalité et les résidus. Ces composantes sont essentielles pour comprendre le comportement global des séries temporelles et pour effectuer des prévisions précises.

Enfin, nous avons discuté des avantages et des inconvénients des différentes méthodes de décomposition, ainsi que de leur application dans divers contextes. Nous avons jeté les bases nécessaires pour aborder des techniques plus avancées telles que les réseaux de neurones récurrents (RNN) et les réseaux de neurones à mémoire à long terme (LSTM), qui offrent des solutions puissantes pour la modélisation et la prévision des séries temporelles.

Dans le prochain chapitre, nous explorerons en détail les RNN et LSTM, en mettant en évidence leur fonctionnement, leurs applications et leur potentiel pour l'analyse et la modélisation avancée des séries temporelles.

Chapitre 4

Réseaux de neurones récurrents (RNN)

Ce chapitre explore les méthodes avancées pour l’analyse et la modélisation des séries temporelles en utilisant des techniques de réseaux de neurones récurrents (RNN), de mémoire à court terme (LSTM) et de mise à jour de portes récurrentes (GRU). Nous plongerons dans les concepts fondamentaux des réseaux de neurones, en mettant en évidence leur architecture et leur fonctionnement. Ensuite, nous explorerons en détail les RNN, suivis des LSTM, une extension puissante conçue pour surmonter les limitations des RNN traditionnels. Nous examinerons également les GRU, qui partagent certaines des capacités des LSTM mais avec une architecture plus simple. Enfin, nous découvrirons diverses applications des RNN, LSTM et GRU dans l’analyse et la prédiction des séries temporelles, offrant un aperçu complet de leur potentiel pour traiter les données temporelles complexes.

Introduction

Dans ce chapitre, nous plongerons dans le monde des réseaux de neurones récurrents (RNN). Les séries temporelles présentent souvent des structures complexes et des dépendances temporelles qui ne peuvent pas être entièrement capturées par les méthodes traditionnelles. Les RNN offrent une approche puissante pour modéliser et prédire ces séries, en exploitant leur capacité à mémoriser les dépendances temporelles sur de longues séquences de données.

Dans ce chapitre, nous commencerons par une introduction aux concepts de base des réseaux de neurones, en mettant en évidence leur architecture et leur fonctionnement général. Ensuite, nous plongerons dans les détails des RNN, en explorant comment ils peuvent être utilisés pour modéliser des séquences de données et capturer des dépendances temporelles.

Nous aborderons ensuite les LSTM, une extension des RNN qui surmonte certains des problèmes de rétention de mémoire à long terme rencontrés par les RNN traditionnels. Les LSTM sont particulièrement efficaces pour traiter les séries temporelles en raison de leur capacité à apprendre et à mémoriser des motifs temporels complexes sur de longues périodes.

Nous introduirons également les réseaux de neurones à portes (GRU), une autre variante des RNN qui partagent certaines des capacités des LSTM mais avec une architecture plus simple.

Enfin, nous explorerons diverses applications des RNN, LSTM et GRU dans l'analyse et la prédition des séries temporelles. Nous découvrirons comment ces modèles peuvent être utilisés pour effectuer des prévisions précises, détecter des anomalies, analyser des tendances et bien plus encore.

En résumé, ce chapitre nous plongera dans l'univers passionnant des réseaux de neurones récurrents, des LSTM et des GRU, offrant un aperçu approfondi de leur fonctionnement et de leur potentiel pour l'analyse avancée des séries temporelles.

4.1 Réseau de neurones

4.1.1 Définition

En apprentissage automatique, un réseau de neurones (également appelé réseau neuronal artificiel ou réseau de neurones, abrégé en ANN ou NN) est un modèle inspiré par la structure et la fonction des réseaux neuronaux biologiques dans les cerveaux des animaux.

Un ANN est composé d'unités ou de nœuds connectés appelés neurones artificiels, qui modélisent de manière approximative les neurones dans un cerveau. Ceux-ci sont reliés par des arêtes, qui modélisent les synapses dans un cerveau. Chaque neurone artificiel reçoit des signaux des neurones connectés, les traite, puis envoie un signal à d'autres neurones connectés. Le "signal" est un nombre réel, et la sortie de chaque neurone est calculée par une fonction non linéaire de la somme de ses entrées, appelée fonction d'activation. La force du signal à chaque connexion est déterminée par un poids, qui s'ajuste pendant le processus d'apprentissage.

Typiquement, les neurones sont regroupés en couches. Différentes couches peuvent effectuer différentes transformations sur leurs entrées. Les signaux voyagent de la première couche (la couche d'entrée) à la dernière couche (la couche de sortie), en passant éventuellement par plusieurs couches intermédiaires (couches cachées). Un réseau est généralement appelé réseau neuronal profond s'il comporte au moins 2 couches cachées.

Les réseaux neuronaux artificiels sont utilisés pour diverses tâches, y compris la modélisation prédictive, le contrôle adaptatif et la résolution de problèmes en intelligence artificielle. Ils peuvent apprendre de l'expérience et peuvent tirer des conclusions à partir d'un ensemble complexe et apparemment non lié d'informations.

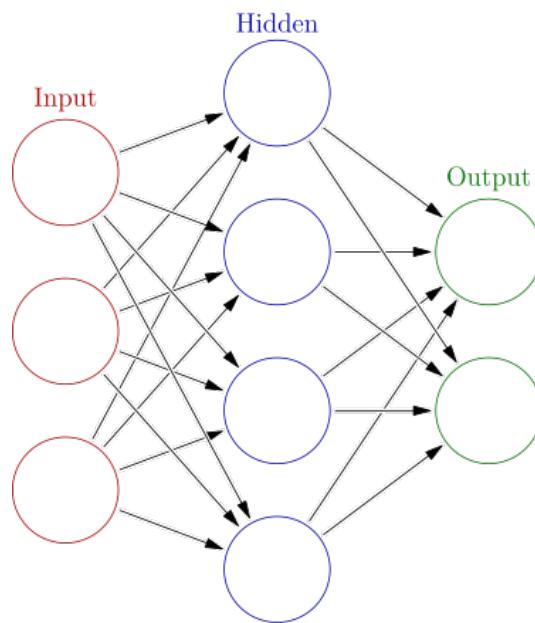


FIGURE 4.1 – Réseau de neurones artificiels simplifié

Dans la figure, nous voyons un réseau de neurones artificiels, qui est un groupe interconnecté de nœuds, inspiré par une simplification des neurones dans un cerveau. Ici, chaque nœud circulaire représente un neurone artificiel et une flèche représente une connexion de la sortie d'un neurone artificiel à l'entrée d'un autre.

4.1.2 Types des réseaux de neurones

Il existe de nombreux types de réseaux neuronaux disponibles ou en cours de développement. Ils peuvent être classés en fonction de leur : la structure, le flux de données, les neurones utilisés et leur densité, les couches et leurs filtres d'activation en profondeur, etc.

Parmi ces réseaux de neurones, on cite :

Perceptron

Le perceptron peut être vu comme le type de réseau de neurones le plus simple. C'est un classifieur linéaire. Ce type de réseau neuronal ne contient aucun cycle (il s'agit d'un réseau de neurones à propagation avant). Dans sa version simplifiée, le perceptron est mono-couche et n'a qu'une seule sortie (booléenne) à laquelle toutes les entrées (booléennes) sont connectées. Plus généralement, les entrées peuvent être des nombres réels.

Un perceptron à n entrées $x = (x_1, \dots, x_n)$ et à une seule sortie o est défini par la donnée de n poids (aussi appelés coefficients synaptiques) (w_1, \dots, w_n) et un biais (ou seuil) θ par :

$$o = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i \geq \theta \\ 0 & \text{sinon} \end{cases}$$

La sortie o résulte alors de l'application de la fonction de Heaviside au potentiel post-synaptique

$$z = \sum_{i=1}^n w_i x_i - \theta$$

où la Fonction de Heaviside est :

$$\forall z \in \mathbb{R}, \quad H(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z \geq 0. \end{cases}$$

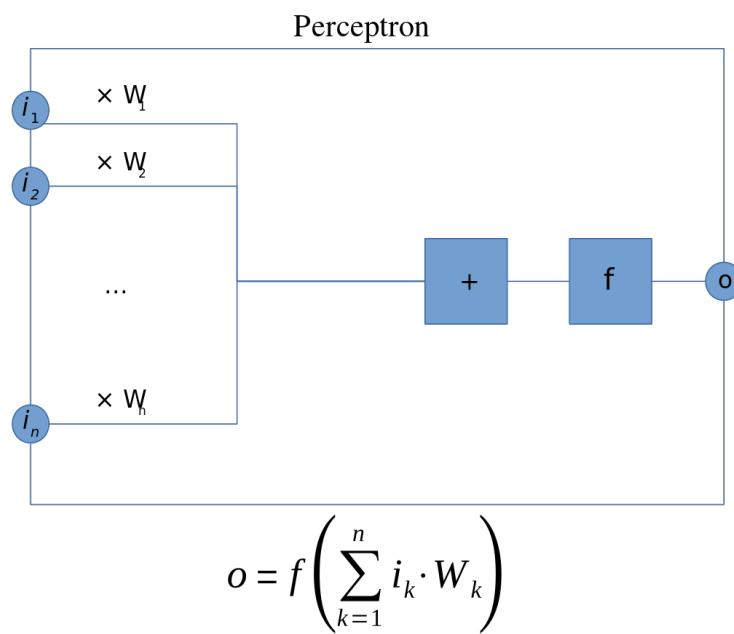


FIGURE 4.2 – Perceptron

Dans la figure, les entrées i_1 à i_n sont multipliées avec les poids W_1 à W_n , puis sommées, avant qu'une fonction d'activation soit appliquée.

Réseau de neurones à action directe

Le réseau de neurones à action directe est le premier type de réseau neuronal artificiel conçu. C'est aussi le plus simple. Dans ce réseau, l'information ne se déplace que dans une seule direction, vers l'avant, à partir des noeuds d'entrée, en passant par les couches cachées (le cas échéant) et vers les noeuds de sortie. Il n'y a pas de cycles ou de boucles dans le réseau.

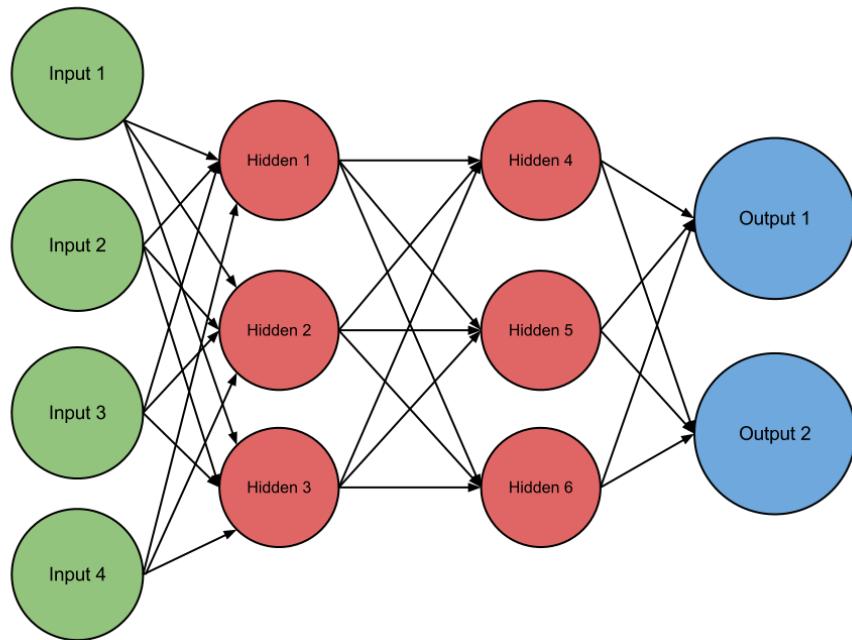


FIGURE 4.3 – Réseau de neurones à action directe

Perceptron multicouche

Le perceptron multicouche (multilayer perceptron MLP en anglais) est un type de réseau neuronal artificiel organisé en plusieurs couches. Un perceptron multicouche possède au moins trois couches : une couche d'entrée, au moins une couche cachée, et une couche de sortie. Chaque couche est constituée d'un nombre (potentiellement différent) de neurones. L'information circule de la couche d'entrée vers la couche de sortie uniquement : il s'agit donc d'un réseau à propagation directe (feedforward). Les neurones de la dernière couche sont les sorties du système global.

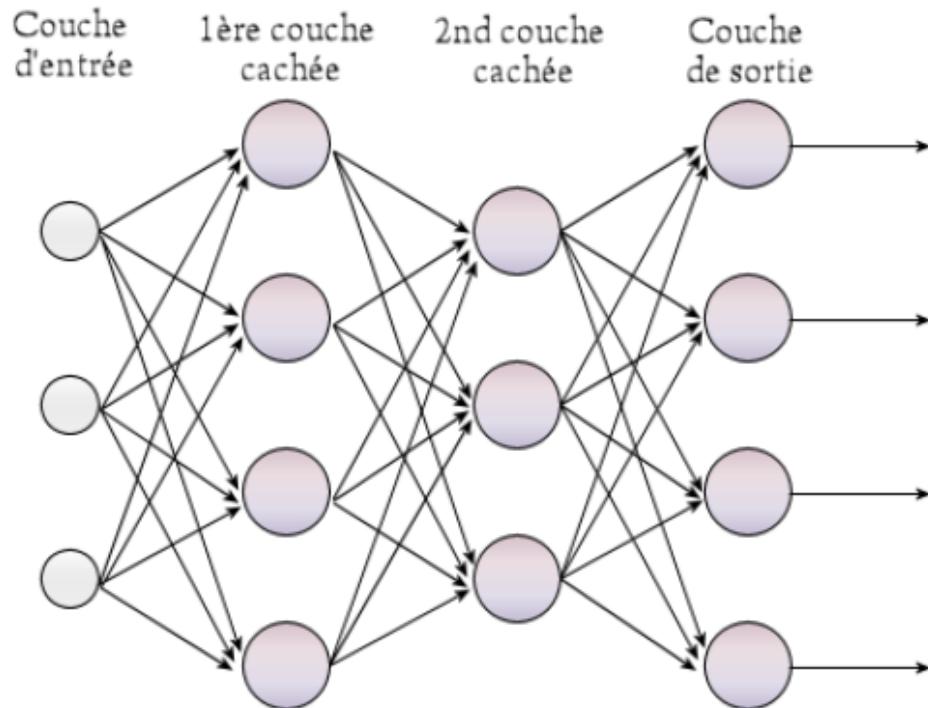


FIGURE 4.4 – Un perceptron multicouche

Réseau de neurones convolutifs

Un réseau de neurones convolutifs ou réseau de neurones à convolution (en anglais CNN ou ConvNet pour convolutional neural networks) est un type de réseau de neurones artificiels acycliques (feed-forward), dans lequel le motif de connexion entre les neurones est inspiré par le cortex visuel des animaux. Ils consistent en un empilage multicouche de perceptrons, dont le but est de prétraiter de petites quantités d'informations. Les réseaux neuronaux convolutifs ont de larges applications dans la reconnaissance d'image et vidéo, les systèmes de recommandation et le traitement du langage naturel.

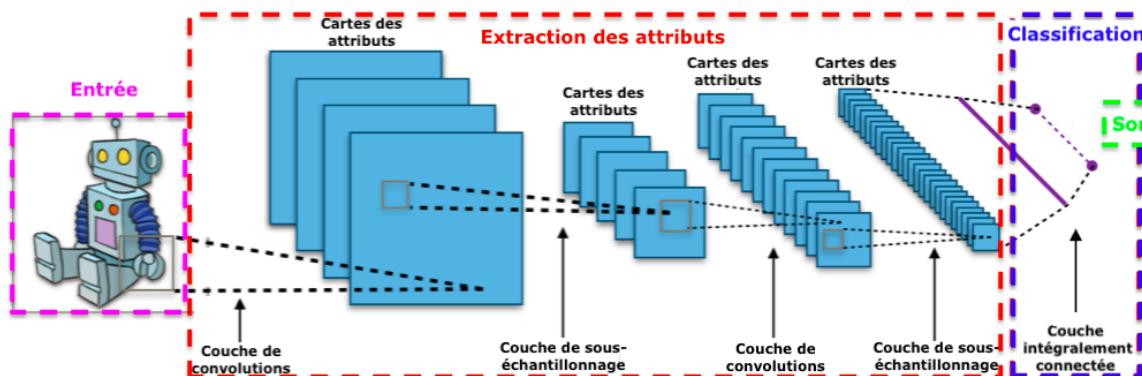


FIGURE 4.5 – Architecture standard d'un réseau à convolutions

La forme la plus commune d'une architecture de réseau de neurones convolutifs empile quelques couches Conv-ReLU, les suit avec des couches Pool, et répète ce schéma jusqu'à ce que l'entrée soit réduite dans un espace d'une taille suffisamment petite. À un moment, il est fréquent de placer des couches entièrement connectées (FC). La dernière couche entièrement connectée est reliée vers la sortie.

L'empilage des couches CONV avec de petits filtres de pooling permet un traitement plus puissant, avec moins de paramètres. Cependant, avec l'inconvénient de demander plus de puissance de calcul (pour contenir tous les résultats intermédiaires de la couche CONV).

Réseau de neurones modulaire

Un réseau neuronal modulaire est un réseau neuronal artificiel caractérisé par une série de réseaux neuronaux indépendants modérés par un intermédiaire. Chaque réseau neuronal indépendant sert de module et fonctionne sur des entrées distinctes pour accomplir une sous-tâche de la tâche que le réseau espère réaliser. L'intermédiaire prend les sorties de chaque module et les traite pour produire la sortie du réseau dans son ensemble. L'intermédiaire n'accepte que les sorties des modules ; il ne répond pas aux modules et ne leur envoie pas d'autres signaux. De même, les modules n'interagissent pas entre eux.

Réseau de neurones récurrents

Un réseau de neurones récurrents (RNN pour recurrent neural network en anglais) est un réseau de neurones artificiels présentant des connexions récurrentes. Un réseau de neurones récurrents est constitué d'unités (neurones) interconnectées interagissant non-linéairement et pour lequel il existe au moins un cycle dans la structure. Les unités sont reliées par des arcs (synapses) qui possèdent un poids. La sortie d'un neurone est une combinaison non linéaire de ses entrées.

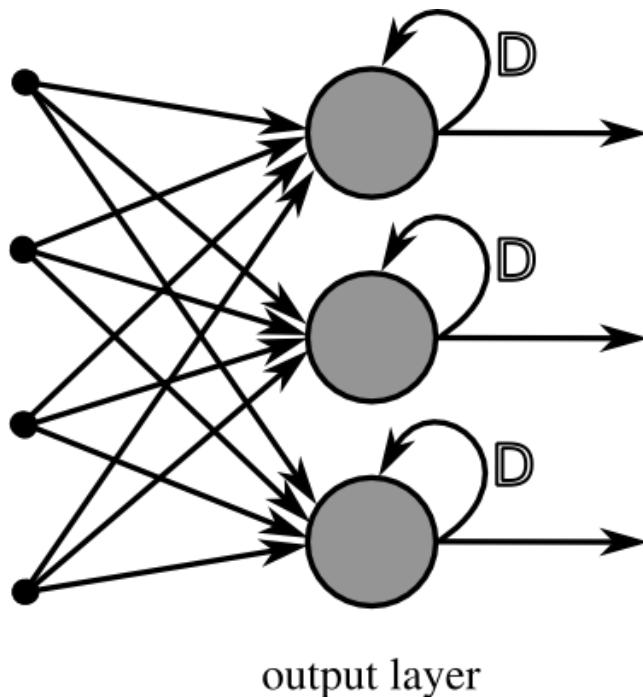


FIGURE 4.6 – Réseau de neurones récurrents

Maintenant que nous avons une vue d'ensemble des différents types de réseaux de neurones, nous allons nous concentrer sur les réseaux de neurones récurrents (RNN). Les RNN sont particulièrement adaptés pour analyser et modéliser les séries temporelles en raison de leur capacité à conserver des informations à travers les séquences de données.

4.2 Réseau de neurones récurrents

Les réseaux de neurones récurrents sont particulièrement adaptés aux données d'entrée de taille variable, ce qui les rend idéaux pour l'analyse de séries temporelles. Ils sont largement utilisés dans des domaines tels que la reconnaissance automatique de la parole et de l'écriture manuscrite, la reconnaissance de formes en général, ainsi que la traduction automatique.

Lorsqu'ils sont "dépliés", ces réseaux ressemblent à des réseaux de neurones classiques avec des contraintes d'égalité sur les poids (figure 4.7 à droite). Leurs techniques d'entraînement sont similaires à celles des réseaux classiques, notamment la Rétropropagation du gradient. Cependant, les réseaux de neurones récurrents rencontrent des difficultés avec la disparition du gradient, ce qui complique l'apprentissage de la mémorisation d'événements passés. Pour pallier ce problème, des architectures spécifiques ont été développées, parmi lesquelles les réseaux à mémoire à long court terme (LSTM) sont particulièrement notables.

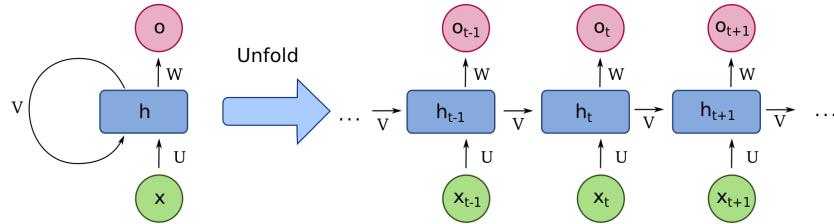


FIGURE 4.7 – Schéma d'un réseau de neurones récurrents à une unité reliant l'entrée et la sortie du réseau

4.2.1 Fonctionnement

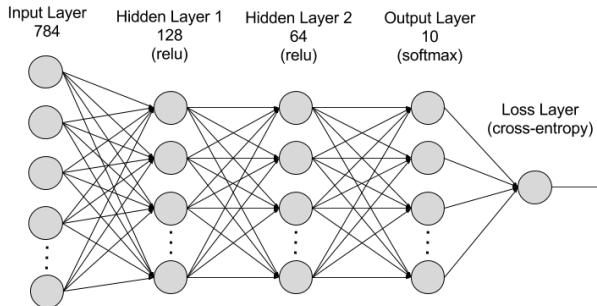


FIGURE 4.8 – Schéma d'un RNN

Les RNN sont constitués de neurones : des nœuds de traitement de données qui travaillent ensemble pour effectuer des tâches complexes. Les neurones sont organisés en couches d'entrée, de sortie et cachées. La couche d'entrée reçoit les informations à traiter et la couche de sortie fournit le résultat. Le traitement, l'analyse et la prévision des données ont lieu dans la couche cachée. [24]

Couche cachée

Les RNN fonctionnent en transmettant les données séquentielles qu'ils reçoivent aux couches cachées étape par étape. Cependant, ils disposent également d'un flux de travail en boucle automatique ou récurrent : la couche cachée peut mémoriser et utiliser les entrées précédentes pour les prévisions futures dans un composant de mémoire à court terme. Elle utilise l'entrée actuelle et la mémoire stockée pour prédire la séquence suivante.

Par exemple, considérez la séquence : La pomme est rouge. Vous voulez que le RNN prédise le rouge lorsqu'il reçoit la séquence d'entrée pomme est. Lorsque la couche cachée traite le mot pomme, elle en stocke une copie dans sa mémoire. Ensuite, lorsqu'elle voit le mot est, il rappelle pomme de sa mémoire et comprend la séquence complète : pomme est pour le contexte. Il peut ensuite prédire rouge pour une meilleure précision. Cela rend les RNN utiles pour la reconnaissance vocale, la traduction automatique et d'autres tâches de modélisation linguistique.

4.2.2 Types de réseaux neuronaux récurrents

L'architecture RNN peut varier selon le problème que nous essayons de résoudre. De ceux avec une seule entrée et sortie à ceux avec beaucoup. Voici quelques exemples d'architectures RNN qui peuvent nous aider à mieux comprendre cela.

Un à Un

Le type le plus simple de RNN est Un à Un (One-to-One), il a une seule entrée et une seule sortie. Ses tailles d'entrée et de sortie sont fixes et agit comme un réseau neuronal traditionnel. L'application de Un à Un peut être utilisée dans la classification d'image.



FIGURE 4.9 – Réseau de neurones récurrents Un à Un

Un à Plusieurs

Un à Plusieurs (One-To-Many) est un type de RNN qui donne plusieurs sorties lorsqu'il est donné une seule entrée. Il prend une taille d'entrée fixe et donne une séquence de sorties de données. Ses modèles peuvent d'appliquer à la génération de musique et sous-titrage d'image.

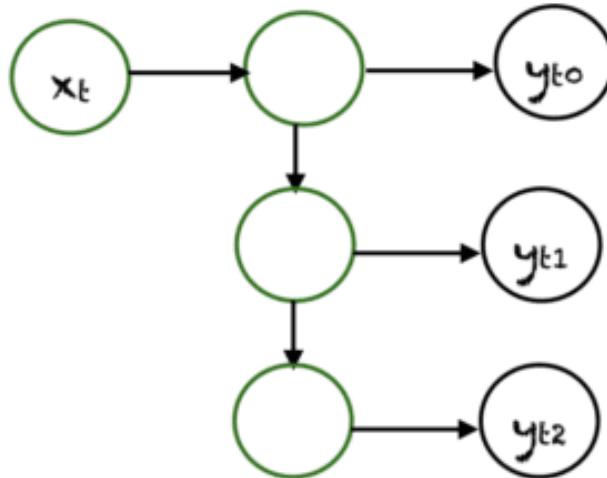


FIGURE 4.10 – Réseau de neurones récurrents Un à Plusieurs

Plusieurs à Un

Plusieurs à Un (Many-To-One) est utilisé lorsqu'une seule sortie est requise à partir de plusieurs unités d'entrée ou d'une séquence d'entre elles. Il faut une séquence d'entrées pour afficher une sortie fixe. L'analyse des sentiments est un exemple courant de ce type de réseau neuronal récurrent.

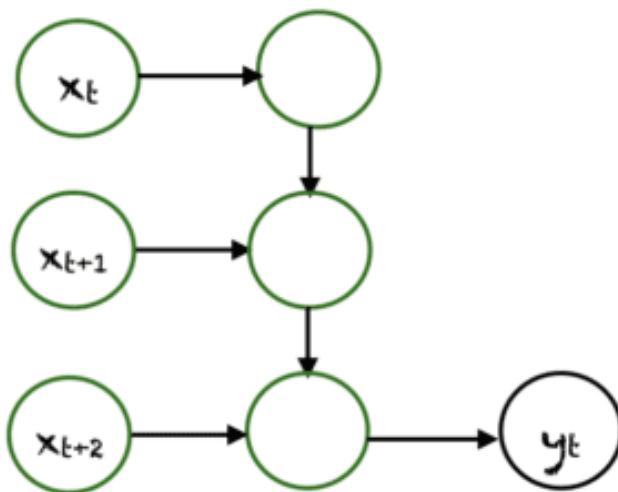


FIGURE 4.11 – Réseau de neurones récurrents Plusieurs à Un

Plusieurs à Plusieurs

Plusieurs à Plusieurs (Many-To-Many) est utilisé pour générer une séquence de données de sortie à partir d'une séquence d'unités d'entrée. Ce type de RNN est divisé en deux sous-catégories :

1. **Taille égale des unités** : Dans ce cas, le nombre d'unités d'entrée et de sortie est le même. Une application commune peut être trouvée dans *Name-Entity Recognition*.

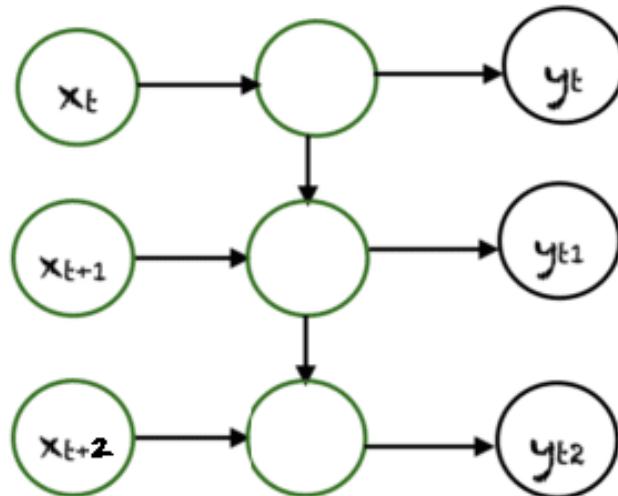


FIGURE 4.12 – Réseau de neurones récurrents Plusieurs à Un à taille égale des unités

2. **Taille inégale des unités** : Dans ce cas, les entrées et les sorties ont un nombre d'unités différent. Son application peut être trouvée dans la *traduction automatique*.

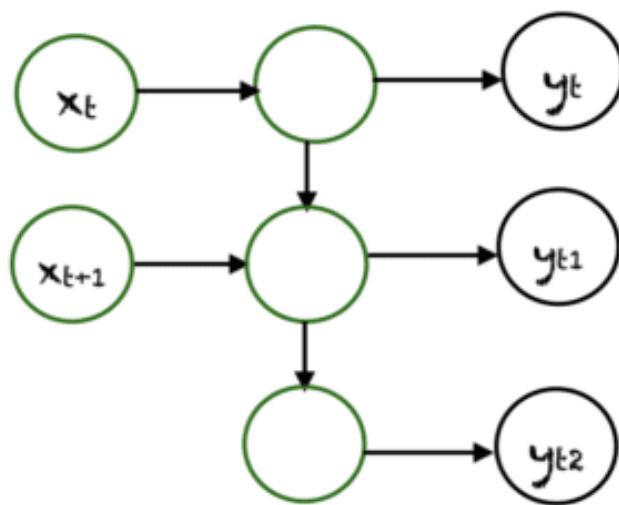


FIGURE 4.13 – Réseau de neurones récurrents Plusieurs à Un à taille inégale des unités

4.2.3 Comment les réseaux neuronaux récurrents se comparent-ils aux autres réseaux de deep learning ?

Les RNN sont l'une des nombreuses architectures de réseaux neuronaux.

Réseau neuronal récurrent vs réseau de neurones à propagation avant

Comme les RNN, les réseaux neuronaux à propagation avant sont des réseaux neuronaux artificiels qui transmettent des informations d'un bout à l'autre de l'architecture. Un réseau de neurones à propagation avant peut effectuer des tâches simples de classification, de régression ou de reconnaissance, mais il ne se souvient pas de l'entrée précédente qu'il a traitée. Par exemple, il oublie *Pomme* au moment où son neurone traite le mot *est*. Le RNN surmonte cette limitation de mémoire en incluant un état de mémoire caché dans le neurone.

Réseau neuronal récurrent vs Réseaux neuronaux convolutifs

Les réseaux neuronaux convolutifs sont des réseaux neuronaux artificiels conçus pour traiter des données temporelles. Vous pouvez utiliser des réseaux neuronaux convolutifs pour extraire des informations spatiales de vidéos et d'images en les faisant passer par une série de couches convolutives et mutualisées du réseau neuronal. Les RNN sont conçus pour capturer les dépendances à long terme dans les données séquentielles.

4.2.4 Entraînement

La descente de gradient est un algorithme d'optimisation itératif de premier ordre pour trouver le minimum d'une fonction. Dans les réseaux de neurones, elle peut être utilisée pour minimiser le terme d'erreur en modifiant chaque poids proportionnellement à la dérivée de l'erreur par rapport à ce poids, à condition que les fonctions d'activation non linéaires soient différentiables. La méthode standard est appelée "*Rétropropagation à travers le temps*" ou BPTT, et est une généralisation de la rétropropagation pour les réseaux feed-forward. Comme cette méthode, elle est une instance de la différentiation automatique dans le mode d'accumulation inverse du principe du minimum de Pontryagin. Une variante en ligne plus coûteuse en termes de calcul est appelée "*apprentissage récurrent en temps réel*" ou RTRL, qui est une instance de la différentiation automatique dans le mode d'accumulation directe avec des vecteurs tangents empilés. Contrairement à la BPTT, cet algorithme est local dans le temps mais pas local dans l'espace.

Dans ce contexte, local dans l'espace signifie que le vecteur de poids d'une unité peut être mis à jour en utilisant uniquement les informations stockées dans les unités connectées et l'unité elle-même, de sorte que la complexité de la mise à jour d'une seule unité est linéaire par rapport à la dimensionnalité du vecteur de poids. Local dans le temps signifie que les mises à jour ont lieu continuellement (en ligne) et ne dépendent que de l'étape de temps la plus récente plutôt que de plusieurs étapes de temps dans un horizon donné comme dans la BPTT.

Pour calculer de manière récursive les dérivées partielles, le RTRL a une complexité temporelle de $O(\text{nombre de neurones cachés} * \text{nombre de poids})$ par étape de temps pour calculer les matrices jacobiniennes, tandis que la BPTT ne prend que

$O(\text{nombre de poids})$ par étape de temps, au prix de stocker toutes les activations en avant dans l'horizon de temps donné. Une version en ligne hybride entre la BPTT et le RTRL avec une complexité intermédiaire existe, ainsi que des variantes pour le temps continu.

Un problème majeur avec la descente de gradient pour les architectures RNN standard est que les gradients d'erreur disparaissent très rapidement de manière exponentielle avec la taille du décalage temporel entre des événements importants. Les LSTM, combinés avec une méthode d'apprentissage hybride BPTT/RTRL, tentent de surmonter ces problèmes. Ce problème est également résolu dans le réseau neuronal récurrent indépendant (IndRNN)[17] en réduisant le contexte d'un neurone à son propre état passé et les informations inter-neurones peuvent ensuite être explorées dans les couches suivantes. Les mémoires de différentes gammes, y compris la mémoire à long terme, peuvent être apprises sans le problème de disparition et d'explosion du gradient.

L'algorithme en ligne appelé rétropropagation recursive causale (CRBP), met en œuvre et combine les paradigmes BPTT et RTRL pour les réseaux localement récurrents. Il fonctionne avec les réseaux localement récurrents les plus généraux. L'algorithme CRBP peut minimiser le terme d'erreur global. Ce fait améliore la stabilité de l'algorithme, offrant une vue unifiée des techniques de calcul des gradients pour les réseaux récurrents avec rétroaction locale.

Une approche du calcul de l'information sur le gradient dans les RNN avec des architectures arbitraires est basée sur la dérivation diagrammatique des graphes de flux de signal. Elle utilise l'algorithme BPTT par lot, basé sur le théorème de Lee pour les calculs de sensibilité du réseau. Elle a été proposée par Wan et Beaufays, tandis que sa version rapide en ligne a été proposée par Campolucci, Uncini et Piazza[7].

4.2.5 Limitations

Les réseaux de neurones récurrents (RNN) souffrent d'un problème appelé *disparition du gradient*, qui limite leur capacité à apprendre à partir d'événements passés éloignés. Ce problème survient lors de l'entraînement du réseau, qui utilise un algorithme appelé rétropropagation du gradient à travers le temps (BPTT). En minimisant une fonction d'erreur dépendant de la sortie, le réseau ajuste ses poids en fonction des gradients calculés. Cependant, la valeur du gradient peut diminuer exponentiellement à chaque étape temporelle, en particulier pour des événements lointains, rendant ces gradients presque nuls et empêchant le réseau de mémoriser efficacement les informations anciennes. En conséquence, les RNN ont une mémoire à court terme, où les entrées passées ont de moins en moins d'influence sur les sorties actuelles. Cette limitation est commune à d'autres algorithmes de réseaux neuronaux et est due à la nature même de la rétropropagation. Des architectures spécifiques comme les réseaux à mémoire à long court terme (LSTM) ont été développées pour atténuer ce problème en permettant aux réseaux de mieux gérer et mémoriser les dépendances à long terme.

Le problème du gradient de fuite peut être résolu par différentes variantes des RNNs. Deux d'entre eux qui vont être discutés plus dans les chapitres suivants sont :

- **LSTM (Long Short-Term Memory)** : Ils sont capables de se souvenir des entrées sur une longue période de temps. C'est parce qu'ils contiennent des informations dans une mémoire, un peu comme la mémoire d'un ordinateur. Le LSTM peut lire, écrire et supprimer des informations de sa mémoire. Dans cette mémoire, on décide à chaque fois de stocker ou de supprimer des informations en fonction de l'importance qu'elle accorde à l'information. L'attribution de l'importance se fait par des poids, qui sont également appris par l'algorithme.
- **GRU (Gated Recurrent Units)** : Ce modèle est similaire aux LSTM car il permet aussi de résoudre le problème de mémoire à court terme des modèles RNN. Pour résoudre ce problème, GRU intègre les deux mécanismes de fonctionnement de porte appelés Update gate et Reset gate : les portails de réinitialisation et de mise à jour sont concrètement des vecteurs qui contrôlent les informations à conserver et à négliger. [25]

Conclusion

En résumé, ce chapitre a abordé les bases des réseaux de neurones récurrents (RNN) en exposant leur architecture, leurs types et leur fonctionnement. Les RNN sont des modèles puissants pour l'analyse et la modélisation des séries temporelles en raison de leur capacité à capturer les dépendances séquentielles. Cependant, ils présentent des limitations comme la disparition du gradient, ce qui complique l'apprentissage de longues dépendances temporelles.

Nous avons également discuté des variantes de RNN, comme les réseaux à mémoire à court et long terme (LSTM) et les unités récurrentes à portes (GRU), conçues pour surmonter ces limitations. Ces architectures avancées permettent de mieux mémoriser et d'exploiter les informations sur de longues séquences de données.

En conclusion, les RNN et leurs variantes constituent des outils essentiels pour traiter des données séquentielles complexes, avec des applications allant de la reconnaissance vocale à la traduction automatique. Les techniques d'entraînement et les améliorations architecturales jouent un rôle crucial pour tirer pleinement parti du potentiel des RNN.

Maintenant que nous avons une compréhension approfondie des réseaux de neurones récurrents, nous allons explorer en détail les réseaux de neurones à mémoire à court et long terme (LSTM). Ces modèles sophistiqués surmontent les limitations des RNN traditionnels et sont particulièrement efficaces pour traiter les séries temporelles en raison de leur capacité à apprendre et à mémoriser des motifs temporels complexes sur de longues périodes. Passons donc au chapitre dédié aux LSTM pour découvrir leur architecture, leur fonctionnement et leurs applications.

Chapitre 5

Réseaux de neurones à mémoire à long terme (LSTM)

Ce chapitre explore les réseaux de neurones à mémoire à long terme, plus connus sous le sigle LSTM (Long Short-Term Memory). Ces réseaux de neurones récurrents (RNN) ont été introduits pour résoudre les limitations des RNN classiques, notamment les problèmes de gradient évanescence et explosif. Nous aborderons les concepts de base des LSTM, leur architecture spécifique, leurs applications, ainsi que les techniques d'entraînement et d'optimisation utilisées pour améliorer leur performance.

Introduction

Les réseaux de neurones à mémoire à long terme, ou LSTM (Long Short-Term Memory), représentent une avancée significative dans le domaine de l'apprentissage automatique, en particulier pour le traitement des données séquentielles et temporelles. Introduits par Sepp Hochreiter et Jürgen Schmidhuber en 1997, les LSTM ont été conçus pour surmonter les limitations des réseaux de neurones récurrents (RNN) classiques, qui peinent souvent à gérer les dépendances à long terme en raison des problèmes de gradient évanescents et explosifs.

5.1 Disparition du gradient

Les réseaux de neurones récurrents classiques sont exposés au problème de disparition de gradient qui les empêche de modifier leurs poids en fonction d'événements trop anciens. Lors de l'entraînement, le réseau essaie de minimiser une **fonction d'erreur** dépendant en particulier de la sortie $E(o_t)$. Pour l'exemple donné ci-dessus, suivant la règle de dérivation en chaîne, la contribution au gradient $\frac{\partial E(o_t)}{\partial U_{mn}}$ de l'entrée x_{t-k} est

$$\frac{\partial E(o_t)}{\partial U_{mn}} = \dots + \frac{\partial E(o_t)}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} \frac{\partial h_{t-k}}{\partial U_{mn}} + \dots$$

$$\frac{\partial E(o_t)}{\partial U_{mn}} = \dots + \frac{\partial E(o_t)}{\partial o_t} \frac{\partial o_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{t-k+1}}{\partial h_{t-k}} \sigma'_h(Ux_{t-k} + Vh_{t-k-1})E_{mn}x_{t-k} + \dots$$

où E_{mn} représente la matrice telle que le seul terme non nul est le terme (m, n) égal à 1. Pour des valeurs standard de la fonction d'activation et des poids du réseau, le produit $\frac{\partial h_t}{\partial h_{t-1}} \dots \frac{\partial h_{t-k+1}}{\partial h_{t-k}}$ décroît exponentiellement en fonction de k . Par conséquent, la correction de l'erreur, si cela doit faire intervenir un événement lointain, diminuera exponentiellement avec l'intervalle de temps entre cet événement et le présent. Le réseau sera incapable de prendre en compte les entrées passées.

Autrement, en utilisant la rétropropagation, un réseau récurrent peut retracer les dépendances arbitraires qu'il trouve dans les données d'entrée. Cependant les gradients à long terme qui sont rétropropagés peuvent tendre vers zéro (on dit qu'ils disparaissent) ou peuvent tendre vers l'infini (on dit qu'ils explosent). Dans les deux cas on perd l'information qu'on voulait garder en mémoire.

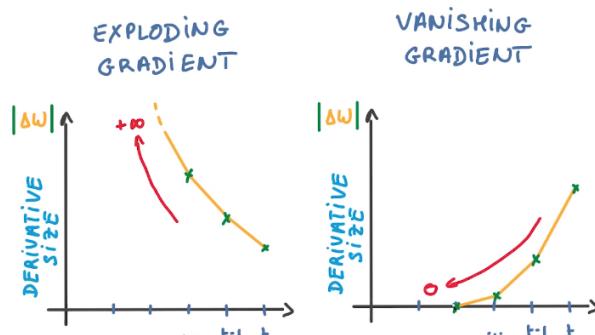


FIGURE 5.1 – Schémas de la disparition et de l'explosion du gradient

5.2 Définition

La mémoire à long court terme (LSTM) [5] est un type de réseau de neurones récurrents (RNN) conçu pour traiter le problème du gradient évanescents [3] présent dans les RNN traditionnels. Son insensibilité relative à la longueur des écarts constitue son avantage sur les autres RNN, les Modèles de Markov Caché et autres méthodes d'apprentissage de séquences. Il vise à fournir une mémoire à court terme pour les RNN qui peut durer des milliers de pas de temps, d'où le terme "mémoire à long court terme" [5]. Il est applicable à la classification, au traitement et à la prédiction des données basées sur les séries temporelles, comme dans l'écriture manuscrite, la reconnaissance vocale, la traduction automatique, la détection d'activité vocale, le contrôle de robots, les jeux vidéo et les soins de santé.

5.3 Structure

Pour résoudre le problème de l'atténuation et de l'explosion des gradients dans un réseau de neurones récurrents profonds, de nombreuses variations ont été développées. L'une des plus célèbres est le réseau Long Short Term Memory (LSTM). Conceptuellement, une unité récurrente LSTM essaie de "se souvenir" de toutes les connaissances passées que le réseau a vues jusqu'à présent et de "oublier" les données non pertinentes. Cela est réalisé en introduisant différentes couches de fonction d'activation appelées "portes" pour des objectifs différents. Chaque unité récurrente LSTM maintient également un vecteur appelé l'état de cellule interne, qui décrit conceptuellement les informations choisies pour être retenues par l'unité récurrente LSTM précédente.

Les réseaux LSTM sont la variation la plus couramment utilisée des réseaux de neurones récurrents (RNN). Le composant crucial du LSTM est la cellule mémoire et les portes (y compris la porte d'oubli mais aussi la porte d'entrée), le contenu interne de la cellule mémoire étant modulé par les portes d'entrée et d'oubli. En supposant que les deux portes soient fermées, le contenu de la cellule mémoire restera inchangé entre deux pas de temps, et la structure de régulation des gradients permet de retenir l'information sur de nombreux pas de temps, permettant ainsi aux gradients de circuler sur de nombreux pas de temps. Cela permet au modèle LSTM de surmonter le problème de l'atténuation des gradients qui se produit avec la plupart des modèles de réseaux de neurones récurrents.

Regardons la différence entre les RNNs et les LSTMs.

Dans les RNNs, nous avons une structure très simple avec une seule fonction d'activation (tanh).

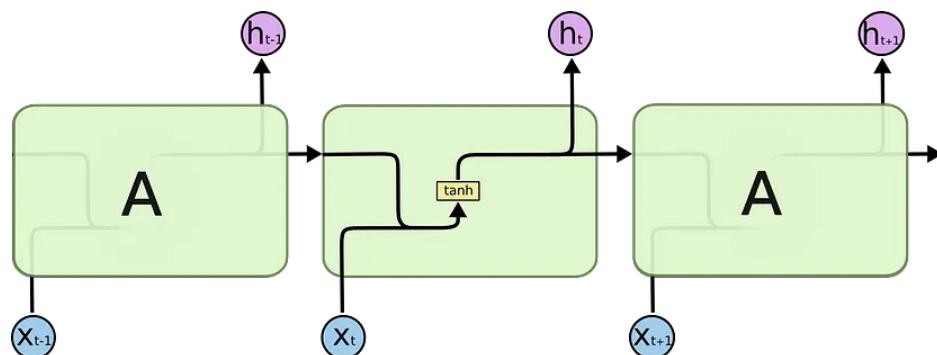


FIGURE 5.2 – Réseau RNN

Dans les LSTMs, au lieu d'un simple réseau avec une seule fonction d'activation, nous avons plusieurs composants, donnant au réseau le pouvoir d'oublier et de se souvenir des informations.

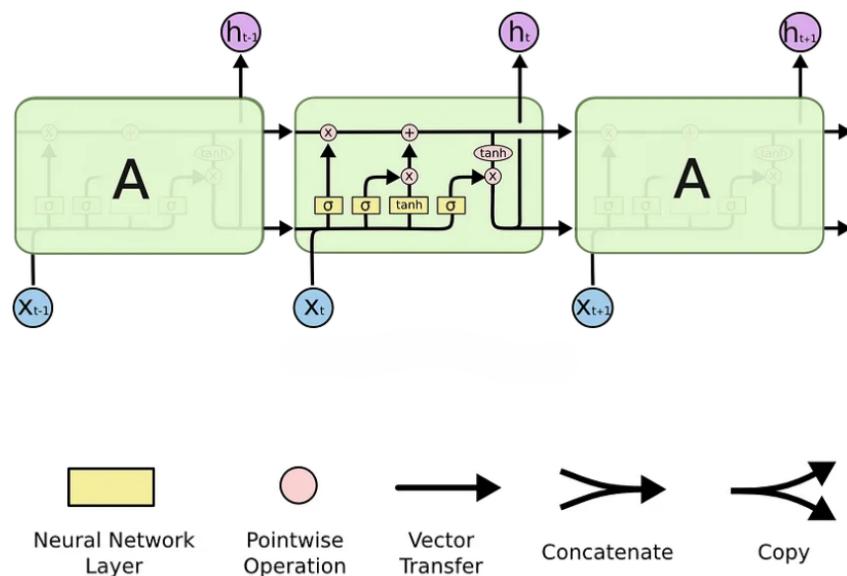


FIGURE 5.3 – Réseau LSTM

Les réseaux LSTM possèdent 4 composants différents, à savoir :

- **État de la cellule** (Cell state ou Memory cell)
- **Porte d'oubli** (Forget gate)
- **Porte d'entrée** (Input gate)
- **Porte de sortie** (Output gate)

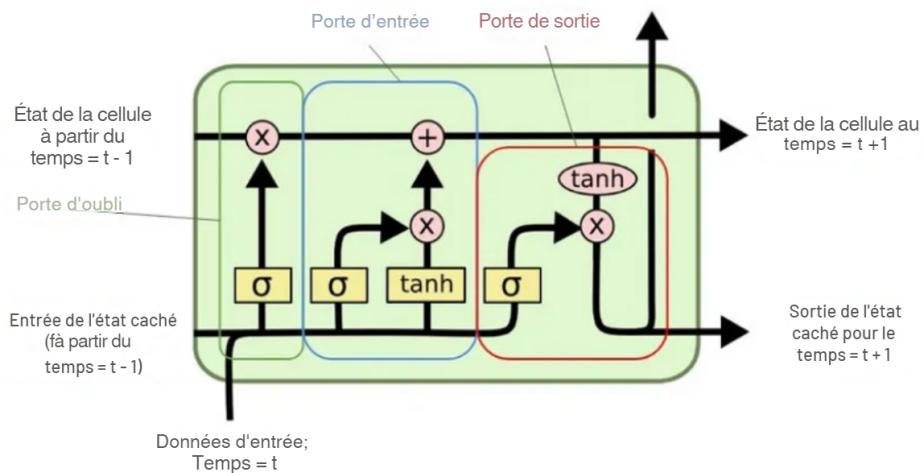


FIGURE 5.4 – Composants du LSTM

5.3.1 Cellule d'État (Cellule de Mémoire)

La cellule d'état est le premier composant de l'unité LSTM et traverse l'ensemble de l'unité. On peut la considérer comme un tapis roulant.

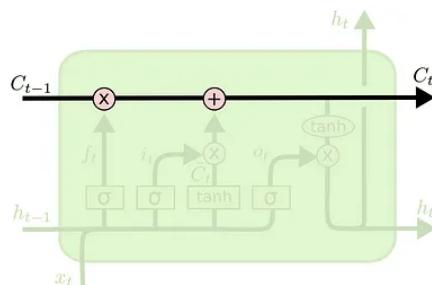


FIGURE 5.5 – Cellule de mémoire

Cette cellule d'état est responsable de la mémorisation et de l'oubli, basée sur le contexte de l'entrée. Cela signifie que certaines des informations précédentes doivent être mémorisées tandis que d'autres doivent être oubliées, et certaines nouvelles informations doivent être ajoutées à la mémoire. La première opération (\times) est l'opération pointwise, qui consiste à multiplier l'état de la cellule par un tableau de $[-1, 0, 1]$. L'information multipliée par 0 sera oubliée par le LSTM. Une autre opération est (+), qui est responsable de l'ajout de nouvelles informations à l'état.

5.3.2 Porte d'oubli

Comme son nom l'indique, la porte d'oubli de l'LSTM décide quelles informations doivent être oubliées. Une couche sigmoïde est utilisée pour prendre cette décision. Cette couche sigmoïde est appelée la "couche de la porte d'oubli".

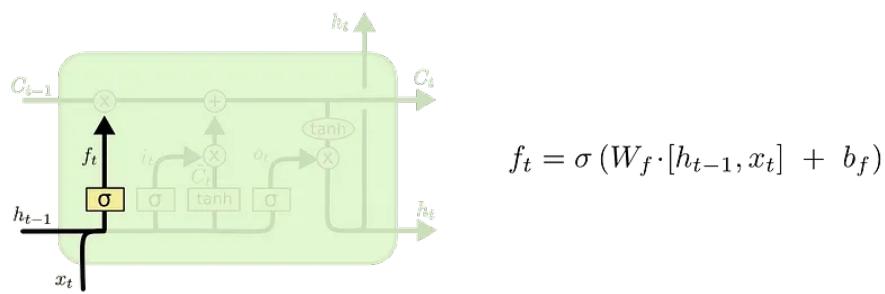


FIGURE 5.6 – Porte d'oubli

Elle effectue un produit scalaire de $h(t - 1)$ et $x(t)$ et, avec l'aide de la couche sigmoïde, produit un nombre entre 0 et 1 pour chaque nombre dans l'état de la cellule $C(t - 1)$. Si la sortie est un '1', cela signifie que nous allons la conserver. Un '0' signifie l'oublier complètement.

5.3.3 Porte d'entrée

La porte d'entrée donne de nouvelles informations au LSTM et décide si ces nouvelles informations vont être stockées dans l'état de la cellule.

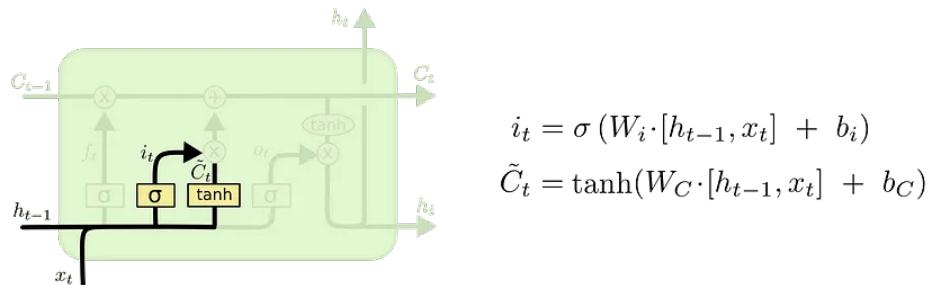


FIGURE 5.7 – Porte d'entrée

Cela comporte trois parties :

1. Une couche sigmoïde décide des valeurs à mettre à jour. Cette couche est appelée la "couche de la porte d'entrée".
2. Une couche de fonction d'activation tanh crée un vecteur de nouvelles valeurs candidates, $\tilde{C}(t)$, qui pourraient être ajoutées à l'état.
3. Nous combinons ensuite ces deux sorties, $i(t) * \tilde{C}(t)$, et mettons à jour l'état de la cellule.

Le nouvel état de la cellule $C(t)$ est obtenu en ajoutant la sortie des portes d'oubli et d'entrée.

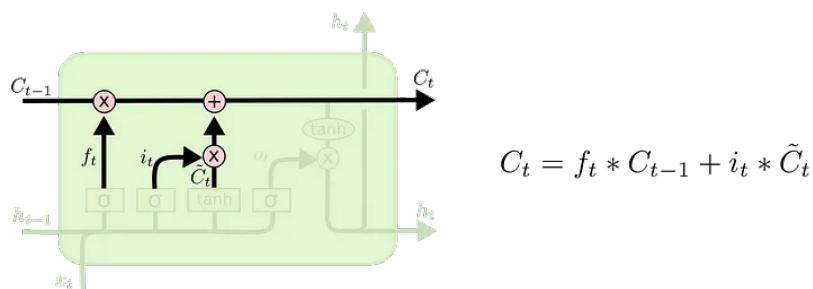


FIGURE 5.8 – Nouvelle cellule de mémoire

5.3.4 Porte de sortie

La sortie de l'unité LSTM dépend du nouvel état de la cellule.

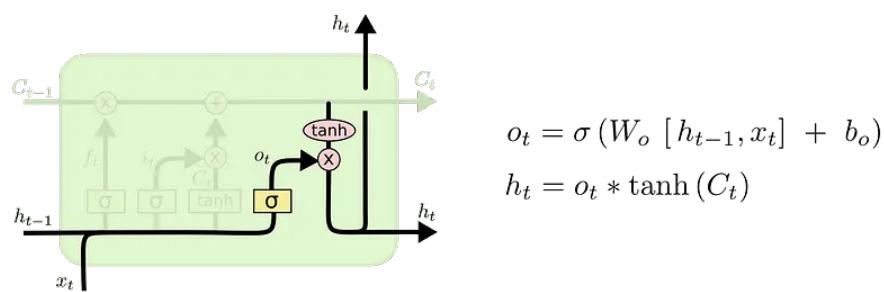


FIGURE 5.9 – Porte de sortie

Tout d'abord, une couche sigmoïde décide quelles parties de l'état de la cellule nous allons sortir. Ensuite, une couche **tanh** est utilisée sur l'état de la cellule pour limiter les valeurs entre -1 et 1, ce qui est finalement multiplié par la sortie de la porte sigmoïde.

5.4 Architectures

5.4.1 Réseau Autoencodeur LSTM

Un réseau autoencodeur LSTM combine les capacités des LSTM (Long Short-Term Memory) avec la structure des autoencodeurs. Les autoencodeurs sont des réseaux neuronaux utilisés pour apprendre une représentation compressée des données d'entrée. Un réseau autoencodeur LSTM est composé de deux parties principales : un encodeur et un décodeur.

- Encodeur** : L'encodeur est une série de couches LSTM qui prennent une séquence d'entrée et la convertissent en une représentation codée (ou état latent). Cette représentation codée est une sorte de résumé compact de la séquence d'entrée, capturant ses caractéristiques essentielles.
- Décodeur** : Le décodeur est une autre série de couches LSTM qui prennent la représentation codée produite par l'encodeur et génèrent une séquence en sortie. L'objectif est que la séquence de sortie soit aussi proche que possible de la séquence d'entrée originale.

```
array([[0.3 ,  0.027],
       [0.4 ,  0.064],
       [0.5 ,  0.125]],

      [[0.4 ,  0.064],
       [0.5 ,  0.125],
       [0.6 ,  0.216]],

      [[0.5 ,  0.125],
       [0.6 ,  0.216],
       [0.7 ,  0.343]],

      [[0.6 ,  0.216],
       [0.7 ,  0.343],
       [0.8 ,  0.512]],

      [[0.7 ,  0.343],
       [0.8 ,  0.512],
       [0.9 ,  0.729]]])
```

Input data converted into 3D
array of size n_samples x
timesteps x n_features

FIGURE 5.10 – Données transformées en 3D pour un réseau LSTM

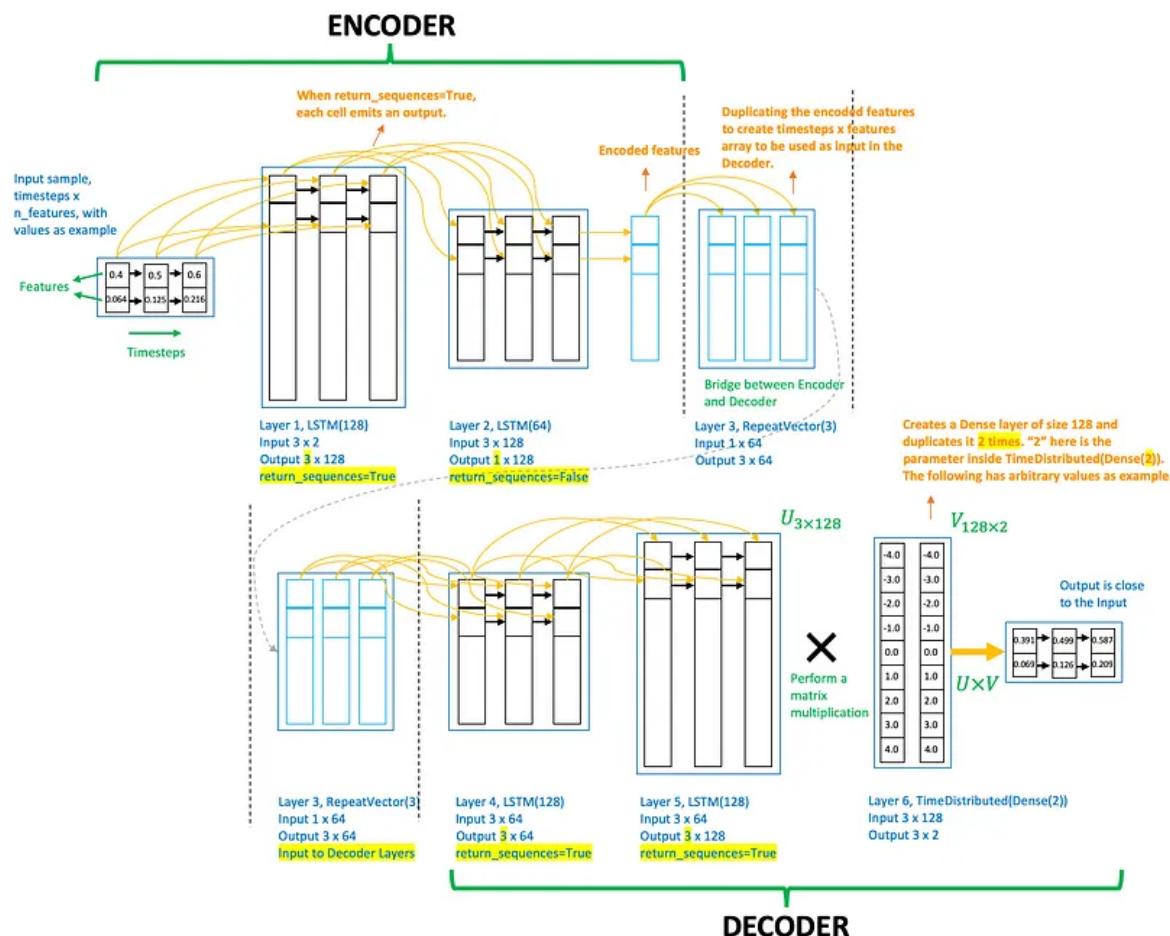
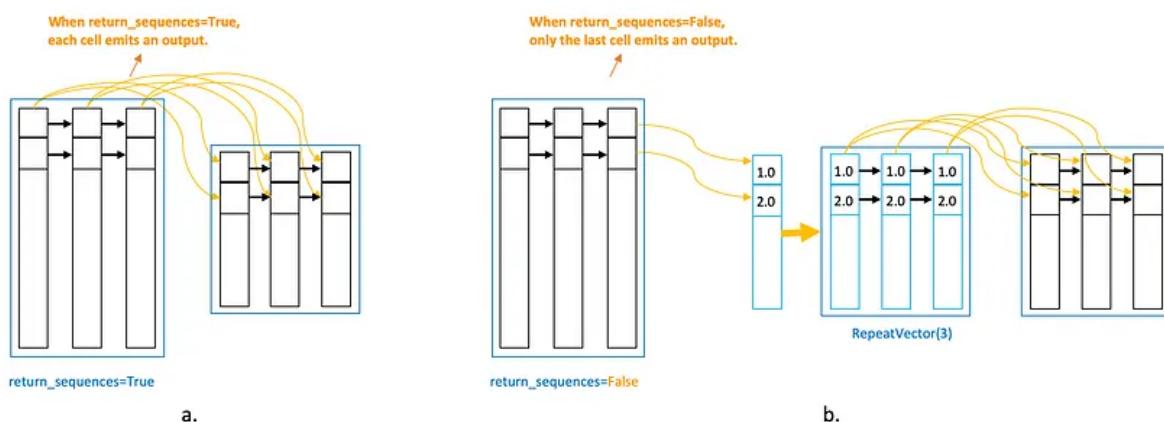


FIGURE 5.11 – Diagramme de flux d'un réseau autoencodeur LSTM

Le diagramme illustre le flux de données à travers les couches d'un réseau autoencodeur LSTM pour un échantillon de données. Un échantillon de données est une instance d'un ensemble de données. Dans notre exemple, un échantillon est un sous-tableau de taille 3x2 comme indiqué dans la Figure 5.10.

D'après ce diagramme, nous apprenons :

- Le réseau LSTM prend un tableau 2D comme entrée.
- Une couche de LSTM a autant de cellules que de pas de temps.
- En réglant `return_sequences=True`, chaque cellule par pas de temps émet un signal.

FIGURE 5.12 – Différence entre `return_sequences=True` et `return_sequences=False`

- En Fig. 5.12a, le signal d'une cellule à un pas de temps dans une couche est reçu par la cellule du même pas de temps dans la couche suivante.
- Dans les modules d'encodeur et de décodeur d'un autoencodeur LSTM, il est important d'avoir des connexions directes entre les cellules de pas de temps respectifs dans les couches LSTM consécutives comme en Fig. 5.12a.
- En Fig. 5.12b, seule la dernière cellule de pas de temps émet des signaux. La sortie est donc un vecteur.
- Comme montré en Fig. 5.12b, si la couche suivante est une LSTM, nous dupliquons ce vecteur en utilisant `RepeatVector(timesteps)` pour obtenir un tableau 2D pour la prochaine couche.
- Aucune transformation n'est nécessaire si la couche suivante est Dense (car une couche Dense attend un vecteur en entrée).

Revenons à l'autoencodeur LSTM dans la Fig. 5.11 :

- Les données d'entrée ont 3 pas de temps et 2 caractéristiques.
- La couche 1, `LSTM(128)`, lit les données d'entrée et produit 128 caractéristiques avec 3 pas de temps pour chacune car `return_sequences=True`.
- La couche 2, `LSTM(64)`, prend l'entrée 3×128 de la couche 1 et réduit la taille des caractéristiques à 64. Comme `return_sequences=False`, elle produit un vecteur de caractéristiques de taille 1×64 .
- La sortie de cette couche est le vecteur de caractéristiques encodées des données d'entrée. Ce vecteur de caractéristiques encodées peut être extrait et utilisé comme compression de données ou comme caractéristiques pour tout autre apprentissage supervisé ou non supervisé.
- La couche 3, `RepeatVector(3)`, réplique le vecteur de caractéristiques 3 fois.
- La couche `RepeatVector` agit comme un pont entre les modules d'encodeur et de décodeur. Elle prépare l'entrée sous forme de tableau 2D pour la première couche LSTM dans le décodeur.
- La couche de décodeur est conçue pour déplier l'encodage. Par conséquent, les couches de décodeur sont empilées dans l'ordre inverse de celui de l'encodeur.
- La couche 4, `LSTM(64)`, et la couche 5, `LSTM(128)`, sont les images miroirs de la couche 2 et de la couche 1, respectivement.
- La couche 6, `TimeDistributed(Dense(2))`, est ajoutée à la fin pour obtenir la sortie, où "2" est le nombre de caractéristiques dans les données d'entrée. La couche `TimeDistributed` crée un vecteur de longueur égale au nombre de caractéristiques produites par la couche précédente. Dans ce réseau, la couche 5 produit 128 caractéristiques. Par conséquent, la couche `TimeDistributed` crée un vecteur de longueur 128 et le duplique 2 fois (= `n_features`).
- La sortie de la couche 5 est un tableau 3×128 que nous notons U et celle de `TimeDistributed` dans la couche 6 est un tableau 128×2 noté V. Une multiplication matricielle entre U et V donne une sortie 3×2 .
- L'objectif de l'ajustement du réseau est de rendre cette sortie proche de l'entrée. Notez que ce réseau lui-même a assuré que les dimensions d'entrée et de sortie correspondent.

5.4.2 LSTM empilée

Les mêmes avantages peuvent être exploités avec les LSTM.

Étant donné que les LSTM opèrent sur des données de séquence, cela signifie que l'ajout de couches ajoute des niveaux d'abstraction des observations d'entrée au fil du temps. En effet, cela découpe les observations dans le temps ou représente le problème à différentes échelles de temps.

Les LSTM empilées ou LSTM profondes ont été introduites par Graves, et al. dans leur application des LSTM à la reconnaissance vocale, battant une référence sur un problème standard difficile.

Dans le même travail, ils ont constaté que la profondeur du réseau était plus importante que le nombre de cellules de mémoire dans une couche donnée pour modéliser la compétence.

Les LSTM empilées sont maintenant une technique stable pour les problèmes de prédiction de séquence difficiles. Une architecture LSTM empilée peut être définie comme un modèle LSTM composé de plusieurs couches LSTM. Une couche LSTM ci-dessus fournit une sortie de séquence plutôt qu'une seule valeur de sortie à la couche LSTM ci-dessous. Plus précisément, une sortie par pas de temps d'entrée, plutôt qu'un pas de temps de sortie pour tous les pas de temps d'entrée.

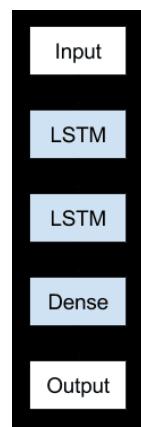


FIGURE 5.13 – LSTM empilée

5.5 Variantes

Il existe des variantes de LSTM, telles que :

5.5.1 Peephole LSTM

Les LSTM Peephole ajoutent des connexions directes des cellules d'état aux portes d'entrée, de sortie et d'oubli, permettant ainsi à ces portes de regarder dans l'état de la cellule.

La figure 5.14 est une représentation graphique d'une unité LSTM avec des connexions peephole. Les connexions peephole permettent aux portes d'accéder au carrousel d'erreur constante (CEC), dont l'activation est l'état de la cellule. h_{t-1} n'est pas utilisé, c_{t-1} est utilisé à la place dans la plupart des endroits.

$$f_t = \sigma_g(W_f x_t + U_f c_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i c_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o c_{t-1} + b_o)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(W_c x_t + b_c)$$

$$h_t = o_t \odot \sigma_h(c_t)$$

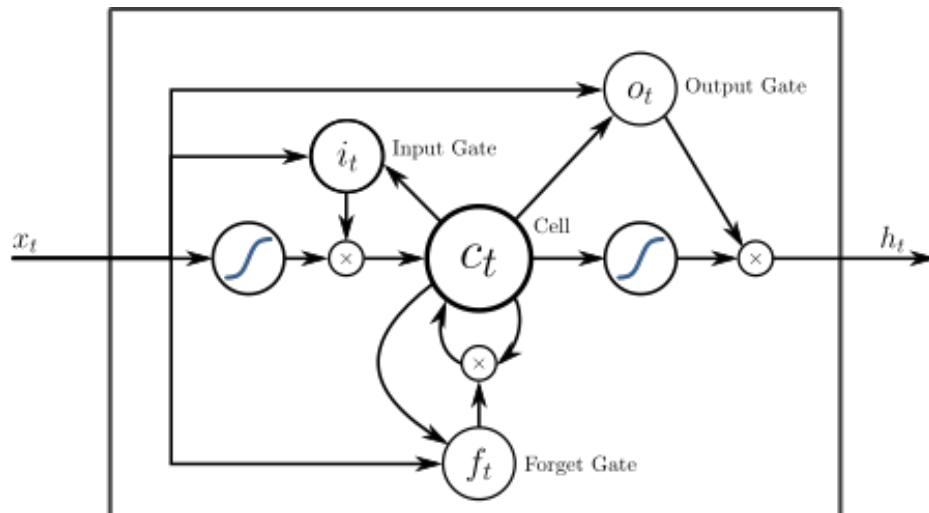


FIGURE 5.14 – Peephole LSTM

Chacune des portes peut être considérée comme un neurone "standard" dans un réseau neuronal feed-forward (ou multi-couches) : elles calculent une activation (en utilisant une fonction d'activation) d'une somme pondérée. i_t , o_t et f_t représentent respectivement les activations des portes d'entrée, de sortie et d'oubli, à l'instant t .

Les 3 flèches sortant de la cellule mémoire c vers les 3 portes i , o et f représentent les connexions de surveillance. Ces connexions peephole représentent en fait les contributions de l'activation de la cellule mémoire c à l'instant $t - 1$, c'est-à-dire la contribution de c_{t-1} (et non c_t , comme la figure 5.14 pourrait le suggérer). En d'autres termes, les portes i , o et f calculent leurs activations à l'instant t (c'est-à-dire, respectivement, i_t , o_t et f_t) en tenant également compte de l'activation de la cellule mémoire c à l'instant $t - 1$, c'est-à-dire c_{t-1} .

La seule flèche sortant de la cellule mémoire de gauche à droite n'est pas une connexion peephole et représente c_t .

Les petits cercles contenant un symbole \times représentent une multiplication élément par élément entre leurs entrées. Les grands cercles contenant une courbe en forme de S représentent l'application d'une fonction différentiable (comme la fonction sigmoïde) à une somme pondérée.

5.5.2 Bi-LSTM (LSTM bidirectionnel)

Les LSTM bidirectionnels (Bi-LSTM) permettent de tenir compte dans une séquence de l'influence de la suite sur ce qui précède, en plus de l'habituelle influence du précédent sur ce qui suit. Cela signifie que l'information peut être passée dans les deux directions, améliorant ainsi les capacités du modèle à comprendre les dépendances dans les séquences. Il combine la puissance des LSTM avec un traitement bidirectionnel, permettant au modèle de capturer à la fois le contexte passé et futur de la séquence d'entrée.

Pour comprendre le Bi-LSTM, examinons ses composants et sa fonctionnalité :

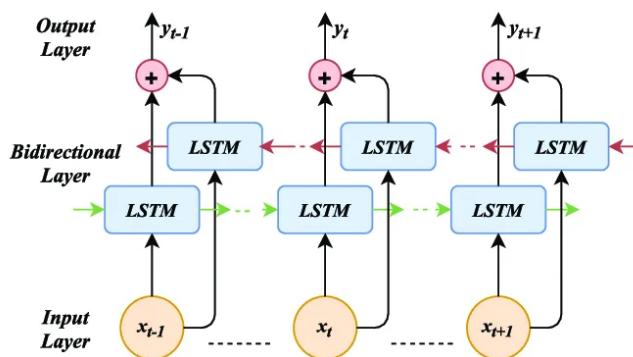


FIGURE 5.15 – LSTM Bidirectionnel

- LSTM (Long Short-Term Memory)** : LSTM est un type de RNN conçu pour surmonter les limitations des RNN traditionnels dans la capture des dépendances à long terme dans les données séquentielles. Il introduit des cellules de mémoire et des mécanismes de porte pour conserver et oublier sélectivement des informations au fil du temps. Les LSTM ont un état de mémoire interne qui peut stocker des informations pendant de longues durées, leur permettant de capturer des dépendances qui peuvent s'étendre sur de nombreux pas de temps.
- Traitement bidirectionnel** : Contrairement aux RNN traditionnels qui traitent les séquences d'entrée dans une seule direction (soit vers l'avant, soit vers l'arrière), le Bi-LSTM traite la séquence dans les deux directions simultanément. Il se compose de deux couches LSTM : l'une traitant la séquence dans la direction avant et l'autre dans la direction arrière. Chaque couche maintient ses propres états cachés et cellules de mémoire.
- Passe avant** : Pendant la passe avant, la séquence d'entrée est alimentée dans la couche LSTM avant du premier pas de temps au dernier. À chaque pas de temps, le LSTM avant calcule son état caché et met à jour sa cellule de mémoire en fonction de l'entrée actuelle et de l'état caché et de la cellule de mémoire précédents.
- Passe arrière** : Simultanément, la séquence d'entrée est également alimentée dans la couche LSTM arrière dans l'ordre inverse, du dernier pas de temps au premier. De manière similaire à la passe avant, le LSTM arrière calcule son état caché et met à jour sa cellule de mémoire en fonction de l'entrée actuelle et de l'état caché et de la cellule de mémoire précédents.
- Combinaison des états avant et arrière** : Une fois que les passes avant et arrière sont terminées, les états cachés des deux couches LSTM sont combinés à chaque pas de temps. Cette combinaison peut être aussi simple que la concaténation des états cachés ou l'application d'une autre transformation. Le bénéfice du Bi-LSTM est qu'il capture non seulement le contexte qui précède un pas de temps spécifique (comme dans les RNN traditionnels) mais aussi le contexte qui suit. En considérant à la fois les informations passées et futures, le Bi-LSTM peut capturer des dépendances plus riches dans la séquence d'entrée.

Architecture

Examinons chaque composant de l'architecture :



FIGURE 5.16 – Architecture du Bi-LSTM

- Séquence d'entrée** : La séquence d'entrée est une séquence de points de données, tels que des mots dans une phrase ou des caractères dans un texte. Chaque point de données est généralement représenté sous forme de vecteur ou de représentation incorporée.
- Incorporation** : La séquence d'entrée est souvent transformée en représentations vectorielles denses appelées incorporations. Les incorporations capturent le sens sémantique des points de données et fournissent une représentation plus compacte et significative pour les couches suivantes.
- Bi-LSTM** : La couche Bi-LSTM est le composant central de l'architecture. Elle se compose de deux couches LSTM : l'une traitant la séquence d'entrée dans la direction avant et l'autre dans la direction arrière. Chaque couche LSTM a son propre ensemble de paramètres.
- Sortie** : La sortie de la couche Bi-LSTM est la combinaison des états cachés des deux couches LSTM avant et arrière à chaque pas de temps. La méthode de combinaison spécifique peut varier, comme la concaténation des états cachés ou l'application d'une transformation différente.

La couche Bi-LSTM traite la séquence d'entrée dans les deux directions avant et arrière simultanément. Pendant la passe avant, la couche LSTM capture des informations du passé (pas de temps précédents), tandis que pendant la passe arrière, elle capture des informations du futur (pas de temps suivants). Ce traitement bidirectionnel permet au modèle de capturer efficacement des dépendances à long terme dans la séquence d'entrée.

La sortie de la couche Bi-LSTM peut être utilisée à diverses fins selon la tâche spécifique. Par exemple, dans la classification de texte, la sortie peut être passée à travers une couche entièrement connectée suivie d'une activation **softmax** pour obtenir les probabilités de classe. Dans les tâches d'étiquetage de séquence comme la reconnaissance d'entités nommées, la sortie peut être directement utilisée pour prédire l'étiquette de chaque jeton d'entrée.

L'architecture d'un Bi-LSTM peut être étendue ou modifiée en fonction des besoins spécifiques de la tâche. Des couches supplémentaires, telles que des couches entièrement connectées ou des mécanismes d'attention, peuvent être ajoutées au-dessus de la couche Bi-LSTM pour améliorer davantage les capacités et les performances du modèle.

5.6 Étapes

Un réseau LSTM effectue durant chaque passage les 5 étapes suivantes :

- Détection des informations passées dans le **cell state** via le forget gate
- Choix des informations pertinentes à long terme à travers l'**input gate**
- Ajout des informations choisies au **cell state**
- Détection des informations importantes à court terme dans le **cell state**
- Génération du nouveau **hidden state** à travers l'**output gate**

5.7 Niveaux d'itération

Un échantillon dans un LSTM est une fenêtre d'observations par pas de temps. Dans les LSTM, les itérations se terminent à un pas de temps. Au sein d'un pas de temps, toutes les opérations de la cellule décrites en Section 5.6 sont exécutées.

Les opérations des cellules dans un pas de temps sont séquentielles. La sortie d'un pas de temps devient l'entrée du suivant. En raison de cela, les pas de temps sont traités un par un. De plus, les étapes au sein d'un pas de temps sont également en ordre.

En raison des opérations séquentielles, les itérations des LSTM sont intensives en calcul. Cependant, les échantillons au sein d'un lot n'interagissent pas dans les LSTM sans état et, par conséquent, un lot est traité ensemble en utilisant des opérations tensorielles.[26]

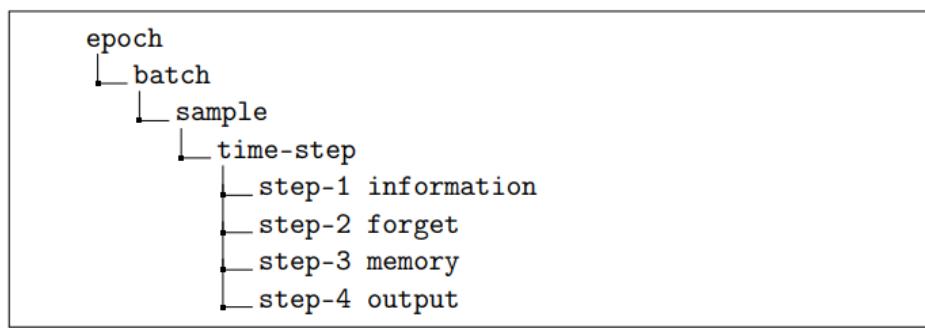


FIGURE 5.17 – Niveaux d’itération pour l’entraînement des LSTM

5.8 Avantages et Inconvénients

5.8.1 Avantages

- **Modélisation des dépendances à long terme** : Les LSTM sont bien adaptés à la modélisation des dépendances à long terme dans les données séquentielles, car ils peuvent " se souvenir " ou " oublier " sélectivement des informations au fil du temps. Cela les rend utiles pour des tâches telles que la reconnaissance vocale, la traduction automatique et la génération de texte.
- **Robustesse aux données bruitées** : Les LSTM sont plus robustes aux données bruitées ou manquantes que d’autres types de réseaux neuronaux récurrents, car ils peuvent filtrer sélectivement les informations non pertinentes ou bruitées en utilisant la porte d’oubli.
- **Flexibilité** : Les LSTM peuvent être utilisés pour une grande variété de tâches, y compris la classification, la régression et l’apprentissage séquence à séquence.
- **Interprétabilité** : Comme les LSTM maintiennent un vecteur d’état de cellule qui représente la " mémoire " du réseau à chaque pas de temps, ils peuvent être plus interprétables que d’autres types de réseaux neuronaux récurrents.

5.8.2 Inconvénients

- **Coût computationnel élevé** : Les LSTM peuvent être coûteux à entraîner et à évaluer, en particulier pour des séquences longues ou des ensembles de données volumineux.
- **Sensibilité au surapprentissage** : Les LSTM peuvent être sensibles au surapprentissage sur de petits ensembles de données, surtout si l’architecture du modèle est trop complexe.
- **Réglage des hyperparamètres** : Les LSTM ont de nombreux hyperparamètres qui doivent être réglés pour obtenir des performances optimales, y compris le nombre d’unités cachées, le taux d’apprentissage et le taux de dropout.
- **Exigences en matière de données** : Les LSTM nécessitent une grande quantité de données d’entraînement pour apprendre des motifs complexes dans les données. Si la quantité de données disponibles est insuffisante, le modèle peut ne pas bien fonctionner.

Conclusion

Les réseaux LSTM ont révolutionné le domaine de l’apprentissage automatique, en particulier pour le traitement de données séquentielles. Leur capacité à modéliser les dépendances à long terme, leur robustesse aux données bruitées et leur flexibilité en font des outils puissants pour une variété de tâches, telles que la reconnaissance vocale, la traduction automatique, la génération de texte et bien d’autres. Cependant, ils ne sont pas sans inconvénients, notamment leur coût computationnel élevé, leur sensibilité au surapprentissage et leurs exigences en matière de données. En fin de compte, les LSTM restent un domaine de recherche dynamique, avec de nombreuses variantes et améliorations en cours de développement pour relever les défis actuels et exploiter tout leur potentiel dans des applications du monde réel.

Chapitre 6

Unités Récurrentes à Portes (GRU)

Ce chapitre présentera les unités récurrentes à portes, également connues sous l'acronyme GRU (Gated Recurrent Unit). Les GRU sont une variante des réseaux de neurones récurrents conçue pour résoudre les mêmes problèmes que les LSTM, tels que le défi du gradient évanescence et explosif, tout en offrant une architecture plus simple et plus efficace en termes de calcul. Nous explorerons les principes fondamentaux des GRU, leur structure interne, leurs avantages par rapport aux LSTM, ainsi que leurs applications dans divers domaines de l'apprentissage automatique et du traitement du langage naturel.

6.1 Définition

Les unités récurrentes fermées (GRU) sont un système de porte dans les réseaux de neurones récurrents, introduit en 2014 par Kyunghyun Cho et al[10]. Le GRU est comme une longue mémoire à court terme (LSTM) avec une porte d'oubli[6], mais a moins de paramètres que LSTM, car il n'a pas de porte de sortie. Les performances de GRU sur certaines tâches de modélisation de musique polyphonique, de modélisation de signaux vocaux et de traitement du langage naturel se sont avérées similaires à celles de LSTM.

6.2 Structure

Les unités récurrentes fermées, ou GRU (Gated Recurrent Units), ont simplifié l'architecture des LSTM en se débarrassant de l'état de cellule et en utilisant l'état caché pour transférer les informations. Elles ne possèdent également que deux portes : une porte de réinitialisation et une porte de mise à jour.

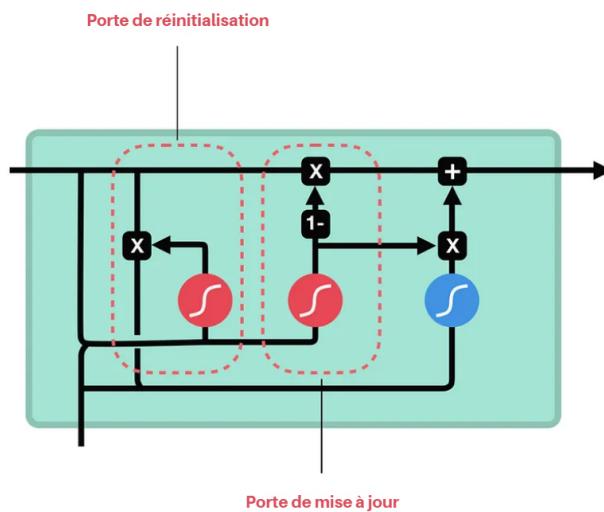


FIGURE 6.1 – Cellule GRU et ses portes

- **Porte de mise à jour (z_t)** : Cette porte fonctionne de manière similaire aux portes d'oubli et d'entrée des LSTM. Elle décide quelles informations doivent être oubliées et quelles nouvelles informations doivent être ajoutées.
- **Porte de réinitialisation (r_t)** : Cette porte est utilisée pour décider quelle quantité d'informations passées doit être oubliée.

6.3 Pourquoi GRU ?

Les mémoires à long terme (LSTM) peuvent résoudre de nombreuses tâches impossibles à résoudre par les algorithmes d'apprentissage précédents pour les réseaux de neurones récurrents (RNN). Nous identifions une faiblesse des réseaux LSTM lorsqu'ils traitent des flux d'entrée continus sans marques explicites de fin de séquence. Sans réinitialisations, les valeurs d'état interne peuvent croître indéfiniment et finalement causer la défaillance du réseau. Notre solution consiste en une "porte d'oubli" adaptative qui permet à une cellule LSTM d'apprendre à se réinitialiser aux moments appropriés, libérant ainsi des ressources internes.

6.4 Fonctionnement des GRU

Voyons maintenant le fonctionnement de ces portes. Pour trouver l'état caché H_t dans un GRU, cela suit un processus en deux étapes. La première étape consiste à générer ce que l'on appelle l'état caché candidat.

État Caché Candidat

$$\hat{h}_t = \tanh(x_t * U_g + (r_t \odot h_{t-1}) * W_g)$$

L'état caché candidat prend en entrée l'état caché de l'instant précédent $t-1$, multiplié par la sortie de la porte de réinitialisation r_t . Cette information est ensuite passée à la fonction **tanh** pour obtenir l'état caché candidat.

$$\hat{h}_t = \tanh(x_t * U_g + (\textcolor{red}{r_t} \odot \textcolor{blue}{h_{t-1}}) * W_g)$$

La partie la plus importante de cette équation est l'utilisation de la valeur de la porte de réinitialisation pour contrôler l'influence de l'état caché précédent sur l'état candidat. Si la valeur de r_t est égale à 1, cela signifie que l'intégralité de l'information de l'état caché précédent h_{t-1} est prise en compte. De même, si la valeur de r_t est 0, cela signifie que l'information de l'état caché précédent est complètement ignorée.

État Caché

Une fois que nous avons l'état candidat, il est utilisé pour générer l'état caché actuel h_t . C'est là qu'intervient la porte de mise à jour. Cette équation est très intéressante, car au lieu d'utiliser une porte distincte comme dans les LSTM, le GRU utilise une seule porte de mise à jour pour contrôler à la fois l'information historique h_{t-1} et la nouvelle information provenant de l'état candidat.

$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \hat{h}_t$$

Maintenant, supposons que la valeur de z_t soit proche de 0, alors le premier terme de l'équation s'annulera, ce qui signifie que le nouvel état caché n'aura pas beaucoup d'information de l'état caché précédent. En revanche, le second terme deviendra presque égal à un, ce qui signifie que l'état caché à l'instant actuel consistera uniquement en l'information provenant de l'état candidat.

$$h_t = \textcolor{red}{u_t} \odot \textcolor{blue}{h_{t-1}} + (1 - u_t) \odot \hat{h}_t$$

De même, si la valeur de z_t est proche de 1, alors le second terme deviendra entièrement nul et l'état caché actuel dépendra entièrement du premier terme, c'est-à-dire de l'information provenant de l'état caché précédent h_{t-1} .

$$h_t = u_t \odot h_{t-1} + (\textcolor{red}{1} - \textcolor{red}{u_t}) \odot \hat{h}_t$$

Nous pouvons donc conclure que la valeur de z_t est très critique dans cette équation et elle peut varier de 0 à 1.

6.5 Architectures

Il existe plusieurs variantes de l'unité récurrente fermée dans lesquels l'unité est activé en utilisant diverses combinaisons de l'état caché et du biais précédent, ainsi une forme simplifiée appelée unité fermée minimale.

6.5.1 Unité entièrement fermée

Au départ, pour $t = 0$, le vecteur de sortie est $h_0 = 0$. Les équations qui régissent le fonctionnement du mécanisme de mise à jour et de réinitialisation dans un GRU sont les suivantes :

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$\hat{h}_t = \phi_h(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t$$

où les variables sont définies comme suit :

- x_t : vecteur d'entrée à l'instant t
- h_t : vecteur de sortie à l'instant t
- \hat{h}_t : vecteur d'activation candidat à l'instant t
- z_t : mise à jour du vecteur de porte à l'instant t
- r_t : réinitialisation du vecteur de porte à l'instant t
- W , U et b : matrices de paramètres et vecteur de biais

Les fonctions d'activation utilisées sont les suivantes :

- σ_g : fonction sigmoïde
- ϕ_h : fonction d'activation (originellement tangente hyperbolique)

Des fonctions d'activation alternatives peuvent être utilisées, à condition que $\sigma_g(x) \in [0, 1]$.

Des formes alternatives peuvent être créées en modifiant z_t et r_t comme suit :

- **Type 1** : Chaque porte ne dépend que de l'état caché précédent et du biais.

$$z_t = \sigma_g(U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(U_r h_{t-1} + b_r)$$

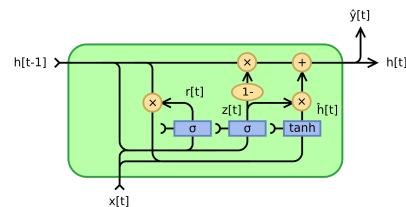


FIGURE 6.2 – Unité récurrente fermée, version entièrement fermée

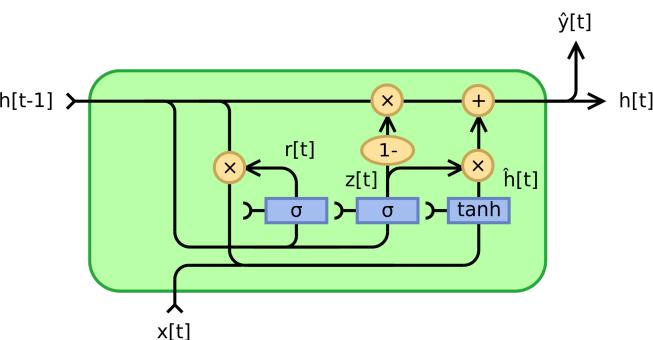


FIGURE 6.3 – GRU Type 1

- **Type 2** : Chaque porte ne dépend que de l'état caché précédent.

$$z_t = \sigma_g(U_z h_{t-1})$$

$$r_t = \sigma_g(U_r h_{t-1})$$

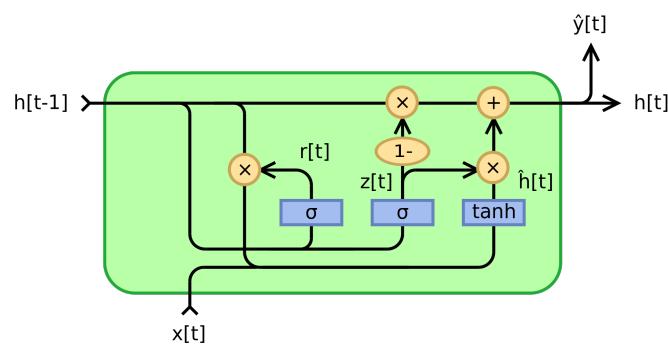


FIGURE 6.4 – GRU Type 2

- **Type 3 :** Chaque porte est calculée en utilisant uniquement le biais.

$$z_t = \sigma_g(b_z)$$

$$r_t = \sigma_g(b_r)$$

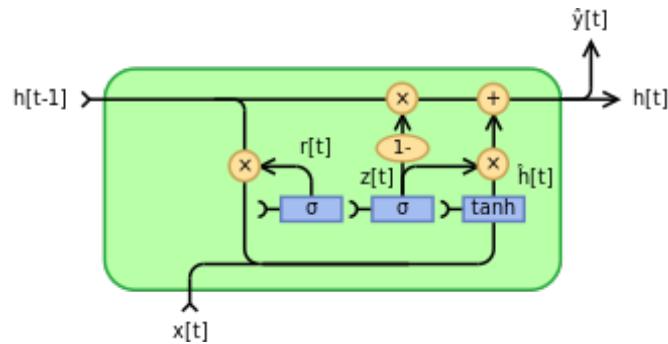


FIGURE 6.5 – GRU Type 3

6.5.2 Unité fermée minimale

L'unité fermée minimale est similaire à l'unité entièrement fermée, mais le vecteur d'activation de la porte de mise à jour et de la réinitialisation sont fusionnés dans une porte d'oubli. Cela implique également que l'équation du vecteur de sortie doit être modifiée :

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ \hat{h}_t &= \phi_h(W_h x_t + U_h (f_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - f_t) \odot h_{t-1} + f_t \odot \hat{h}_t \end{aligned}$$

où les variables sont définies comme suit :

- x_t : vecteur d'entrée à l'instant t
- h_t : vecteur de sortie à l'instant t
- \hat{h}_t : vecteur d'activation candidat à l'instant t
- f_t : vecteur de porte d'oubli à l'instant t
- W , U et b : matrices de paramètres et vecteur de biais

Conclusion

En conclusion, les Gated Recurrent Units (GRU) représentent une avancée significative dans le domaine des réseaux de neurones récurrents, offrant une alternative efficace aux LSTM (Long Short-Term Memory). Grâce à leur architecture simplifiée, qui intègre uniquement deux portes - la porte de réinitialisation et la porte de mise à jour - les GRU parviennent à conserver et à manipuler efficacement les informations temporelles tout en réduisant la complexité computationnelle.

Le mécanisme de la porte de réinitialisation permet de contrôler l'influence de l'état caché précédent, tandis que la porte de mise à jour joue un rôle crucial en équilibrant l'apport des nouvelles informations et des informations historiques. Cette double capacité permet aux GRU de s'adapter dynamiquement à différentes séquences de données, ce qui les rend particulièrement utiles pour les tâches de traitement du langage naturel, de reconnaissance vocale et d'autres applications séquentielles.

Bien que les GRU et les LSTM aient des performances comparables, le choix entre les deux architectures dépend souvent du cas d'utilisation spécifique et des exigences en termes de vitesse d'entraînement et de capacité de mémorisation. Les chercheurs et les ingénieurs continuent d'explorer ces modèles pour déterminer lequel offre les meilleurs résultats selon les contextes particuliers.

Ainsi, les GRU, avec leur structure plus simple et leur efficacité, demeurent un outil précieux pour les problèmes de prédition de séquences complexes, offrant un équilibre optimal entre performance et complexité.

Chapitre 7

Languages et outils utilisés

Dans ce projet, nous avons exploité une variété de langages de programmation et d'outils de développement pour mener à bien l'ensemble des étapes de la modélisation prédictive, allant de la prétraitement des données à l'évaluation des performances des modèles. L'utilisation de ces technologies a permis de gérer efficacement les données volumineuses, de construire et d'entraîner des modèles de réseaux de neurones complexes, ainsi que de visualiser et interpréter les résultats de manière claire et concise.

7.1 Langages de Programmation

Le principal langage de programmation utilisé dans ce projet est **Python**.

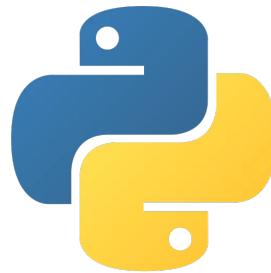


FIGURE 7.1 – Logo du Langage Python

Python est largement reconnu pour sa simplicité et sa lisibilité, ce qui en fait un choix idéal pour la science des données et le développement de modèles de machine learning. Les bibliothèques Python suivantes ont été particulièrement utiles :

- **NumPy** : Utilisé pour les opérations numériques et la manipulation de matrices.



FIGURE 7.2 – Logo du Numpy

- **Pandas** : Employé pour la manipulation et l'analyse des données, permettant une gestion facile des structures de données complexes.



FIGURE 7.3 – Logo du Pandas

- **Scikit-learn** : Fournit des outils simples et efficaces pour l'analyse des données et le machine learning.



FIGURE 7.4 – Logo du Scikit-learn

- **Matplotlib et Seaborn** : Utilisés pour la visualisation des données et des résultats des modèles.



(a) Logo du Seaborn



(b) Logo du Matplotlib

- **Flask** : Utilisé pour le backend de l’application, permettant de gérer les requêtes et les réponses HTTP.



FIGURE 7.6 – Logo du Flask

- **Chart.js** : Utilisé pour afficher les prévisions de consommation sous forme de graphiques interactifs.



FIGURE 7.7 – Logo du Chart.js

7.2 Bibliothèques de Deep Learning

Pour le développement et l’entraînement des modèles de réseaux de neurones, nous avons utilisé les bibliothèques suivantes :

- **TensorFlow** : Une plateforme open-source de bout en bout pour le machine learning, TensorFlow a été utilisé pour créer, entraîner et déployer des modèles de machine learning à grande échelle.



FIGURE 7.8 – Logo du TensorFlow

- **Keras** : Une API de haut niveau pour TensorFlow, Keras simplifie la construction et l’entraînement des réseaux de neurones grâce à son interface intuitive et ses abstractions simplifiées.



FIGURE 7.9 – Logo du Keras

7.3 Environnement de Développement

L'environnement de développement utilisé pour ce projet inclut :

- **Visual Studio Code (VS Code)** : Un éditeur de code source développé par Microsoft, VS Code est connu pour sa légèreté et sa flexibilité. Il supporte une grande variété d'extensions, ce qui le rend idéal pour le développement Python et la gestion des notebooks Jupyter.

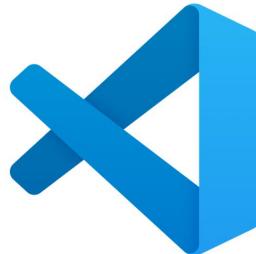


FIGURE 7.10 – Logo du VS Code

- **Google Colab** : Une plateforme de notebooks basée sur le cloud qui permet d'accéder gratuitement à des GPU et des TPU, facilitant ainsi l'entraînement des modèles complexes sur des matériels accélérés.



FIGURE 7.11 – Logo du Google Colab

7.4 Outils de Gestion de Versions

Pour la gestion du code et la collaboration, nous avons utilisé :

- **Git** : Un système de contrôle de version distribué, Git a permis de suivre les modifications du code.



FIGURE 7.12 – Git

- **GitHub** : Une plateforme de développement collaboratif basée sur Git, utilisée pour héberger le dépôt de code, gérer les versions et faciliter le travail en équipe.



FIGURE 7.13 – Logo du Github

L'intégration de ces langages et outils a permis une approche systématique et organisée du projet, assurant une gestion efficace des données, une modélisation robuste, et une évaluation rigoureuse des performances des modèles.

Chapitre 8

Réalisation

Ce chapitre marque le début de la mise en œuvre concrète du projet de prévision de la demande électrique multi-horizons. Après avoir posé les bases conceptuelles et méthodologiques dans le cadre général du projet, nous entamons maintenant les étapes de développement du modèle de prévision, en nous appuyant sur les techniques avancées d'apprentissage profond.

Dans cette section, nous détaillerons les différentes phases de réalisation du projet, allant de la collecte et de la préparation des données à la mise en œuvre et à l'évaluation des modèles. Nous explorerons également les défis techniques rencontrés et les solutions adoptées pour surmonter ces obstacles. Enfin, nous discuterons des résultats obtenus et des perspectives pour l'amélioration continue du modèle de prévision.

En alignement avec les objectifs définis dans le cadre général du projet, cette section vise à fournir une compréhension approfondie du processus de développement du modèle de prévision de la demande électrique, tout en mettant en lumière les contributions et les innovations spécifiques apportées dans le cadre de ce projet.

8.1 Collection des données

8.1.1 Données de consommation

Dans le cadre de ce projet, l'objectif principal est de prévoir la demande électrique future en se basant sur des données historiques de consommation. Les données initiales fournies couvrent une période étendue, allant de 2010 à 2023, et sont enregistrées à une résolution horaire. Chaque enregistrement dans cet ensemble de données contient deux champs essentiels :

- **Datetime** : La date et l'heure du relevé, au format %Y-%m-%d %H:%M:%S.
- **Consommation** : La consommation électrique en kilowattheure (kWh).

Ces données offrent une base riche pour l'analyse des tendances temporelles de la consommation électrique et la construction de modèles de prévision.

Contexte et Importance

La prévision précise de la demande électrique est cruciale pour plusieurs raisons. Elle permet aux gestionnaires de réseaux électriques de planifier les opérations, d'optimiser la production et la distribution d'électricité, et de garantir la stabilité du réseau. Une bonne prévision aide également à réduire les coûts opérationnels et à minimiser les risques de pénurie ou de surcharge, améliorant ainsi l'efficacité énergétique globale.

Limites des Données Disponibles

Cependant, les données initialement disponibles présentent une limitation notable : elles se concentrent exclusivement sur les historiques de consommation d'électricité sans inclure d'autres variables explicatives potentielles telles que les conditions météorologiques, les jours fériés, ou les types de jours (jours ouvrables contre week-ends). Cette restriction peut limiter la précision des modèles prédictifs car la consommation d'électricité est influencée par divers facteurs exogènes.

Approche Proposée

Pour surmonter cette limitation, il est crucial d'explorer des méthodes qui intègrent non seulement les données de consommation historique mais aussi d'autres variables pertinentes. Cela pourrait inclure :

- **Données météorologiques**
- **Données calendaires**

L'intégration de ces facteurs peut améliorer significativement la précision des prévisions de la demande électrique, permettant ainsi de mieux répondre aux défis posés par la variabilité et l'incertitude inhérentes à la consommation d'énergie.

	Date	Time	Hourly_Value
254209	2010-01-01	01:00:00	2369.0
254210	2010-01-01	02:00:00	2247.0
254211	2010-01-01	03:00:00	2128.0
254212	2010-01-01	04:00:00	2105.0
254213	2010-01-01	05:00:00	2135.0
...
8755	2023-12-31	19:00:00	5731.0
8756	2023-12-31	20:00:00	5738.0
8757	2023-12-31	21:00:00	5568.0
8758	2023-12-31	22:00:00	5259.0
8759	2023-12-31	23:00:00	5016.0
122711 rows × 2 columns			

FIGURE 8.1 – Données initiales de consommation électrique

8.1.2 Données de température

Pour enrichir notre modèle, nous avons intégré des données météorologiques provenant de 25 stations différentes. Afin de réduire la redondance et simplifier l'analyse, nous avons étudié la corrélation entre ces stations. En utilisant l'algorithme de clustering K-Means, nous avons regroupé les stations en clusters et sélectionné une station représentative par cluster. Ainsi, nous avons réduit le nombre de stations de 25 à 5, tout en conservant une diversité représentative des conditions météorologiques locales.

La figure 8.2 ci-dessous montre les coefficients de corrélation des températures entre différentes villes du Maroc. Ces corrélations nous permettent de comprendre comment les températures dans une ville sont associées à celles d'autres villes.

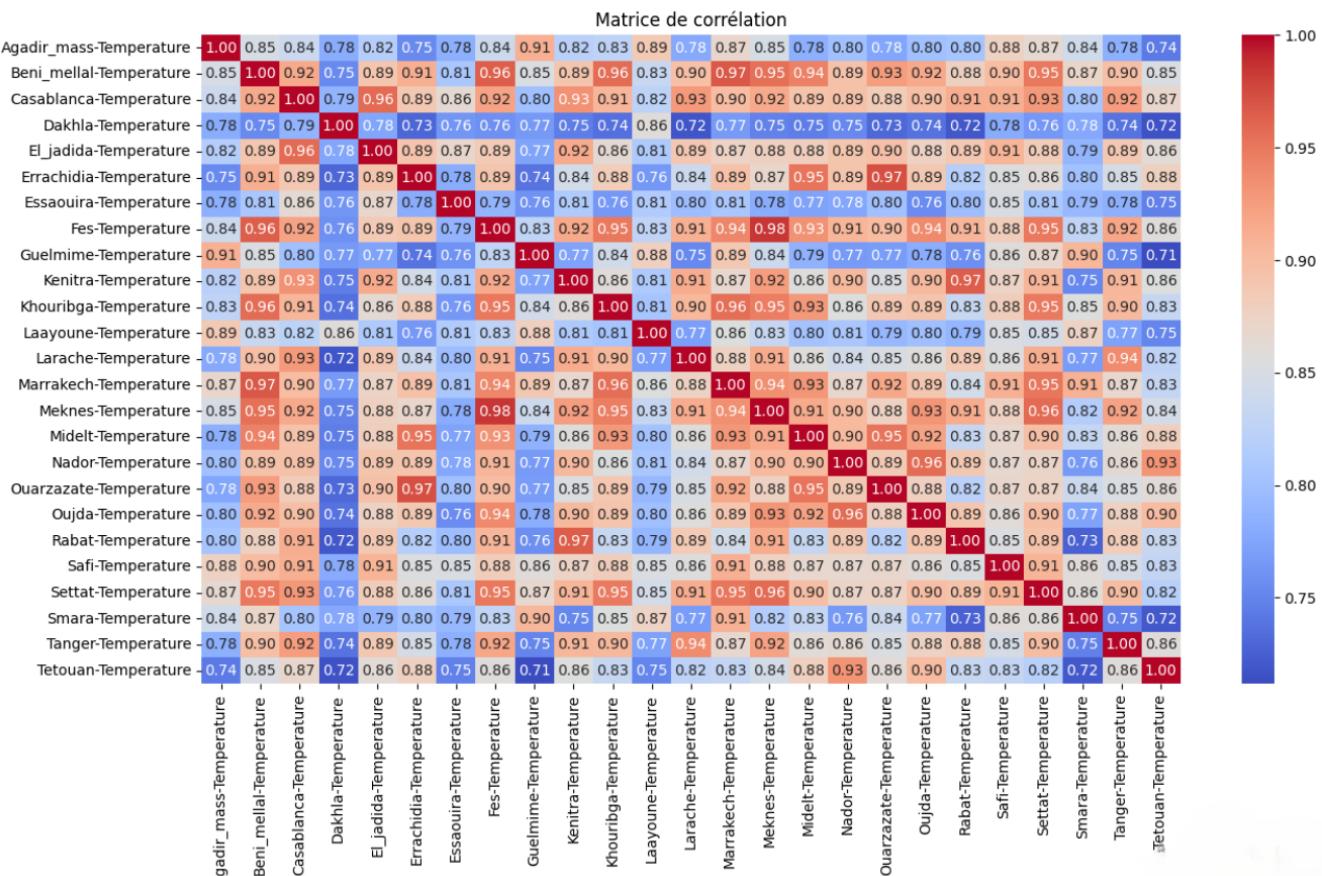


FIGURE 8.2 – Matrice de corrélation entre différentes villes du Maroc

Pour cette analyse, nous avons utilisé l'algorithme de clustering K-means afin de regrouper les stations météorologiques selon leurs températures corrélées. Voici les résultats obtenus et une analyse détaillée de chaque cluster :

```

Cluster 1: ['Errachidia', 'Midelt', 'Nador', 'Ouarzazate', 'Oujda', 'Tetouan']
Cluster 2: ['Beni_mellal', 'Fes', 'Khouribga', 'Marrakech', 'Meknes', 'Safi', 'Settat']
Cluster 3: ['Agadir_mass', 'Guelmim', 'Laayoune', 'Smara']
Cluster 4: ['Dakhla', 'Essaouira']
Cluster 5: ['Casablanca', 'El_jadida', 'Kenitra', 'Larache', 'Rabat', 'Tanger']

```

FIGURE 8.3 – Regroupement des stations météorologiques

Cluster 1

Villes : Errachidia, Midelt, Nador, Ouarzazate, Oujda, Tetouan

Caractéristiques :

- Régions Variées :** Ce cluster comprend des villes situées dans des régions géographiquement diversifiées, y compris le nord-est (Nador, Oujda), le sud-est (Errachidia, Ouarzazate) et le nord (Tetouan).
- Influences Climatiques Mixtes :** Les températures dans ces villes peuvent être influencées par des facteurs géographiques divers, tels que les montagnes de l'Atlas, le désert du Sahara et la proximité de la Méditerranée.

Cluster 2

Villes : Beni Mellal, Fes, Khouribga, Marrakech, Meknes, Safi, Settat

Caractéristiques :

- Centres Urbains et Agricoles :** Ce cluster comprend plusieurs grandes villes du centre du Maroc, connues pour leurs centres urbains et agricoles.
- Climat Tempéré :** Les températures de ces villes sont probablement influencées par un climat méditerranéen tempéré avec des variations saisonnières modérées.

Cluster 3

Villes : Agadir, Guelmim, Laayoune, Smara

Caractéristiques :

- Régions du Sud :** Ces villes sont situées dans le sud du Maroc, avec une influence notable de l'océan Atlantique pour Agadir.
- Climat Désertique et Océanique :** La présence de villes proches de l'océan (Agadir) et des régions désertiques (Guelmim, Laayoune, Smara) suggère une combinaison de climats désertique et océanique.

Cluster 4

Villes : Dakhla, Essaouira

Caractéristiques :

- Villes Côtierres :** Ces deux villes sont situées le long de la côte atlantique.
- Climat Océanique :** Les températures dans ces villes sont fortement influencées par l'océan Atlantique, avec des variations modérées et des températures relativement douces toute l'année.

Cluster 5

Villes : Casablanca, El Jadida, Kenitra, Larache, Rabat, Tanger

Caractéristiques :

- Grandes Villes Côtierres :** Ce cluster comprend les principales villes côtières le long de l'Atlantique, du nord au sud.
- Influence Océanique Prononcée :** Les températures sont modérées par l'océan Atlantique, avec des hivers doux et des étés tempérés, typiques du climat méditerranéen.

Méthodologie de l'entraînement du modèle K-Means

L'entraînement du modèle K-Means pour la classification des stations météorologiques a été réalisé en utilisant exclusivement les données de température de l'année 2023. En raison de la disponibilité limitée des données historiques, seules les données de cette année ont été utilisées pour former le modèle.

Ce choix a été motivé par la nécessité de travailler avec les données disponibles et de fournir une analyse basée sur des informations récentes. Bien que l'utilisation des données d'une seule année puisse présenter certaines limitations en termes de généralisation, elle permet néanmoins de capturer les tendances et les schémas de variation des températures pour cette période spécifique.

Il convient de noter que l'utilisation exclusive des données de 2023 pour l'entraînement du modèle K-Means pourrait limiter sa capacité à généraliser à d'autres années ou à d'autres périodes temporelles. Cependant, dans le cadre de cette étude, cette approche a été jugée adéquate pour fournir une segmentation initiale des stations météorologiques en fonction de leurs caractéristiques thermiques pour l'année 2023.

Vu que les données de consommation d'électricité étaient disponibles sur une période plus longue (2010 - 2023), elles ont pu être utilisées pour d'autres aspects de l'analyse. Cependant, pour les températures, l'utilisation de l'API <https://open-meteo.com/en/docs/historical-weather-api> était nécessaire pour obtenir des données historiques étendues.

Il est important de noter que l'obtention des données de température historiques via l'API n'était pas destinée à l'entraînement

du modèle K-Means, mais plutôt pour une utilisation ultérieure dans d'autres modèles, bien précisément LSTM et GRU. Ces données historiques permettront d'enrichir les analyses futures et de comprendre les tendances à long terme des conditions météorologiques dans différentes régions.

La décision d'utiliser uniquement les données de 2023 pour l'entraînement du modèle K-Means était basée sur la nécessité d'effectuer une segmentation initiale des stations météorologiques en clusters pour cette année spécifique, afin de fournir une base pour des analyses plus approfondies par la suite.

Après avoir effectué la segmentation des stations météorologiques en clusters à l'aide du modèle K-Means, des données de température supplémentaires ont été collectées pour une seule ville représentative de chaque cluster. Les villes sélectionnées pour représenter chaque cluster étaient Casablanca, Agadir, Marrakech, Nador et Dakhla.

Cette approche a été adoptée pour avoir une compréhension plus détaillée des conditions météorologiques dans des zones clés représentatives de chaque cluster. En obtenant les données de température pour ces villes spécifiques, nous avons pu évaluer plus précisément les caractéristiques thermiques et climatiques distinctives de chaque cluster, en tenant compte des variations géographiques et des influences environnementales.

	Temperature_Casa	Temperature_Marrakech	Temperature_Agadir	Temperature_Nador	Temperature_Dakhla
DateTime					
2010-01-01 01:00:00	11.656500	10.467500	12.752	13.198000	17.867000
2010-01-01 02:00:00	11.706500	9.867499	12.302	12.848000	17.367000
2010-01-01 03:00:00	11.556500	9.317500	12.002	12.698000	16.817001
2010-01-01 04:00:00	11.306500	8.517500	11.352	12.648000	16.267000
2010-01-01 05:00:00	11.106501	7.767500	10.802	12.497999	15.717000
...
2023-12-31 19:00:00	13.634500	13.275001	17.800	14.634500	19.472000
2023-12-31 20:00:00	12.834500	12.125000	16.700	14.484500	19.572000
2023-12-31 21:00:00	9.984500	12.325000	15.100	13.584500	19.572000
2023-12-31 22:00:00	11.384500	12.925000	13.650	13.934500	19.522001
2023-12-31 23:00:00	10.484500	11.525001	12.800	13.084500	19.522001

122711 rows × 5 columns

FIGURE 8.4 – Températures collectées de l'API des 5 stations représentatives

8.2 Prétraitement des données

Cette section détaillera les diverses techniques et méthodes utilisées pour préparer les données pour l'analyse.

8.2.1 Feature Engineering

Le *feature engineering* consiste à créer de nouvelles variables (*features*) à partir des données brutes pour améliorer les performances des modèles de machine learning. Voici les principales variables dérivées créées pour cette étude :

- **Month :**
 - **Variable :** Month
 - **Description :** Le mois de l'année, utile pour capturer les variations saisonnières dans la consommation d'électricité.
- **Saison :**
 - **Variable :** Saison
 - **Description :** La saison de l'année (printemps, été, automne, hiver), importante pour modéliser les variations saisonnières de la consommation d'électricité.
- **Day Name :**
 - **Variable :** Day_Name
 - **Description :** Le jour de la semaine (lundi, mardi, etc.). La consommation d'électricité peut varier en fonction du jour de la semaine.
- **Hour :**

- **Variable :** Hour
- **Description :** L'heure de la journée, cruciale pour capturer les cycles quotidiens de la consommation d'électricité.
- **Hour Cosine et Sine :**
 - **Variables :** hour_cos, hour_sin
 - **Description :** Représentation circulaire de l'heure de la journée pour capturer les périodicités cycliques de manière plus efficace dans les modèles.
- **Period of Day :**
 - **Variable :** Period_of_Day
 - **Description :** Périodes de la journée (matin, après-midi, soir, nuit), pour modéliser les variations de consommation en fonction des différentes périodes.
- **Friday Prayer Time :**
 - **Variable :** Friday_Prayer_Time
 - **Description :** Indicateur binaire pour l'heure de la prière du vendredi, pouvant affecter la consommation d'électricité.
- **Is Ramadan :**
 - **Variable :** is_Ramadan
 - **Description :** Indicateur binaire pour les jours pendant le mois de Ramadan. La consommation d'électricité peut être affectée par les changements dans les routines quotidiennes et les heures de repas.
- **Islamic Holiday :**
 - **Variable :** Islamic_Holiday
 - **Description :** Indicateur binaire pour les principales fêtes islamiques (Aïd al-Fitr, Aïd al-Adha, etc.), susceptibles d'influencer la consommation d'électricité.
- **National Holiday :**
 - **Variable :** National_Holiday
 - **Description :** Indicateur binaire pour les jours fériés nationaux, influençant souvent la consommation d'électricité.
- **Temperatures :**
 - **Variables :** Temperature_Casa, Temperature_Agadir, Temperature_Marrakech, Temperature_Dakhla, Temperature_Nador
 - **Description :** Températures enregistrées dans les villes représentatives de chaque cluster, influençant directement la consommation d'électricité.

	Month	Hour	hour_cos	hour_sin	Friday_Prayer_Time	Temperature_Casa	Temperature_Agadir	Temperature_Marrakech	Temperature_Dakhla	Temperature_Nador	Hourly_Value
count	376920.000000	376920.000000	3.769200e+05	3.769200e+05	376920.000000	376920.000000	376920.000000	376920.000000	376920.000000	376920.000000	376920.000000
mean	6.523145	11.500000	-5.55123e-17	-1.850490e-17	0.017861	17.761815	19.785832	18.391231	20.325804	18.604080	2448.188466
std	3.448648	6.922196	7.071077e-01	7.071077e-01	0.132445	6.201054	6.514778	8.564558	3.629085	5.552917	2365.419069
min	1.000000	0.000000	-1.000000e+00	-1.000000e+00	0.000000	-1.993500	1.952000	-4.782500	6.917000	2.048000	0.000000
25%	4.000000	5.750000	-7.071068e-01	-7.071068e-01	0.000000	13.256500	15.252000	12.167500	17.867000	14.297999	1045.000000
50%	7.000000	11.500000	-6.123234e-17	6.123234e-17	0.000000	17.534500	19.100000	17.917501	20.067001	18.198000	1895.000000
75%	10.000000	17.250000	7.071068e-01	7.071068e-01	0.000000	21.534500	23.752000	24.025000	22.467001	22.698000	3598.000000
max	12.000000	23.000000	1.000000e+00	1.000000e+00	1.000000	45.006500	50.750000	47.325000	39.617000	39.684500	65327.000000

FIGURE 8.5 – Statistiques descriptives des variables

Les statistiques descriptives dans la figure 8.5 montrent une distribution des températures et des valeurs horaires. Les températures varient significativement selon les villes, avec des moyennes allant de 17.76°C à Casablanca à 20.33°C à Dakhla. Les valeurs horaires de la consommation d'électricité montrent une distribution large avec une moyenne de 2,448 et un maximum de 65,327, indiquant des pics de consommation importants. Les variables sinus et cosinus de l'heure montrent une distribution symétrique attendue pour des fonctions périodiques. Le temps de prière du vendredi est une variable binaire avec peu de valeurs non nulles.

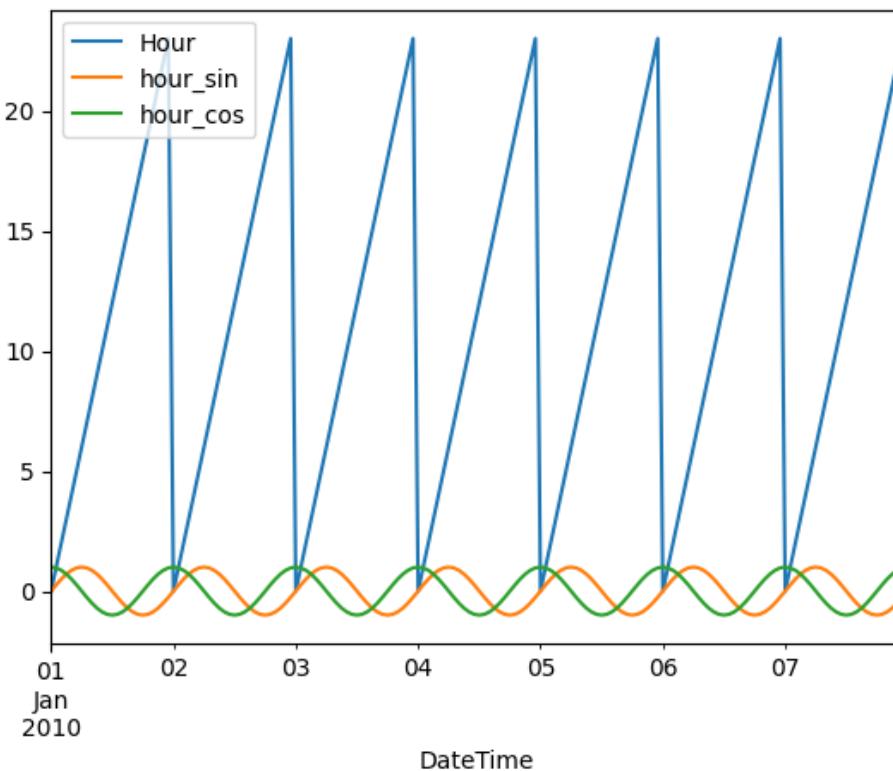


FIGURE 8.6 – Représentation de l'heure de la journée et de ses composantes sinusoïdales (sinus et cosinus) sur une semaine

Cette figure illustre les variations horaires sur une semaine, avec trois courbes distinctes : l'heure de la journée (en bleu), le sinus de l'heure (en orange), et le cosinus de l'heure (en vert). Les courbes sinus et cosinus sont ajoutées pour capturer la nature cyclique des données horaires, ce qui est crucial pour les modèles de série temporelle.

Pourquoi ajouter le cosinus et le sinus de l'heure ?

1. Saisonnalité Capturée par les Fonctions Trigonométriques :

- Les fonctions sinus et cosinus permettent de représenter des cycles périodiques. Étant donné que la consommation électrique suit une tendance quotidienne (24 heures), ces fonctions peuvent efficacement modéliser ces variations cycliques.

2. Éviter la Nature Linéaire des Heures :

- Utiliser l'heure de la journée directement dans les modèles pourrait introduire une fausse tendance linéaire, alors que le comportement réel est cyclique. Les fonctions trigonométriques, en revanche, respectent cette périodicité et éliminent les fausses tendances.

3. Préparation des Données pour les Modèles :

- Les modèles de série temporelle, tels que les modèles de régression linéaire, les réseaux de neurones récurrents (RNN) ou les modèles ARIMA, bénéficient souvent de l'ajout de ces composantes saisonnières explicites pour améliorer la précision des prévisions.

4. Capturer les Schémas Complexes :

- Les schémas de consommation peuvent être influencés par des variations horaires complexes (comme les pics de consommation le matin et le soir). Les composantes sinusoïdales aident à capturer ces variations complexes de manière plus précise.

8.2.2 Nettoyage des données

Pour traiter les valeurs manquantes, nous avons utilisé une méthode d'interpolation linéaire afin de garantir la continuité des données temporelles. Pour combler les valeurs manquantes, nous avons utilisé l'interpolation linéaire, qui consiste à « joindre les points » donnés par des segments de droite. Elle peut servir à estimer les points de la courbe situés entre ceux donnés au

départ. Entre deux points p_1 et p_2 de coordonnées respectives (x_1, y_1) et (x_2, y_2) , l'interpolation est donnée par la formule suivante :

$$y = p \cdot (x - x_1) + y_1$$

avec la pente p qui s'exprime comme

$$p = \frac{y_2 - y_1}{x_2 - x_1}.$$

8.2.3 Traitement des valeurs aberrantes

Dans le cadre de cette étude, nous avons opté pour une approche spécifique concernant les valeurs aberrantes. Les modèles LSTM (Long Short-Term Memory) et GRU (Gated Recurrent Unit) utilisés pour la modélisation sont connus pour leur capacité à gérer efficacement les séries temporelles, même en présence de valeurs aberrantes.

Les valeurs aberrantes sont des points de données qui se distinguent nettement des autres observations et peuvent potentiellement influencer les modèles d'apprentissage de manière négative. Toutefois, grâce à la structure des réseaux de neurones récurrents (RNN) comme LSTM et GRU, ces modèles sont capables de traiter les séquences de données avec des variations importantes sans nécessiter un prétraitement intensif des valeurs aberrantes.

Les LSTM et GRU possèdent des mécanismes internes de gestion de la mémoire à long terme et à court terme, leur permettant de se concentrer sur les tendances et les schémas pertinents tout en atténuant l'impact des fluctuations extrêmes. Ces mécanismes incluent des cellules de mémoire et des portes (gates) qui régulent le flux d'information, filtrant ainsi les informations non pertinentes ou aberrantes.

Par conséquent, nous avons décidé de ne pas appliquer de techniques spécifiques pour identifier ou éliminer les valeurs aberrantes dans notre jeu de données. Cette décision repose sur la robustesse intrinsèque des modèles LSTM et GRU face aux anomalies et sur leur capacité à apprendre des séquences temporelles complexes même en présence de valeurs extrêmes.

Pour résumer, bien que les valeurs aberrantes puissent représenter un défi pour de nombreux modèles de machine learning, les architectures LSTM et GRU utilisées dans cette étude nous permettent de gérer ces anomalies de manière efficace, sans nécessiter d'étapes de prétraitement supplémentaires.

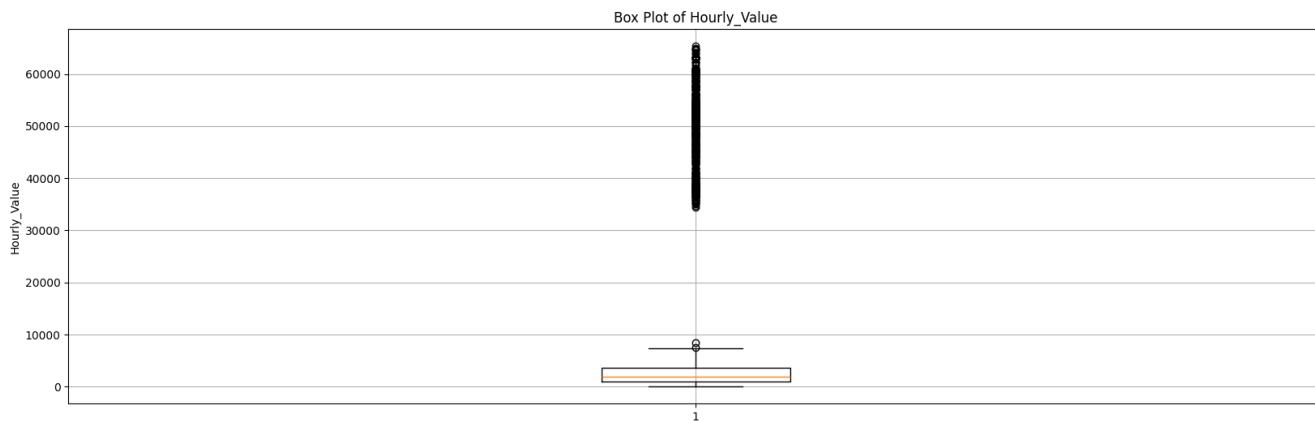


FIGURE 8.7 – Diagramme en boîte (boxplot) des valeurs horaires de la consommation d'électricité

Dans la figure 8.7 ci-dessus, on observe que :

- La ligne médiane orange montre que la moitié des valeurs horaires sont en dessous de cette valeur, indiquant une tendance centrale.
- L'intervalle interquartile (IQR) est relativement petit par rapport à l'échelle totale des valeurs, ce qui indique que la majorité des données se regroupent autour de la médiane.
- Il y a un grand nombre de valeurs aberrantes au-dessus de la boîte et des moustaches, atteignant des valeurs horaires très élevées (jusqu'à environ 60,000). Ces valeurs montrent des pics de consommation d'électricité qui ne suivent pas la tendance générale des données.

8.3 Analyse Exploratoire des Données

8.3.1 Description de la série temporelle

La série chronologique examinée dans cette étude représente la demande d'électricité sur une période de 14 ans, de 2010 à 2023, avec une résolution horaire, ce qui se traduit par un total de 122 712 observations. Cette série offre une vision détaillée de l'évolution de la demande d'électricité au Maroc, permettant une analyse fine des variations saisonnières, des tendances à long terme et des fluctuations intra-journalières. Les données horaires fournissent un aperçu complet des schémas de consommation d'électricité tout au long de la journée, permettant ainsi une évaluation précise des besoins en énergie à différentes échelles de temps. Un aperçu concis de cette série est présenté dans la figure 8.8 suivante :

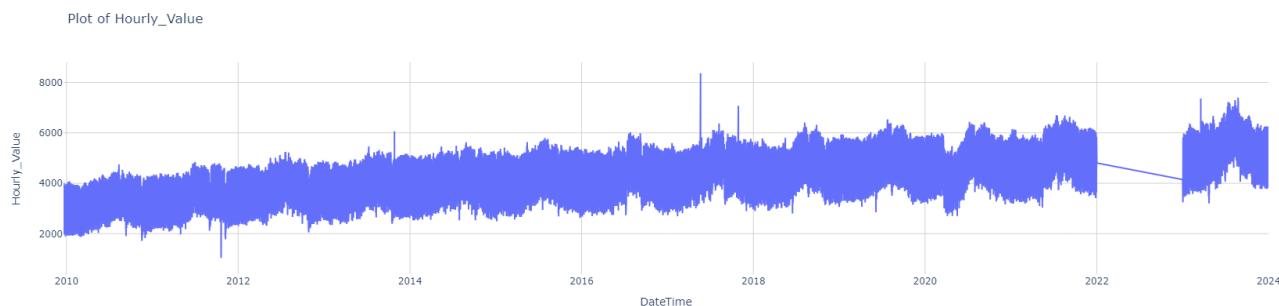


FIGURE 8.8 – Évolution temporelle des données pendant la période donnée

Le graphique dans la figure 8.8 de la série temporelle de la consommation électrique horaire de 2010 à 2024 montre une tendance générale à la hausse, reflétant potentiellement une croissance économique, une augmentation de la population ou une dépendance accrue à l'électricité. Cette période est marquée par des pics significatifs de consommation autour de 2013-2014 et 2016-2020, probablement dus à des événements spécifiques ou à des variations saisonnières. La variabilité croissante au fil des ans suggère également des fluctuations dues à des conditions climatiques extrêmes ou à des changements dans les habitudes de consommation. L'analyse statistique, basée sur 122 712 points de données, confirme la richesse et la granularité de la base de données utilisée pour cette étude.

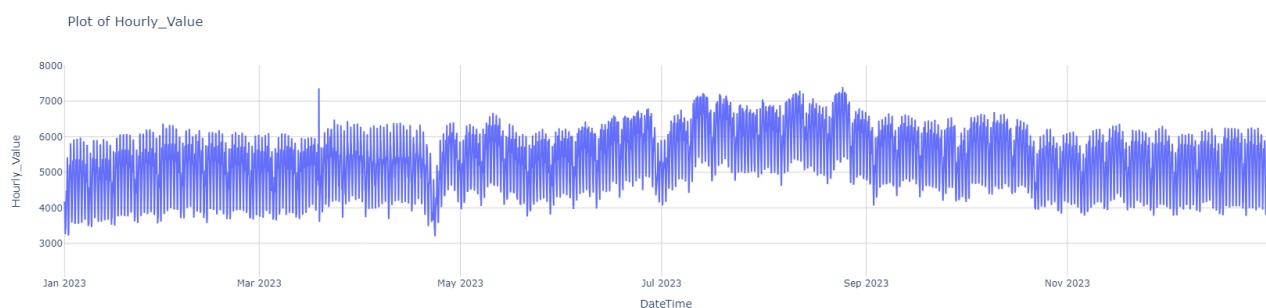


FIGURE 8.9 – Évolution temporelle des données pendant une année

Le graphique dans la figure 8.9 illustre l'évolution temporelle de la consommation électrique horaire tout au long de l'année 2023. Les données montrent une variation diurne marquée, avec des pics récurrents qui semblent correspondre aux heures de pointe quotidiennes, typiquement le matin et le soir.

La consommation augmente progressivement à partir de janvier, atteignant des niveaux plus élevés pendant les mois d'été, notamment en juillet et août. Cette hausse estivale peut être attribuée à une utilisation accrue des systèmes de climatisation en réponse aux températures élevées. En mars, on observe un pic notable, peut-être lié à un événement spécifique ou à une demande accrue pour des raisons inconnues.

De septembre à décembre, la consommation diminue légèrement, mais elle reste plus élevée que les niveaux observés au début de l'année. Cette tendance pourrait refléter une baisse de l'utilisation des climatisations avec la fin de l'été, ainsi qu'une possible augmentation des besoins de chauffage en hiver, bien que moins marquée.

Les variations journalières persistent tout au long de l'année, illustrant les cycles de consommation réguliers. Le graphique démontre également une stabilité relative dans les niveaux de consommation pendant les mois intermédiaires, sans fluctuations extrêmes après l'été.

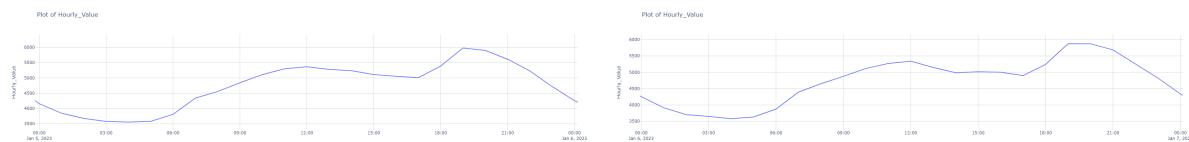
En résumé, la consommation électrique horaire en 2023 présente des variations saisonnières et journalières attendues, avec des périodes de forte demande en été et une diminution progressive vers la fin de l'année.



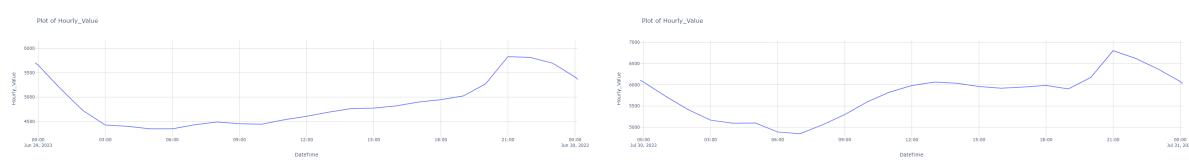
FIGURE 8.10 – Évolution temporelle des données pendant deux semaines

Le graphique présente l'évolution de la consommation électrique horaire sur une période de 15 jours, du 1er janvier 2023 au 15 janvier 2023. On observe des cycles quotidiens bien définis, caractérisés par des pics et des creux réguliers. Chaque jour, la consommation atteint généralement son maximum en deux périodes principales : en fin de matinée et en début de soirée, ce qui correspond probablement aux heures de pointe lorsque les activités domestiques et industrielles sont à leur apogée. Entre ces deux périodes, la consommation diminue, atteignant son minimum pendant la nuit et les premières heures de la matinée.

Au début de chaque jour, il y a une montée progressive de la consommation électrique qui atteint un pic avant de redescendre légèrement et de remonter à nouveau pour un deuxième pic en début de soirée. Cette variation suit un schéma répétitif avec des fluctuations mineures d'un jour à l'autre, suggérant une routine quotidienne constante.



(a) Évolution temporelle de la consommation électrique pendant une journée normale (b) Évolution temporelle de la consommation électrique pendant une journée de Vendredi



(c) Évolution temporelle de la consommation électrique pendant une journée d'une fête islamique (Eid Al-Adha) (d) Évolution temporelle de la consommation électrique pendant une journée d'une fête nationale (Fête du trône)

FIGURE 8.11 – Évolution temporelle de la consommation électrique pendant une journée pour différents types de jour

La figure 8.11 présente l'évolution temporelle de la consommation électrique pendant une journée pour différents types de jour.

- La première sous-figure 8.11a montre l'évolution de la consommation électrique pendant une journée normale. La courbe présente une variation typique avec des creux et des pics correspondant aux périodes de faible et de forte demande.
- La deuxième sous-figure 8.11b illustre la consommation électrique pendant une journée de Vendredi. On observe une tendance similaire à celle d'une journée normale, mais avec des variations potentielles dues aux spécificités de cette journée (les prières du vendredi).

- La troisième sous-figure 8.11c représente la consommation pendant une journée de fête islamique, en l'occurrence l'Eid Al-Adha. La courbe montre des variations distinctes, reflétant les comportements particuliers de consommation lors de cette fête religieuse.
- La quatrième sous-figure 8.11d présente la consommation électrique pendant une journée de fête nationale, ici la Fête du trône. Cette courbe affiche également des motifs distincts, suggérant une influence des célébrations nationales sur la demande en électricité.

L'ensemble de ces sous-figures permet de comparer les différentes tendances de consommation électrique selon le type de jour, mettant en évidence les fluctuations et les particularités propres à chaque contexte.

8.3.2 Décomposition de la série temporelle

La décomposition STL (Seasonal-Trend Decomposition using Loess) est une méthode robuste et flexible pour décomposer une série temporelle en trois composants principaux : tendance, saisonnalité et résidus. La méthode utilise des régressions locales (LOESS) pour séparer la série temporelle en ces composants, ce qui permet d'identifier et de visualiser les variations à long terme (tendance), les fluctuations périodiques (saisonnalité) et les bruits ou anomalies (résidus). La décomposition STL est particulièrement utile pour analyser des séries temporelles non stationnaires avec des comportements saisonniers complexes.



FIGURE 8.12 – Décomposition STL de la série temporelle de la consommation électrique

La figure 8.12 montre la décomposition de la série temporelle de la consommation électrique en quatre composants principaux : la série observée (Hourly_Value), la tendance (Trend), la saisonnalité (Season), et les résidus (Resid).

1. **Hourly_Value (Série observée)** : Cette série montre les valeurs horaires de la consommation électrique de 2010 à 2023. On observe une tendance générale à la hausse avec des variations saisonnières et quelques anomalies ponctuelles.

2. **Trend (Tendance)** : Ce composant montre la tendance à long terme de la consommation électrique. On peut voir une augmentation progressive de la consommation électrique au fil du temps, avec une diminution notable autour de l'année 2022.
3. **Season (Saisonnalité)** : Ce composant montre les variations saisonnières répétitives de la série temporelle. Il capture les motifs récurrents qui se produisent dans la série temporelle, tels que les fluctuations hebdomadaires et saisonnières. Les variations saisonnières semblent relativement constantes au fil des années, mais il y a des changements visibles après 2021.
4. **Resid (Résidus)** : Ce composant représente les résidus ou les variations aléatoires qui ne sont pas expliquées par la tendance ou la saisonnalité. Les résidus sont généralement centrés autour de zéro, mais quelques valeurs anormalement élevées ou basses apparaissent, indiquant des événements inhabituels ou des anomalies dans les données.

Cette décomposition STL permet de mieux comprendre les différentes composantes de la série temporelle de la consommation électrique, en séparant les effets de tendance, de saisonnalité et d'anomalies aléatoires.

8.3.3 Test de stationnarité (Test de Dickey-Fuller Augmenté)

```
Test Statistic: -6.833653769198311
P-value: 1.8661083955917185e-09
Critical Values:
 1%: -3.4304074235158244
 5%: -2.861565380328433
 10%: -2.5667835090547686
```

FIGURE 8.13 – Test ADF de la série du consommation électrique

Les résultats du test de Dickey-Fuller augmenté dans la figure 8.13 montrent que la série temporelle de la consommation électrique est stationnaire. La statistique de test est significativement plus basse que les valeurs critiques à tous les niveaux de signification, et la p-value est extrêmement faible, ce qui fournit une forte évidence pour rejeter l'hypothèse nulle de non-stationnarité. Cela signifie que la moyenne et la variance de la série temporelle restent constantes dans le temps, ce qui est une condition essentielle pour de nombreux modèles de prévision des séries temporelles.

8.3.4 Fonction d'autocorrélation (ACF) et d'autocorrélation partielle (PACF)

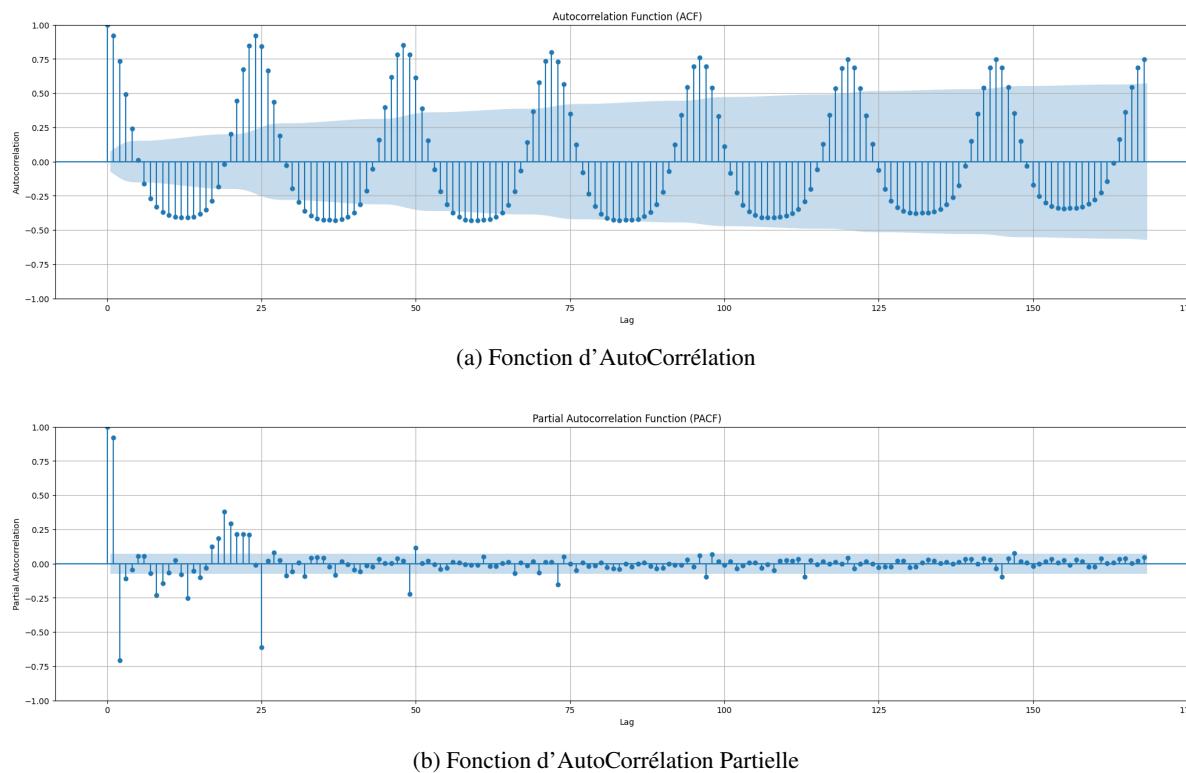


FIGURE 8.14 – ACF / PACF

La figure attachée présente les graphiques de la fonction d'autocorrélation (ACF) et de la fonction d'autocorrélation partielle (PACF) pour les 720 premières heures de la série temporelle de consommation électrique, avec un décalage (lag) maximum de 168 périodes. Voici la description de chaque graphique :

Graphique ACF (Autocorrelation Function)

1. Périodicité et Saison :

- Les pics d'autocorrélation positifs à des lags multiples de 24 (environ 24, 48, 72, etc.) indiquent une forte composante saisonnière quotidienne, ce qui est logique pour une série temporelle de consommation électrique où la demande fluctue de manière répétitive chaque jour.
- Les pics d'autocorrélation diminuent en amplitude à mesure que le décalage augmente, ce qui est attendu pour les données de consommation qui tendent à être plus similaires sur de courtes périodes.

2. Autocorrélation significative :

- Les barres dépassant la zone bleue (intervalle de confiance) indiquent une autocorrélation significative. Les lags à environ 24 heures montrent la plus forte autocorrélation, suggérant que les valeurs de consommation à une heure donnée sont fortement corrélées avec les valeurs à la même heure du jour précédent.

Graphique PACF (Partial Autocorrelation Function)

1. Déterminer l'ordre AR :

- Les premiers lags (1, 2, 3, ...) montrent une autocorrélation partielle significative. Cela suggère que les valeurs immédiates précédentes influencent directement la valeur actuelle.
- L'autocorrélation partielle devient non significative après quelques lags initiaux, ce qui indique qu'après avoir pris en compte l'effet des premiers lags, il ne reste pas beaucoup de corrélation à expliquer par les lags suivants.

2. Interprétation de la coupure :

- Le premier lag (1) montre la plus haute valeur de l'autocorrélation partielle, ce qui est typique des séries temporelles où les valeurs passées proches ont un fort impact sur les valeurs actuelles.

- La significativité des lags diminuant rapidement après les premiers lags suggère un modèle AR (AutoRegressive) de faible ordre pourrait bien s'ajuster aux données.

Ces graphiques révèlent des comportements importants de la série temporelle de consommation électrique. La forte autocorrélation aux lags multiples de 24 heures dans l'ACF confirme la présence d'une saisonnalité quotidienne, tandis que le PACF indique que l'influence des valeurs passées se limite principalement aux quelques lags immédiats.

8.4 Implémentation des modèles

8.4.1 Encodage des données

Il est important de se rappeler que l'encodage des données est une étape importante du prétraitement des données et peut avoir un impact significatif sur les performances d'un modèle. Pour cela il est crucial de choisir la technique d'encodage appropriée pour les données spécifiques. Pour notre cas d'étude, nous avons utilisé LabelEncoder pour l'encodage des données catégorielles (Saison, Day_Name, Period_of_Day, Islamic_Holiday, National_Holiday) puisque :

- LabelEncoder est une méthode d'encodage simple et facile à comprendre.
- LabelEncoder est une méthode d'encodage computationnellement efficace, particulièrement adaptée aux ensembles de données volumineux comme dans le cas de notre étude.
- Il convertit les labels catégoriels en représentations numériques de manière rapide et efficiente, ce qui peut être crucial pour les modèles d'apprentissage.
- LabelEncoder est compatible avec une large gamme de modèles d'apprentissage automatique et d'outils d'analyse statistique.

DateTi...	Month	Saison	Day_N...	Hour	hour ...	hour_sin	Period...	Friday...	is Ra...	Islam...	Natio...	Temp...	Temp...	Temp...	Temp...	Temp...	Hour...
Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
2010-01-...	1	1	0	0	1	0	3	0	0	4	0	11.9565	12.852	10.5175	18.217001	13.098	2664
2010-01-...	1	1	0	1	0.965925...	0.258819...	3	0	0	4	0	11.6565	12.752	10.4675	17.867	13.198	2369
2010-01-...	1	1	0	2	0.866025...	0.5	3	0	0	4	0	11.7065	12.302	9.867499	17.367	12.848	2247
2010-01-...	1	1	0	3	0.707106...	0.707106...	3	0	0	4	0	11.5565	12.002	9.3175	16.817001	12.698	2128
2010-01-...	1	1	0	4	0.5	0.866025...	3	0	0	4	0	11.3065	11.352	8.5175	16.267	12.648	2105
2010-01-...	1	1	0	5	0.258819...	0.965925...	3	0	0	4	0	11.106501	10.802	7.7675004	15.717	12.497999	2135
2010-01-...	1	1	0	6	6.123233...	1	2	0	0	4	0	10.9065	10.252	7.0175004	15.266999	12.198	2199
2010-01-...	1	1	0	7	-0.25881...	0.965925...	2	0	0	4	0	11.3065	11.202	7.9675	14.816999	12.497999	2166
2010-01-...	1	1	0	8	-0.5	0.866025...	2	0	0	4	0	11.4065	11.202	7.9175	14.316999	12.547999	2346
2010-01-...	1	1	0	9	-0.70710...	0.707106...	2	0	0	4	0	12.356501	12.202	9.9175	14.717	13.348	2650
2010-01-...	1	1	0	10	-0.86602...	0.5	2	0	0	4	0	14.356501	14.402	11.5675	16.367	13.948	2902
2010-01-...	1	1	0	11	-0.96592...	0.258819...	2	0	0	4	0	15.4565	15.252	12.8175	18.567001	14.497999	2940
2010-01-...	1	1	0	12	-1	1.224646...	0	1	0	4	0	16.0565	16.001999	13.7675	20.867	14.648	2896
2010-01-...	1	1	0	13	-0.96592...	-0.25881...	0	1	0	4	0	16.456501	16.552	14.5175	22.867	15.098	2785
2010-01-...	1	1	0	14	-0.86602...	-0.5	0	1	0	4	0	16.3565	17.102	15.1675	24.267	15.247999	2815
2010-01-...	1	1	0	15	-0.70710...	-0.70710...	0	0	0	4	0	16.206501	17.452	15.5675	24.967001	15.247999	2778
2010-01-...	1	1	0	16	-0.5	-0.86602...	0	0	0	4	0	16.1065	17.352	15.2675	25.367	15.098	2779
2010-01-...	1	1	0	17	-0.25881...	-0.96592...	0	0	0	4	0	15.5065	16.802	14.1675	25.367	14.747999	3148
2010-01-...	1	1	0	18	-1.83697...	-1	1	0	0	4	0	14.0065	15.702	12.2675	24.617	14.497999	3696
2010-01-...	1	1	0	19	0.258819...	-0.96592...	1	0	0	4	0	12.8065	14.902	11.5675	23.117	14.448	3587

FIGURE 8.15 – Extrait de jeu de données encodé

8.4.2 Division des données

```
data_train, data_val, data_test = filtered_data.loc[(filtered_data.index >= '2010-01-01') & (filtered_data.index <= '2018-12-31')], \
| filtered_data.loc[(filtered_data.index >= '2019-01-01') & (filtered_data.index <= '2020-12-31')], \
| filtered_data.loc[filtered_data.index >= '2021-01-01']
```

FIGURE 8.16 – Divion des données (Entraînement / Validation / Test)

Le code ci-dessus crée trois DataFrames distincts :

- data_train : Ce DataFrame contient des lignes où la date de l'index est comprise entre '2010-01-01' (inclusive) et '2018-12-31' (inclusive). Cela représente les données d'entraînement pour nos modèles.
- data_val : Ce DataFrame contient des lignes où la date de l'index est comprise entre '2019-01-01' (inclusive) et '2020-12-31' (inclusive). Cela représente les données de validation pour nos modèles.

- `data_test` : Ce DataFrame contient des lignes dont la date d'index est à partir de '2021-01-01' (inclus). Cela représente les données de test pour nos modèles.

8.4.3 Normalisation des données

La normalisation est une étape essentielle du prétraitement des données pour les modèles d'apprentissage automatique, et il s'agit d'une technique de mise à l'échelle des caractéristiques. La normalisation est particulièrement cruciale pour la manipulation des données, la réduction ou l'augmentation de l'étendue des données avant qu'elles ne soient utilisées pour les étapes suivantes dans le processus de réalisation du modèle. La mise à l'échelle min-max et la normalisation Z-Score (standardisation) sont les deux méthodes les plus fréquemment utilisées pour la normalisation dans le cadre de la mise à l'échelle des caractéristiques.

L'objectif de la normalisation est de transformer les caractéristiques sur une échelle similaire. Cela permet d'éviter que des valeurs d'entrée extrêmes n'aient un impact disproportionné sur le processus d'apprentissage du modèle et ne conduisent à une convergence difficile ou à une mauvaise performance.

Nous avons utilisé, comme illustré dans le code de la figure 8.17, le `MinMaxScaler` comme technique de mise à l'échelle qui normalise les données entre une plage de -1 à 1 pour aligner les valeurs d'entrée avec la plage de sortie de la fonction d'activation `tanh` utilisée dans les modèles LSTM et GRU.

```
values = data_train.values

scaler = MinMaxScaler(feature_range=(-1, 1))
scaled_train = scaler.fit_transform(values)
scaled_test = scaler.transform(data_test)
scaled_val = scaler.transform(data_val)
```

FIGURE 8.17 – Code Python de mise à l'échelle

Le code ci-dessus produit des données mises à l'échelle entre une plage de -1 et 1, comme illustré dans la figure 8.18 suivante :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
0	-1.0	-0.333333	-1.0	-1.000000	1.000000e+00	0.000000e+00	1.000000	-1.0	-1.0	1.0	-1.0	-0.495657	-0.567749	-0.435619	-0.445783	-0.419178	-0.555737
1	-1.0	-0.333333	-1.0	-0.913043	9.659258e-01	2.588190e-01	1.000000	-1.0	-1.0	1.0	-1.0	-0.509667	-0.572228	-0.437649	-0.469880	-0.413699	-0.636240
2	-1.0	-0.333333	-1.0	-0.826087	8.660254e-01	5.000000e-01	1.000000	-1.0	-1.0	1.0	-1.0	-0.507332	-0.592385	-0.462011	-0.504303	-0.432877	-0.669532
3	-1.0	-0.333333	-1.0	-0.739130	7.071068e-01	1.000000	-1.0	-1.0	1.0	-1.0	-0.514336	-0.605823	-0.484342	-0.542169	-0.441096	-0.702006	
4	-1.0	-0.333333	-1.0	-0.652174	5.000000e-01	8.660254e-01	1.000000	-1.0	-1.0	1.0	-1.0	-0.526011	-0.634938	-0.516825	-0.580034	-0.443836	-0.708282
5	-1.0	-0.333333	-1.0	-0.565217	2.588190e-01	9.659258e-01	1.000000	-1.0	-1.0	1.0	-1.0	-0.535351	-0.659574	-0.547277	-0.617900	-0.452055	-0.700096
6	-1.0	-0.333333	-1.0	-0.478261	6.123234e-17	1.000000e+00	0.333333	-1.0	-1.0	1.0	-1.0	-0.544690	-0.684211	-0.577729	-0.648881	-0.468493	-0.682631
7	-1.0	-0.333333	-1.0	-0.391304	2.588190e-01	9.659258e-01	0.333333	-1.0	-1.0	1.0	-1.0	-0.526011	-0.641657	-0.539156	-0.679862	-0.452055	-0.691636
8	-1.0	-0.333333	-1.0	-0.304348	5.000000e-01	8.660254e-01	0.333333	-1.0	-1.0	1.0	-1.0	-0.521341	-0.614657	-0.541187	-0.714286	-0.449315	-0.642516
9	-1.0	-0.333333	-1.0	-0.217391	7.071068e-01	1.000000	0.333333	-1.0	-1.0	1.0	-1.0	-0.476978	-0.596864	-0.459981	-0.686747	-0.405479	-0.559558
10	-1.0	-0.333333	-1.0	-0.130435	8.660254e-01	5.000000e-01	0.333333	-1.0	-1.0	1.0	-1.0	-0.383581	-0.498320	-0.392986	-0.573150	-0.372603	-0.490790
11	-1.0	-0.333333	-1.0	-0.043478	9.659258e-01	2.588190e-01	0.333333	-1.0	-1.0	1.0	-1.0	-0.332213	-0.460246	-0.342232	-0.421687	-0.342466	-0.480420
12	-1.0	-0.333333	-1.0	0.043478	-1.000000e+00	1.224647e-16	-1.000000	1.0	-1.0	1.0	-1.0	-0.304194	-0.426652	-0.303659	-0.263339	-0.334247	-0.492427
13	-1.0	-0.333333	-1.0	0.130435	9.659258e-01	2.588190e-01	-1.000000	1.0	-1.0	1.0	-1.0	-0.285514	-0.402016	-0.273207	-0.125645	-0.309589	-0.522718
14	-1.0	-0.333333	-1.0	0.217391	8.660254e-01	-5.000000e-01	-1.000000	1.0	-1.0	1.0	-1.0	-0.290184	-0.377380	-0.246815	-0.029260	-0.301370	-0.514531
15	-1.0	-0.333333	-1.0	0.304348	7.071068e-01	-1.000000	-1.0	-1.0	1.0	-1.0	-0.297189	-0.361702	-0.230574	0.018933	-0.301370	-0.524628	
16	-1.0	-0.333333	-1.0	0.391304	-5.000000e-01	8.660254e-01	-1.000000	1.0	-1.0	1.0	-1.0	-0.301859	-0.366181	-0.242755	0.046472	-0.309589	-0.524355
17	-1.0	-0.333333	-1.0	0.478261	-2.588190e-01	9.659258e-01	-1.000000	1.0	-1.0	1.0	-1.0	-0.329878	-0.390817	-0.287418	0.046472	-0.328767	-0.423659
18	-1.0	-0.333333	-1.0	0.565217	-1.836970e-16	-1.000000e+00	-0.333333	-1.0	-1.0	1.0	-1.0	-0.399925	-0.440090	-0.364564	-0.005164	-0.342466	-0.274117
19	-1.0	-0.333333	-1.0	0.652174	2.588190e-01	-9.659258e-01	-0.333333	-1.0	-1.0	1.0	-1.0	-0.455963	-0.475924	-0.392986	-0.108434	-0.345205	-0.303861

FIGURE 8.18 – Extrait des données d'entraînement mises à l'échelle

8.4.4 Modélisation

Comment transformer une série temporelle en un problème d'apprentissage supervisé

La prévision des séries temporelles peut être définie comme un problème d'apprentissage supervisé. Ce redimensionnement de nos données nous permet d'accéder à la suite d'algorithmes d'apprentissage automatique linéaires et non linéaires standard pour notre problème.

Fenêtre glissante Les données de séries temporelles peuvent être formulées comme un problème d'apprentissage supervisé. Étant donné une séquence de nombres pour un ensemble de données de séries temporelles, nous pouvons restructurer les données pour qu'elles ressemblent à un problème d'apprentissage supervisé. Nous pouvons le faire en utilisant les étapes de temps précédentes comme variables d'entrée et en utilisant l'étape de temps suivante comme variable de sortie. Rendons cela concret avec un exemple. Imaginons que nous ayons une série temporelle comme suit :

time	measure
1,	100
2,	110
3,	108
4,	115
5,	120

FIGURE 8.19 – Exemple d'un petit ensemble de données de séries temporelles fictif

Nous pouvons restructurer cet ensemble de données de séries temporelles en un problème d'apprentissage supervisé en utilisant la valeur de l'étape temporelle précédente pour prédire la valeur de l'étape temporelle suivante. En réorganisant l'ensemble de données de séries temporelles de cette manière, les données ressembleraient à ceci :

x	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

FIGURE 8.20 – Exemple d'un ensemble de données de séries temporelles comme apprentissage supervisé

- Nous pouvons voir que l'étape temporelle précédente est l'entrée (X) et l'étape temporelle suivante est la sortie (y) dans notre problème d'apprentissage supervisé.
- Nous pouvons voir que l'ordre entre les observations est préservé et doit continuer à être préservé lors de l'utilisation de cet ensemble de données pour entraîner un modèle supervisé.
- Nous pouvons constater qu'il n'y a pas de valeur précédente que nous puissions utiliser pour prédire la première valeur de la séquence. Nous supprimerons cette ligne car nous ne pouvons pas l'utiliser.
- Nous pouvons également voir qu'il n'y a pas de valeur suivante connue à prédire pour la dernière valeur de la séquence. Nous voudrons peut-être aussi supprimer cette valeur lors de l'entraînement de notre modèle supervisé.

L'utilisation des étapes temporelles précédentes pour prédire l'étape temporelle suivante est appelée la méthode de la *fenêtre glissante*. En abrégé, elle peut être appelée méthode de la fenêtre dans certaines littératures. En statistiques et en analyse de séries temporelles, cela est appelé un décalage ou méthode de décalage (lag). Le nombre d'étapes temporelles précédentes est appelé la largeur de la fenêtre ou la taille du décalage. Cette fenêtre glissante est la base permettant de transformer tout ensemble de données de séries temporelles en un problème d'apprentissage supervisé. À partir de cet exemple simple, nous pouvons remarquer quelques points :

- Nous pouvons voir comment cela peut fonctionner pour transformer une série temporelle en un problème d'apprentissage supervisé de régression ou de classification pour des valeurs de séries temporelles réelles ou étiquetées.
- Nous pouvons voir comment, une fois qu'un ensemble de données de séries temporelles est préparé de cette manière, n'importe quel algorithme standard d'apprentissage automatique linéaire ou non linéaire peut être appliqué, tant que l'ordre des lignes est préservé.
- Nous pouvons voir comment la largeur de la fenêtre glissante peut être augmentée pour inclure davantage d'étapes temporelles précédentes.
- Nous pouvons voir comment l'approche de la fenêtre glissante peut être utilisée sur une série temporelle comportant plus d'une valeur, ou une série temporelle dite multivariée.

Fenêtre glissante avec variables multiples Le nombre d'observations enregistrées pour un moment donné dans un ensemble de données de séries temporelles est important. Traditionnellement, différents noms sont utilisés :

- **Série temporelle univariée** : Il s'agit d'ensembles de données où une seule variable est observée à chaque instant, comme la température chaque heure. L'exemple de la section précédente est un ensemble de données de séries temporelles univariées.

- **Série temporelle multivariée** : Il s'agit d'ensembles de données où deux variables ou plus sont observées à chaque instant.

La plupart des méthodes d'analyse de séries temporelles, se concentrent sur les données univariées. Cela est dû au fait qu'elles sont les plus simples à comprendre et à manipuler. Les données multivariées sont souvent plus difficiles à traiter. Elles sont plus complexes à modéliser et souvent, de nombreuses méthodes classiques ne donnent pas de bons résultats.

Le point fort de l'utilisation de l'apprentissage automatique pour les séries temporelles est là où les méthodes classiques échouent. Cela peut se produire avec des séries temporelles univariées complexes et est plus probable avec des séries temporelles multivariées en raison de leur complexité supplémentaire. Voici un autre exemple concret pour illustrer la méthode de la fenêtre glissante pour les séries temporelles multivariées. Supposons que nous ayons l'ensemble de données fictif de séries temporelles multivariées ci-dessous avec deux observations à chaque étape temporelle. Supposons également que nous ne nous intéressons qu'à la prédiction de la mesure2.

time	measure1	measure2
1,	0.2,	88
2,	0.5,	89
3,	0.7,	87
4,	0.4,	88
5,	1.0,	90

FIGURE 8.21 – Exemple d'un petit ensemble de données fictif de séries temporelles multivariées

Nous pouvons reformuler cet ensemble de données de séries temporelles comme un problème d'apprentissage supervisé avec une largeur de fenêtre de un. Cela signifie que nous utiliserons les valeurs des étapes temporales précédentes de Measure1 et Measure2. Nous aurons également la valeur de l'étape temporelle suivante pour Measure1 disponible. Ensuite, nous prédirons la valeur de l'étape temporelle suivante de Measure2. Cela nous donnera 3 caractéristiques d'entrée et une valeur de sortie à prédire pour chaque modèle d'entraînement.

X1	X2	X3	y
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

FIGURE 8.22 – Exemple d'ensemble de données de séries temporelles multivariées transformé en un problème d'apprentissage supervisé

Nous pouvons voir qu'à l'instar de l'exemple de série temporelle univariée ci-dessus, nous devrons peut-être supprimer les premières et dernières lignes afin de former notre modèle d'apprentissage supervisé. Cet exemple soulève la question de savoir ce que nous ferions si nous voulions prédire à la fois measure1 et measure2 pour l'étape temporelle suivante ? L'approche de la fenêtre glissante peut également être utilisée dans ce cas. En utilisant le même ensemble de données de séries temporelles ci-dessus, nous pouvons le formuler comme un problème d'apprentissage supervisé où nous prédisons à la fois measure1 et measure2 avec la même largeur de fenêtre de un, comme suit.

X1	X2	y1	y2
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

FIGURE 8.23 – Exemple d'ensemble de données de séries temporelles multivariées formulé comme un problème d'apprentissage supervisé de prédiction multi-étape ou de séquence

En effet, peu de méthodes d'apprentissage supervisé peuvent gérer la prédiction de plusieurs valeurs de sortie sans modification, mais certaines méthodes, comme les réseaux neuronaux artificiels, n'ont généralement pas de difficulté à le faire. Nous pouvons considérer la prédiction de plus d'une valeur comme la prédiction d'une séquence. Dans ce cas, nous prédisons deux variables de sortie différentes, mais nous pourrions vouloir prédire plusieurs étapes temporelles à l'avance pour une seule variable de sortie. Cela s'appelle la prévision multi-étape.

Fenêtre glissante avec plusieurs étapes Le nombre d'étapes temporelles à prévoir est important. Encore une fois, il est traditionnel d'utiliser différents noms pour le problème en fonction du nombre d'étapes temporelles à prévoir :

- Prévision à une étape : C'est là où la prochaine étape temporelle ($t + 1$) est prédite.
- Prévision multi-étapes : C'est là où deux étapes temporelles ou plus doivent être prédites.

Tous les exemples que nous avons examinés jusqu'à présent ont été des prévisions à une étape. Il existe plusieurs façons de modéliser la prévision multi-étapes en tant que problème d'apprentissage supervisé. Pour l'instant, nous nous concentrerons sur le cadrage de la prévision multi-étapes en utilisant la méthode de la fenêtre glissante. Considérons le même ensemble de données de séries temporelles univariées de l'exemple de la première fenêtre glissante ci-dessus dans la figure 8.19. Nous pouvons formuler cette série temporelle comme un ensemble de données de prévision à deux étapes pour l'apprentissage supervisé avec une largeur de fenêtre de un, comme suit :

X1	y1	y2
?	100	110
100	110	108
110	108	115
108	115	120
115	120	?
120	?	?

FIGURE 8.24 – Exemple d'ensemble de données de séries temporelles univariées formulé comme un problème d'apprentissage supervisé de prédiction multi-étapes ou de séquence

Nous pouvons voir que la première ligne et les deux dernières lignes ne peuvent pas être utilisées pour entraîner un modèle supervisé. C'est également un bon exemple pour illustrer la charge sur les variables d'entrée. En particulier, un modèle supervisé dispose uniquement de $X1$ pour prédire à la fois $y1$ et $y2$. Une réflexion approfondie et des expérimentations sont nécessaires sur notre problème pour trouver une largeur de fenêtre qui donne des performances de modèle acceptables.

L'analyse de la fonction d'autocorrélation (ACF) et de la fonction d'autocorrélation partielle (PACF) est une méthode couramment utilisée pour déterminer les décalages temporels significatifs dans une série temporelle. Cela nous permet de comprendre quelles sont les dépendances temporelles importantes entre les observations passées et les observations actuelles. En identifiant les pas de temps précédents qui ont un impact sur le pas de temps actuel, nous pouvons choisir une largeur de fenêtre appropriée pour notre modèle de prédiction, ce qui contribue à améliorer ses performances. C'est une démarche essentielle dans la modélisation des séries temporelles afin de construire un modèle robuste et précis.

```

#splitamultivariatesequenceintosamples
Codumate: Options | Test this function
def split_sequences(sequences,n_steps_in,n_steps_out):
    X, y= list(), list()
    for i in range(len(sequences)):
        #find the end of this pattern
        end_ix = i + n_steps_in
        #out_end_ix = end_ix + n_steps_out-1
        out_start_y = end_ix + n_steps_out
        #check if we are beyond the dataset
        if out_start_y > len(sequences):
            break
        #gather input and output parts of the pattern
        seq_x, seq_y=sequences[i:end_ix, :], sequences[end_ix:(end_ix + n_steps_out), :]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

# choose a number of time steps
n_steps_in, n_steps_out = 168, 168
#n_steps_in, n_steps_out = 10, 5
# covert into input/output
X_train, y_train = split_sequences(scaled_train, n_steps_in, n_steps_out)
X_test, y_test = split_sequences(scaled_test, n_steps_in, n_steps_out)
X_val, y_val = split_sequences(scaled_val, n_steps_in, n_steps_out)

print(f"X_train[-1]: {X_train[-1]}")
print(f"y_train[-1]: {y_train[-1]}")
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
print(X_val.shape)
print(y_val.shape)

```

FIGURE 8.25 – Code de conversion des données en un problème d'apprentissage supervisé

Ce code définit une fonction nommée `split_sequences` qui prend une séquence temporelle multivariée en entrée et la divise en échantillons pour l'apprentissage supervisé. Chaque échantillon est composé d'un nombre défini d'étapes temporelles en entrée (`n_steps_in`) et un nombre correspondant d'étapes en sortie (`n_steps_out`). La fonction parcourt chaque élément de la séquence et crée des échantillons en extrayant les séquences d'entrée et de sortie. Si la taille des échantillons dépasse la longueur de la séquence, la fonction arrête le processus. Les échantillons résultants sont ensuite utilisés pour créer les ensembles d'entraînement, de test et de validation, chacun étant séparé selon les besoins spécifiés par les valeurs `n_steps_in` et `n_steps_out`.

Si on jette un coup d'œil sur le résultat de ce bout de code dans la figure 8.26, on voit que les données brutes ont été bien transformées sous forme des séquences d'où les dimensions affichées présentent (*batch, timesteps, features*).

```

X_train[-1]: [[ 1.           -0.33333333 -0.66666667 ... -0.40413081 -0.26923288
   -0.33769955]
[ 1.           -0.33333333 -0.66666667 ... -0.37314967 -0.28567123
   -0.38272616]
[ 1.           -0.33333333 -0.66666667 ... -0.36626492 -0.28567123
   -0.39282303]
...
[ 1.           -0.33333333  0.           ... -0.30430286 -0.63909589
   0.00218311]
[ 1.           -0.33333333  0.           ... -0.32151456 -0.62539726
   -0.12907627]
[ 1.           -0.33333333 -0.66666667 ... -0.31118761 -0.6089589
   -0.24491745]]
y_train[-1]: [[ 1.0000000e+00 -3.3333333e-01 -6.6666667e-01 ... -3.18072289e-01
   -6.41835616e-01 -3.24873789e-01]
[ 1.0000000e+00 -3.3333333e-01 -6.6666667e-01 ... -3.45611015e-01
   -6.74712329e-01 -3.60349297e-01]
[ 1.0000000e+00 -3.3333333e-01 -6.6666667e-01 ... -3.80034423e-01
   -6.93890411e-01 -3.71810615e-01]
...
[ 1.0000000e+00 -3.3333333e-01  0.0000000e+00 ... -4.52323442e-01
   -3.78821918e-01  5.45777050e-04]
[ 1.0000000e+00 -3.3333333e-01  0.0000000e+00 ... -4.55765921e-01
   -3.9800000e-01 -1.71373994e-01]
[ 1.0000000e+00 -3.3333333e-01 -6.6666667e-01 ... -4.55765921e-01
   ...
(17185, 168, 17)
(17185, 168, 17)
(17186, 168, 17)
(17186, 168, 17)

```

FIGURE 8.26 – Les données transformées sous forme de séquences

8.4.5 LSTM

Architecture

L'architecture du modèle LSTM dans la figure 8.27 développé pour la prédiction de la consommation électrique se compose de plusieurs couches structurées pour optimiser les performances de prédiction. Le modèle commence par une couche LSTM avec 512 unités et une activation `tanh`, conçue pour capturer les dépendances temporelles complexes dans les données d'entrée, avec une forme d'entrée déterminée par `n_steps_in` et `X_train.shape[2]`. Une couche `Dropout` avec un taux de 20% suit cette couche, ce qui aide à prévenir le surapprentissage en éteignant aléatoirement 20% des neurones durant l'entraînement. Ensuite, une couche `RepeatVector` est utilisée pour répéter la représentation en sortie de la première couche LSTM sur plusieurs pas de temps, correspondant à `n_steps_out`.

```

opt = Adam(0.0001, clipnorm=1.)

model=Sequential()
model.add(LSTM(512,activation='tanh',input_shape=(n_steps_in,X_train.shape[2])))
#model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(256,activation='tanh',return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(128, activation='tanh', return_sequences=True))
model.add(Dropout(0.2))
model.add(TimeDistributed(Dense(X_train.shape[2])))
model.compile(optimizer=opt, loss='mse')

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# fit model
history = model.fit(X_train, y_train, epochs=100, verbose=1, validation_data=(X_val, y_val), callbacks=[early_stopping])

loss = model.evaluate(X_test, y_test)

# Access training history
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

# Plot training and validation loss
epochs = range(1, len(training_loss) + 1)
plt.plot(epochs, training_loss, 'b', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

(a) Définition du modèle LSTM

Model: "sequential"		
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 512)	1085440
dropout (Dropout)	(None, 512)	0
repeat_vector (RepeatVector)	(None, 168, 512)	0
lstm_1 (LSTM)	(None, 168, 256)	787456
dropout_1 (Dropout)	(None, 168, 256)	0
lstm_2 (LSTM)	(None, 168, 128)	197120
dropout_2 (Dropout)	(None, 168, 128)	0
time_distributed (TimeDistributed)	(None, 168, 17)	2193
<hr/>		
Total params: 2072209 (7.90 MB)		
Trainable params: 2072209 (7.90 MB)		
Non-trainable params: 0 (0.00 Byte)		

(b) Sommaire du modèle LSTM

FIGURE 8.27 – Architecture du modèle LSTM

La structure comprend ensuite deux couches LSTM supplémentaires, avec 256 et 128 unités respectivement, chacune utilisant une activation `tanh` et configurées pour retourner des séquences. Ces couches sont également suivies de couches `Dropout` avec le même taux de 20%, ajoutant une robustesse supplémentaire contre le surapprentissage. Enfin, une couche `TimeDistributed` avec une couche `Dense` interne est utilisée pour appliquer une transformation dense à chaque pas de temps de la séquence, égalant le nombre de caractéristiques d'entrée (`X_train.shape[2]`). L'optimiseur Adam avec un taux d'apprentissage de 0.0001 et une normalisation des gradients (`clipnorm=1`) est employé pour la compilation du modèle, en utilisant la perte quadratique moyenne (`mse`) comme fonction de perte.

Pour améliorer la performance et éviter le surapprentissage, nous avons utilisé le mécanisme d'*early stopping*. Le `callback EarlyStopping` surveille la perte de validation (`val_loss`) et arrête l'entraînement si cette perte ne diminue plus après 10 époques consécutives, en restaurant les poids du modèle correspondant à la meilleure performance de validation. Le modèle a été entraîné pour un maximum de 100 époques, mais le processus d'*early stopping* a permis de réduire ce nombre en arrêtant l'entraînement dès que le modèle a cessé de s'améliorer sur les données de validation.

Apprentissage

La courbe d'apprentissage dans la figure 8.28 montre l'évolution des pertes d'entraînement et de validation au cours des différentes époques.

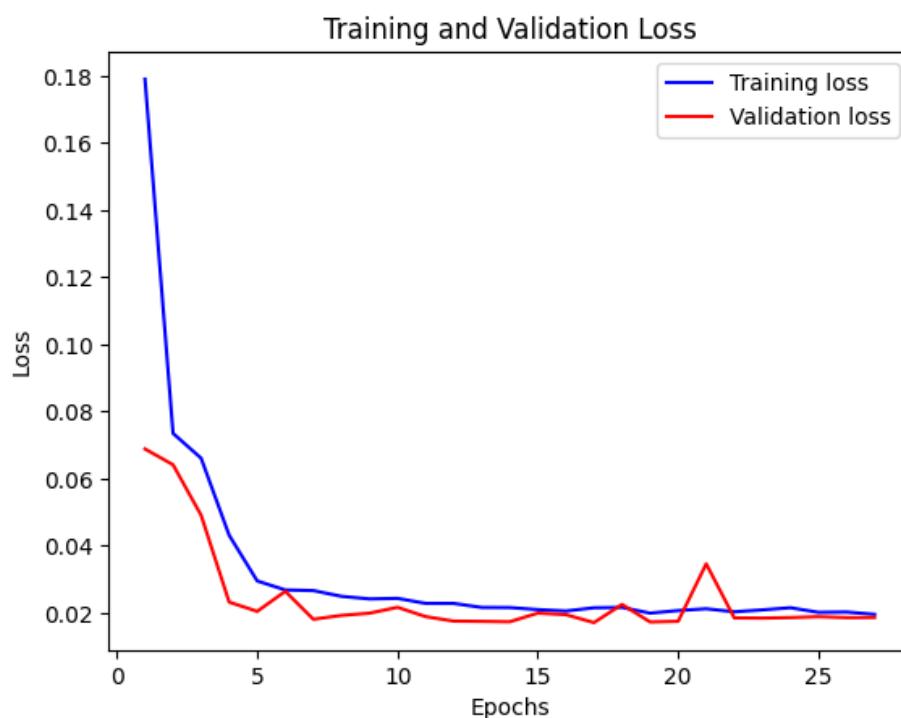


FIGURE 8.28 – Courbe d'apprentissage du modèle LSTM

On observe dans la figure ci-dessus que la perte d'entraînement commence à une valeur élevée (0.18), ce qui est attendu au début de l'entraînement.

La perte diminue rapidement au cours des premières époques, indiquant que le modèle apprend rapidement des données.

Après environ 10 à 15 époques, la perte d'entraînement se stabilise autour de 0.02, suggérant que le modèle atteint un point où il ne peut plus apprendre de nouvelles informations significatives sans surajustement.

La perte de validation diminue également rapidement au début, ce qui est un bon signe car cela montre que le modèle généralise bien aux données de validation.

Après environ 10 époques, la perte de validation se stabilise autour de 0.02 avec quelques fluctuations légères. Les fluctuations peuvent indiquer des variations dans les mini-lots de validation, mais elles restent minimes et globalement la perte de validation reste basse.

Interprétation

- **Pas de Surapprentissage** : Les courbes de perte d'entraînement et de validation sont proches et suivent une trajectoire similaire, ce qui est un bon signe. Cela indique que le modèle ne surapprend pas les données d'entraînement et généralise bien aux données de validation.
- **Convergence** : Les pertes se stabilisent rapidement, ce qui indique que le modèle converge bien et de manière efficace.
- **Performance Globale** : Une perte autour de 0.02 sur les données de validation montre que le modèle est performant et prédit bien les valeurs de consommation électrique.

8.4.6 GRU

Architecture

```

Click to add a breakpoint import GRU

opt = Adam(0.0001, clipnorm=1.)

model = Sequential()
model.add(GRU(512, activation='tanh', input_shape=(n_steps_in, X_train.shape[2])))
# model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(RepeatVector(n_steps_out))
model.add(GRU(256, activation='tanh', return_sequences=True))
model.add(Dropout(0.2))
model.add(GRU(128, activation='tanh', return_sequences=True))
model.add(Dropout(0.2))
model.add(TimeDistributed(Dense(X_train.shape[2])))
model.compile(optimizer=opt, loss='mse')

early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Fit model
history = model.fit(X_train, y_train, epochs=100, verbose=1, validation_data=(X_val, y_val), callbacks=[early_stopping])

loss = model.evaluate(X_test, y_test)

# Access training history
training_loss = history.history['loss']
validation_loss = history.history['val_loss']

# Plot training and validation loss
epochs = range(1, len(training_loss) + 1)
plt.plot(epochs, training_loss, 'b', label='Training loss')
plt.plot(epochs, validation_loss, 'r', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

(a) Définition du modèle GRU

Model: "sequential_2"		
Layer (type)	Output Shape	Param #
gru (GRU)	(None, 512)	815616
dropout_6 (Dropout)	(None, 512)	0
repeat_vector_2 (RepeatVector)	(None, 168, 512)	0
gru_1 (GRU)	(None, 168, 256)	591360
dropout_7 (Dropout)	(None, 168, 256)	0
gru_2 (GRU)	(None, 168, 128)	148224
dropout_8 (Dropout)	(None, 168, 128)	0
time_distributed_2 (TimeDistributed)	(None, 168, 17)	2193

Total params: 1557393 (5.94 MB)
Trainable params: 1557393 (5.94 MB)
Non-trainable params: 0 (0.00 Byte)

(b) Sommaire du modèle GRU

FIGURE 8.29 – Architecture du modèle GRU

Le modèle GRU (*Gated Recurrent Unit*) développé pour cette étude de cas présente une architecture séquentielle composée de plusieurs couches de traitement. Voici une description détaillée de chaque couche du modèle :

- **Première couche GRU** : Une couche GRU avec 512 unités et une fonction d'activation \tanh . Cette couche traite

les séquences d'entrée avec une dimension de pas temporels `n_steps_in` et le nombre de caractéristiques par pas `X_train.shape[2]`.

- **Dropout** : Une couche de régularisation Dropout avec un taux de 20%, utilisée pour prévenir le surapprentissage en désactivant aléatoirement une fraction des unités pendant l'entraînement.
- **RepeatVector** : Cette couche répète l'entrée afin de préparer les données pour la couche GRU suivante en sortie.
- **Deuxième couche GRU** : Une deuxième couche GRU avec 256 unités et une fonction d'activation `tanh`, configurée pour retourner des séquences complètes.
- **Dropout** : Une autre couche Dropout avec un taux de 20%, similaire à la précédente pour continuer à régulariser le modèle.
- **Troisième couche GRU** : Une troisième couche GRU avec 128 unités et une fonction d'activation `tanh`, également configurée pour retourner des séquences.
- **Dropout** : Une troisième couche Dropout avec un taux de 20%.
- **TimeDistributed** : Une couche `TimeDistributed` enveloppant une couche `Dense` qui applique une transformation linéaire à chaque pas temporel des séquences de sortie, avec une dimension de sortie égale au nombre de caractéristiques d'entrée `X_train.shape[2]`.

Le modèle est compilé avec l'optimiseur Adam utilisant un taux d'apprentissage de 0.0001 et une norme de coupure (`clipnorm`) de 1. Le critère de perte utilisé est l'erreur quadratique moyenne (`mse`).

Une technique d'arrêt anticipé (*early stopping*) est appliquée pour surveiller la perte de validation (`val_loss`) avec une patience de 10 époques. Cette technique permet de restaurer les meilleurs poids du modèle lorsque la performance sur les données de validation ne s'améliore plus.

Le modèle est entraîné pendant un maximum de 100 époques, avec une évaluation continue sur un jeu de validation pour prévenir le surapprentissage.

Apprentissage

Les courbes de perte d'entraînement et de validation sont tracées dans la figure 8.30 pour visualiser la convergence du modèle.

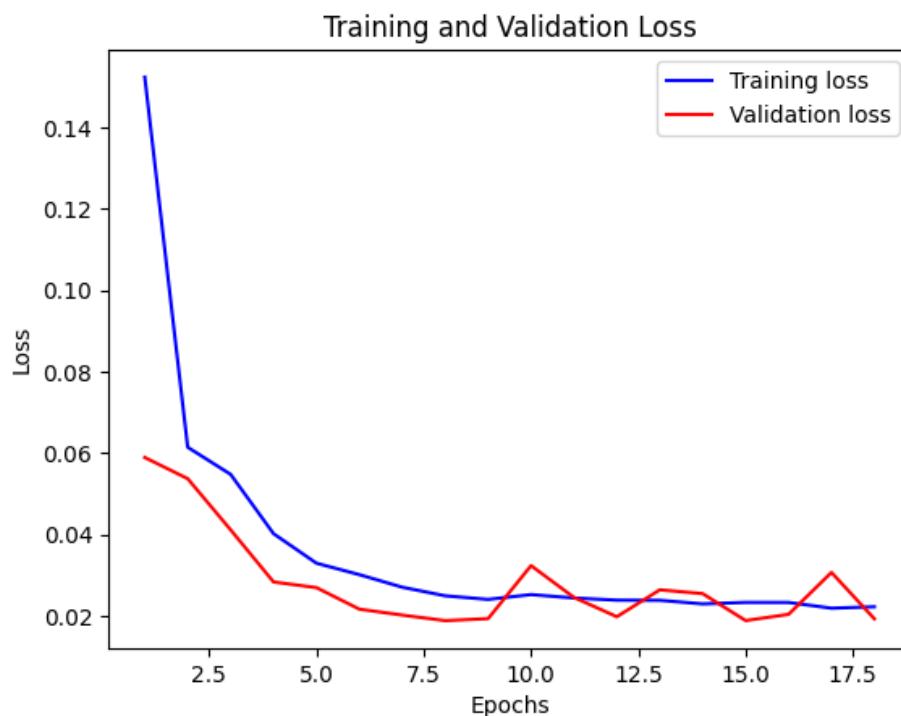


FIGURE 8.30 – Courbe d'apprentissage du modèle

On observe dans la figure ci-dessus que la perte d'entraînement commence à une valeur élevée (0.15), ce qui est attendu au début de l'entraînement. La perte diminue rapidement au cours des premières époques, indiquant que le modèle apprend

rapidement des données. Après environ 10 à 15 époques, la perte d'entraînement se stabilise autour de 0.02, suggérant que le modèle atteint un point où il ne peut plus apprendre de nouvelles informations significatives sans surajustement.

La perte de validation diminue également rapidement au début, ce qui est un bon signe car cela montre que le modèle généralise bien aux données de validation. Après environ 10 époques, la perte de validation se stabilise autour de 0.02 avec quelques fluctuations légères. Les fluctuations peuvent indiquer des variations dans les mini-lots de validation, mais elles restent minimes et globalement la perte de validation reste basse.

Interprétation

- **Pas de Surapprentissage** : Les courbes de perte d'entraînement et de validation sont proches et suivent une trajectoire similaire, ce qui est un bon signe. Cela indique que le modèle ne surapprend pas les données d'entraînement et généralise bien aux données de validation.
- **Convergence** : Les pertes se stabilisent rapidement, ce qui indique que le modèle converge bien et de manière efficace.
- **Performance Globale** : Une perte autour de 0.02 sur les données de validation montre que le modèle est performant et prédit bien les valeurs de consommation électrique.

8.5 Évaluation des performances

Dans cette section, nous évaluons les performances des modèles LSTM et GRU en utilisant les métriques MSE (Mean Squared Error), RMSE (Root Mean Squared Error), et MAE (Mean Absolute Error). Les prédictions des modèles sont comparées aux valeurs réelles pour déterminer la précision et la robustesse de chaque modèle.

8.5.1 Performance du Modèle LSTM

Le modèle LSTM a été évalué en utilisant l'ensemble de test. Les prédictions ont été comparées aux valeurs réelles après une transformation inverse pour obtenir les valeurs dans leur échelle d'origine. La figure 8.31 montre le code utilisé pour l'évaluation et les résultats obtenus.

```
1/1 [=====] - 1s 1s/step
MSE: 118961.85142444039
RMSE: 344.9084681831404
MAE: 283.4573000953311
```

FIGURE 8.31 – Évaluation des performances du modèle LSTM

Les métriques de performance pour le modèle LSTM sont les suivantes :

- **MSE** : 118961.85142444309
- **RMSE** : 344.9084681381044
- **MAE** : 283.4573000953111

Ces résultats indiquent que le modèle LSTM est relativement précis dans ses prédictions de la consommation électrique, bien que certaines erreurs persistent, comme le montrent les valeurs de RMSE et MAE.

8.5.2 Performance du Modèle GRU

De manière similaire, le modèle GRU a été évalué en utilisant les mêmes métriques. La figure 8.32 présente le code utilisé pour l'évaluation et les résultats obtenus.

```
1/1 [=====] - 1s 1s/step
MSE: 83988.00956453526
RMSE: 289.8068487191689
MAE: 225.57440185546875
```

FIGURE 8.32 – Évaluation des performances du modèle LSTM

Les métriques de performance pour le modèle GRU sont les suivantes :

- **MSE** : 83988.02956453326

- **RMSE** : 289.808347191689
- **MAE** : 225.57440185546875

Ces résultats montrent que le modèle GRU offre une meilleure performance avec des erreurs plus faibles par rapport au modèle LSTM, comme l'indiquent les valeurs de MSE, RMSE et MAE.

8.5.3 Comparaison des Modèles

En comparant les métriques de performance des deux modèles, il apparaît que le modèle GRU surpassé le modèle LSTM dans la prédiction de la consommation électrique. Le modèle GRU a une MSE, une RMSE et une MAE plus basses, ce qui indique une meilleure précision globale.

Ces résultats peuvent être influencés par la complexité des modèles et les spécificités des données utilisées pour l'entraînement et la validation. Les deux modèles montrent une capacité à apprendre et à généraliser les tendances des données, mais le GRU semble mieux adapté pour cette tâche spécifique de prédiction de la consommation électrique.

8.6 Application

Une application web a été développée pour la prévision de la consommation électrique. Cette application utilise un modèle de réseau de neurones récurrents (RNN) de type GRU (Gated Recurrent Unit) pour réaliser des prévisions basées sur des données historiques.

L'application a été développée en utilisant le framework Flask pour la partie backend et diverses bibliothèques pour le frontend et le traitement des données.

8.6.1 Fonctionnalités

L'application offre les fonctionnalités suivantes :

- **Téléchargement de fichiers Excel** : L'utilisateur peut téléverser un fichier Excel contenant les données historiques de consommation électrique.
- **Prétraitement des données** : Les données sont prétraitées pour ajouter des informations contextuelles telles que les fêtes islamiques, les jours fériés nationaux, et les périodes de Ramadan.
- **Prévision de la consommation** : Le modèle GRU utilise les données prétraitées pour prévoir la consommation électrique sur une période donnée.
- **Visualisation des résultats** : Les prévisions sont affichées sous forme de graphique interactif permettant de visualiser les tendances et les valeurs prévues.
- **Téléchargement des prévisions** : L'utilisateur peut télécharger les prévisions sous forme de fichier Excel.

8.6.2 Processus de prévision

Le processus de prévision se déroule en plusieurs étapes :

1. **Téléversement des données** : L'utilisateur téléverse un fichier Excel contenant les données historiques de consommation.

Prévision de consommation électrique

Soumettre fichier Excel

Choose file: testflask.xlsx

(a) Téléversement des données historiques de consommation électrique

datetime	Consommation
2024-05-27 00:00:00	5412
2024-05-27 01:00:00	5413
2024-05-27 02:00:00	5414
2024-05-27 03:00:00	5415
2024-05-27 04:00:00	5416
2024-05-27 05:00:00	5417
2024-05-27 06:00:00	5418
2024-05-27 07:00:00	5419
2024-05-27 08:00:00	5420
2024-05-27 09:00:00	5421
2024-05-27 10:00:00	5422
2024-05-27 11:00:00	5423
2024-05-27 12:00:00	5424
2024-05-27 13:00:00	5425
2024-05-27 14:00:00	5426
2024-05-27 15:00:00	5427
2024-05-27 16:00:00	5428
2024-05-27 17:00:00	5429
2024-05-27 18:00:00	5430
2024-05-27 19:00:00	5431

(b) Extrait du fichier de consommation téléversé

FIGURE 8.33 – Téléversement des données historiques de consommation électrique

2. **Prétraitement** : Les données sont transformées pour inclure des variables additionnelles et sont normalisées.

```
@app.route('/upload', methods=['POST'])
def upload():
    file = request.files['file']
    if file:
        df = pd.read_excel(file)
        df.rename(columns={'datetime': 'DateTime', 'Consommation': 'Hourly_Value'}, inplace=True)
        df['Hijri_Date'] = df['DateTime'].apply(lambda x: gregorian_to_hijri(x))
        df['Islamic_Holiday'] = df['DateTime'].apply(lambda x: is_islamic_holiday(x))
        df['National_Holiday'] = df['DateTime'].apply(lambda x: is_national_holiday(x))
        df['is_Ramadan'] = df['DateTime'].apply(lambda x: is_ramadan(x))
        df['Day_Name'] = df['DateTime'].dt.day_name()
        df['Month'] = df['DateTime'].dt.month
        df['Hour'] = df['DateTime'].dt.hour
        df['hour_sin'] = np.sin(2 * np.pi * df['Hour']/24.0)
        df['hour_cos'] = np.cos(2 * np.pi * df['Hour']/24.0)
        df['DayOfWeek'] = df['DateTime'].dt.dayofweek
        df['Type_de_Jour'] = df['Day_Name'].apply(lambda x: map_day_to_type(x))
        df['Saison'] = df['Month'].map(saisons)
        df['Period_of_Day'] = df['Hour'].apply(map_hour_to_period)
        df['Friday_Prayer_Time'] = df.apply(lambda row: is_friday_prayer_time(row['Hour'], row['Day_Name']), axis=1)
        start_date = pd.to_datetime(df['DateTime'].iloc[0], format='%Y-%m-%d %H:%M:%S').strftime('%Y-%m-%d')
        end_date = pd.to_datetime(df['DateTime'].iloc[-1], format='%Y-%m-%d %H:%M:%S').strftime('%Y-%m-%d')
        end_datetime = pd.to_datetime(df['DateTime'].iloc[-1], format='%Y-%m-%d %H:%M:%S')
        print(start_date)
        print(end_date)
        df_weather = get_weather_data(start_date, end_date)
        df_weather = handle_nan_values(df_weather)
        df = pd.merge(df, df_weather, left_on='DateTime', right_on='date', how='left')
        df = df.tail(168)
        df = encode_dataframe(df)
        df = df[['Month', 'Saison', 'Day_Name', 'Hour', 'hour_cos', 'hour_sin', 'Period_of_Day', 'Friday_Prayer_Time', 'is_Ramadan', 'Islamic_Holiday', 'National_Holiday', 'Temperature_Casa', 'Temperature_Agadir', 'Temperature_Marrakech', 'Temperature_Dakhla', 'Temperature_Nador', 'Hourly_Value']]
        values = df.values
        scaler = joblib.load('scaler.joblib')
        scaled_data = scaler.transform(values)
```

FIGURE 8.34 – Code du prétraitement des données

3. **Prévision** : Les données prétraitées sont passées dans le modèle GRU pour générer les prévisions.

```

model = load_model('model8.h5')
y_pred_scaled = model.predict(sequenced_data)
y_pred_scaled_reshaped = y_pred_scaled.reshape(y_pred_scaled.shape[0] * y_pred_scaled.shape[1], y_pred_scaled.shape[2])
y_pred_original = scaler.inverse_transform(y_pred_scaled_reshaped)
y_pred_original_values = y_pred_original[:, -1]
df['Predicted_Consumption'] = y_pred_original_values

new_datetimes = pd.date_range(start=end_datetime + pd.Timedelta(hours=1), periods=168, freq='H')
df['DateTime'] = new_datetimes

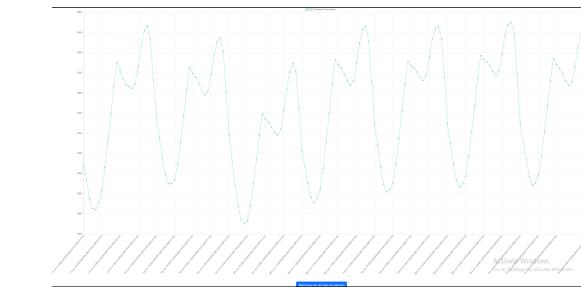
df = df[['DateTime', 'Predicted_Consumption']]
df = pd.DataFrame(df)

df['DateTime'] = df['DateTime'].dt.strftime('%Y-%m-%d %H:%M:%S')
result_json = df.to_json(orient='records')
return jsonify(result_json)

```

FIGURE 8.35 – Code de génération des prévisions de la consommation électrique

4. **Affichage et téléchargement** : Les résultats sont affichés sous forme de graphique et peuvent être téléchargés.



(a) Graphe des prévisions

A	B
DateTime	Predicted_Consumption
2024-06-14 00:00:00	4088.125732
2024-06-14 01:00:00	3936.411621
2024-06-14 02:00:00	3749.909668
2024-06-14 03:00:00	3651.793701
2024-06-14 04:00:00	3638.743408
2024-06-14 05:00:00	3710.218994
2024-06-14 06:00:00	3832.481689
2024-06-14 07:00:00	4059.239014
2024-06-14 08:00:00	4320.979004
2024-06-14 09:00:00	4591.919434
2024-06-14 10:00:00	4868.910156
2024-06-14 11:00:00	5101.619141
2024-06-14 12:00:00	5024.375488
2024-06-14 13:00:00	4944.327148
2024-06-14 14:00:00	4880.908203
2024-06-14 15:00:00	4864.712402
2024-06-14 16:00:00	4843.109375
2024-06-14 17:00:00	4891.693848
2024-06-14 18:00:00	5061.851074
2024-06-14 19:00:00	5271.031738
2024-06-14 20:00:00	5418.158203

(b) Les prévisions en format Excel

FIGURE 8.36 – Résultats de prévision

Cette application fournit un outil puissant et intuitif pour prévoir la consommation électrique, en combinant des techniques avancées de machine learning avec une interface utilisateur conviviale. Elle permet aux utilisateurs de prendre des décisions informées basées sur des prévisions précises et visualisées.

Résumé Le modèle GRU, avec une structure plus simplifiée mais efficace, a montré une performance supérieure par rapport au modèle LSTM. Toutefois, le modèle LSTM reste une alternative robuste et pourrait être préférée dans des scénarios où des dépendances temporelles plus complexes doivent être capturées.

Conclusion et Perspectives

Dans ce projet, nous avons développé et évalué plusieurs modèles de réseaux de neurones récurrents (RNN) pour la prédiction de la consommation électrique. Nous avons utilisé des architectures LSTM (Long Short-Term Memory) et GRU (Gated Recurrent Unit), chacune optimisée et entraînée sur un ensemble de données historiques. Les performances de ces modèles ont été comparées en utilisant des métriques telles que la MSE (Mean Squared Error), la RMSE (Root Mean Squared Error) et la MAE (Mean Absolute Error).

Les résultats obtenus montrent que les deux architectures, LSTM et GRU, sont capables de capturer efficacement les dépendances temporelles dans les données de consommation électrique, avec des performances similaires. Cependant, le modèle GRU a montré une légère supériorité en termes de précision de prédiction et de rapidité d'entraînement, ce qui en fait une option attrayante pour ce type d'application.

Malgré les résultats prometteurs obtenus, plusieurs améliorations peuvent être envisagées pour accroître encore les performances des modèles :

- **Augmentation des Données** : L'utilisation de jeux de données plus volumineux et diversifiés pourrait améliorer la robustesse et la généralisation des modèles.
- **Optimisation des Hyperparamètres** : Une recherche plus approfondie des hyperparamètres via des techniques comme la recherche en grille (Grid Search) ou la recherche bayésienne pourrait conduire à de meilleures configurations de modèles.
- **Incorporation de Caractéristiques Externes** : L'intégration de données supplémentaires telles que les conditions météorologiques, les jours fériés et les prix de l'électricité pourrait améliorer la précision des prédictions.

En plus des améliorations des modèles actuels, l'exploration de nouvelles architectures de réseaux de neurones pourrait offrir des gains supplémentaires :

- **Transformers** : Les modèles basés sur les Transformers, connus pour leurs performances exceptionnelles dans le traitement du langage naturel, pourraient être adaptés pour la prédiction de séries temporelles.
- **Modèles Hybrides** : La combinaison de RNN avec des modèles de régression classiques ou des réseaux de neurones convolutifs (CNN) pourrait capturer des aspects différents des données.

Enfin, le déploiement des modèles dans un environnement de production présente plusieurs défis et opportunités :

- **Déploiement en Temps Réel** : Le développement d'un système de prédiction en temps réel intégré dans des infrastructures existantes pour une gestion optimale de la consommation énergétique.
- **Applications Pratiques** : L'application de ces modèles dans des secteurs spécifiques comme les smart grids, la gestion de l'énergie domestique, et la planification des ressources énergétiques.
- **Feedback et Adaptation** : L'incorporation de mécanismes de feedback pour l'adaptation continue des modèles aux nouvelles données et aux changements de comportement des consommateurs.

En conclusion, bien que les résultats actuels soient encourageants, le domaine de la prédiction de la consommation énergétique offre de nombreuses pistes de recherche et de développement. Les travaux futurs porteront sur l'amélioration continue des modèles, l'exploration de nouvelles approches et le déploiement pratique des solutions développées.

Glossaire

ARIMA AutoRegressive Integrated Moving Average, est un modèle statistique utilisé pour l'analyse et la prévision des séries temporelles. Il combine trois composants principaux : l'autorégression (AR), l'intégration (I) et la moyenne mobile (MA).

Auto-corrélation mesure la corrélation entre les valeurs d'une série temporelle à différents moments. Pour un décalage h , elle est définie par :

$$\rho_h = \frac{\text{Cov}(Y_t, Y_{t+h})}{\sqrt{\text{Var}(Y_t) \cdot \text{Var}(Y_{t+h})}}$$

Bruit blanc est une réalisation d'un processus aléatoire dans lequel la densité spectrale de puissance est la même pour toutes les fréquences de la bande passante.

Espérance également connue sous le nom de moyenne ou valeur attendue, est une mesure de la tendance centrale d'une série temporelle. Elle est définie par :

$$E(Y_t) = \mu$$

Pour une série de T observations, l'espérance empirique est :

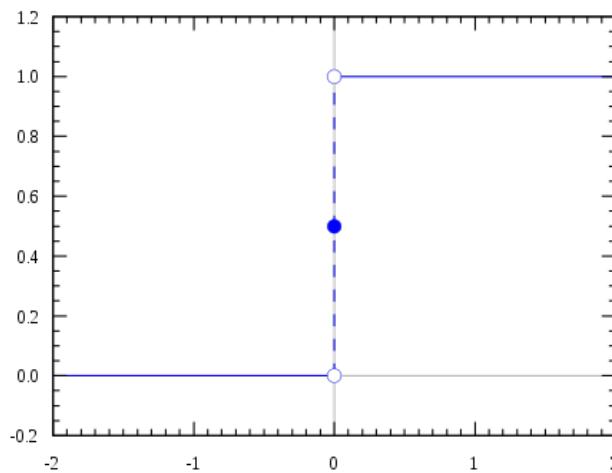
$$\bar{Y} = \frac{1}{T} \sum_{t=1}^T Y_t$$

Fonction de Heaviside est définie comme :

$$\forall x \in \mathbb{R}, \quad H(x) = \begin{cases} 0 & \text{si } x < 0 \\ \frac{1}{2} & \text{si } x = 0 \\ 1 & \text{si } x > 0. \end{cases}$$

La valeur de la fonction en 0 est parfois notée avec un indice: la fonction H_a satisfait l'égalité $H_a(0) = a$ pour a un réel quelconque.

La fonction est utilisée dans les mathématiques du traitement du signal pour représenter un signal obtenu en fermant un interrupteur à un instant donné et en le maintenant fermé indéfiniment.

FIGURE 8.37 – La fonction $H_{0,5}$ de Heaviside.

K-Means est une méthode de partitionnement de données et un problème d'optimisation combinatoire. Étant donnés des points et un entier k , le problème est de diviser les points en k groupes, souvent appelés clusters, de façon à minimiser une certaine fonction. On considère la distance d'un point à la moyenne des points de son cluster ; la fonction à minimiser est la somme des carrés de ces distances..

LSTM est un type de réseau de neurones récurrents (RNN) conçu pour traiter des séquences de données et surmonter certaines limitations des RNN traditionnels.

MA- m (Moyenne Mobile d'ordre m) peut être écrite comme

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^k y_{t+j},$$

où $m = 2k + 1$. Autrement dit, l'estimation de la tendance-cycle à l'instant t est obtenue en moyennant les valeurs de la série temporelle dans un intervalle de k périodes autour de t . Les observations proches dans le temps sont également susceptibles d'être proches en valeur. Par conséquent, la moyenne élimine une partie de l'aléatoire dans les données, laissant une composante tendance-cycle lissée. Nous appelons cela une m -MA, c'est-à-dire une moyenne mobile d'ordre m .

Modèles de Markov Caché Un modèle de Markov caché (MMC) — en anglais : *hidden Markov model (HMM)* —, ou plus correctement (mais non employé) automate de Markov à états cachés, est un modèle statistique dans lequel le système modélisé est supposé être un processus markovien de paramètres inconnus. Contrairement à une chaîne de Markov classique, où les transitions prises sont inconnues de l'utilisateur mais où les états d'une exécution sont connus, dans un modèle de Markov caché, les états d'une exécution sont inconnus de l'utilisateur (seuls certains paramètres, comme la température, etc. sont connus de l'utilisateur).

Formellement, un MMC est défini par :

$S = \{s_1, \dots, s_N\}$ l'ensemble discret des N états du modèle ; on désigne un état au temps t par $q_t \in S$.

$V = \{v_1, \dots, v_M\}$ l'ensemble discret des M symboles observables ; on désigne un symbole au temps t par $o_t \in V$.

$\pi = \{\pi_1, \dots, \pi_N\}$ la probabilité de commencer dans l'état i ($1 \leq i \leq N$).

$A = \{a_{ij}\}_{1 \leq i,j \leq N}$, où $a_{ij} = P(q_{t+1} = s_j | q_t = s_i)$ pour les modèles d'ordre 1 ; A est la matrice des probabilités de transitions entre états.

$B = \{b_j(k)\}_{1 \leq j \leq N, 1 \leq k \leq M}$, où $b_j(k) = P(o_t = v_k | q_t = s_j)$ B est la matrice des probabilités d'observation dans les états.

Habituellement, on désigne un MMC par le triplet $\{A, B, \pi\}$.

Les contraintes classiques d'un MMC sont :

$\sum_i \pi_i = 1$, $\forall s_i \in S$ la somme des probabilités des états initiaux est égale à 1.

$$\sum_{j=1}^N a_{ij} = 1, \quad \forall s_i, s_j \in S \text{ la somme des probabilités des transitions partant d'un état est égale à 1.}$$

$$\sum_{j=1}^N b_j(k) = 1, \quad \forall k \in V \text{ la somme des probabilités des observations d'un état est égale à 1.}$$

Moyenne mobile ou moyenne glissante (en anglais, Moving Average), est un type de moyenne statistique utilisée pour analyser des séries ordonnées de données, le plus souvent des séries temporelles, en supprimant les fluctuations transitoires de façon à en souligner les tendances à plus long terme. Cette moyenne est dite mobile parce qu'elle est recalculée de façon continue, en utilisant à chaque calcul un sous-ensemble d'éléments dans lequel un nouvel élément remplace le plus ancien ou s'ajoute au sous-ensemble.

Processus stationnaire si ses propriétés statistiques caractérisées par des espérances mathématiques sont indépendantes du temps.

RNN est un réseau de neurones artificiels présentant des connexions récurrentes.

Rétropropagation du gradient est une méthode pour entraîner un réseau de neurones. Elle consiste à mettre à jour les poids de chaque neurone de la dernière couche vers la première. Elle vise à corriger les erreurs selon l'importance de la contribution de chaque élément à celles-ci. Dans le cas des réseaux de neurones, les poids synaptiques qui contribuent plus à une erreur seront modifiés de manière plus importante que les poids qui provoquent une erreur marginale. Les poids dans le réseau de neurones sont au préalable initialisés avec des valeurs aléatoires. On considère ensuite un ensemble de données qui vont servir à l'apprentissage. Chaque échantillon possède ses valeurs cibles qui sont celles que le réseau de neurones doit à terme prédire lorsqu'on lui présente le même échantillon. L'algorithme suit les étapes suivantes. Soient un échantillon \vec{x} que l'on présente à l'entrée du réseau de neurones et \vec{t} la sortie désirée pour cet échantillon. On propage le signal en avant dans les couches du réseau de neurones : $x_k^{(n-1)} \mapsto x_j^{(n)}$, avec n le numéro de la couche, et k et j les numéros des neurones sur leur couche respective. La propagation vers l'avant se calcule à l'aide de la fonction d'activation g , de la fonction d'agrégation h (souvent un produit scalaire entre les poids et les entrées du neurone) et des poids synaptiques \vec{w}_{jk} entre le neurone $x_k^{(n-1)}$ et le neurone $x_j^{(n)}$. La notation est alors inversée : \vec{w}_{jk} indique bien un poids de k vers j .

$$x_j^{(n)} = g^{(n)}(h_j^{(n)}) = g^{(n)}\left(\sum_k w_{jk}^{(n)} x_k^{(n-1)}\right)$$

Lorsque la propagation vers l'avant est terminée, on obtient à la sortie le résultat \vec{y} , c-à-d, $\vec{y}_i = x_i^{\text{sortie}}$. On calcule alors l'erreur entre la sortie donnée par le réseau \vec{y} et le vecteur \vec{t} désiré à la sortie pour cet échantillon. Pour chaque neurone i dans la couche de sortie, on calcule (g' étant la dérivée de g):

$$e_i^{\text{sortie}} = g'(h_i^{\text{sortie}})(y_i - t_i)$$

On propage l'erreur vers l'arrière $e_i^{(n)} \mapsto e_j^{(n-1)}$ grâce à la formule suivante :

$$e_j^{(n-1)} = g'^{(n-1)}(h_j^{(n-1)}) \sum_i w_{ij}^{(n)} e_i^{(n)}$$

note :

$$e_i^{(n)} = e_i^{\text{sortie}} = (y_i - t_i) \frac{\partial y_i}{\partial h_i^{(n)}}$$

On met à jour les poids dans toutes les couches :

$$w_{ij}^{(n)} = w_{ij}^{(n)} - \lambda e_i^{(n)} x_j^{(n-1)}$$

où λ représente le taux d'apprentissage (de faible magnitude et compris entre 0,0 et 1,0).

Rétropropagation à travers le temps ou BPTT est une technique de gradient pour entraîner certains types de réseaux de neurones récurrents. Les données d'apprentissage d'un réseau neuronal récurrent sont une séquence ordonnée de k paires d'entrées-sorties, $\langle \mathbf{a}_0, \mathbf{y}_0 \rangle, \langle \mathbf{a}_1, \mathbf{y}_1 \rangle, \langle \mathbf{a}_2, \mathbf{y}_2 \rangle, \dots, \langle \mathbf{a}_{k-1}, \mathbf{y}_{k-1} \rangle$.

Une valeur initiale doit être spécifiée pour l'état caché \mathbf{x}_0 . En règle générale, un vecteur de tous les zéros est utilisé à cette fin. La rétropropagation à travers le temps commence par déplier un réseau neuronal récurrent dans le temps. Le réseau déplié contient k entrées et sorties, mais chaque copie du réseau partage les mêmes paramètres. Ensuite, l'algorithme de rétropropagation est utilisé pour trouver le gradient de la fonction de coût par rapport à tous les paramètres du réseau.

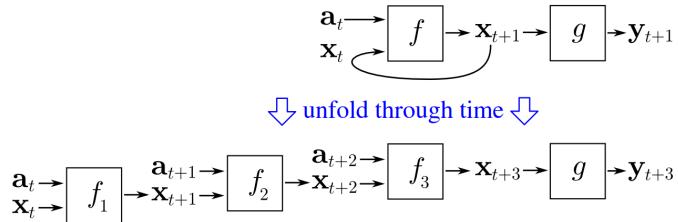


FIGURE 8.38 – L'algorithme de rétropropagation à travers le temps « déplie » un réseau neuronal récurrent dans le temps.

La rétropropagation à travers le temps a tendance à être beaucoup plus rapide pour former des réseaux de neurones récurrents que les techniques d'optimisation à usage général telles que l'optimisation évolutive .

Saisonalité d'une série temporelle capture les variations périodiques ou cycliques qui se répètent à intervalles réguliers. Dans un modèle additif, la série temporelle Y_t est la somme de la tendance T_t , de la composante saisonnière S_t , et de l'erreur E_t :

$$Y_t = T_t + S_t + E_t$$

Dans un modèle multiplicatif, la série temporelle Y_t est le produit de la tendance T_t , de la composante saisonnière S_t , et de l'erreur E_t :

$$Y_t = T_t \cdot S_t \cdot E_t$$

Tendance d'une série temporelle représente la composante à long terme qui montre une direction générale dans laquelle les données évoluent au fil du temps. Elle peut être modélisée par des fonctions simples telles que des lignes droites ou des courbes polynomiales.

Une tendance linéaire peut être modélisée par l'équation :

$$T_t = \beta_0 + \beta_1 t$$

où β_0 est l'ordonnée à l'origine et β_1 est la pente de la ligne.

Une tendance quadratique peut être modélisée par l'équation :

$$T_t = \beta_0 + \beta_1 t + \beta_2 t^2$$

où β_0 , β_1 , et β_2 sont les coefficients de l'équation quadratique..

Variance mesure la dispersion des valeurs autour de leur moyenne. Elle est définie par :

$$\text{Var}(Y_t) = \sigma^2 = E[(Y_t - \mu)^2]$$

Empiriquement, pour une série de T observations, la variance est :

$$\sigma^2 = \frac{1}{T} \sum_{t=1}^T (Y_t - \bar{Y})^2$$

Références

- [1] D. A. DICKEY et W. A. FULLER. “Distribution of the Estimators for Autoregressive Time Series with a Unit Root”. In : *Journal of the American Statistical Association* 74.366 (1979). JSTOR 2286348, p. 427-431. DOI : 10.1080/01621459.1979.10482531. URL : <https://doi.org/10.1080/01621459.1979.10482531>.
- [2] R. B. CLEVELAND et al. “STL : A seasonal-trend decomposition procedure based on loess”. In : *Journal of Official Statistics* 6.1 (1990), p. 3-33.
- [3] Yoshua BENGIO, Patrice SIMARD et Paolo FRASCONI. “Learning long-term dependencies with gradient descent is difficult”. In : *IEEE Transactions on Neural Networks* 5.2 (1994), p. 157-166. DOI : 10.1109/72.279181.
- [4] Sepp HOCHREITER et Jürgen SCHMIDHUBER. “Long short-term memory”. In : *Neural Computation* 9.8 (1997), p. 1735-1780.
- [5] Sepp HOCHREITER et Jürgen SCHMIDHUBER. “Long short-term memory”. In : *Neural Computation* 9.8 (1997), p. 1735-1780. DOI : 10.1162/neco.1997.9.8.1735.
- [7] Paolo CAMPOLUCCI, Aurelio UNCINI et Francesco PIAZZA. “A Signal-Flow-Graph Approach to On-line Gradient Calculation”. In : *Neural Computation* 12.8 (2000), p. 1901-1927. DOI : 10.1162/089976600300015196.
- [8] G Peter ZHANG. “Time series forecasting using a hybrid ARIMA and neural network model”. In : *Neurocomputing* 50 (2003), p. 159-175.
- [9] Shu FAN et Rob J HYNDMAN. “Short-term load forecasting based on a semi-parametric additive model”. In : *IEEE Transactions on Power Systems* 27.1 (2012), p. 134-141.
- [10] Kyunghyun CHO et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In : *arXiv preprint arXiv:1406.1078* (oct. 2014), p. 1724-1734. DOI : 10.48550/arXiv.1406.1078.
- [12] Junhua MAO et al. “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)”. In : *arXiv* (2014). arXiv : 1412.6632 [cs.CV].
- [15] Daniel L MARINO, Kithmin AMARASINGHE et Milos MANIC. “Building energy load forecasting using deep neural networks”. In : *IEMCON 2016–7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference* (2016), p. 1-7.
- [16] Sungho RYU, Kwangyong NOH et Heung-No KIM. “Deep neural network based demand side short term load forecasting”. In : *Energies* 10.1 (2017), p. 3.
- [17] Shuai LI et al. “Independently Recurrent Neural Network (IndRNN) : Building A Longer and Deeper RNN”. In : (2018), p. 5457-5466.
- [18] Stephen HABEN, Georgios GIASEMIDIS et Florian ZIEL. “Forecasting day-ahead electricity load using a multiple linear regression model”. In : *Energy Systems* 10 (2019), p. 1-21.
- [19] Slawek SMYL. “A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting”. In : *International Journal of Forecasting* 36.1 (2020), p. 75-85.
- [27] Luis HERNANDEZ et al. “A Survey on Electric Power Demand Forecasting :Future Trends in Smart Grids, Microgrids andSmart Buildings”. In : *IEEE Communications Surveys and Tutorials* 16 (4 Avril 2014).

Actes de conférence

- [6] Felix GERS, Jürgen SCHMIDHUBER et Fred CUMMINS. “Learning to Forget : Continual Prediction with LSTM”. In : *Proceedings of the International Conference on Artificial Neural Networks (ICANN'99)*. T. 1999. IEE. London, 1999, p. 850-855. ISBN : 0-85296-721-7. DOI : 10.1049/cp:19991218. URL : <https://ieeexplore.ieee.org/document/818045>.
- [11] Alex GRAVES et Navdeep JAITLEY. “Towards End-to-End Speech Recognition with Recurrent Neural Networks”. In : *Proceedings of the International Conference on Machine Learning (ICML)*. Beijing, China, 21–26 June 2014, p. 1764-1772.
- [13] Ilya SUTSKEVER, Oriol VINYALS et Quoc V. LE. “Sequence to Sequence Learning with Neural Networks”. In : *Proceedings of the Neural Information Processing Systems (NIPS)*. Montréal, QC, Canada, août 2014, p. 3104-3112.
- [28] S. HOCHREITER et J SCHMIDHUBER. “LSTM can solve hard long time lag problems”. In : *Advances in Neural Information Processing Systems*. Denver, CO, USA, December 2-5, 1996.

Livres

- [14] George E. P. BOX et al. *Time Series Analysis : Forecasting and Control*. 5th. 2015.
- [26] Chitta RANJAN. *Understanding Deep Learning : Application in Rare Event Prediction*. 1st. ISBN : 9798586189561.

Webographie

- [20] <https://www.youtube.com/watch?v=EY5OQ2iVA50>.
- [21] <https://towardsdatascience.com/detecting-stationarity-in-time-series-data-d29e0a21e638>.
- [22] <https://www.geeksforgeeks.org/autocorrelation/>.
- [23] <https://otexts.com/fpp2/>.
- [24] <https://aws.amazon.com/fr/what-is/recurrent-neural-network/>.
- [25] <https://blent.ai/blog/a/rnn-on-vous-explique-tout>.

