الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

جامعة سعد دحلب البليدة
Université SAAD DAHLAB de BLIDA

كليـــــة العلـــــوم
Faculté des Sciences

قسم الإعلام الآلي
Département Informatique

MÉMOIRE DE MASTER

EN INFORMATIQUE

Option : Systèmes Informatiques et Réseaux

Présenté par :

Mr. ALOUANI Faouzi **&** **Mr. ZERRAD Samir**

PRÉDICTION DES CRIMES EN UTILISANT LES RÉSEAUX DE NEURONES RÉCURRENTS

Soutenu devant le jury composé de :

Président : Mr. BENYAHIA Mohamed, Maître Assistant A, USDB

Examinatrice : Mme. ZAHRA Fatma Zohra, Maître Assistant A, USDB

Encadreur : Mr. KAMECHE Abdallah Hicham, Maître Assistant B, USDB

Blida, Septembre 2020

RÉSUMÉ

ملخص:

في السنوات الأخيرة، شهدت برامج الشرطة تطوًرا كبيًرا، مستفيدة من التقدم التكنولوجي في الإعلام الآلي. ومع ذلك، فإن هذا التطور يتعلق فقط بمجال الإحصاء والرسوم البيانية في أنظمة المساعدة على إتخاذ القرار (لوحة المعلومات، الحوصلة، إلخ).

في الواقع، هذه البرامج (مثال: ICO Police، Citigraf، إلخ.) تتجاهل تمًاما جانب التنبؤ الشرطي، مثل التنبؤ بالجرائم بمرور الوقت من حيث الكم والنوع من أجل اتخاذ القرارات المناسبة لمجابهة بشكل أفضل ضد هذه الآفة، من خلال تكييف خطط العمل الوقائية، وترشيد الموارد اللازمة. هذه هي نقطة البداية لعملنا، والتي تتمثل في إقتراح متنبئ للجريمة. هذا المتنبئ مسؤول عن توقع الجريمة في المحور الزمني من حيث الكم والنوع.

يعتمد المتنبئ لدينا على الشبكات العصبية المتكررة RNN مرورا بمختلف الأنواع من LSTM و GRU.

تخص دراستنا تقييم الأساليب والنماذج المختلفة المقترحة في التنبؤ.

الحلول المقترحة قد تم تدريبها وتجريبها على قاعدة بيانات فعلية.

الكلمات المفتاحية: التعلم العميق، الشبكة العصبية المتكررة، النماذج التنبؤية، الشرطة التنبؤية.

Résumé:

Ces dernières années, les logiciels de police ont connu un grand développement, profitants ainsi des avances technologiques de l'informatique. Cependant ce développement concerne uniquement la partie statistique et graphique d'aide à la décision (tableau de bord, bilan,…etc.).

En effet, ces logiciels (Exemple: ICO Police, Citigraf,…etc.) négligent complètement l'aspect de la prédiction policière, tel que la prédiction des crimes dans le temps en nombre et en type afin de mieux prendre les décisions idoines pour lutter contre ce fléau, en adaptant les plans d'actions préventifs, tout en optimisant les ressources nécessaires. C'est le point de départ de notre travail, qui consiste à proposer un prédicteur de crime. Ce prédicteur est chargé de prévoir le crime dans l'axe du temps en nombre et en type.

Notre prédicteur, est basé sur les réseaux de neurones récurrents à savoir les RNN tout en passant par leurs variantes LSTM (Long Short-Term Memory) et GRU (Gated recurrent units).

Notre étude consiste à évaluer les différentes architectures et approches de prédiction proposées.

Les solutions suggérées ont été entrainées et testées sur une base de données réelle.

Mots clés: Deep Learning, réseau de neurones récurrents, modèles prédictifs, police prédictive.

Abstract:

In recent years, police software has experienced a great development, taking advantage of technological advances in computing. However, this development only concerns the statistical and graphical part of decision support (dashboard, report, etc.).

Indeed, these software (Example: ICO Police, Citigraf, etc.) completely neglect the aspect of police prediction, such as the prediction of crimes over time in number and type in order to better take the appropriate decisions to stop this fact, by adapting preventive action plans, while optimizing the necessary resources. This is the starting point of our work, which consists in proposing a crime predictor. This predictor is responsible for predicting crime in the time axis in number and type.

Our predictor is based on recurrent neural networks namely RNNs while passing through their variants LSTM (Long Short-Term Memory) and GRU (Gated recurrent units).

This study consists in evaluating the different architectures and prediction approaches proposed.

Keywords: Deep Learning, recurrent neural network, predictive models, predictive policing.

REMERCIEMENTS

Ce travail n'aurait pas vu le jour sans l'aide considérable, le soutien, la dextérité et la revue critique du mémoire, tant sur le fond que sur la forme de notre encadreur Mr KAMECHE Abdallah Hicham, qui n'a épargné aucun effort pour nous guider dans le chemin du savoir.

Nous remercions également Mr GUESMIA Abdelkader le chef du département d'informatique pour ses remarques et son suivi durant toutes les étapes de nos études et de notre projet.

Nous tenons aussi à exprimer toute notre gratitude à tous les enseignants qui ont contribué à notre formation.

Pour leur soutien moral et leurs encouragements, un grand merci à tous les proches, les amis et les collègues.

En terminant, il nous est agréable de remercier infiniment ceux et celles qui ont participé de près ou de loin afin d'achever ce mémoire

À ma très chère mère,

À mon vénéré père,

À ma petite famille, ma femme et mes enfants,

À mes frères et mes sœurs,

À toute ma grande famille,

À tous mes amis.

Faouzi

Je dédie ce modeste travail à :

Mes parents qui sans eux je n'aurai pas pu arriver à ce stade,

Ma petite famille, ma femme et mes enfants,

Mes frères et mes sœurs,

Ma grande famille,

Mes amis.

Samir

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES DIAGRAMMES

LISTE DES ACRONYMES ET ABRÉVIATIONS

Acronyme	Intitulé
PredPol :	Predictive policing software par PredPol Company
Vanilla RNN :	Vanilla Recurrent Neural Network (Simple RNN)
LSTM :	Long Short-Term Memory
GRU :	Gated Recurrent Unit
GAN :	Generative Adversarial Network
Word2Vec :	Algorithme "Word to Vectors" par l'équipe de Tomas Mikolov
GPU :	Graphical Processing Unit
Tanh :	Tangente Hyperbolique
RNN :	Recurrent Neural Network
CNN :	Convolutional Neural Network
BP :	Back Propagation
SRN :	Simple Recurrent Networks
ML :	Machine Learning
SVM :	Support Vector Machine
MLP :	Multilayer Perceptron
ST-ResNet :	Spatio-Temporal Residual Networks
DBScan :	Density-Based Spatial Clustering of Applications with Noise
EM :	Expectation–Maximization
GMM :	Gaussian Mixture Models
GBM :	Gradient Boosting Machine
CART :	Classification And Regression Trees
XGBoost :	eXtreme Gradient Boosting
Catboost :	CATegory BOOSTing
Logit boost :	LOGIsTic BOOST
CCRBoost :	Cluster Confidence Rate Boosting
AG :	Algorithmes génétiques
Q-Learning :	Quality-LEARNING
ReLU :	Rectified Linear Unit (fonction d'activation)
RMSE :	Root Mean Square Error
MSE :	Mean Square Error
MAE :	Mean Absolute Error
OLS :	Ordinary Least Squares
Palantir :	Software of predictive policing par l'enterprise Palantir technologies
HunchLab :	Software of predictive policing par l'enterprise Azavea
SQL :	Structured Query Language
DataSet :	Ensemble de données d'entrée d'un modèle de ML
CSV :	Comma-separated values
TPE :	Tree-structured parzen estimator
UML	Unified Modeling Language
CPU :	Central Processing Unit
RAM :	Random Access Memory
QGis :	Quantum Geographic Information System
SAGA GIS :	System for Automated Geoscientific Analyses Geographic Information System
ArcGIS :	Arc (view) Geographic Information System

INTRODUCTION GÉNÉRALE

Les services de police ont consacré beaucoup de temps et de ressources à élaborer et à analyser les tendances du taux de criminalité et aussi à adapter leurs techniques policières à ces tendances: c'est ce qu'on appelle la prédiction policière.

La prédiction policière permet de mieux comprendre ces tendances et de prendre les décisions idoines pour lutter contre la criminalité, en adaptant les plans d'actions préventifs, optimisant les ressources nécessaires. Au début, les services de police se limitaient à examiner les tendances historiques en utilisant les méthodes statistiques traditionnelles et à extrapoler pour l'avenir, cependant avec l'amélioration des techniques d'analyse des données, des modèles prédictifs ont apparus depuis l'année **1998 [1]**, basés sur le Machine Learning, mais limités par la puissance de calcul à l'époque. Le premier modèle commercialisé de la police prédictive était "**PredPol**" en **2012 [1]**.

1. Motivation:

Avant l'apprentissage automatique, les premières techniques extrapolaient simplement les tendances historiques en utilisant les méthodes statistiques traditionnelles, cependant elles ne peuvent pas prévoir les écarts par rapport à ces tendances à l'exception de certains techniques plus avancées qui ont amélioré le pouvoir prédictif, telle que la régression multi-variée, mais ne reposaient toujours que sur des données historiques sur la criminalité **[2] [3]**.

Les services de police ont recours à l'exploitation des données massives des applications à vocation opérationnelle ce qui augmente considérablement le nombre d'entités, et d'autre part les nouvelles méthodes d'apprentissage automatique sont mieux adaptées pour gérer un grand nombre de caractéristiques et de trouver les relations complexes entre ces facteurs.

Avec le développement des ordinateurs de hautes performances, le coût de ces techniques d'apprentissage automatique a été considérablement réduit et donnant des résultats impressionnants.

La prédiction de la criminalité est un domaine qui n'a pas reçu beaucoup d'attention dans le Machine Learning en utilisant des ensembles de données sur la criminalité. En outre, la complexité inhérente du problème le rend approprié pour les réseaux de neurones et le Deep Learning.

2. Objectifs:

Notre objectif principal est la conception et l'implémentation d'un prédicteur de crimes basé sur les réseaux de neurones récurrents. Il s'agit d'une comparaison et une évaluation des trois modèles récurrents (Vanilla RNN, LSTM et GRU). Dans ce prédicteur on utilise des données réelles, et étant donné les paramètres et les caractéristiques à résoudre, le prédicteur au final devra donner les prédictions de crimes en nombre et en type dans l'axe du temps, avec une évaluation des modèles prédictifs.

Pour atteindre notre but, l'arbre à objectifs est le suivant:

1) Faire un état de lieu sur les techniques basées sur le Machine Learning dont les réseaux de neurones récurrents font partie, pour la prédiction des crimes.
2) Implémenter un prédicteur de crimes basé sur les réseaux de neurones récurrents.
3) Évaluer la performance du prédicteur proposé sur une base de données réelle.

3. Plan du mémoire:

Dans ce travail, nous allons d'abord sensibiliser le lecteur à la théorie du domaine de Machine Learning, les définitions et les types d'apprentissage automatique dont les réseaux de neurones font partie prenante.

Dans la deuxième partie, nous allons faire une étude bibliographique des algorithmes de Machine Learning, puis une synthèse sur les applications similaires en présentant les limites de chacun des algorithmes dans la prédiction.

Troisièmement, proposer une modélisation et une approche qui apporte des solutions à l'analyse et le prétraitement de données ainsi que les critères de choix de modèles de réseaux de neurones récurrents.

Enfin, la dernière partie sera consacrée au développement de notre prédicteur de crimes et à l'évaluation de ses performances.

Nos suggestions et perspectives dans le domaine, viennent clôturer nos travaux.

CHAPITRE I

INTRODUCTION AUX RÉSEAUX DE NEURONES

I.1 Introduction:

Au cours des deux dernières décennies, nous avons pu constater un développement fulgurant des *réseaux de neurones*. Cet intérêt a démarré avec l'application réussie de cette technique puissante pour des problématiques très différentes, et dans des domaines aussi divers que la finance, la médecine, la production industrielle, la géologie, la physique ou encore la police prédictive.

Le succès croissant des réseaux de neurones sur la plupart des autres techniques statistiques **[4] [5]** peut s'attribuer à leur puissance, leur polyvalence et à leur simplicité d'utilisation. Les réseaux de neurones sont des techniques extrêmement sophistiquées de modélisation et de prévision (prédiction), en mesure de modéliser des relations entre des données ou des fonctions particulièrement complexes.

La possibilité d'apprendre sur la base d'exemples constitue l'une des nombreuses fonctionnalités des réseaux de neurones qui permettent à l'utilisateur de modéliser ses données et établir des règles précises qui vont guider les relations sous-jacentes entre différents attributs de données. L'utilisateur des réseaux de neurones collecte des données représentatives puis il fait appel aux *algorithmes d'apprentissage*, qui vont apprendre automatiquement la structure des données.

I.2 Apprentissage automatique:

Il consiste à la recherche automatique d'un modèle à partir de données pouvant être réutilisé dans un nouvel environnement ou une nouvelle situation (ex prédiction) **[6]**.

L'apprentissage automatique est un concept pour définir l'acquisition de connaissance et réutiliser ces nouvelles connaissances. Pour nous les humains, notre apprentissage se fait tout au long de notre vie. Nous apprenons à partir de perception de l'environnement avec les cinq sens, les expériences de la vie, de répétition des événements avec la mémoire, et de notre jugement (libre arbitre ou intelligence). Pour les machines, comme elles ne sont pas dotées de sens ni de jugement dynamique,

elles obéissent aux instructions dans un programme avec des données d'entrée et une donnée sortie ou réponse du programme **[7]**.

I.2.1 Définition:

L'apprentissage automatique est un domaine qui permet à une machine d'avoir des connaissances sans être formellement programmé ou sans intervention humaine **[9]**.

Une définition formelle de l'apprentissage automatique qui lui considère comme un programme informatique qui apprend à partir d'une donnée d'expérience **E** une tâche spécifique **T** avec une mesure de performance **P**. Par exemple dans une étude météo, **E** = sera l'historique météo de plusieurs jours, **T** = l'estimation de la météo du lendemain et **P** = la probabilité de la météo estimée par le programme **[8]**.

L'apprentissage nécessite une expérience **E**. Celle-ci consiste généralement en une base de données d'apprentissage **Btrain** que le modèle analyse lors du processus d'apprentissage. Cet apprentissage d'une tâche **T** se fait à l'aide d'une fonction de coût **J**. Cette fonction de coût est calculée sur une base de données de test **Btest**, distincte de **Btrain**, afin de mesurer les performances du modèle appris; c'est la mesure de performance **P [8]**.

Lors de l'apprentissage, le modèle doit donc être capable de modifier ses paramètres à l'aide de la base de données d'apprentissage **Btrain** pour améliorer ses performances mesurées par **P**. Le fait que la performance soit mesurée sur une base de données **Btest** distincte de **Btrain** implique une capacité de généralisation du modèle, c.-à-d., une capacité à répondre à des cas qu'il n'a pas vu lors de son expérience **E**. Le but du modèle est de faire tendre ses paramètres vers un optimal qui minimise **J** sur **Btest** (**la mesure P**) **[8]**.

I.2.2 Types:

Dans l'apprentissage automatique, les tâches sont généralement classées en grandes catégories. Ces catégories sont basées sur la façon dont l'apprentissage est reçu ou comment le feedback sur l'apprentissage est donné au système développé.

Deux des types d'apprentissage automatique les plus largement adoptées sont l'**apprentissage supervisé** qui forme des algorithmes basés sur des données d'entrée et de sortie étiquetées par l'homme et l'apprentissage **non supervisé** qui ne fournit pas à l'algorithme des données étiquetées pour lui permettre de trouver une structure et de découvrir une logique dans les données entrées. Explorons donc ces types plus en détail.

Il est à noter, qu'il existe aussi un **apprentissage semi-supervisé** et **par renforcement**.

I.2.2.1 Apprentissage supervisé:

Dans l'apprentissage supervisé, l'ordinateur est fourni avec des exemples d'entrées qui sont étiquetés avec les sorties souhaitées. Le but de cette méthode est que l'algorithme puisse

«apprendre» en comparant sa sortie réelle avec les sorties «enseignées» pour trouver des erreurs et modifier le modèle en conséquence. L'apprentissage supervisé utilise donc des modèles pour prédire les valeurs d'étiquettes sur des données non étiquetées supplémentaires.

Par exemple, avec un apprentissage supervisé, un algorithme peut être alimenté avec des images de requins étiquetés **Poisson** des images d'océans étiquetés comme **Ocean**. En étant formé sur ces données, l'algorithme d'apprentissage supervisé devrait être capable d'identifier plus tard des images de requin non marquées comme Poisson des images océaniques non étiquetées **Ocean**. Un cas d'utilisation de l'apprentissage supervisé consiste à utiliser des données historiques pour prédire des événements futurs statistiquement probables. Il peut utiliser les informations historiques sur les marchés boursiers pour anticiper les fluctuations à venir ou être utilisé pour filtrer les courriers indésirables. Dans l'apprentissage supervisé, des photos étiquetées de chiens peuvent être utilisées comme données d'entrée pour classer les photos non marquées de chiens **[9]**.

Nous présentons ici les tâches d'apprentissage supervisé: **[8]**

- **La classification:** Lorsque les données servent à prédire une variable catégorielle, l'apprentissage supervisé est également appelé classification. C'est le cas par exemple lorsqu'une étiquette ou un indicateur (par exemple « chien » ou « chat ») est attribué à une image. Lorsqu'il n'y a que deux étiquettes, on parle de classification binaire. Lorsqu'il y a plus de deux catégories, on parle de classification en classes multiples.

- La régression: Lorsqu'on procède à la prédiction de valeurs continues, on parle de régression, elle consiste à associer l'entrée $\boldsymbol{x}$ à un $\boldsymbol{y \epsilon R^m}$. La principale différence avec la classification est l'espace de sortie Y qui est continu.

I.2.2.1.1 Prévision (Prédiction):

Il s'agit de faire des prédictions à partir de données passées et présentes. Ce type de processus sert le plus souvent à analyser des tendances, par exemple estimer les ventes de l'année prochaine à partir des ventes de l'année en cours et des années précédentes.

I.2.2.2 Apprentissage non supervisé:

Dans l'apprentissage non supervisé, les données sont non étiquetées, de sorte que l'algorithme d'apprentissage trouve tout seul des points communs parmi ses données d'entrée. Les données non étiquetées étant plus abondantes que les données étiquetées, les méthodes d'apprentissage automatique qui facilitent l'apprentissage non supervisé sont particulièrement utiles.

L'objectif de l'apprentissage non supervisé peut être aussi simple que de découvrir des modèles cachés dans un ensemble de données, mais il peut aussi avoir un objectif d'apprentissage

des caractéristiques, qui permet à la machine intelligente de découvrir automatiquement les représentations nécessaires pour classer les données brutes.

L'apprentissage non supervisé est couramment utilisé pour les données transactionnelles. Vous pouvez avoir un grand ensemble de données sur les clients et leurs achats, mais en tant qu'être humain, vous ne serez probablement pas en mesure de comprendre quels attributs similaires peuvent être tirés des profils de clients et de leurs types d'achats. Avec ces données introduites dans un algorithme d'apprentissage non supervisé, on peut déterminer que les femmes d'une certaine tranche d'âge qui achètent des savons non parfumés sont susceptibles d'être enceintes, et donc une campagne de marketing liée à la grossesse et aux produits pour bébés pour augmenter leur nombre d'achats.

Sans une réponse «correcte», les méthodes d'apprentissage non supervisées peuvent examiner des données complexes, plus expansives et apparemment sans point commun, afin de les organiser de manière potentiellement significative. L'apprentissage non supervisé est souvent utilisé pour la détection d'anomalies, y compris pour les achats frauduleux de cartes de crédit et les systèmes de recommandation qui conseille sur les produits à acheter ensuite. Dans l'apprentissage non supervisé, les photos non marquées des chiens peuvent être utilisées comme données d'entrée pour l'algorithme afin de trouver des similitudes et de classer les photos de chiens ensemble **[9]**.

Les 03 principales tâches qui peuvent être réalisées à partir d'un apprentissage non supervisé sont les suivantes : **[8]**

- Clustering: Cette tâche consiste à séparer les données d'entrée en différents groupes en fonction de leur structure ou de leurs similarités. Contrairement aux tâches de classification, les groupes ne sont pas connus par avance. Dans cette catégorie, il est possible de citer les algorithmes des **K-moyennes** ou de **clustering hiérarchique**.

- Distribution de densité: Cette tâche consiste à trouver ou capturer la distribution (inconnue) des données en entrée. L'un des exemples le plus récent est le modèle des réseaux antagonistes génératifs (GAN).

- Réduction de la dimensionnalité: Cette tâche consiste à compresser les données d'entrée (par exemple, représentées dans $\boldsymbol{R^n}$) sur un espace de représentation plus petit (par exemple, $\boldsymbol{R^m}$ avec $\boldsymbol{n >> m}$). Outre le simple gain en termes de place, cette réduction est également intéressante pour représenter les données. Réduire la dimensionnalité permet d'éviter les problématiques liées à la "malédiction de la dimension". Cette malédiction fait référence aux problématiques que l'on rencontre sur des données de grandes dimensions. Par exemple, la recherche de plus proches voisins obtient de très mauvais résultats sur des données avec beaucoup de dimensions. Un exemple intéressant de réduction de dimension est le modèle du **Word2Vec**, capable de représenter des mots, par des vecteurs

denses de quelques centaines de dimensions. Cette représentation a tendance à rapprocher les mots avec des sens proches les uns des autres.

I.2.2.3 Apprentissage par renforcement:

Des tentatives profondes d'apprentissage pour imiter comment le cerveau humain peut traiter des stimuli lumineux et sonores dans la vision et l'ouïe sont à l'étude. Une architecture d'apprentissage par renforcement est inspirée par les réseaux neuronaux biologiques et se compose de plusieurs couches dans un réseau neuronal artificiel composé de matériel et de **GPU**.

L'apprentissage par renforcement ou approfondies (Deep Learning) utilise une cascade de couches d'unités de traitement non linéaires afin d'extraire ou de transformer les caractéristiques (ou représentations) des données. La sortie d'une couche sert d'entrée de la couche suivante. Dans l'apprentissage par renforcement, les algorithmes peuvent être supervisés et servir à classer les données, ou non supervisés et à effectuer une analyse de modèle. Parmi les algorithmes d'apprentissage machine actuellement utilisés et développés, l'apprentissage par renforcement absorbe le plus de données et a été capable de battre les humains dans certaines tâches cognitives. En raison de ces attributs, l'apprentissage par renforcement est devenu l'approche avec un potentiel significatif dans le monde de l'intelligence artificielle. La reconnaissance faciale par ordinateur et la reconnaissance vocale ont toutes deux permis de réaliser des progrès significatifs grâce à des approches d'apprentissage approfondies. **IBM Watson** est un exemple bien connu d'un système qui exploite l'apprentissage par renforcement **[9]**.

I.3 Réseau de neurones:

L'origine des réseaux de neurones vient de l'essai de modélisation mathématique du cerveau humain les premiers travaux datent de 1943 et sont l'œuvre de **W.M. Culloch** et **W. Pitts [8].** Ils supposent que l'impulsion nerveuse est le résultat d'un calcul simple effectué par chaque neurone et que la pensée née grâce à l'effet collectif d'un réseau de neurone interconnecté **[8]**.

I.3.1 Définition:

Un réseau de neurones est un assemblage de constituants élémentaires interconnectés (appelés «neurones» en hommage à leur modèle biologique), qui réalisent chacun un traitement simple mais dont l'ensemble en interaction fait émerger des propriétés globales complexes. Chaque neurone fonctionne indépendamment des autres de telle sorte que l'ensemble forme un système massivement parallèle. L'information est stockée de manière distribuée dans le réseau sous forme de coefficients synaptiques ou de fonctions d'activation, il n'y a donc pas de zone de mémoire et de zone de calcul, l'une et l'autre sont intimement liés.

Le réseau de neurones artificiel ne se programme pas, est un algorithme d'apprentissage automatique qui s'inspire de quelques aspects des neurones animaux (**Figure 1.1**). Il s'agit d'un des plus puissants classifieurs aujourd'hui. Il est modélisé mathématiquement par un réseau dans un graphe, plus ou moins complexe, hiérarchique sous forme de couches dont les nœuds élémentaires sont appelés neurones. Les tâches particulièrement adaptées au traitement par réseau de neurones sont: l'association, la classification, la discrimination, la prévision ou l'estimation, et la commande de processus complexes.

Il est composé généralement de trois types de couches: la couche d'entrée (input layer), couche(s) cachée(s) (hidden layer) et la couche de sortie (output layer). La couche d'entrée est composée de neurones qui correspondent aux caractéristiques des données d'entrée représentées par une grille multidimensionnelle (par exemple la matrice de pixels de l'image ou la forme vectorielle de la donnée). La couche de sortie représente les résultats de la tâche assignée au réseau. Les couches cachées sont les couches intermédiaires entre l'entrée et la sortie. L'ensemble du réseau ainsi formé est généralement vu comme une boîte noire. La taille d'un réseau de neurones est mesurée par le nombre de couches, nombre de nœuds et la façon dont les nœuds/neurones sont connectés (**Figure 1.2**) **[7]**.

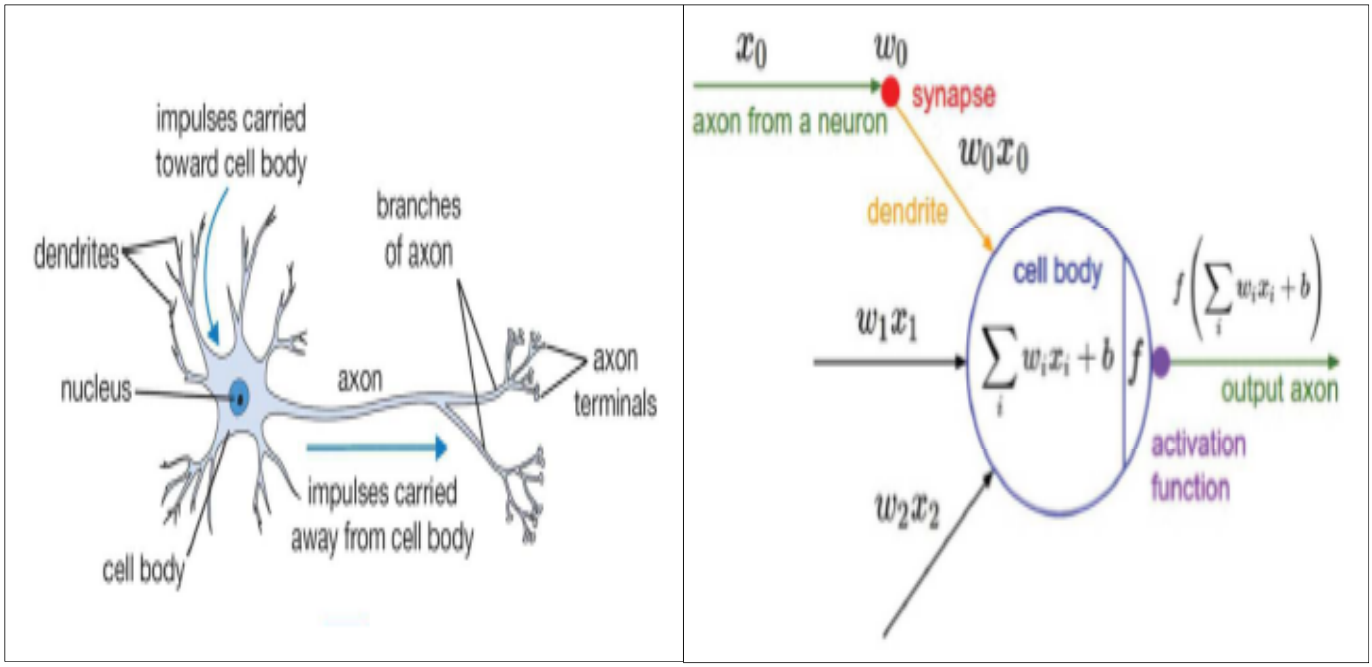

Figure I.1 - Comparaison entre un neurone animal et le modèle mathématique d'un neurone artificiel. Dans les réseaux biologiques, les données d'entrée proviennent d'un autre neurone qui passe dans la dendrite, le corps cellulaire fait les calculs et le résultat sort par l'axone. Le fonctionnement d'un neurone artificiel est calqué sur celui d'un neurone biologique **[7]**.

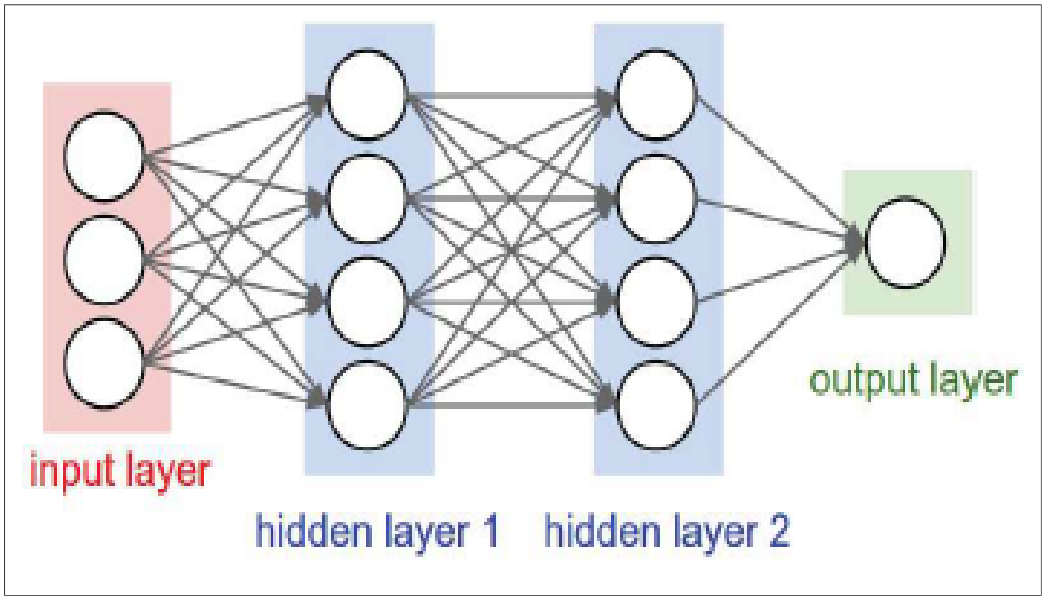

Figure I.2 - Réseau de neurones ou réseaux de neurones avec deux couches cachées; Réseau de neurones pleinement connecté **[7]**.

I.3.2 Modèle mathématique:

I.3.2.1 Les neurones formels:

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

Un neurone est donc avant tout un opérateur mathématique, dont on peut calculer la valeur numérique par quelques lignes de logiciel. On a pris l'habitude de représenter graphiquement un neurone comme indiqué sur la **figure 1.3**.

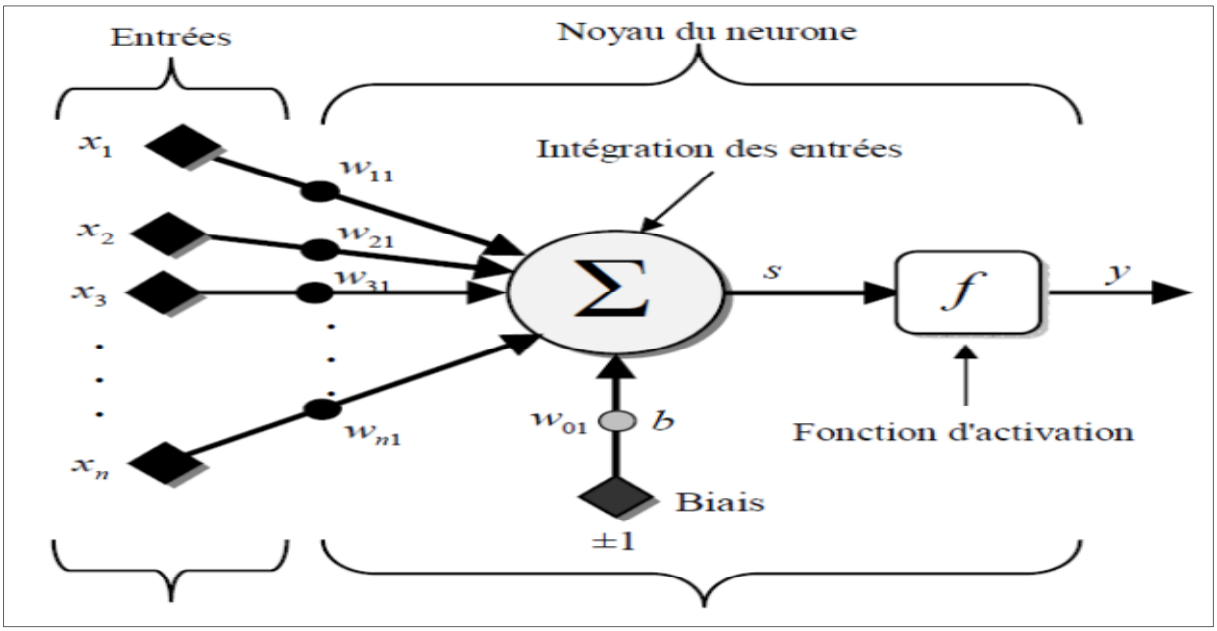

Figure I.3 - Modèle d'un neurone artificiel **[8]**.

Des observations de neurone biologique, découle le modèle du neurone formel proposé par **W.M. Culloch** et **W. Pitts** en 1943**[8]** :

- Les x_i représentent les vecteurs d'entrées, elles proviennent soit des sorties d'autres neurones, soit de stimuli sensoriels (capteur visuel, sonore…) ;
- Les w_{ij}sont les poids synaptiques du neurone j. Ils correspondent à l'efficacité synaptique dans les neurones biologiques ($w_{ij} > 0$: synapse excitatrice, $w_{ij} < 0$: synapse inhibitrice). Ces poids pondèrent les entrées et peuvent être modifiés par apprentissage ;
- Biais : entrée prend souvent les valeurs -1 ou +1 qui permet d'ajouter de la flexibilité au réseau en permettant de varier le seuil de déclenchement du neurone par l'ajustement des poids et du biais lors de l'apprentissage ;
- Noyau : intègre toutes les entrées et le biais et calcul la sortie du neurone selon une fonction d'activation qui est souvent non linéaire pour donner une plus grande flexibilité d'apprentissage.

I.3.2.2 Modélisation d'un neurone formel:

La modélisation consiste à mettre en œuvre un système de réseau de neurones sous un aspect non pas biologique mais artificiel, cela suppose que d'après le principe biologique on aura une correspondance pour chaque élément composant le neurone biologique, donc une modélisation pour chacun d'entre eux.

On pourra résumer cette modélisation par le tableau 1.1, qui nous permettra de voir clairement la transition entre le neurone biologique et le neurone formel.

Neurone biologique	Neurone formel
Synapses	Poids des connexions
Axones	Signal de sortie
Dendrites	Signal d'entrée
Noyau ou Somma	Fonction d'activation

Tableau I.1 - Analogie entre le neurone biologique et le neurone formel **[8]**.

Le modèle mathématique d'un neurone artificiel est déjà illustré à la figure 1.3. Un neurone est essentiellement constitué d'un intégrateur qui effectue la somme pondérée de ses entrées. Le résultat s de cette somme est ensuite transformé par une fonction de transfert f qui produit la sortie y du neurone. En suivant les notations présentées à la section précédente, les n entrées du neurone correspondent au vecteur $x = [x_1, x_2, x_3 \dots x_n]^T$, alors que $w = [w_{11}, w_{21}, w_{31} \dots w_{n1}]^T$, représente le vecteur des poids du neurone.

La sortie s de l'intégrateur est donnée par l'équation suivante:

$$s = \sum_{i=1}^{n} w_{il}\, x_i \pm b = w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + \cdots + w_{n1}x_n \pm b \qquad \textbf{(I.1)}$$

Que l'on peut aussi écrire sous forme matricielle :

$$s = w^T x \pm b \quad \textbf{(I.2)}$$

Cette sortie correspond à une somme pondérée des poids et des entrées plus ce qu'on nomme le biais b du neurone. Le résultat s de la somme pondérée s'appelle le niveau d'activation du neurone. Le biais b s'appelle aussi le seuil d'activation du neurone. Lorsque le niveau d'activation atteint ou dépasse le seuil b, alors l'argument de f devient positif (ou nul). Sinon, il est négatif.

On peut faire un parallèle entre ce modèle mathématique et certaines informations que l'on connaît (ou que l'on croit connaître) à propos du neurone biologique. Ce dernier possède trois principales composantes: les dendrites, le corps cellulaire et l'axone (Figure 1.1). Les dendrites forment un maillage de récepteurs nerveux qui permettent d'acheminer vers le corps du neurone des signaux électriques en provenance d'autres neurones. Celui-ci agit comme une espèce d'intégrateur en accumulant des charges électriques. Lorsque le neurone devient suffisamment excité (lorsque la charge accumulée dépasse un certain seuil), par un processus électrochimique, il engendre un potentiel électrique qui se propage à travers son axone pour éventuellement venir exciter d'autres neurones. Le point de contact entre l'axone d'un neurone et la dendrite d'un autre neurone s'appelle la synapse. Il semble que c'est l'arrangement spatial des neurones et de leur axone, ainsi que la qualité des connexions synaptiques individuelles qui détermine la fonction précise d'un réseau de neurones biologique. C'est en se basant sur ces connaissances que le modèle mathématique décrit ci-dessus a été défini.

Un poids d'un neurone artificiel représente donc l'efficacité d'une connexion synaptique. Un poids négatif vient inhiber une entrée, alors qu'un poids positif vient l'accentuer. Il importe de retenir que ceci est une grossière approximation d'une véritable synapse qui résulte en fait d'un processus chimique très complexe et dépendant de nombreux facteurs extérieurs encore mal connus. Il faut bien comprendre que notre neurone artificiel est un modèle pragmatique qui, comme nous le verrons ci-après, nous permettra d'accomplir des tâches intéressantes. La vraisemblance biologique de ce modèle ne nous importe peu. Ce qui compte est le résultat que ce modèle nous permettrons d'atteindre.

Un autre facteur limitatif dans le modèle que nous nous sommes donnés concerne son caractère discret. En effet, pour pouvoir simuler un réseau de neurones, nous allons rendre le temps discret dans nos équations. Autrement dit, nous allons supposer que tous les neurones sont synchrones, c'est à dire qu'à chaque temps t, ils vont simultanément calculer leur somme pondérée et produire une sortie:

$$y(t) = f(s(t)) \quad \textbf{(I.3)}$$

Dans les réseaux de neurones biologiques, tous les neurones sont en fait asynchrones. Revenons donc à notre modèle artificiel tel que formulé par l'équation (I.2) et ajoutons la fonction d'activation f pour obtenir la sortie du neurone :

$$y = f(s) = f(w^T x \pm b) \quad \textbf{(I.4)}$$

En remplaçant w^T par une matrice $W = w^T$ d'une seule ligne, on obtient une forme générale :

$$y = f(Wx \pm b) \quad \textbf{(I.5)}$$

L'équation (I.5) nous amène à introduire un schéma de notre modèle plus compact que celui de la Figure **1.3**. La Figure **1.4** illustre celui-ci. On y représente les n entrées comme un rectangle noir. De ce rectangle sort le vecteur x dont la dimension matricielle est $\boldsymbol{n}$ x 1. Ce vecteur est multiplié par une matrice W qui contient les poids (synaptiques) du neurone. Dans le cas d'un neurone simple, cette matrice possède la dimension $1 \times n$. Le résultat de la multiplication correspond au niveau d'activation qui est ensuite comparé au seuil b (un scalaire) par soustraction. Finalement, la sortie du neurone est calculée par la fonction d'activation f. La sortie d'un neurone est toujours un scalaire **[9]**.

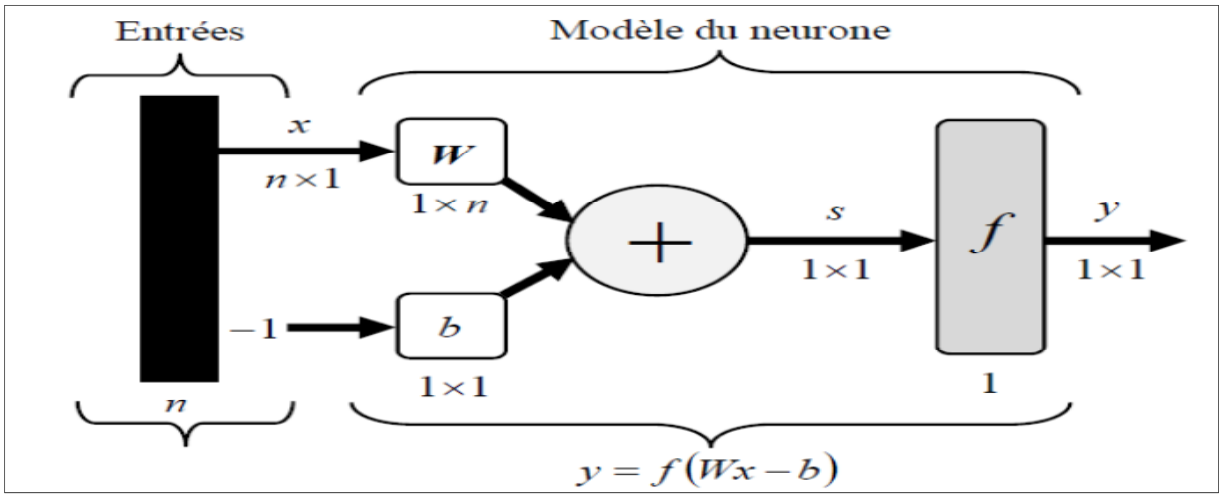

Figure I.4 - Représentation matricielle du modèle d'un neurone artificiel **[9]**.

I.4 Types des réseaux de neurones:

Le type d'un réseau de neurones reflète l'architecture et l'organisation des neurones entre eux au sein d'un même réseau. Autrement dit, il s'agit de la façon dont ils ordonnés et connectés. La majorité des réseaux de neurones utilise le même type de neurones. Quelques architectures plus rares se basent sur des neurones dédiés. L'architecture d'un réseau de neurones dépend de la tâche à apprendre.

Un réseau de neurone est en général composé de plusieurs couches de neurones, des entrées jusqu'aux sorties. On distingue deux grands types d'architectures de réseaux de neurones: les réseaux de neurones non récurrents et les réseaux de neurones récurrents.

Une Chronologie des modèles les plus marquants des réseaux de neurones est illustrée dans la figure 1.7.

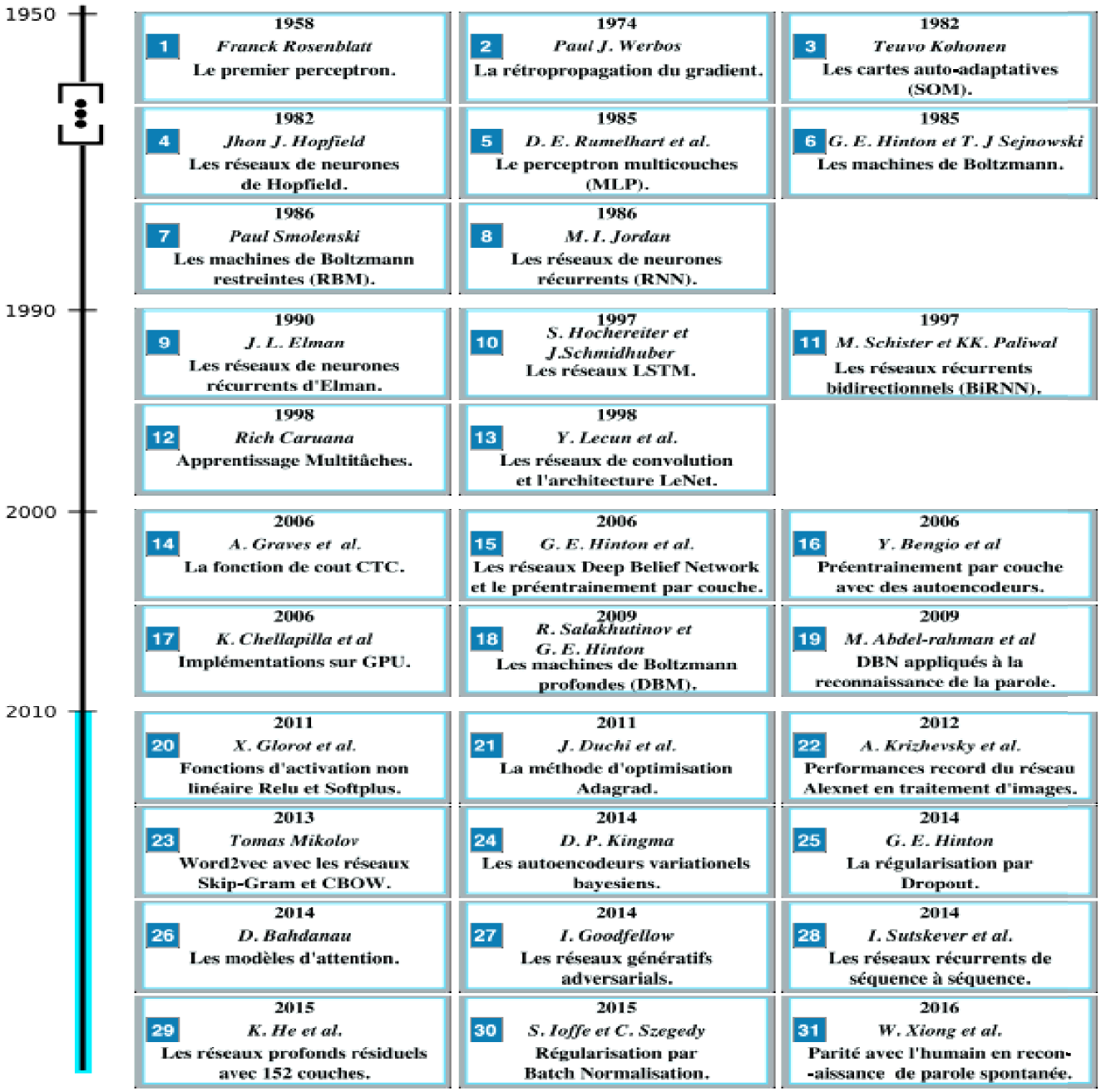

Figure I.5 - Chronologie des modèles les plus marquants des réseaux de neurones **[9]**.

I.4.1 Convolutionnels (CNN):

L'origine des réseaux de neurones convolutifs « CNN » (*Convolutional Neural Network*) est liée aux travaux sur les mécanismes biologiques de la vision. Ainsi on peut remonter dans les années 1950-1960 avec les travaux de caractérisation du cortex visuel de Hubel et Wiesel qui ont identifié deux types de cellules dans le cortex visuel **[12]**, les cellules simples et les cellules

complexes. Les couches de cellules simples sont utilisées pour l'extraction de motifs et les couches de cellules complexes réunissent les motifs reconnus, rendant l'application moins sensible aux variations de position. Ils ont ainsi introduit les concepts de champs réceptifs 2D des neurones et de pooling. Dans les années 1980, la notion de Neocognitron inspirée par les travaux de Hubel et Wiesel, fut introduite par Kunihiko Fukushima **[12]**. La structure Neocognitron est une structure en couches, où celles-ci sont interconnectées en cascade. Ces couches sont composées de cellules simples ou complexes et chacune possède son propre jeu de pondérations tel qu'illustré ci-dessous. Finalement, Neocognitron fut utilisée dans une application de reconnaissance de chiffres. En 1986, la structure du Neocognitron a été réutilisée avec l'idée de partage des mêmes pondérations entre plusieurs neurones d'une même couche associée aux principes de base de l'apprentissage formel par optimisation du gradient, la rétro-propagation du gradient (back-propagation, BP). Ainsi, les bases structurelles des réseaux convolutionnels ont été posées avec une première formalisation d'un algorithme d'apprentissage. Par la suite, en 1998, Yann Lecun et al ont proposé une structure de réseaux de convolutions, LeNet-5 (Figure 1.8), associé à un algorithme d'apprentissage pour la classification de chiffres **[12]**.

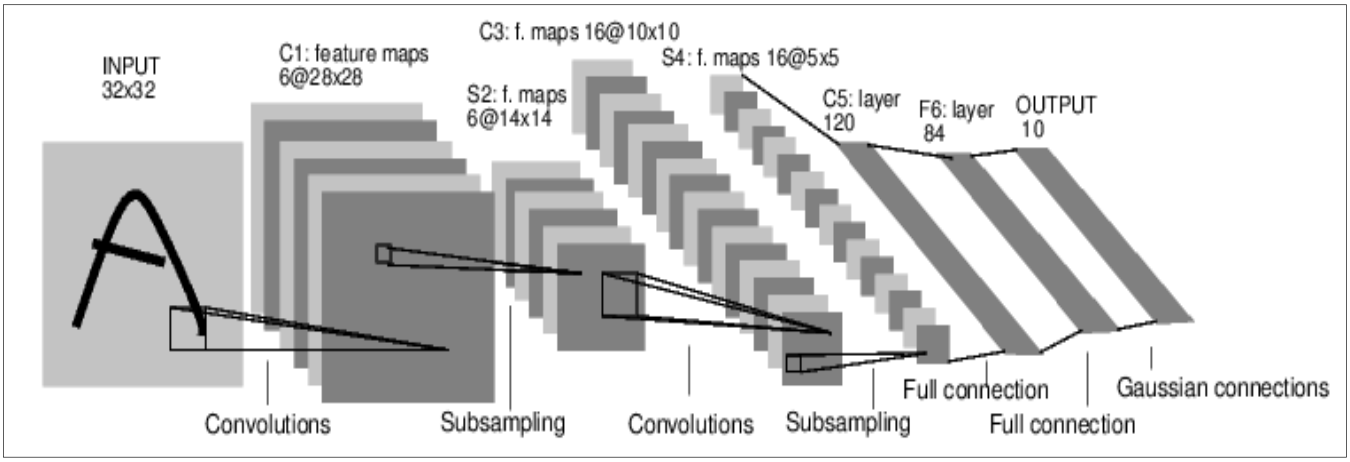

Figure I.6 - Réseaux convolutionnels LeNet-5 **[12]**.

Au final, les CNN sont composés de couches non rebouclées, chaque couche est composée de neurones qui sont connectés à la couche précédente par leur champ récepteur. Il existe aussi plusieurs types de couches. Et dans ces couches un certain nombre de pondérations sont communes à plusieurs neurones.

I.4.2 Récurrents:

Contrairement aux réseaux de neurones non récurrents les réseaux de neurones récurrents dont le graphe de connexions est acyclique, les réseaux de neurones récurrents peuvent avoir une topologie de connexions quelconque, comprenant notamment des boucles qui ramènent aux entrées la/les valeur(s) d'une ou de plusieurs sorties. Pour qu'un tel système soit causal, il faut évidemment qu'à toute boucle soit associé un retard: un réseau de neurones récurrents est donc un

système dynamique, régi par des équations différentielles; comme l'immense majorité des applications sont réalisées par des programmes d'ordinateurs, on se place dans le cadre des systèmes à temps discret, où les équations différentielles sont remplacées par des équations aux différences. Il s'agit donc de réseaux de neurones avec retour en arrière (feedback network or récurrent network), (Figure 1.9) **[10]**.

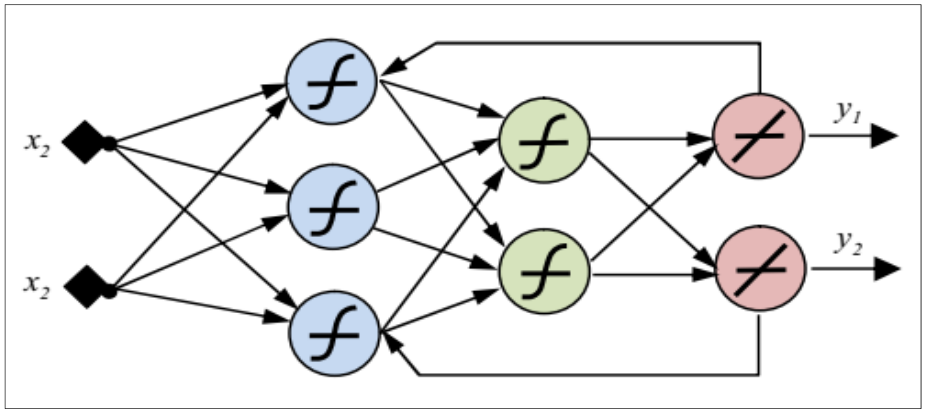

Figure I.7 - Schéma de réseau de neurones récurrent (bouclé) **[10]**.

I.4.2.1 RNN:

Les Réseaux de Neurones récurrents « RNN » (*Recurrent Neural Networks*) traitent l'information en cycle. Ces cycles permettent au réseau de traiter l'information plusieurs fois en la renvoyant à chaque fois au sein du réseau.

La force des Réseaux de neurones récurrents réside dans leur capacité de prendre en compte des informations contextuelles suite à la récurrence du traitement de la même information. Cette dynamique auto-entretient le réseau.

Les Réseaux de neurones récurrents se composent d'une ou plusieurs couches. Le modèle de ***Hopfield*** (réseau temporel) **[10]** est le réseau de neurones récurrent d'une seule couche le plus connu.

Les Réseaux de neurones récurrents à couches multiples revendiquent quant à eux la particularité de posséder des couples (entrée/sortie) comme les perceptrons entre lesquels la donnée véhicule à la fois en propagation en avant et en rétro propagation **[10]** (Figure 1.10).

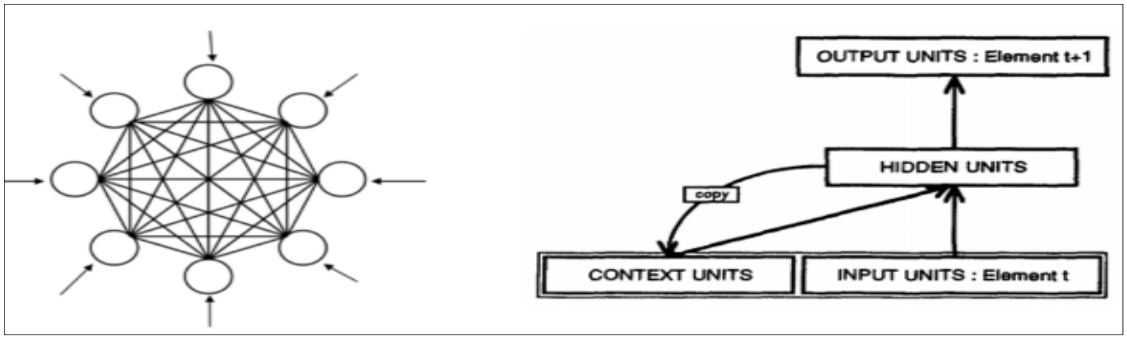

Figure I.8 - le modèle de Hopfield à gauche **[16]** et le modèle de Elman, SRN à droite **[17]**.

I.4.2.2 Cellules LSTM / GRU:

I.4.2.2.1 LSTM:

LSTM, qui signifie *Long Short-Term Memory*, est une cellule composée de trois « portes » : ce sont des zones de calculs qui régulent le flux d'informations (en réalisant des actions spécifiques). On a également deux types de sorties (nommées états) (Figure 1.11).

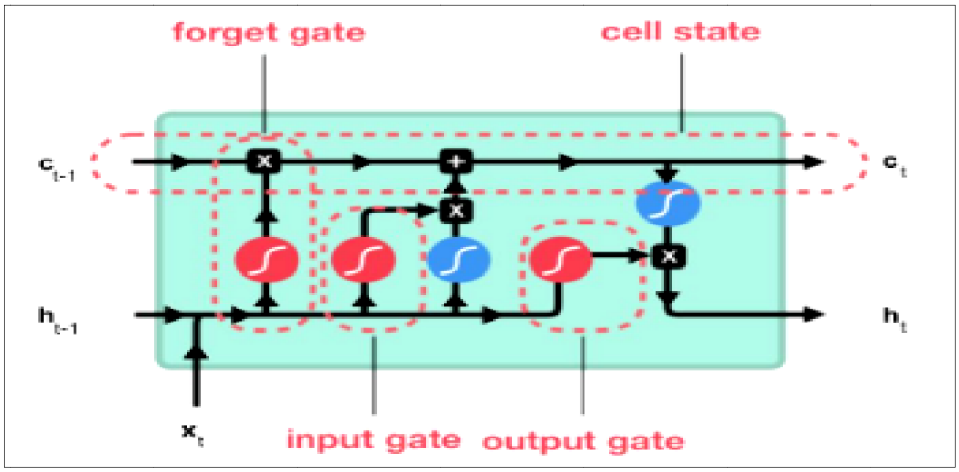

Figure I.9 - Cellule LSTM **[13]**.

Ces opérations dans les portes (gate) permettent au LSTM de **conserver ou supprimer** des informations qu'il a en mémoire. Par exemple, dans notre phrase « Hier soir j'ai mangé un hamburger et des », il est important de retenir les mots « hamburger » et « manger » tandis que les déterminants « un », « et » peuvent être oubliés par le réseau.

Les données stockées dans la mémoire du réseau sont en fait un vecteur noté c_t : l'état de la cellule. Comme cet état dépend de l'état précédent c_{t-1}, qui lui-même dépend d'états encore précédents, le réseau peut conserver des informations qu'il a vu longtemps auparavant (contrairement au RNN classique).

- **Forget gate (porte d'oubli):**

Cette porte décide de quelle information doit être conservée ou jetée : l'information de l'état caché précédent est concaténé à la donnée en entrée (par exemple le mot « des » vectorisé) puis on y applique la fonction sigmoïde afin de normaliser les valeurs entre 0 et 1. Si la sortie de la sigmoïde est proche de 0, cela signifie que l'on doit **oublier l'information** et si on est proche de 1 alors il faut **la mémoriser** pour la suite.

- **Input gate (porte d'entrée):**

La porte d'entrée a pour rôle d'extraire l'information de la donnée courante (le mot « des » par exemple) : on va appliquer en parallèle une sigmoïde aux deux données concaténées (c_f porte précédente) et une tanh.

- **Sigmoïde** va renvoyer un vecteur pour lequel une coordonnée proche de 0 signifie que la coordonnée en position équivalente dans le vecteur concaténé n'est pas importante. A l'inverse, une coordonnée proche de 1 sera jugée « importante » (c.-à-d. **utile pour la prédiction** que cherche à faire le LSTM).
- **Tanh** va simplement normaliser les valeurs (les écraser) entre -1 et 1 pour éviter les problèmes de surcharge de l'ordinateur en calculs.
- Le **produit** des deux permettra donc de ne garder que les informations importantes, les autres étant quasiment remplacées par 0.

- **Cell state (état de la cellule):**

On parle de l'état de la cellule avant d'aborder la dernière porte (porte de sortie), car la valeur calculée ici est utilisée dedans.

L'état de la cellule se calcule assez simplement à partir de la porte d'oubli et de la porte d'entrée : d'abord on multiplie coordonnée à coordonnée la sortie de l'oubli avec l'ancien état de la cellule. Cela permet d'oublier certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, on additionne le tout (coordonnée à coordonnée) avec la sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule ce que le LSTM (parmi les entrées et l'état caché précédent) a jugé pertinent.

- **Output gate (porte de sortie):**

Dernière étape : la porte de sortie doit décider de quel sera le prochain état caché, qui contient des informations sur les entrées précédentes du réseau et sert aux prédictions.

Pour ce faire, le nouvel état de la cellule calculé juste avant est normalisé entre -1 et 1 grâce à tanh. Le vecteur concaténé de l'entrée courante avec l'état caché précédent passe, pour sa part, dans une fonction sigmoïde dont le but est de décider des **informations à conserver** (proche de 0 signifie que l'on oublie, et proche de 1 que l'on va conserver cette coordonnée de l'état de la cellule).

- **Hidden state (état caché):** Même que celui d'un RNN.

Tout cela peut sembler magique en ce sens où on dirait que le réseau doit deviner ce qu'il doit retenir dans un vecteur à la volée, mais rappelons bien qu'une ***matrice de poids*** est appliquée en entrée. C'est cette matrice qui va, concrètement, stocker le fait que telle information est importante ou non à partir des milliers d'exemples qu'aura vu le réseau **[13]**.

I.4.2.2.2 GRU:

GRU, *Gated Recurrent Unit*, pour sa part dispose de deux portes et un état en sortie : (Figure 1.12)

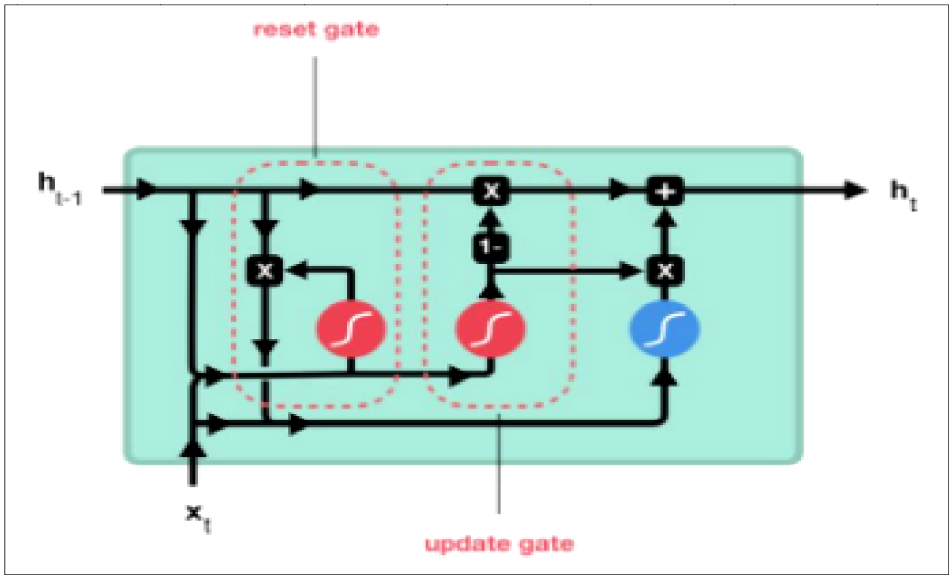

Figure I.10 - Cellule GRU **[13]**.

Une fois que l'on a compris le fonctionnement du LSTM, celui du GRU n'est pas très éloigné (quoique plus simple). A noter l'apparition d'une nouvelle opération qui consiste à inverser un vecteur par rapport à 1 (on calcule 1 – le vecteur). Ceci a pour objectif d'inverser les conclusions d'une porte (par exemple, que tout ce qui a été jugé important devienne anodin et vice versa) : en logique, c'est une porte NOT.

- **Porte de reset (reset gate):**

Cette porte sert à contrôler combien d'information passée au réseau doit oublier. L'état caché précédent, concaténé avec les données d'entrée, passe par une sigmoïde (pour ne conserver que les coordonnées pertinentes) puis est multiplié par l'ancien état caché: on n'en conserve donc que les coordonnées importantes (telles qu'elles) de l'état précédent (on a donc perdu une partie de l'état précédent dans cette porte).

- **Porte de mise à jour (update gate):**

Cette porte agit exactement de la même manière que les portes "oubli" et "d'entrée" du LSTM: elle décide des informations à conserver et de celles à oublier.

Les données d'entrées et l'ancien état caché sont concaténés et passent par une fonction sigmoïde dont le rôle est de déterminer quelles sont les composantes importantes.

- **Sortie du réseau GRU:**

L'état caché précédent (partiellement effacé par la porte de reset) est combiné avec l'entrée du réseau et normalisé par un tanh entre -1 et 1. On vient ensuite annuler toutes ses coordonnées jugées « inutiles pour les prédictions » (grâce à la sortie de la porte de mise à jour), puis on y ajoute les coordonnées de l'état caché précédent jugées « inutiles » (en ayant, cette fois, annulé toutes les coordonnées pertinentes).

Les calculs opérés par le GRU sont plus rapides et plus simples. Notons cependant que les capacités/l'efficacité de ce dernier ne sont plus à prouver, et il est autant utilisé que le LSTM **[13]**.

I.4.3 Génératifs:

Les types ou les modèles génératifs sont des modèles de Machine Learning capables de synthétiser des objets complexes, comme des textes, des images ou de sons, « similaires » à ceux d'une liste d'exemples. En termes un peu plus techniques, on peut dire que les modèles génératifs sont capables d'apprendre une distribution de probabilité sur des objets complexes, distribution que l'on pourra ensuite échantillonner pour produire des exemplaires inédits mais ressemblants aux exemples.

Les applications de ces modèles génératifs sont multiples:

- La création, interactive ou non, d'images ou de vidéos dans un contexte artistique ou marketing.
- La simulation d'évolutions vraisemblables d'environnements physiques en vue de planifier de tâches en robotique ou dans un contexte d'apprentissage par renforcement.
- La mise à l'épreuve de notre aptitude à comprendre la structure d'objets complexes du monde réel en essayant d'en générer des exemplaires crédibles.
- La génération d'images en haute résolution réalistes à partir d'image en basse résolution.
- L'enrichissement d'un jeu de données dans un contexte d'apprentissage semi-supervisé où l'on dispose de peu de données étiquetées.

Ce type se base sur un algorithme qui se contenterait de régurgiter dans le désordre les images fournies en entrée et arguer, avec un zeste de mauvaise foi, que les images ainsi produites sont bien similaires aux exemples. On conçoit par conséquent que, pour être utile, un algorithme génératif se doit d'incorporer une notion de régularité qui lui permettra d'interpoler entre les exemples qu'il a vus pour en générer de nouveaux qui ne soient pas strictement identiques. La définition mathématique rigoureuse de cette notion de similarité reste à ce jour un problème ouvert pour les objets complexes du monde physique. Par chance, les techniques de Deep

Learning sont aujourd'hui en avance sur les développements théoriques. De fait, on sait construire des modèles génératifs qui font l'affaire en pratique, même si nous comprenons encore mal la nature de la régularité qu'ils exploitent implicitement **[13]**.

I.4.3.1 GAN:

Les réseaux antagonistes génératifs, ou GAN (pour *Generative Adversarial Network*), introduit en 2014 par le chercheur américain Ian Goodfellow **[12]**, ils présentent en effet la particularité de pouvoir créer des données (par exemple des images dans le cas des deepfakes). Et ce, sans que leur apprentissage soit supervisé, sont des algorithmes d'apprentissage non supervisé qui permettent de générer des données artificielles avec un fort degré de réalisme.

Les GANs sont des modèles dits génératifs qui diffèrent des techniques traditionnelles d'analyse de données comme la classification. Alors que cette technique vise à apprendre à discriminer les données issues de différentes classes en fonction de leurs descripteurs, les algorithmes génératifs visent à faire le contraire: étant donnée une classe, les GANs cherchent à générer des données qui lui seraient associées.

Les GANs sont composés de deux réseaux de neurones concurrents. Le **générateur**, qui a pour objectif de créer des images aussi réalistes que possible. Et le **discriminateur**, chargé de reconnaître si les images produites par le générateur sont ou non des faux.

Dans les faits, on commence par nourrir le générateur de données comprenant du "bruit" (des images aléatoires), dont il va tenter d'extraire lui-même, en les encodant, certaines règles. Celles-ci lui permettront, en les décodant ensuite, de produire des images. Au départ, le générateur ne va rien produire de très intéressant puisque ses premières images ressembleront au bruit qu'on lui a donné en entrée **[12]** (Figure 1.13).

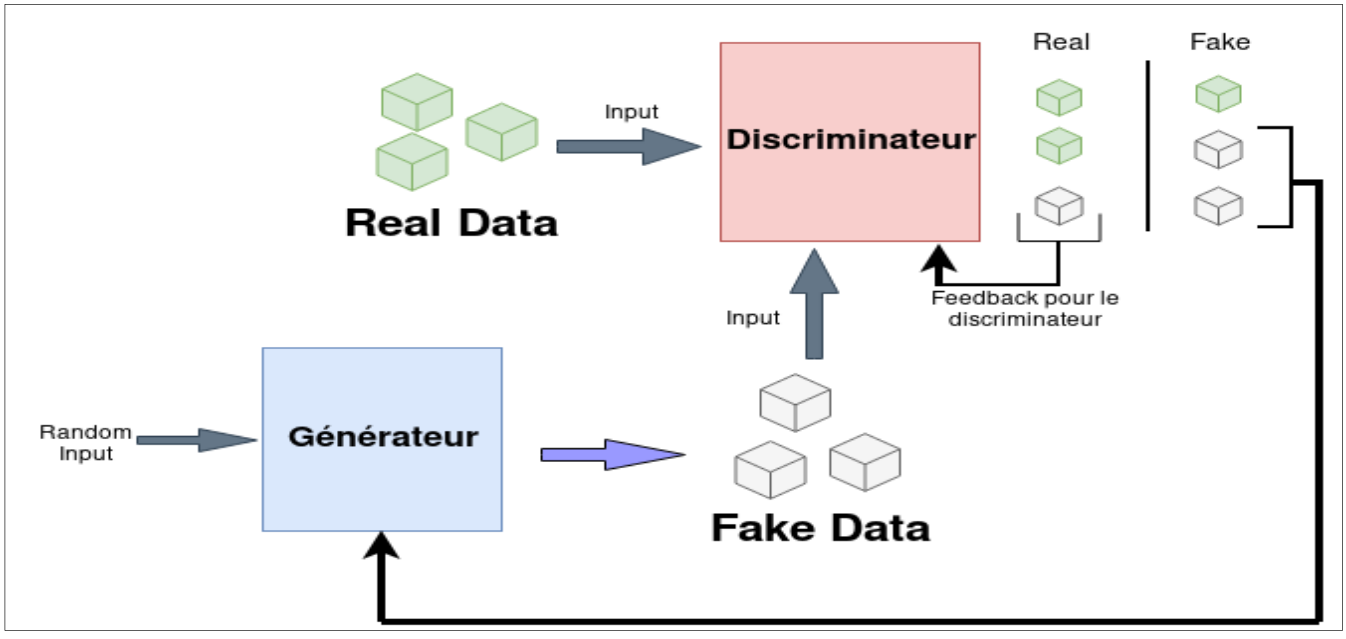

Figure I.11 - feedback pour le générateur (les échantillons ne sont pas assez réalistes) **[12]**.

I.4.3.2 Auto-encodeurs:

Un autoencodeur est une sorte de réseau de neurones utilisé en apprentissage non-supervisé. Contrairement à l'apprentissage supervisé, l'ensemble de données utilisé pour l'entraînement du modèle est composé d'exemples qui n'ont pas de cible. Pour un autoencodeur, l'entrée x devient aussi sa cible. Ce modèle apprend donc à reconstruire son entrée, ce qui a pour but d'apprendre une représentation des données compactes mais riche en information. Cette représentation sera utile pour accomplir certaines tâches ou encore pour faire de la réduction de dimensionnalité. En général, un autoencodeur prend en entrée un vecteur x et calcule une couche cachée avec une fonction d'activation. Ensuite, il construit le vecteur de sortie xb, prédiction de l'entrée x qui devient aussi la cible du modèle (Figure 1.14). Ici x remplace le vecteur cible y. Notons qu'un autoencodeur peut prendre n'importe quelle formes. L'important est que l'entrée soit aussi la cible **[11]**.

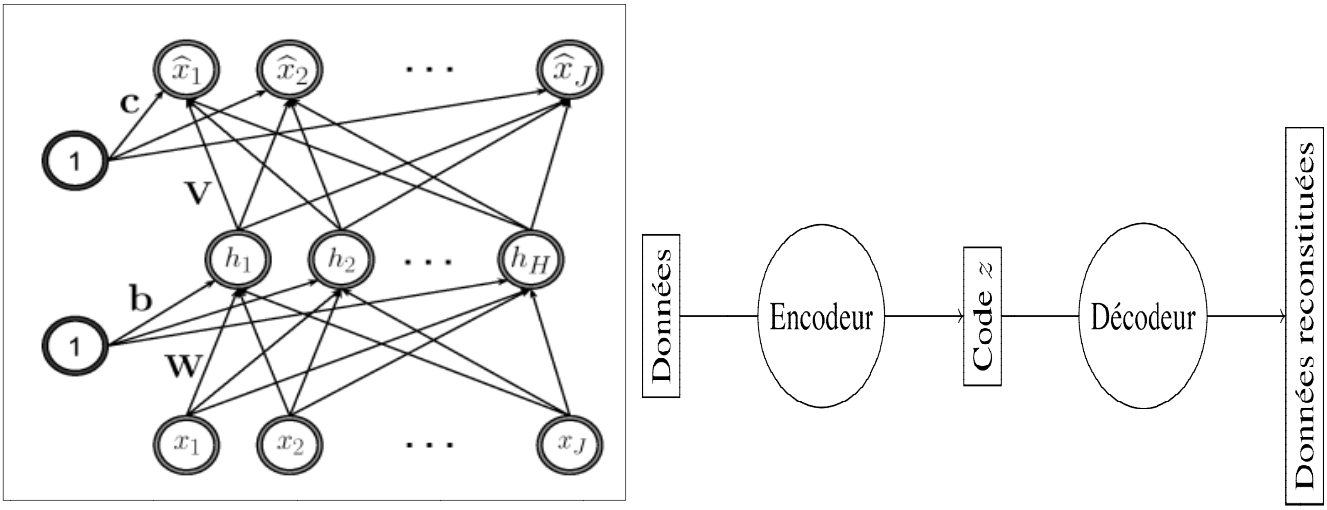

Figure I.12 - Autoencodeur typique et Schéma général d'un autoencodeur **[11]**.

I.4.4 Fonction d'activation:

Jusqu'à présent, nous n'avons pas spécifié la nature de la fonction d'activation. Il se trouve que plusieurs possibilités existent. Différentes fonctions de transfert pouvant être utilisées comme fonction d'activation du neurone sont énumérées à la figure 1.6. Les fonctions d'activations les plus utilisées sont les fonctions «seuil» (en anglais «hard limit»), «linéaire» et «sigmoïde». Comme son nom l'indique, la fonction seuil applique un seuil sur son entrée. Plus précisément, une entrée négative ne passe pas le seuil, la fonction retourne alors la valeur 0 (on peut interpréter ce 0 comme signifiant faux), alors qu'une entrée positive ou nulle dépasse le seuil, et la fonction retourne à 1 (vrai). Utilisée dans le contexte d'un neurone, cette fonction est illustrée à la figure 1.5.a. On remarque alors que le biais *b* dans l'expression de

$y = hard\ lim(w^T x - b)$ (équation I.4) détermine l'emplacement du seuil sur l'axe $w^T x$, où la fonction passe de 0 à 1. Donc cette fonction permet de prendre des décisions binaires.

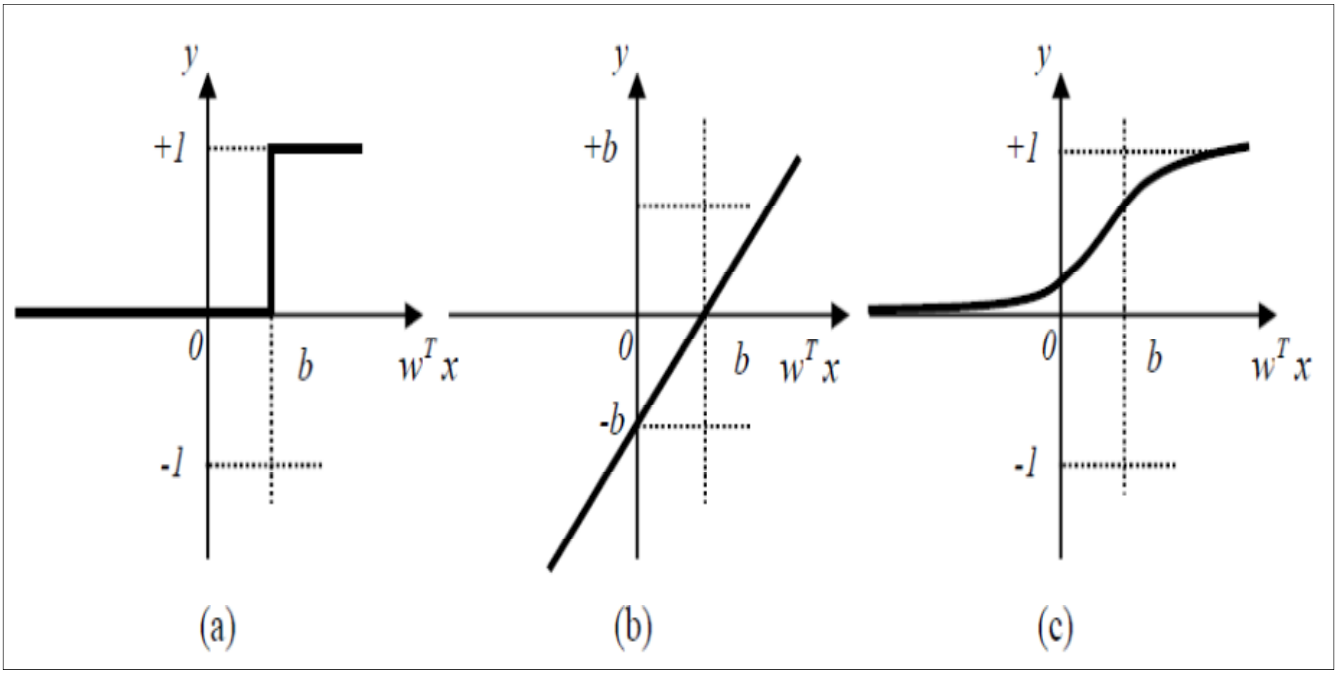

Figure I.13 - Fonctions d'activations: (a) du neurone «seuil» ; (b) du neurone «linéaire», et (c) du neurone «sigmoïde» **[10]**.

La fonction linéaire est très simple, elle affecte directement son entrée à sa sortie :

$$y = s \qquad \textbf{(I.6)}$$

Appliquée dans le contexte d'un neurone, cette fonction est illustrée à la figure 1.5.b. Dans ce cas, la sortie du neurone correspond à son niveau d'activation dont le passage à zéro se produit lorsque :

$$w^T x = b \qquad \textbf{(I.7)}$$

La fonction de transfert sigmoïde est quant à elle illustrée à la figure 1.5.c. Son équation est donnée par :

$$y = \frac{1}{1+ exp^{-s}} \qquad \textbf{(I.8)}$$

Elle ressemble soit à la fonction seuil, soit à la fonction linéaire, selon que l'on est loin ou près de b, respectivement. La fonction seuil est donc non linéaire car il y a une discontinuité lorsque $w^T x = b$. De son côté, la fonction linéaire est tout à fait linéaire. Elle ne comporte aucun changement de pente. La sigmoïde est un compromis intéressant entre les deux précédentes. Notons finalement, que la fonction «tangente hyperbolique (tanh)» est une version symétrique de la sigmoïde **[10]**.

Nom de la fonction	Relation entrée/sortie	Icône
Seuil	$y = 0 \quad si \quad s < 0$ $y = 1 \quad si \quad s \geq 0$	
Seuil symétrique	$y = -1 \quad si \quad s < 0$ $y = 1 \quad si \quad s \geq 0$	
Linéaire	$y = s$	
Linéaire saturée	$y = 0 \quad si \quad s \leq 0$ $y = s \quad si \quad 0 \leq s \leq 1$ $y = 1 \quad si \quad s \geq 1$	
Linéaire saturée symétrique	$y = -1 \quad si \quad s < -1$ $y = s \quad si \quad -1 \leq s \leq 1$ $y = 1 \quad si \quad s > 1$	
Linéaire positive	$y = 0 \quad si \quad s \leq 0$ $y = s \quad si \quad s \geq 0$	
Sigmoïde	$y = \frac{1}{1+exp^{-s}}$	
Tangente hyperbolique	$y = \frac{e^{s}-e^{-s}}{e^{s}+e^{-s}}$	
Compétitive	$y = 1 \quad si \quad s \quad maximum$ $y = 0 \quad autrement$	

Figure I.14 - Différentes formes de la Fonction d'activation **[10]**.

I.4.4.1 Algorithme d'apprentissage de réseaux de neurones:

On s'intéresse à l'algorithme d'apprentissage supervisé (back propagation), cet algorithme d'apprentissage ne peut être utilisé que lorsque les combinaisons d'entrées-sorties désirés sont connues. L'apprentissage est alors facilité et par là, beaucoup plus rapide que pour les deux autres algorithmes (non supervisé et celui par renforcement) puisque l'ajustement des poids est fait directement à partir de l'erreur, soit la différence entre la sortie obtenue par le réseau de neurones et la sortie désirée.

Le réseau apprend à classifier des entrées en s'appuyant sur l'ensemble des exemples avec les réponses corrects associées (plus généralement il s'agit de la régression non-linaire).

La fonction de coût peut être minimisée avec tout algorithme reposant sur le calcul du gradient, quasi-newtonien, ou autre. Le gradient est calculé avec l'algorithme de back propagation. On procède donc de la même manière que pour un réseau non récurrents, à ceci près que les paramètres des différentes copies doivent être identiques (poids partagés) **[6] [8]**.

L'apprentissage d'un réseau de neurones récurrent s'effectue alors de la façon suivante :

1. Initialiser les poids des liens entre les neurones. Souvent une valeur entre 0 et 1, déterminée aléatoirement, est assignée à chacun des poids.
2. Application d'un vecteur entrées-sorties à apprendre.
3. Calcul des sorties du réseau de neurones à partir des entrées qui lui sont appliquées et calcul de l'erreur entre ces sorties et les sorties idéales à apprendre.
4. Correction des poids des liens entre les neurones de la couche de sortie et de la première couche cachée selon l'erreur présente en sortie.
5. Propagation de l'erreur sur la couche précédente et correction des poids des liens entre les neurones de la couche cachée et ceux en entrées.
6. Boucler à la 2^{e} étape avec un nouveau vecteur d'entrées-sorties tant que les performances du réseau de neurones (erreur sur les sorties) ne sont pas satisfaisantes.

I.4.5 Explosion et vanishing gradient:

Le problème du gradient de disparition (gradient de fuite) est un problème qui se pose parfois lors de la formation d'algorithmes d'apprentissage automatique par **descente de gradient** (ce dernier sera détaillé dans la section des algorithmes du chapitre suivant). Cela se produit le plus souvent dans les réseaux de neurones qui ont plusieurs couches neuronales comme dans un système d'apprentissage en profondeur, mais se produit également dans les réseaux de neurones récurrents. Le point clé est que les dérivées partielles calculées permettent de calculer le gradient à mesure que l'on s'enfonce dans le réseau. Étant donné que les gradients contrôlent ce que le réseau apprend pendant la formation, si les gradients sont très faibles ou nuls, peu ou pas de formation peut avoir lieu, conduisant à de mauvaises performances prédictives **[14]**.

En outre, la formation d'un réseau neuronal récurrent (RNN) semble simple car nous ne disposons que d'un ensemble de matrices de poids, mais il est extrêmement difficile en raison de ses connexions récurrentes. Nous pouvons analyser et comprendre la raison de cette dureté en utilisant l'outil de règle de rétropropagation (propagation arrière) et de chaîne. Par exemple, lorsque nous multiplions toutes les matrices de poids dans la propagation vers l'avant, nous devons faire de même dans la propagation arrière. En reculant, le signal peut devenir trop fort ou trop faible; c'est le problème d'explosion ou de disparition du gradient, respectivement. Essentiellement, au fur et à mesure que nous remontons dans le temps, les gradients deviennent plus forts ou plus faibles, par conséquent, pendant quelques étapes, nous n'observons pas le problème d'explosion ou de disparition, mais il apparaît furieusement à mesure que la séquence de temps augmente. Le gradient disparaissant rende difficile de savoir dans quelle direction les paramètres doivent se déplacer pour améliorer les fonctions de perte, tandis que le gradient explosif rende l'apprentissage instable. **[15]**

I.5 Conclusion:

Le domaine de Machine Learning était très vaste et complexe, nous sommes limités à une partie de ce domaine, celle de l'apprentissage automatique par les réseaux de neurones.

Nous constatons ici que les réseaux de neurones avec leur diversité peuvent satisfaire les aspects fixés dans nos objectifs.

Dans la suite de ce manuscrit nous allons nous intéresser particulièrement aux réseaux de neurones récurrents.

CHAPITRE II
DÉTECTION ET PRÉDICTION DES CRIMES

II.1 Introduction:

De plus en plus de services de police ont recours à l'exploitation des données dans les domaines de la gestion de la sécurité et de la réduction du taux de criminalité.

Actuellement les services de police utilisent déjà plusieurs systèmes qui relèvent de l'intelligence artificielle, à savoir la reconnaissance faciale et l'identification des empreintes digitales, néanmoins la prédiction policière reste toujours un domaine de recherche dont l'objectif principal est de développer des machines qui permet de prédire les crimes, en tirant profit des algorithmes de Machine Learning et de l'accessibilité croissante à une diversité de données.

Cet appui de l'intelligence artificielle à travers la prédiction policière peut contribuer efficacement dans la lutte contre la criminalité.

A cet effet, il existe des solutions de prédiction des crimes dans le temps basée sur les techniques issues de **Machine Learning (ML)** et de **statistique** pour déduire le nombre de crimes par type, afin de moderniser l'action policière.

Dans ce qui suit, nous nous présentons un état de lieu sur les techniques basées sur le **Machine Learning** et la **statistique** pour la prédiction des crimes.

II.2 Algorithmes de base:

II.2.1 Régression linéaire (Linear Regression):

Les algorithmes de régression linéaire modélisent la relation entre des variables prédictives et une variable cible. La relation est modélisée par une fonction mathématique de prédiction. Le cas le plus simple est **la régression linéaire univariée**. Elle va trouver une fonction sous forme de droite pour

estimer la relation. **La régression linéaire multivariée** intervient quand plusieurs variables explicatives interviennent dans la fonction de prédiction. Et finalement, **la régression polynomiale** qui permet de modéliser des relations complexes qui ne sont pas forcément linéaires.

II.2.2 Régression logistique (Logistic Regression):

La régression logistique est une méthode statistique inventée afin d'effectuer des classifications binaires. Elle prend en entrée des variables prédictives qualitatives et/ou ordinales et mesure la probabilité de la valeur de sortie en utilisant la fonction sigmoïd.

On peut effectuer la classification multi-classes (par exemple classifier une photo en trois possibilités comme moto, voiture, tramway). En utilisant la régression logistique et la méthode un-contre-tous (One-Versus-All classification).

La régression logistique permettra de répondre à des problèmes comme :

- Est-ce que le client est solvable pour lui accorder un crédit ?
- Est-ce que la tumeur diagnostiquée est bénigne ou maline ?

II.2.3 Machine à Vecteurs de Support (Support Vector Machine - SVM):

Machine à Vecteurs de Support (SVM) est lui aussi un algorithme de classification binaire. Tout comme la régression logistique, elles existent deux classes (Imaginons qu'il s'agit de e-mails, et que les mails Spam sont en rouge et les non spam sont en bleu). Le SVM va opter à séparer les deux classes d'une manière différente à celle de la régression logistique.

Sans entrer dans les détails, et pour des considérations mathématiques, le SVM choisira la séparation la plus nette possible entre les deux classes. C'est pour cela qu'on le nomme aussi "**Large Margins classifier**" (classifieur aux marges larges).

II.2.4 Naïve Bayes:

Naïve Bayes est un classifieur assez intuitif à comprendre. Il se base sur le théorème de Bayes des probabilités conditionnelles.

Naïve Bayes assume une hypothèse forte (naïve). En effet, il suppose que les variables sont indépendantes entre elles. Cela permet de simplifier le calcul des probabilités.

Généralement, le Naïve Bayes est utilisé pour les classifications de texte (en se basant sur le nombre d'occurrences de mots).

II.2.5 Détection d'anomalies (Anomaly Detection):

Détection d'anomalies est un algorithme de Machine Learning pour détecter des patterns anormaux. Par exemple vous receviez dans votre compte en banque 2000€ mensuellement et que un jour vous déposiez 10 000€ d'un coup. L'algorithme détectera cela comme une anomalie.

Cet algorithme est très utile pour la détection de fraudes dans les transactions bancaires, et les détections d'intrusions.

II.2.6 Arbres de décision (Decision Trees):

L'arbre de décision est un algorithme qui se base sur un modèle de graphe (les arbres) pour définir la décision finale. Chaque nœud comporte une condition, et les branchements sont en fonction de cette condition (Vrai ou Faux). Plus on descend dans l'arbre, plus on cumule les conditions.

Il est à noter, qu'il existe aussi un algorithme plus complexe qui se fonde sur l'algorithme de l'arbre de décision, c'est l'algorithme des forêts aléatoires (**Random Forest**).

II.2.7 Réseaux de neurones (Neurals Networks):

Les réseaux de neurones sont inspirés des neurones du système nerveux humains. Ils permettent de trouver des patterns complexes dans les données. Ces réseaux de neurones apprennent une tâche spécifique en fonction des données d'entrainement.

Les réseaux de neurones se composent de nœuds. Dans ces réseaux, on retrouve le tiers d'entrée (Input Layer) qui va recevoir les données d'entrées. L'Input Layer va propager les données par la suite aux tiers cachés (Hidden Layers). Finalement le Tiers de sortie permet de produire le résultat de classification. Chaque tiers du réseau de neurones est un ensemble d'interconnexions des nœuds d'un tiers avec ceux des autres tiers

Parmi les types des réseaux de neurones, en trouve par exemple RNN, CNN, MLP, ST-ResNet, ...etc. (revenir au chapitre précédent pour plus de détails).

II.2.8 k-moyennes (k-Means):

Si vous souhaitiez lancer une campagne publicitaire et que vous vouliez envoyer un message publicitaire différent en fonction du public visé. Vous devez dans un premier lieu regrouper la population ciblée sous forme de groupes. Les individus de chaque groupe auront un degré de similarité (âge, salaire, etc…). C'est ce que fera l'algorithme K-Means !

K-Means est un algorithme de **clustering** en apprentissage non supervisé. On lui donne un ensemble d'éléments (des données), et un nombre de groupes K. K-means va segmenter les éléments en K groupes. Le groupement s'effectue en minimisant la distance euclidienne entre le centre du cluster et un élément donné.

Il est à noter qu'il existe beaucoup d'algorithmes de clustering, à savoir: DBScan (Density-Based Spatial Clustering of Applications with Noise), Mean-Shift Clustering, Expectation–Maximization (EM) Clustering using Gaussian Mixture Models (GMM), Agglomerative Hierarchical Clustering.

II.2.9 K voisins les plus proches (K Nearest Neighbors - KNN):

Il s'agit d'un algorithme d'apprentissage supervisé. Il sert aussi bien pour la classification que la régression. Son fonctionnement peut être assimilé à l'analogie suivante *"dis moi qui sont tes voisins, je te dirais qui tu es…"*.

K voisins les plus proches est un algorithme simple qui stocke tous les cas disponibles et classe les nouveaux cas par un vote majoritaire de ses k voisins. Le cas attribué à la classe est le plus courant parmi ses K voisins les plus proches mesurés par une fonction de distance (ces fonctions de distance peuvent être la distance euclidienne, Manhattan, Minkowski et Hamming).

Pour effectuer une prédiction, l'algorithme K-NN ne va pas calculer un modèle prédictif à partir d'un *Training Set* (jeu de données) comme c'est le cas pour la régression logistique ou la régression linéaire. En effet il effectue des prédictions justes à temps (à la volée) en calculant la similarité entre une observation en entrée et les différentes observations du jeu de données.

II.2.10 Gradient Descent:

Vu son importance, on a ajouté l'algorithme Gradient Descent dans cette liste bien qu'il ne soit pas "vraiment" un algorithme de Machine Learning. En effet, Gradient Descent est un algorithme itératif de minimisation de fonction de coût (ou de perte) dans le sens de la descente la plus raide définie par le négatif du gradient. Cette minimisation servira à produire des modèles prédictifs comme la régression logistique et la régression linéaire.

En apprentissage automatique, nous utilisons la descente de gradient pour mettre à jour les paramètres de notre modèle. Les paramètres font référence aux coefficients dans la régression linéaire et aux poids dans les réseaux de neurones. **[2]**

II.3 Autres méthodes et techniques:

II.3.1 Algorithme de Gradient Boosting:

L'idée de base ressemble à celle du **bagging** (algorithme d'ensemble ancêtre de celui des forêts aléatoires **"Random Forest"** fondé sur les arbres de décision). Plutôt que d'utiliser un seul modèle, nous en utilisons plusieurs que nous agrégeons ensuite pour obtenir un seul résultat. Dans la construction des modèles, le Boosting travaille de manière séquentielle. Il commence par construire un premier modèle qu'il va évaluer. A partir de cette mesure, chaque individu va être pondéré en fonction de la performance de la prédiction. L'objectif est de donner un poids plus important aux individus pour lesquels la valeur a été mal prédite pour la construction du modèle suivant. Le fait de corriger les poids au fur et à mesure permet de mieux prédire les valeurs difficiles.

Ces algorithmes sont utilisés beaucoup plus dans les problématiques de simulation.

II.3.1.1 GBM (Gradient Boosting Machine):

Cet algorithme utilise le **gradient de la fonction de perte** pour le calcul des poids des individus lors de la construction de chaque nouveau modèle. Cela ressemble un peu à la descente de gradient pour les réseaux de neurones.

GBM utilise généralement des arbres CART « Classification And Regression Trees » (cette dernière fondées sur l'algorithme d'arbre de décision) et on peut personnaliser l'algorithme en utilisant différents paramètres, différentes fonctions pour faire une prédiction avec une puissance de prédiction élevée.

II.3.1.2 XGBoost (eXtreme Gradient Boosting):

XGBoost a un pouvoir prédictif extrêmement élevé, ce qui en fait le meilleur choix pour la précision des événements car il possède à la fois un modèle linéaire et l'algorithme d'apprentissage d'arbre, ce qui rend l'algorithme presque plus rapide que les techniques existantes de rappel de gradient.

II.3.1.3 LightGBM:

LightGBM est basé sur des algorithmes d'arbre de décision, il divise la feuille d'arbre avec le meilleur ajustement tandis que d'autres algorithmes de stimulation divisent la profondeur de l'arbre ou le niveau plutôt que la feuille. Ainsi, lors de la croissance sur la même feuille dans LightGBM, l'algorithme feuille par feuille peut réduire plus de pertes que l'algorithme par niveau et par conséquent, donne une bien meilleure précision qui peut rarement être obtenue par l'un des algorithmes de boosting existants. En outre, il est étonnamment très rapide, d'où le mot «Light».

II.3.1.4 Catboost (CATegory BOOSTing):

Catboost peut traiter automatiquement les variables catégorielles (une variété de formats) sans afficher l'erreur de conversion de type, ce qui vous aide à vous concentrer sur l'optimisation de votre modèle plutôt que sur le tri des erreurs banales, ainsi qu'il ne nécessite pas de formation approfondie sur les données comme les autres modèles de Machine Learning.

II.3.1.5 Logit boost (LOGIsTic BOOST):

L'algorithme Logit boost a été conçu comme une solution alternative pour remédier aux limites d'Adaboost dans la gestion du bruit et des valeurs aberrantes. Le Logit boost utilise log-likelihood binomiale qui modifie la fonction de perte de façon linéaire. En revanche, Adaboost utilise une fonction de perte exponentielle qui change exponentiellement avec l'erreur de classification. C'est la raison pour laquelle Logit boost a tendance à être moins sensible aux valeurs aberrantes et au bruit.

Logit Boost est une variante de Boosting populaire qui peut être appliquée à la classification binaire ou à plusieurs classes. D'un point de vue statistique, Logit Boost peut être considéré comme une régression additive d'arbre en minimisant la perte logistique.

II.3.1.6 Cluster Confidence Rate Boosting (CCRBoost):

Le CCRBoost commence par un multi-clustering suivi de processus d'apprentissage des fonctionnalités (caractéristiques) locales pour découvrir tous les modèles distribués possibles à partir de distributions de différentes formes, tailles et périodes. L'étiquette de classification finale est produite en utilisant des groupements des modèles distribués les plus appropriés.

II.3.2 Algorithmes de réduction de dimensionnalité:

La réduction de dimension fait référence au processus de conversion d'un ensemble de données ayant de vastes dimensions en données avec des dimensions moindres garantissant qu'elles véhiculent des informations similaires de manière concise. Ces techniques sont généralement utilisées lors de la résolution de **problèmes d'apprentissage automatique** pour obtenir de meilleures fonctionnalités pour une tâche de classification ou de régression (l'une des applications les plus courantes de cette technique est le **traitement d'image**) en se basant sur divers méthodes dont on a illustré auparavant.

II.3.3 Algorithmes génétiques (AG):

Les algorithmes génétiques (AGs) sont des algorithmes d'optimisation stochastique fondés sur les mécanismes de la sélection naturelle et de la génétique. Les AGs sont basés sur des mécanismes très simples (manipulations élémentaires de chaînes binaires).

Ils sont robustes car ils peuvent résoudre des problèmes fortement non-linéaires et discontinus, et efficaces car ils font évoluer non pas une solution mais toute une population de solutions potentielles. Ils bénéficient donc d'un parallélisme puissant.

Dans le cas de l'apprentissage, ce dernier augmente considérablement l'efficacité de l'apprentissage et ces performances.

II.3.4 Algorithmes Q-Learning (Quality-LEARNING):

Le Q-Learning est un algorithme d'apprentissage **basé sur des valeurs**. Les algorithmes basés sur la valeur mettent à jour la fonction de valeur basée sur une équation (en particulier l'équation de Bellman). Alors que l'autre type, **basé sur la politique**, estime la fonction de valeur avec une politique gourmande obtenue à partir de la dernière amélioration de politique.

II.4 Travaux connexes:

On présente dans cette section, trois (3) travaux connexes à notre travail, et qui utilisent des techniques de prédiction différentes (le réseau de neurones, l'arbre de décision, le k-means, le DBScan et la régression linéaire).

On conclut cette section, par un quatrième travail connexe qui est considéré comme un travail de critique entre les différentes techniques utilisées pour réaliser un prédicteur de crimes.

II.4.1 Arbre de décision, k-means et DBScan:

Commençant par le premier travail, c'est celui de "Baboo, S. S. (2011). An enhanced algorithm to predict a future crime using data mining"**[21]**.

Il utilise la technique d'arbres de décision hybridée avec les algorithmes de clustering, ce qui a donné une prédiction de crimes de 86% sur une base de données d'un centre de recherche indien, contenant des crimes commis en inde dans la période entre 2001 et 2009, puis il compare leur résultat à un modèle basé uniquement sur l'arbres de décision.

Le prédicteur proposé utilise une seule fonctionnalité ou caractéristique (feature), c'est le nombre de crime dans le temps "ALL CRIMES" afin de déduire le taux de criminalité en hybridant les techniques k-means et DBScan (clustering).

Cette approche est décrite comme suit, étant donné un ensemble d'objets, le clustering est le processus de découverte de classe, où les objets sont regroupés en clusters et les classes sont inconnues

au préalable. Deux techniques de clustering, K-means et DBScan (Density-Based Spatial Clustering Application with Noise) sont envisagées à cet effet.

L'algorithme hybride est donné comme suit:

- L'algorithme hybride regroupe les données "**m**" groupes où **m** est prédéfini.

- Entrée : Type de crime, nombre de grappes, nombre d'itérations.

- Les échantillons initiaux peuvent jouer un rôle important dans le résultat final.

- Étape 1: Choisissez au hasard les centres de cluster.

- Étape 2: attribuer des instances aux clusters en fonction de leur distance par rapport aux centres du cluster.

- Étape 3: les centres des groupes sont ajustés.

- Étape 4: passez à l'étape 1 jusqu'à la convergence.

- Étape 5: sortie C0, C1, C2, C3 où C_i sont les classes de types des crimes constatés.

À partir du résultat du regroupement, la tendance de la criminalité pour chaque type de crime a été identifiée pour chaque année. De plus, légèrement modifiant les groupes de clustering. À partir de ces groupes homogènes on peut mesurer les tendances et la méthode utilisée est la suivante:
- Fonction de sortie du taux de criminalité = 1 / taux de criminalité.

L'évaluation finale de ce prédicteur est meilleure par comparaison à un autre qui utilise un algorithme basé uniquement sur l'arbre de décision qui donne seulement une prédiction de crimes de 84% au lieu de 86% avec hybridation avec K-means et DBScan.

II.4.2 Régression linéaire:

En outre, le deuxième travail, c'est celui de " Shingleton, J. S. (2012). Crime trend prediction using regression models for salinas, california"**[22]**.

Il utilise la technique de la régression linéaire, particulièrement trois 03 méthodes de la régression linéaire multivariée (la régression négative binomial, la régression OLS " Ordinary Least Squares" et la régression de poisson) sur une base de donnée du Département de Police de Salinas (SPD)-USA, contenant des crimes concernant cette zone dans la période entre 1980 et 2011.

Cette étude évalue et compare les modèles statistiques issues de la régression linaire à savoir la régression négative binomiale, la régression des moindres carrés ordinaires et la régression de poison

qui utilisent des données et des variables environnementales préalablement établies (multi-variables: nombre de crime et type de crime) pour prédire les tendances de la criminalité.

Des modèles de régression ont été créés pour prédire les niveaux de crimes futurs à l'aide techniques d'analyse. Un programme a également été écrit pour permettre la régression pour une exploration et une analyse plus poussées des données environnementales de Salinas, et ceci en lui passant le nombre et le type de crime ainsi que le type d'algorithme de régression linaire à appliquer sur ces derniers.

Les prédicteurs proposés utilisent deux 02 fonctionnalités ou caractéristiques (features), c'est le nombre de crime dans le temps "ALL CRIMES" et type de crime "CRIME TYPE" (seulement 02 types de crimes).

L'évaluation finale des résultats de prédictions des trois algorithmes, utilise la métrique d'évaluation **RSE** (**E**rreur **S**tandard **R**ésiduelle) par comparaison de valeurs prédites aux observations réelles, où le premier modèle linéaire a donné une prédiction plus proche des taux de criminalité que les deux autres modèles.

II.4.3 Réseau de neurones:

Concernant le troisième travail jointé en référence dans la fiche de présentation de notre Projet de Fin d'Etude, c'est celui de "Krishnan, A., Sarguru, A., & Sheela, A. S. (2018). Predicitve analysis of crime data using deep learning"**[20]**.

Il utilise la technique des réseaux de neurones, particulièrement les réseaux de neurones récurrents (Simple RNN et LSTM avec ReLU comme fonction d'activation) sur une base de données du ministère de la justice indienne, contenant des crimes concernant le code pénal indien et les lois spéciales et locales dans la période entre 2011 et 2018.

Le réseau de neurones récurrent de type LSTM (Long Short-Term Memory) est utilisé avec l'architecture suivante:

- La fonction d'activation : Relu (Rectified Linear Unités).
- La fonction de perte: MSE (Mean Squared Error).
- L'optimiseur: Adams.
- La métrique d'évaluation: RMSE (Root Mean Squared Error).
- Les époques = 200, batch_size = 2, Verbose = 1, ...etc, ont été fixés expérimentalement.
- La validation: est par Holdout.
- L'ensemble de formation et de test est 80% et 20% respectivement du DataSet.

Les modèles prédictifs proposés utilisent une seule fonctionnalité ou caractéristique (feature), c'est le nombre de crimes "ALL CRIME" dans le temps.

La précision de ce modèle proposé est comparée à un modèle basique (simple RNN) où on peut conclure que ce modèle est meilleur en offrant une meilleure efficacité que le modèle basique et ceci en basant sur le calcule de la racine carrée de l'erreur quadratique moyenne (RMSE) des prédictions.

II.5 Étude comparative de différents modèles:

Stalidis & Al **[23]** ont mené une étude comparative de 10 méthodes de pointe contre 03 configurations de Réseau de Neurones récurrents est menée, dont les expériences ont été effectuées avec cinq (05) ensembles de données accessibles au public (cité dans le tableau II.1 ci-après).

II.5.1 Statistiques de base pour les 5 ensembles de données examinés:

N°	BASE DE DONNÉES	ANNÉE DE DÉBUT	ANNÉE DE FIN	NOMBRE D'INCIDENTS
1.	Philadelphia	2006	2017	2.203.785
2.	Seattle	1996	2016	684.472
3.	Minneapolis	2010	2016	136.121
4.	DC Metro	2008	2017	313.410
5.	San Francisco	2003	2015	878.049

Tableau II.1 - Liste de DataSet **[23]**.

II.5.2 Les 10 Techniques de pointes de comparaison:

Les techniques utilisées dans cette étude sont illustrées dans le tableau ci-dessous, dont les critères d'évaluation en trouvent: MSE (Mean Squared Error), MCCE (Multi-Class Cross Entropy), BCE (Binary Cross Entropy), F1Score, AUCPR (Area Under Curve the Precision-Recall), AUROC (Area Under Curve-Receiver Operating Characteristic) et PAI (Prediction Accuracy Index) qui seront discutés dans la section (II.3.4 Évaluation) dans ce chapitre.

N°	ALGORITHME	OBSERVATIONS
1.	CCRBoost	Algorithme de clustering utilisé plus dans la classification.
2.	Decision Trees	Arbre de décision déjà vu dans le premier exemple.
3.	Naive Bayes	Algorithme issu de la statistique utilisé plus dans la classification.
4.	Logit Boost	Algorithme de boosting moins sensible aux bruits et aux valeurs aberrantes utilisé plus dans la classification binaire.
5.	SVM	Algorithme classifieur aux marges larges utilisé plus dans la classification binaire.
6.	Random Forests	Algorithme issu des arbres de décision.
7.	KNN	Algorithme d'apprentissage supervisé utilisé plus dans la classification.
8.	MLP (150)	MLP avec une couche cachée de 150 neurones, type de réseau de neurones (Multilayer Perceptron).
9.	MLP (150, 300, 150, 50)	MLP avec quatre couches cachées de 150, 300, 150 et 50 neurones chacune, type de réseau de neurones (Multilayer Perceptron).
10.	ST-ResNet	Type de réseau de neurones (Spatio-Temporal Residual Networks).

Tableau II.2 - Liste d'algorithmes **[23]**.

II.5.3 Description:

Cette étude avait couvrir deux (02) caractéristiques [nombre de crime dans le temps "ALL CRIMES" et type de crime "CRIME TYPE" (10 types de crimes)] en fixant préalablement les régions, dont les critères d'évaluations ont été réalisées en comparant les avec quatre 04 métriques d'évaluation différentes.

II.5.4 Évaluation:

Tableau des évaluations pour les caractéristiques "ALL CRIMES" et "CRIME TYPE" en utilisant quatre 04 métriques d'évaluation différentes:

Les métriques d'évaluation:

Parmi les métriques d'évaluation utilisées dans cette étude, l'évaluation finale des modèles prédictifs est basée seulement sur les **04** métriques suivantes:

1. F1score : est une mesure de la précision d'un test. Il est calculé à partir de la précision et du rappel du test, où la précision est le nombre de résultats positifs correctement identifiés divisé par le nombre de tous les résultats positifs, y compris ceux qui ne sont pas correctement identifiés, et le rappel est le nombre de résultats positifs correctement identifiés divisé par le nombre de tous les échantillons qui auraient dû être identifiés comme positifs.

Donc F1score est la moyenne harmonique de la précision et du rappel. La valeur la plus élevée possible de F 1 est 1, indiquant une précision et un rappel parfaits, et la valeur la plus basse possible est 0, si la précision ou le rappel est égal à zéro. Le F1score est également connu sous le nom de coefficient de similarité de dés.

2. AUCPR : Area Under Curve the Precision-Recall (Précision moyenne)

Afin de définir AUCPR, on doit définir quelle courbe Précision-Rappel. C'est une courbe qui combine la précision "valeur prédictive positive " (PPV) et le rappel "taux de vrais positifs" (TPR) dans une seule visualisation. Pour chaque seuil, en calculant PPV et TPR et les tracez. Plus la courbe est élevée sur l'axe y, les performances de votre modèle sont meilleures.

On peut utiliser ce graphique pour prendre une décision éclairée en ce qui concerne le dilemme classique précision / rappel. Évidemment, plus le rappel est élevé, plus la précision est faible. Savoir à quel rappel votre précision commence à chuter rapidement cela peut aider pour choisir le seuil et pour fournir un meilleur modèle.

L'AUCPR peut être considéré comme la moyenne des scores de précision calculés pour chaque seuil de rappel. On peut également ajuster cette définition en fonction des besoins en choisissant / en découpant les seuils de rappel si nécessaire.

3. AUCROC : Area Under Curve - Receiver Operating Characteristic

AUC signifie l'aire sous la courbe ROC pour ainsi dire sur le score ROC AUC, où la courbe ROC c'est un graphique qui visualise le compromis entre le taux de vrais positifs (TPR) et les taux de faux positifs (FPR). Fondamentalement, pour chaque seuil, on calcule le TPR et le FPR et le représentons sur un graphique.

Le TPR le plus élevé et le FPR le plus bas sont meilleurs pour chaque seuil et donc les classificateurs/prédicteurs qui ont des courbes plus en haut à gauche sont meilleurs.

Afin d'obtenir un nombre qui nous indique la qualité de notre courbe, on peut calculer l'aire sous la courbe ROC ou le score AUCROC. Plus votre courbe est en haut à gauche, plus l'aire est élevée et donc le score AUCROC plus élevé.

Alternativement, le score AUCROC équivaut au calcul de la corrélation de rang entre les prédictions et les cibles. Du point de vue de l'interprétation, elle est plus utile car elle indique à quel point le modèle est bon pour classer les prédictions. Il indique aussi quelle est la probabilité qu'une instance positive choisie au hasard soit mieux classée qu'une instance négative choisie au hasard.

4. PAI : Prediction Accuracy Index

Utiliser pour la mesure de l'efficacité des prévisions avec l'équation suivante:

$$\mathrm{PAI} = \frac{\frac{r}{R}}{\frac{a}{A}}$$

Où "r" est le nombre de crimes qui se produisent dans une zone de prévision examinée, "R" est le nombre total de crimes sur toute la carte, "a" est la taille de la zone de prévision et "A" est la taille de la zone de toute la carte à l'étude. La métrique **PAI** offre un aperçu utile de l'efficacité de la prévision produite par chaque méthode, car elle mesure non seulement le nombre de prédictions correctes, mais prend en compte l'importance de chaque prédiction, ce qui représente objectif ultime. Cette métrique dépend fortement du pourcentage de la surface totale qui devrait être examinée. Afin de simuler des conditions pragmatiques où un organisme d'application de la loi a une limite aux ressources disponibles pour lutter contre la criminalité, il est nécessaire de limiter à **5%** la zone maximale qui peut être prédite comme zone criminelle de la surface totale et signalons cette métrique comme **PAI @ 5**

Zone a examinée :

La zone (région) présente la surface a examinée, c'est la taille de la grille de la zone sur la carte, elle variée entre les valeurs suivante [16-24-32-40].

Le tableau suivant illustre les résultats de cette étude:

N°	Taille Zone / ALGORITHME	F1score				AUCPR				AUCROC				PAI@5			
		16	24	32	40	16	24	32	40	16	24	32	40	16	24	32	40
1.	**CCRBoost**	0.93	0.92	0.88	0.88	0.99	0.99	0.98	0.97	0.92	0.92	0.90	0.91	1.87	1.94	1.86	2.00
2.	**Decision Trees**	0.93	0.92	0.90	0.89	0.99	0.97	0.97	0.96	0.85	0.80	0.78	0.79	0.59	1.51	1.54	1.80
3.	**Naive Bayes**	0.92	0.87	0.84	0.83	0.98	0.98	0.98	0.97	0.82	0.89	0.86	0.88	1.21	2.82	2.00	1.93
4.	**Logit Boost**	0.94	0.92	0.91	0.89	1.00	0.99	0.99	0.99	0.99	0.96	0.94	0.94	0.81	1.73	2.52	1.91
5.	**SVM**	0.94	0.93	0.91	0.90	0.99	0.99	0.98	0.98	0.91	0.93	0.90	0.90	0.06	0.12	0.19	0.25
6.	**Random Forests**	0.94	0.92	0.90	0.89	1.00	0.99	0.98	0.98	0.97	0.94	0.91	0.92	1.04	2.44	1.93	1.98
7.	**KNN**	0.94	0.92	0.91	0.89	0.99	0.98	0.97	0.97	0.86	0.87	0.84	0.88	0.93	1.66	1.89	2.29
8.	**MLP (150)**	0.94	0.92	0.90	0.89	0.98	0.99	0.98	0.98	0.84	0.94	0.91	0.92	2.21	2.09	2.30	2.95
9.	**MLP (150,300,150,50)**	0.94	0.92	0.90	0.89	0.97	0.99	0.98	0.98	0.79	0.91	0.91	0.90	1.31	2.10	2.15	2.70
10.	**ST-ResNet**	0.91	0.88	0.86	0.82	0.99	0.99	0.99	0.99	0.98	0.96	0.96	0.96	3.30	1.62	2.55	2.56
11.	**Modèle 1**	**0.99**	**0.97**	**0.96**	**0.94**	**1.00**	**1.00**	**1.00**	**0.99**	**0.99**	**0.98**	**0.97**	**0.97**	4.02	4.34	**4.14**	**4.33**
12.	**Modèle 2**	0.99	0.97	0.95	0.94	1.00	0.98	0.99	0.99	0.95	0.81	0.93	0.96	4.30	**4.40**	4.10	4.12
13.	**Modèle 3**	0.99	0.96	0.94	0.92	0.99	0.99	0.99	0.99	0.88	0.91	0.94	0.94	**4.32**	3.41	3.29	3.31

Tableau II.3 - Évaluation des exemples **[23]**.

Le modèle optimal parmi les 03 configurations de réseaux de neurones est celui qui utilise des couches **RNN**, **LSTM** et **ReLU** comme fonction d'activation (**Modèle 1**, dont **p** est la taille de grille de région déjà fixée, varie entre [16, 24, 32, 40] qui sert comme un paramètre dans les 3 modèles [Modèle1, Modèle2 et Modèle 3]).

REMARQUE:

En pratique, il existe une multitude d'application. Certaines d'entre elles sont de puissantes applications commerciales bien connues, parmi lesquelles on trouve **PredPol**, **Palantir** ou encore **HunchLab**.

Le leader du marché de la prédiction policière, l'entreprise californienne américaine **PredPol**, fondateur du premier logiciel prédicteur de crimes commercialisé et ce depuis **2012**, utilisé par plus de 60 départements de police outre-Atlantique, s'appuie sur un algorithme de prédiction des séismes (les crimes se répliquent plus ou moins aux mêmes endroits et peuvent donc être anticipés) ce dernier qui utilise les réseaux de neurones comme Machine Learning. **[24]**

II.6 Conclusion:

Nous avons vu dans ce chapitre que les algorithmes de Machine Learning servaient à deux choses : classifier et prédire et qu'ils se divisaient en algorithmes supervisés et non supervisés.

Il n'y a pas de modèles meilleurs que d'autres, tout dépend de ce qu'on veut en faire, cependant les réseaux de neurones récurrents présentent davantage de précision relatif aux problèmes de régression et de prédiction par rapport aux autres méthodes et algorithmes d'apprentissage automatique standard utilisés dans la littérature pour la prédiction policière. Nous allons de ce fait explorer plus l'utilisation et l'exploitation de ces architectures sur une base de données criminelle réelle. C'est ce qui va être détaillé dans le prochain chapitre.

CHAPITRE III

ANALYSE ET CONCEPTION

III.1 Introduction:

L'analyse des phénomènes criminels fait appel à des méthodes statistiques traditionnelles pour comprendre les tendances criminelles, tels que les taux des crimes commis pendant le temps, par régions, par tranches d'âges et par fonctions,…etc.

Néanmoins, ces méthodes ne permettent pas de prédire le nombre des crimes qui peuvent survenues dans le futur, afin de mieux prendre les décisions idoines pour lutter contre ce fléau, en adaptant les plans d'actions préventifs, optimisant les ressources nécessaires.

Dans cette intention, un prédicteur de crimes dans le temps via les réseaux de neurones récurrents est proposé pour prédire les crimes en nombre et en type, afin de rationaliser l'effort de lutte contre la criminalité.

Dans la suite, on va décrire notre constat sur lequel on a défini l'architecture de notre prédicteur, puis la collecte et le prétraitement des données utilisés pour construire le DataSet du prédicteur en passant par la description du comportement et de fonctionnement de ce dernier à travers les diagrammes **UML**[1] et on se termine par les critères de choix de notre modèle prédictif.

[1] Unified Modeling Language.

III.2 Constat:

La problématique de notre travail s'inscrit dans la modélisation prédictive de régression (**Regression Predictive Modeling**) qui consiste à approximer une fonction de mappage (f) des variables d'entrée (X) à une variable de sortie continue (y) [y = f(X)].

Une variable de sortie continue est une valeur réelle, telle qu'une valeur entière ou à virgule flottante. Ce sont souvent des quantités ou des mesures.

Dans notre cas, le problème de régression tente d'estimer la fonction de mappage (f) des variables d'entrée (x) aux variables de sortie numériques ou continues (y) **[y = f(X)]**.

La régression se caractérise par:

- Un problème de régression nécessite la prédiction d'une quantité.
- Une régression peut avoir des variables d'entrée réelles ou discrètes.
- Un problème avec plusieurs variables d'entrée est souvent appelé un problème de régression multi-variée.
- Un problème de régression où les variables d'entrée sont ordonnées par le temps est appelé un problème de prévision de séries chronologiques (*c'est ici où s'enregistre exactement notre problématique*).

Étant donné qu'un modèle prédictif de régression prédit une quantité, la compétence du modèle doit être signalée comme une erreur dans ces prédictions.

Il existe de nombreuses façons d'estimer la compétence d'un modèle prédictif de régression, mais la plus courante est peut-être de calculer "l'erreur quadratique moyenne, abrégée par l'acronyme RMSE" ou de calculer "l'erreur absolue moyenne, abrégée par l'acronyme MAE".

En outre, la quantification et l'analyse statistique de la criminalité dans le temps constitue des lignes de données ordonnées par le temps sous forme de **séries chronologiques** ou **temporelles** (nommées aussi "**données séquentielles**"), d'où l'apparence de l'intérêt primordial des réseaux de neurones à travers le modèle **"Sequential"**.

III.3 Situation:

Nous avons travaillé sur une base de données réelle, en se focalisant sur "**la table CRIME**", contenant **6.932.403** enregistrements dans la période entre **2001** et **2018**, qui représentent **63** types de

crimes, dont **6.739.035** enregistrements reviennent seulement pour **24** types de crimes parmi les **63** (**ces 24 types sont les crimes les plus commis, qui seront l'objet de notre travail**).

Table CRIME : contienne les informations essentielles du crime, c'est une table dynamique (avec clé étrangère) (gain de taille, non redondance et cohérence des données), ces clés étrangères sont référencés vers des tables statiques ou tables de références (sans clé étrangère) dont le nombre est 30 tables statiques.

La **table CRIME** se compose de **40** champs concernant le crime et **13** champs systèmes, à savoir: trace de modification (@IP, user, time, journal, requête SQL, ...etc.), ces derniers ont été éliminés du traitement et de l'exploitation dans ce travail (champs non significatifs pour notre travail).

Le type des champs concernant le crime est détaillé ci-dessous:

N°	CHAMPS	TYPE
1.	CASE_NO	NUMBER
2.	TYPE_CRIME	VARCHAR
3.	CASE_DATE	DATE
4.	CAT	VARCHAR
5.	INSCRIPTION_NO	NUMBER
6.	INSCRIPTION_DATE	DATE
7.	AGE	DATE
8.	COMMUNITY_AREA	VARCHAR
9.	STATE	VARCHAR
10.	GENDER	VARCHAR
11.	CODE_NAT	VARCHAR
12.	DOMESTIC	VARCHAR
13.	ARREST_LOCATION	VARCHAR
14.	ARRESTED	VARCHAR
15.	OLD_CRIME	VARCHAR
16.	STATUS	VARCHAR
17.	REF_MRF1	VARCHAR
18.	REF_MRF2	NUMBER
19.	REF_MRF3	VARCHAR
20.	REF_MRF4	VARCHAR
21.	REF_MRF5	VARCHAR
22.	REF_MRF6	VARCHAR
23.	REF_MRF7	VARCHAR
24.	REF_MRF8	VARCHAR
25.	REF_MRF9	VARCHAR
26.	REF_MRF10	VARCHAR
27.	REF_MRF11	VARCHAR
28.	REF_MRF12	VARCHAR
29.	REF_MRF13	VARCHAR
30.	REF_MRF14	VARCHAR
31.	REF_MRF15	VARCHAR
32.	REF_MRF16	VARCHAR
33.	REF_MRF17	VARCHAR
34.	REF_MRF18	VARCHAR
35.	REF_MRF19	VARCHAR
36.	REF_MRF20	VARCHAR
37.	REF_MRF21	VARCHAR
38.	REF_MRF22	VARCHAR
39.	REF_MRF23	VARCHAR
40.	REF_SPC	VARCHAR

Tableau III.1 - Type des champs de la table CRIME.

Parmi ces **40** champs, il existe **30** champs sont des clés étrangères qui font référence aux tables statiques, comme décrit dans le tableau III.2 ci-dessous.

Il est à noter que les autres **24** champs **REF_ [...]** indiquent la description physique de la personne, à savoir: REF_MRF1, REF_MRF2, jusqu'à REF_SPC indiquent respectivement la couleur de cheveux, la structure physique et les marques particulières.

III.3.1 Architecture du système:

La conception architecturale du système est le processus de conception permettant d'identifier les sous-systèmes constituant le système et le cadre de contrôle et de communication des sous-systèmes. Le but de la conception architecturale est d'établir la structure globale du software du prédicteur.

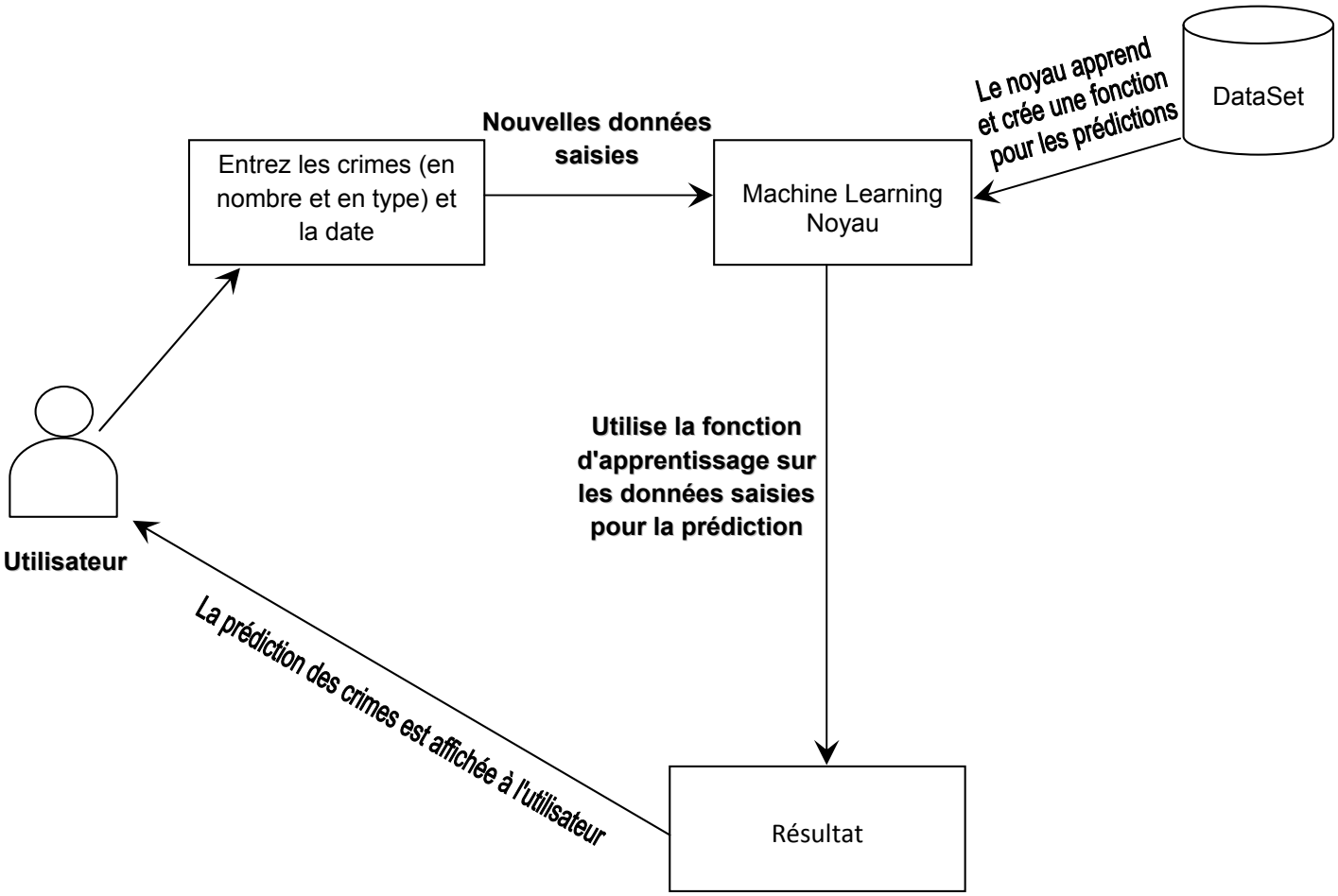

Diagramme III.1 - Architecture système du Prédicteur.

III.4 Traitement des données:

Ce travail a été divisé en deux (2) unités, comme suit:

1. Collecte de données.

2. Sélection de données.

III.4.1 Collecte de données:

Cette étape est déjà réalisée à partir du modèle relationnel des données mentionné dans la section **III.3** de ce chapitre.

III.4.2 Sélection de données:

Les données constituent une partie essentielle d'un système de Machine Learning (ML). Nous devons préparer les données avant de pouvoir les utilisées pour entraîner un modèle de ML et pour obtenir des prédictions à partir du modèle entraîné.

L'objectif fondamental de notre projet est de construire un modèle tel qu'il puisse prédire la criminalité par type qui est susceptible d'un certain ensemble de caractéristiques.

Une fois que nous avons l'ensemble de données prêt, nous devons prétraiter les données brutes pour rendre l'ensemble de données plus adapté au réseau de neurones récurrents, ainsi que plus adapté à l'objectif attendu de notre projet.

III.4.2.1 Présentation de l'ensemble de données:

Les données utilisées dans notre projet de recherche concernent la criminalité issue d'une base de données propre. L'ensemble de données comprend les attributs suivants:

N°	ATTRIBUT	DESIGNATION
1.	**CASE_NO**	Numéro affaire, c'est un champ numérique, Indique le numéro d'affaire (incident) du crime enregistré. Il est analogue au numéro de ligne.
2.	**TYPE_CRIME**	C'est un champ de type "varchar", indique le type de crime (on a 64 types de crimes).
3.	**CASE_DATE**	C'est un champ de type date, indique la date de crime (incident).
4.	**CAT**	C'est un champ de type "varchar", spécifie la catégorie de crime (on a 03 catégories de crimes).
5.	**INSCRIPTION_NO**	C'est un champ de type numérique, indique le numéro de la pièce d'inscription (document juridique).
6.	**INSCRIPTION_DATE**	C'est un champ de type "date", spécifie la date de la pièce d'inscription.
7.	**AGE**	C'est un champ de type "date", indique la date de naissance de la personne qui a commis le crime.
8.	**COMMUNITY_AREA**	C'est un champ de type "varchar". spécifie le lieu de naissance de la personne qui a commis le crime.
9.	**STATE**	C'est un champ de type "varchar", spécifie la circonscription de naissance de la personne qui a commis le crime.
10.	**GENDER**	C'est un champ de type "varchar", indique le sexe de la personne qui a commis le crime.

11.	**CODE_NAT**	C'est un champ de type "varchar", spécifie la nationalité de la personne qui a commis le crime.
12.	**DOMESTIC**	C'est un champ de type "varchar", indique si la personne réside une adresse fixe ou non.
13.	**ARREST_LOCATION**	C'est un champ de type "varchar", spécifie le lieu de crime.
14.	**ARRESTED**	C'est un champ de type "varchar", indique si la personne qui a commis le crime est arrêtée (emprisonner) ou non (en fuite).
15.	**OLD_CRIME**	C'est un champ de type "varchar", indique si la personne qui a commis le crime est récidiviste (l'accusé a des antécédents) ou non.
16.	**STATUS**	C'est un champ de type "varchar", indique si la personne est recherchée ou non.
17. . . . 40.	**REF_ [...]**	Ces **24** champs indiquent la description physique de la personne, à savoir: REF_MRF1, REF_MRF2, jusqu'à REF_SPC indiquent respectivement la couleur de cheveux, la structure physique et les marques particulières.

Tableau III.2 - Attributs de notre DataSet.

Il y a environ **07** millions de lignes dans l'ensemble de données. Il reflète les incidents de criminalité de l'année **2001** à **2018**, qui représentent **63** types de crimes, dont **6.739.035** enregistrements pour **24** types de crimes parmi les **63**, et la taille de l'ensemble de données est d'environ **1,2 GB**.

III.5 Approche:

III.5.1 Diagramme de cas d'utilisation:

Le principe de notre travail est de concevoir et d'implémenter un prédicteur de crimes basé sur les réseaux de neurones récurrents (dont le détail et les avantages de ces architectures ont été entamés dans les deux chapitres précédents).

Ci-dessous le diagramme qui représente le cas d'utilisation de notre prédicteur.

Le diagramme de cas d'utilisation représente le scénario global du prédicteur. Un scénario n'est rien d'autre qu'une séquence d'étapes décrivant une interaction entre un utilisateur et un prédicteur.

Ainsi, le cas d'utilisation est un ensemble de scénarios liés par un objectif. Le diagramme de cas d'utilisation est dessiné pour exposer les fonctionnalités du prédicteur.

- **Les cas d'utilisations du prédicteur:**

Notre prédicteur attend un **DataSet** composé de deux fonctionnalités, le type de crime et la date qui s'exécutent dans un noyau de prédiction, et qui affiche ses résultats à l'achèvement.

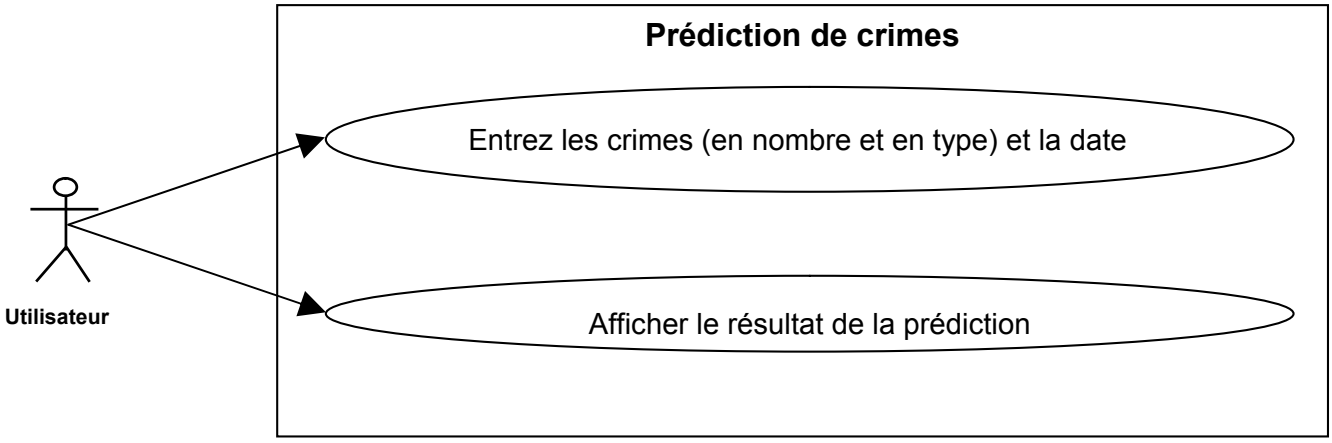

Diagramme III.2 - Diagramme cas d'utilisation.

III.5.2 Diagramme d'activités:

Le diagramme d'activité est une représentation graphique pour montrer le flux d'interaction au niveau de scénarios spécifiques. Il est similaire à un organigramme dans lequel en représente les diverses activités pouvant être effectuées dans le prédicteur.

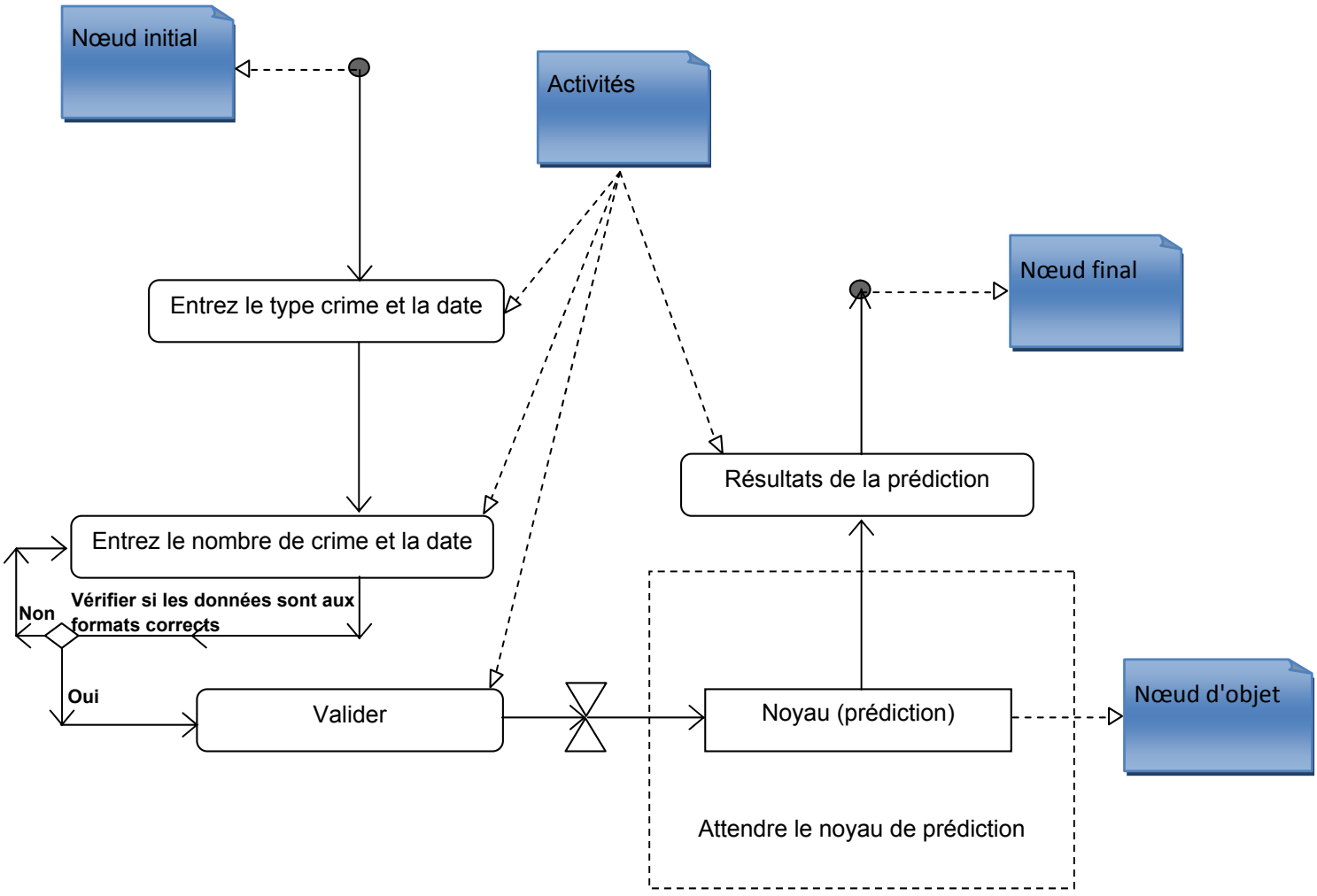

Diagramme III.3 - Diagramme d'activités.

III.5.3 Diagramme de séquence:

Le diagramme de séquence montre comment l'objet interagit avec un autre objet. Il existe des séquences d'événements qui sont représentées par un diagramme de séquence.

Il s'agit d'une vue temporelle de l'interaction entre les objets pour atteindre un objectif comportemental du prédicteur.

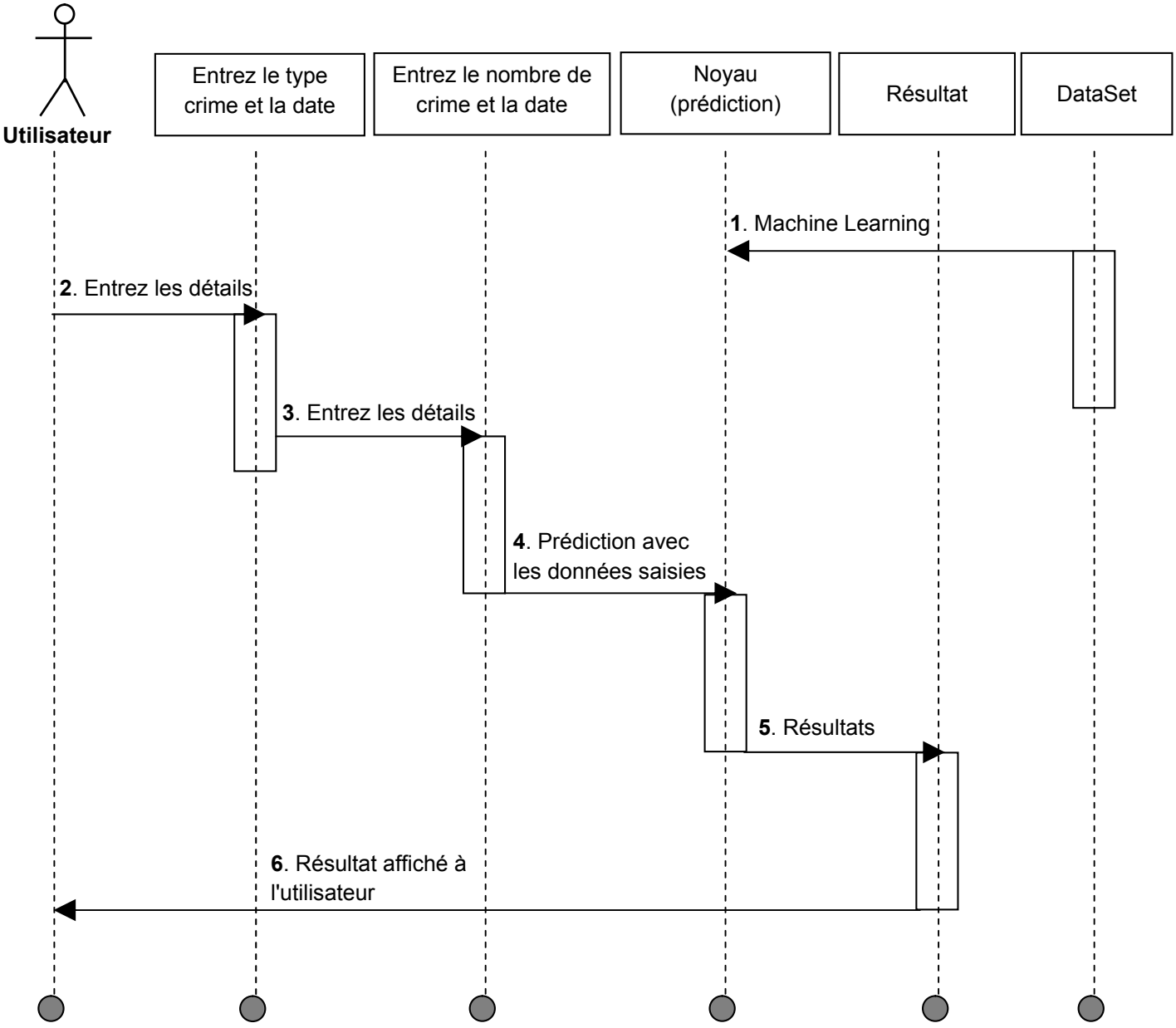

Diagramme III.4 - Diagramme de séquence.

III.6 Critères de choix du modèle:

Comme suggéré par (Fischer et al., 2017) **[25]**, on peut faire bon usage d'un réseau LSTM pour la prédiction des marchés financiers car il s'agit d'une technique artistique pour l'apprentissage des séquences. Même si ce n'est pas le même domaine que notre cas d'utilisation de la criminalité, les auteurs de l'article démontré que les réseaux LSTM peuvent extraire des informations pertinentes des données bruyantes sous-jacentes. Le réseau LSTM était principalement introduit pour surmonter le problème du gradient de fuite (**Vanishing Gradient**) et pour apprendre les dépendances à long terme puisque le simple RNN n'est pas capable de le faire. En outre il existe déjà certaines approches réussies qui prédirent et prévoient la criminalité avec les réseaux LSTM **[26]**.

Aussi le réseau Gated Recurrent Unit (GRU), qui est également un réseau neuronal récurrent, devient également de plus en plus populaire pour la tâche de prédiction lors de la possession de données séquentielles. C'était introduit il y a seulement quelques années (Cho et. al, 2014) **[27]** et il a été démontré que le GRU fonctionne de manière similaire ou même meilleure (simple et rapide), selon les données et la configuration.

Comme déjà illustré, il existe des solutions plus avancées que le simple RNN pour la prédiction de la prochaine étape d'une séquence. Pour surmonter le problème du gradient de fuite, c'est-à-dire avoir des informations cruciales dans un pas de temps très lointain qui pourraient disparaître ou des informations non pertinentes peuvent avoir une trop grande influence sur le pas de temps actuel, *nous suggérons d'utiliser les réseaux LSTM (Long Short-Term Memory) et GRU (Gated Recurrent Unit) et les comparons avec un Simple RNN*.

Dans ce travail, nous utilisons trois types de réseaux de neurones récurrents pour prédire et prévoir le crime (**LSTM**, **GRU** et **Simple RNN**).

Nous voulons être en mesure de faire une comparaison équitable entre les trois différents types de réseaux de neurones, donc nous somme besoin de les ajustés d'une manière similaire, et pour être plus précis, nous devons également ajuster les mêmes hyper-paramètres pour eux en utilisons le sur-ajustement. De plus, pour faciliter l'adaptation de nombreux modèles nous avons développé une classe personnalisée "**Architecture"** pour une comparaison équitable en instancions les mêmes paramètres d'objets et les mêmes squelettes de modèles prédictifs LSTM, GRU et Simple RNN.

A cet effet, nous choisissons une architecture moins complexe, où nous n'avons que "**la couche d'entrée**", une couche cachée que nous appelons "**une cellule**". Cette cellule peut être un objet **SimpleRNN**, **LSTM** ou **GRU**. Enfin, nous ajoutons "**une couche de sortie**" connectée, que nous fournissons avec une fonction d'activation de l'unité de rectification linéaire (**ReLU: Rectified Linear**

Unit), ce choix de cette fonction n'est qu'une précaution, car nous ne veulent pas que notre modèle génère des prédictions inférieures à zéro, ainsi que cette fonction d'activation est l'une des méthodes les plus simples et les plus efficaces. Elle a une convergence six fois meilleure que les fonctions de **tanh régulier** ce qui évite également le problème du gradient de fuite **[20]**.

En raison des restrictions supplémentaires lors de la construction d'un modèle avec état, nous travaillons avec la variante sans état c'est-à-dire tous les états ne sont pas propagés au prochain lot, autrement dit ne pas mémoriser les sous-séquences appartenant au même type de crime. Nous aborderons en détail dans le chapitre 4, le formatage adéquat de données, l'ajustement (réglage), ainsi que l'optimisation des hyper-paramètres et l'implémentation des trois types de réseaux de neurones supra cité.

III.7 Conclusion:

Dans ce chapitre, nous avons modélisé les différents aspects de notre prédicteur. Dans un premier temps nous avons conçus le DataSet (traitement des données d'entrée du modèle prédictif) qui est la ressource précieuse du prédicteur, et qui rentre dans le choix du modèle. Par la suite, nous avons présentés les phases et les tâches du noyau du prédicteur, via l'utilisation des différents diagrammes UML, qui illustrent notre vision et conception du modèle prédictif. De plus ils nous ont permis de dégager l'architecture générale de notre modèle en anticipant la phase d'implémentation, et en fin nous avons montré les critères de choix de notre modèle prédictif.

CHAPITRE IV
IMPLÉMENTATION ET ÉVALUATION

IV.1 Introduction:

L'implémentation du projet se fait à l'aide du langage de programmation **python**. Plus précisément, en utilisant **anaconda** pour des fins d'apprentissage automatique.

Anaconda est l'une des nombreuses distributions Python, cependant c'est une nouvelle distribution. Il était auparavant connu sous le nom de "**Continuum Analytics**". Anaconda propose plus de 100 nouveaux packages, il est utilisé pour le calcul scientifique, la science des données, l'analyse statistique et l'apprentissage automatique.

Concernant la technologie Python, nous avons trouvé qu'Anaconda était plus facile. Puisqu'il résoudre les problèmes suivants:

- Installation de Python sur plusieurs plateformes.
- Séparation entre les différents environnements.
- Ne pas avoir des exigences et des privilèges préalables exacts.
- Mise en service et exécution avec des packages et des bibliothèques spécifiques.

IV.2 Détails du software:

- Anaconda "Individual Edition" pour étudiants (open source distribution).
- Anaconda Distribution (v3 2020.02 64-bit).
- conda (4.8.3).
- Python (3.7.7).

- Packages Used:
  - tensorflow: 2.1.0
  - keras: 2.3.1
  - scipy: 1.5.0
  - numpy: 1.18.5
  - matplotlib: 3.2.2
  - pandas: 1.0.5
  - statsmodels: 0.11.1
  - sklearn: 0.23.1

IV.3 Le langage d'implémentation Python:

Le langage de programmation Python a été créé en 1989 par Guido van Rossum **[29]**, aux Pays-Bas. Le nom Python vient d'un hommage à la série télévisée Monty Python's Flying Circus dont G. van Rossum est fan. La première version publique de ce langage a été publiée en 1991. La dernière version de Python est la version 3. Plus précisément, la version 3.8 a été publiée en février 2020. La version 2 de Python est désormais obsolète et cessera d'être maintenue après le 1er janvier 2020. Python Software Foundation, est l'association qui organise le développement de Python et anime la communauté de développeurs et d'utilisateurs. Ce langage de programmation présente de nombreuses caractéristiques intéressantes :

- Il est multiplateforme, C'est-à-dire qu'il fonctionne sur de nombreux systèmes d'exploitation: Windows, Mac OS X,Linux, Android, iOS, depuis les mini-ordinateurs Raspberry Pi jusqu'aux supercalculateurs.

- Il est gratuit. Vous pouvez l'installer sur autant d'ordinateurs que vous voulez (même sur téléphone).

- C'est un langage de haut niveau. Il demande relativement peu de connaissance sur le fonctionnement d'un ordinateur pour être utilisé.

- C'est un langage interprété. Un script Python n'a pas besoin d'être compilé pour être exécuté, contrairement à des langages comme le C ou le C++.

- Il est orienté objet. C'est-à-dire qu'il est possible de concevoir en Python des entités qui miment celles du monde réel (une cellule, une protéine, un atome, etc.) avec un certain nombre de règles de fonctionnement et d'interactions.

- Il est relativement simple à prendre en main et à métrisé.

- Enfin, il est très utilisé en bio-informatique et plus généralement en analyse de données. Toutes ces caractéristiques font que Python est désormais enseigné dans de nombreuses formations, depuis l'enseignement secondaire jusqu'à l'enseignement supérieur.

IV.4 Détails du hardware pré-requis:

- **Système d'exploitation**: Windows 7 minimum, 64-bit macOS 10.9+, or Linux.
- **Architecture du système**: 64-bit x86, 32-bit x86 Windows ou Linux.
- **CPU**: Intel Core 2 Quad CPU minimum.
- **RAM**: 4 GB minimum.

IV.4.1 Détails de l'environnement:

- **Système d'exploitation:** Windows® 7 Édition Familiale Premium authentique 64 bits avec Service Pack 1.
- **CPU:** Intel ® Core™ i7 de 2^{e} génération - 2670QM CPU @ 2.20GHz.
- **RAM:** 8 GB DDR3 SDRAM.
- **GPU:** 2 GB - NVIDIA ® GeForce ® 540M GT.
- **Disque dur:** 640 GB - SATA 7200 TR/MIN.

IV.5 Détails de l'implémentation:

Les données utilisées ont été extraites et traitées préalablement (voir le chapitre précédent).

Les entrées ont été effectuées juste pour que la machine apprenne tout ce qu'elle a à faire avec les données et ce que la sortie est réellement demandée. Dès que la machine a appris les algorithmes et le processus, la précision des différents algorithmes a été mesurée et l'algorithme le plus précis est utilisé pour le noyau de prédiction après une évaluation.

Pour une mise en œuvre et un fonctionnement correct, plusieurs algorithmes et techniques ont été utilisés. Voici les algorithmes utilisés avec démonstration des modèles (Vanilla RNN, LSTM et GRU):

IV.5.1 Avant-propos:

Afin de remédier et d'affronter aux défis pour implémenter un modèle prédictif fiable, nous allons démontrerez les quatre facteurs influant dans une prédiction à travers un réseau de neurone récurrent (**formatage** de données, **statefull** "modèle avec état", **return_sequences** "mémoriser tous les états cachés" et le problème de gradient de fuite "**Vanishing gradient**").

Ainsi que, pour cerner toutes les données et les informations des crimes emmagasinés dans la table crime illustrée dans le chapitre précédent, nous sommes obligés de récupérer et d'interpréter toutes les clés étrangères dans cette table, d'où on a fait appel au SQL pour effectuer des requêtes de jointures entre la table crime et les tables statiques objet des clés étrangères.

Commençant par la requête permettant de récupérer la table CRIME:

```
select * from CRIME where (CASE_DATE >= to_date('01/01/2001','dd/mm/yyyy'))
and (CASE_DATE < to_date('01/11/2018','dd/mm/yyyy')); (1)
```

Puis on a récupéré et interprété toutes les clés étrangères dans cette table crime (cette opération est pour les **30** champs clés étrangères):

Voici un exemple pour le champ "TYPE_CRIME" de la table "CRIME" qui on a récupéré sa "Label" depuis la table de référence " TYPE_CRIME", où on a évité l'utilisation de la requête **INNER JOIN** (*UPDATE T2 SET T2.Name = T1 .Name FROM Table2 as T2 INNER JOIN Table1 as T1 ON T1.Id = T1.Id;)* vu son coût d'exécution dans une table "CRIME" d'une taille presque de **07 Millions** d'uplets et on a opté à une requête équivalente qui est la suivante:

```
UPDATE CRIME, TYPE_CRIME SET CRIME.TYPE_CRIME = TYPE_CRIME.Label_CRIME
WHERE CRIME.TYPE_CRIME=TYPE_CRIME.ID_ TYPE_CRIME; (2)
```

Idem pour les autres champs clés étrangères dans la table "CRIME" en remplaçant le nom de la table de référence et les champs concernés dans la requête ci-dessus.

Nous somme devant un phénomène chronologique de crime, d'où un tri par jour, mois, année, type_crime et en nombre d'occurrences est nécessaire (on a décomposé le champ "CASE_DATE" momentanément pour ce tri), la requête est la suivante:

```
SELECT count(*), CRIME.ANNEE, CRIME.MOIS, CRIME.TYPE_CRIME FROM CRIME GROUP
BY CRIME.ANNEE, CRIME.MOIS, CRIME.TYPE_CRIME
HAVING CRIME.TYPE_CRIME='Label_CRIME'; (3)
```

Maintenant on a une table "CRIME" qui comporte toutes les données et les informations du crime, pour les éventuels analyses et traitements pour extraire le DATASET final, le nombre d'enregistrements (observations) après cette dernière requête du "GROUB BY" est devenu **253.103** dont **156.312** enregistrements concernant les **24** types de crimes les plus commis.

IV.5.1.1 Prétraitement des données:

L'implémentation du projet a été faite en utilisant le langage de programmation **Python** et donc nous avons utilisé la bibliothèque **Pandas** pour le prétraitement des données (lire les données, analyser les dates et regrouper les données).

D'abord nous allons convertir les données collectées dans la table **CRIME** à un fichier au format **csv**, ce dernier est également appelé valeurs séparées par des virgules, c'est un format basé sur du texte et représente des données qui se trouvent dans des bases de données ou une feuille de calcul. Il est compact et simple, ils peuvent être facilement ouverts dans "**Microsoft Excel**".

Voici quelques autres avantages des fichiers csv:

- Ils sont sûrs et il est facile de différencier entre le texte et les valeurs numériques.
- Ils peuvent être facilement analysés avec un simple code.
- Il est facile d'importer les fichiers à l'aide de la bibliothèque Pandas et aussi les fichiers csv consomment généralement moins de mémoire et sont rapidement importés.
- La manipulation d'un fichier CSV est beaucoup plus simple car ils sont après tous des fichiers texte.

IV.5.1.1.1 Lire les données:

La lecture et la sélection de caractéristiques est généralement définie comme un processus de recherche permettant de trouver un sous-ensemble "**pertinent**" de caractéristiques parmi celles de l'ensemble de départ (en parle sur une réduction basée sur une sélection de caractéristiques [colonnes ou attributs]). La notion de pertinence d'un sous-ensemble de caractéristiques dépend toujours des objectifs et des critères du système et aussi du phénomène étudié (ces caractéristiques constitueraient l'entrée du réseau de neurones).

En raison de la taille gênante de cet ensemble de données (**1,2 GB**), nous ne chargeons que les colonnes que nous souhaitons utiliser. Notez également que nous ne pouvons pas analyser la date pendant la lecture.

Vu notre objectif de recherche "la prédiction des crimes dans le temps pour déduire les nombres des crimes (par type de crime)", nous avons omis beaucoup de colonnes qui contenant du texte difficile à compresser (donc volumineux) et essentiellement redondant, alors nous pouvons réduire la taille des données à quelque chose de plus gérable.

```
datapath = os.path.join("..", "data", "TABLE_CRIME.csv")
columns = ["CASE_NO", "TYPE_CRIME", "CASE_DATE", "CAT", "INSCRIPTION_NO", "INSCRIPTION_DATE", "AGE", "COMMUNITY_AREA",
           "STATE", "GENDER", "CODE_NAT", "DOMESTIC", "ARREST_LOCATION", "ARRESTED", "OLD_CRIME", "STATUS"]
alldata = pd.read_csv(datapath, usecols=columns)
```

IV.5.1.1.2 Regroupement de l'ensemble de données:

Nous sommes intéressés par deux vues de l'ensemble de données:

A- La quantification statistique de la criminalité à savoir le nombre total des incidents et crimes pour tous les types de crimes dans le temps.

B- La quantification statistique des différents types de la criminalité à savoir le nombre des incidents et crimes constatés par type dans le temps.

En utilisant la bibliothèque **pandas** pour regrouper l'ensemble de données comme suit:

A- Tous les crimes (Taux de crimes global):

```
all_crime = alldata.groupby(alldata.index.date).CASE_NO.count().reset_index()
all_crime.to_csv(os.path.join("..", "data", "all_crimes.csv"),
                 header=["time", "total"], index=False)
all_crime.head(10)
```

B- Types de crimes (Taux de crimes par type):

Notez que nous avons des valeurs manquantes pour quelques types de crimes. Cependant, nous pouvons les éliminer en toute sécurité, comme suit:

```
type_crime = alldata.dropna(subset=["TYPE_CRIME"])
type_crime = type_crime.groupby(["TYPE_CRIME", type_crime.index]).CASE_NO.count()
```

Maintenant on va résoudre le problème des entrées dans le **DataSet** où certains jours n'ont aucune valeur de type crime enregistrée, et ceci car les algorithmes utilisés ultérieurement exigeraient que chaque jour dans le **DataSet** ait le même nombre d'échantillons.

Cela signifie que nous devons remplir tous les jours où ces types de crime n'apparaissent pas avec des zéros (en peut les rejetons complètement car ils sont clairement minimes).

```
new_index = pd.MultiIndex.from_product(type_crime.index.levels)
type_crime = type_crime.reindex(new_index).fillna(0)
assert len(type_crime) == alldata.TYPE_CRIME.nunique() * alldata.index.nunique()
type_crime = type_crime.reset_index()
type_crime.to_csv(os.path.join("..", "data", "type_crimes.csv"),
                                header=["Type", "Time", "Crimes"],
                  index=False)
```

Maintenant, notre **dataframe** a la forme souhaitée, en comptant les crimes signalés par jour et par type, constituant en conséquence un vecteur avec trois colonnes contenant la date, le type et les scores (nombres) respectivement.

IV.5.2 Démarche:

1. Créer une liste pour garder une trace des entrées: `liste_entree = []`

   Les entrées utilisées sont celles du fichier "**type_crimes.csv**" déjà collecté dans le chapitre précédent (il contient **156.312** lignes et **03** colonnes: **Type** [type crime], **Time** [date] et **Crimes** [nombre d'incidents]).

2. ***Initialiser*** les propriétés de notre_RNN et initialiser les couches entièrement connectées et la fonction d'activation de l'état caché (hidden state):

```
C_SIZE = 5                  # le nombre d'activations cachées
SEQ_LEN = 10                # la longueur de la séquence
FEATURES_SIZE = 1           # le nombre de fonctionnalités
couche_entree = Dense(C_SIZE, name = 'couche_entree')
couche_cachee = Dense(C_SIZE , name = 'couche_cachee')
tanh = Activation('tanh')
couche_sortie = Dense(FEATURES_SIZE, activation='relu', name = 'couche_sortie')
```

3. Obtenir le vecteur d'entrée et l'ajouter à la liste d'entrée:

```
x = Input(shape = (FEATURES_SIZE,))
liste_entree.append(x)
```

4. Calculer les activations d'entrée (input activations):

```
activ_entree = couche_entree(x)
```

5. L'état caché initial est défini comme un vecteur de zéros et l'ajouter à la liste d'entrée:

```
c0 = K.zeros_like(activ_entree)
liste_entree.append(c0)
ct = Input(tensor = c0, name = 'c0')
```

6. Calculer les activations cachées (hidden activations):

```
ct = tanh(add([couche_cachee(ct), activ_entree]))
```

7. Calculer la sortie:

```
sortie = couche_sortie(ct)
notreRNN = Model(inputs=liste_entree, outputs=sortie)
```

8. Couper la matrice 3-dimension le long du 2ème axe dans la liste des entrées et la transférer du format long au format large:

```
def get_entree(X):
 return([X[:,i,:] for i in range(X.shape[1])])

all_seqs = data.pivot(index='Type', columns='Time', values='Crimes')
```

   Nous obtenons une matrice de 24 types de crimes par 6513 pas de temps (24 longues séries chronologiques), puis nous allons diviser les longues séries chronologiques en sous-séquences, en les traitant comme des échantillons indépendants. La façon la plus simple de le faire est de faire glisser une trame de longueur SEQ_LEN à travers chaque série et de traiter chaque sous-série contenue dans la trame comme un échantillon. Le successeur direct après la trame définit l'étiquette respective de la sous-série.

9. Diviser les données (DataSet) par date à un ensemble de formation (apprentissage) et un ensemble de validation:

```
dates = pd.to_datetime(all_seqs.columns)
lower = pd.to_datetime("2015-01-01")
upper = pd.to_datetime("2017-01-01")
end = pd.to_datetime("2018-01-01")

tr_idx = all_seqs.columns[dates < lower]
val_idx = all_seqs.columns[(dates >=lower) == (dates < upper)]
all_seqs_tr = all_seqs[tr_idx].values
all_seqs_val = all_seqs[val_idx].values
all_seqs_tr.shape, all_seqs_val.shape
```

10. Entrer la matrice de longues séquences [sequences, timesteps] et obtenir une matrice de sous-séquences X [subsequences, SEQ_LEN, 1] et des étiquettes y [subsequences,1]:

```
def split_sequences(long_seqs, SEQ_LEN):
 X = []
 y = []
 for long_seq in long_seqs:
  n = long_seq.shape[0]
```

Nous parcourons les 24 séquences complètes (longues) et en utilisant le découpage de tableau pour créer les sous-séquences. Les sous-séquences résultantes sont empilées verticalement. Les couches inférieures récurrentes dans keras et Notre_RNN attendent une matrice d'entrée des dimensions [batch_size, timesteps, input_dim]. Puisque nous travaillons avec des séries unidimensionnelles, nous avons jusqu'à présent ignoré la troisième dimension. Afin de correspondre à la forme d'entrée requise, une troisième dimension de taille un (01) est ajoutée au tableau.

11. Faire empiler les trames de SEQ_LEN jusqu'à la fin de la séquence:

```
seq_empil = np.stack([compl_seq[i:i+SEQ_LEN] for i in range(n-SEQ_LEN)])
etiquettes = np.array([compl_seq[i] for i in np.arange(SEQ_LEN, n)]).reshape(-1,1)
X.append(seq_empil)
y.append(etiquettes)
```

12. Ajouter un axe pour le nombre de fonctionnalités ou caractéristiques (features) par pas de temps= 1:

```
X = np.vstack(X)

X = X[:,:,np.newaxis]
y = np.vstack(y)
return(X,y)
```

13. Couper des séquences en sous séquences de longueur SEQ_LEN:

```
X_tr, y_tr = split_sequences(all_seqs_tr, SEQ_LEN)
X_val, y_val = split_sequences(all_seqs_val, SEQ_LEN)
X_tr.shape, y_tr.shape, X_val.shape, y_val.shape
```

14. Approche naïve: prédire à partir de la dernière valeur observée:

```
MOY = y_tr.mean().round(3)
y_true = all_seqs_val[:,1:]
predicts = all_seqs_val[:,:-1]
MAE_dernier_val = np.abs(predicts - y_true ).mean().round(3)
print("Taux quotidien de criminalité par type: " + str(int(MOY)))
print("MAE prédire la dernière valeur: " + str(MAE_dernier_val))
```

Pour obtenir une situation de base pour les performances du modèle lors de son construction, nous établissons une approche naïve: nous calculons l'erreur absolue moyenne (MAE) en utilisant la dernière valeur du crime observé comme prédiction du la valeur du crime suivant. Il en résulte un MAE de "6,652", avec une moyenne de 46 crimes par jour et par type, cela ne semble pas mauvais pour une simple heuristique (voir **Démonstration-1**).

15. Les RNN sont entraînés par la rétropropagation, nous utilisons la rétropropagation dans le temps (calculer l'impact sur la perte en considérant toutes les entrées jusqu'au premier pas de temps) en utilisant la méthode "compile" de keras pour transmettre la fonction de perte, optimiseur et MAE comme métrique d'évaluation:

```
BTC_SIZE = 30
EPQ = 50
notreRNN.compile(loss='mean_squared_error', optimizer='adam', metrics = ['mae'])
notreRNN_hist = notreRNN.fit(get_entree(X_tr), y_tr, batch_size = BTC_SIZE, epochs = EPQ , validation_data = (get_entree(X_val),
 y_val), verbose = 0)
```

16. Entrer la matrice de longues séquences [sequences, timesteps] et obtenir une matrice de sous-séquences disjointes X [subsequences, SEQ_LEN, 1] et des étiquettes y [subsequences,SEQ_LEN,1]:

```
def split_sequence_return_state(compl_seqs, SEQ_LEN, split_seq_start = True):
 compl_seqs_X = compl_seqs[:,:-1]
 compl_seqs_y = compl_seqs[:,1:]
 ....
  start = compl_seqs_X.shape[1] % SEQ_LEN
  X = [compl_seqs_X[:,i:i+SEQ_LEN] for i in np.arange(start,compl_seqs_X.shape[1],SEQ_LEN)]
  y = [compl_seqs_y[:,i:i+SEQ_LEN] for i in np.arange(start,compl_seqs_y.shape[1],SEQ_LEN)]
 ...
```

Une manière plus efficace de présenter les données au modèle est utilisée. Au lieu de changer une valeur dans chaque séquence d'échantillon (glisser une trame une étape à la fois dans des séquences, ce qui donne des séquences d'échantillons principalement redondantes), elles sont maintenant définies par des sous-séquences disjointes de longueur SEQ_LEN. Ainsi qu'au lieu de prédire une étiquette après avoir observé une sous-séquence complète, nous attribuons une étiquette correspondant aux séquences de même longueur à chaque séquence d'échantillon, où chaque séquence d'étiquettes commence un pas dans le temps avant la séquence d'échantillons. Cela a réduit la taille de l'ensemble de données d'un facteur de SEQ_LEN, sans perdre aucune information par rapport au formatage précédent (voir la section **Démarche-8**).

17. La sortie est triée d'abord par pas de temps puis par type:

```
 X = np.vstack(X)[:,:,np.newaxis]
 y = np.vstack(y)[:,:,np.newaxis]
 return X, y
X_tr, y_tr = split_sequence_return_state(all_seqs_tr, SEQ_LEN)
X_val, y_val = split_sequence_return_state(all_seqs_val, SEQ_LEN, split_seq_start = False)
```

18. ***Construire*** et former le Simple_RNN avec la propriété return_sequences:

```
model = Sequential()
model.add(SimpleRNN(C_SIZE, input_shape=(SEQ_LEN,FEATURES_SIZE), return_sequences=True) )
model.add(TimeDistributed(Dense(FEATURES_SIZE)))
model.compile(loss='mean_squared_error', optimizer='adam', metrics = ['mae'])
seqRNN_hist = model.fit(X_tr, y_tr, batch_size = BTC_SIZE, epochs = EPQ, validation_data = (X_val, y_val), verbose = 0)
```

Nous remplaçons maintenant notre_RNN par la couche keras SimpleRNN() en combinaison avec une couche de sortie Dense(). Bien que cela implémente le même comportement que notre_RNN, mais il permet des fonctionnalités supplémentaires. Et ceci en définissant le paramètre "return_sequence=true" (il passe tous les états cachés [hidden states], pas seulement le dernier, à la couche suivante) contrairement à notre_RNN où on a opté pour mettre à jours l'état caché à travers la variable "ct" (voir la section **Démarche-6**).

Le calque de sortie renvoie pour chaque pas de temps dans cette séquence d'états cachés une prédiction pour le pas de temps suivant dans la séquence d'étiquettes, en keras c'est plus pratique en encapsulant cette sortie par la méthode "TimeDistributed()".

19. ***Construire*** le Simple_RNN avec état (stateful):

```
BTC_SIZE = 24
model = Sequential()
model.add(SimpleRNN(C_SIZE, batch_input_shape=(BTC_SIZE, SEQ_LEN, FEATURES_SIZE), return_sequences=True, stateful=True))
model.add(TimeDistributed(Dense(FEATURES_SIZE)))
model.compile(loss='mean_squared_error', optimizer='adam', metrics = ['mae'])
```

Une façon d'éviter que l'état caché ne se réinitialise avec des zéros est d'enregistrer le dernier état caché d'une sous-séquence et de l'utiliser comme initialisation lors du passage de la sous-séquence suivante (à partir de la séquence complète initiale). En keras, cela peut être fait en définissant le paramètre "stateful=true" et la taille du lot "batch_size" pour mémoriser les sous-séquences appartenant au même type crime (on a 24 types).

20. Former le RNN avec état en ajoutant la métrique d'évaluation, l'époque et la fonction de perte à la liste:

```
stateful_val_mae = []
stateful_loss = []
stateful_val_loss = []
for i in range(EPQ):
 stateRNN_hist = model.fit(X_tr, y_tr, batch_size = BTC_SIZE, epochs = 1, validation_data = (X_val, y_val), verbose
= 0, shuffle=False)
 model.reset_states()
 stateful_val_mae.append(stateRNN_hist.history['val_mae'])
 stateful_loss.append(stateRNN_hist.history['loss'])
 stateful_val_loss.append(stateRNN_hist.history['val_loss'])
```

Lors de l'appel de la méthode ".fit", keras révise les lignes des données d'entraînement par défaut au début de chaque période d'entraînement, pour cela le paramètre "shuffle" doit être initialisé à "false" lors de l'utilisation du "stateful=true" avec une réinitialisation des états cachés après chaque époque.

21. Tracer l'erreur moyenne absolue "Mean Absolute Error" (MAE) au cours des époques pour les différents modèles:

```
import matplotlib.pyplot as plt
plt.plot(np.repeat(MAE_dernier_val,50))
plt.plot(notreRNN_hist.history['val_mae'])
plt.plot(seqRNN_hist.history['val_mae'])
plt.plot(stateful_val_mae)
plt.title("Erreur moyenne absolue pour l'ensemble de validation")
plt.ylabel('MAE')
plt.xlabel('époque')
axes = plt.gca()
axes.set_ylim([4,20])
plt.rcParams["figure.figsize"] = (8, 6) # (w, h)
plt.legend(['dernière prédiction calculée', 'notre_RNN', 'seq_RNN', 'stateful_RNN'], loc='upper right')
plt.plot()
plt.show()
```

Nous évaluerons le comportement de MAE pour les trois modèles sur l'ensemble de validation au cours des époques d'apprentissage.

IV.5.3 Démonstration:

1- Console:

```
In [1]: runfile('E:/these/M2_SIR_USDB1/code/Démonstration_RNN_MAE.py', wdir='E:/these/M2_SIR_USDB1/code')
Using TensorFlow backend.
Taux quotidien de criminalité par type: 46
MAE prédire la dernière valeur: 6.652
```

2- Tracé:

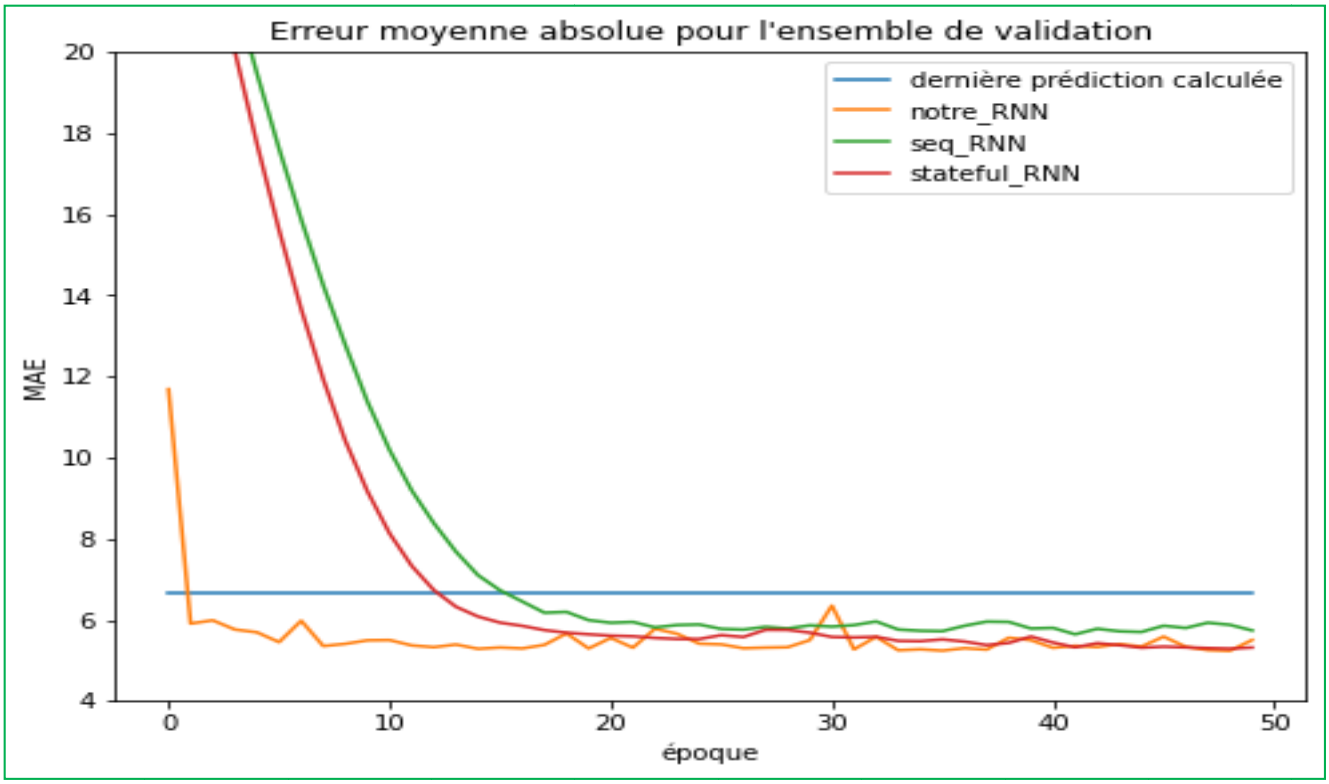

Figure IV.1 - Tracé MAE.

IV.5.4 Constat:

La première chose à noter est que notre_RNN converge nettement plus vite que les deux autres modèles. Il atteint déjà après la première époque un MAE prés de six (**06**). Cela est logique car la taille de l'ensemble d'entraînement était dix fois plus grande. Cependant, le seq_RNN (return_sequences) n'a pas convergé après dix époques. Donc, essentiellement, nous compensons la redondance réduite dans le nouveau format de données en nous entraînant pour plus d'époques. Tous les modèles fonctionnent mieux que l'approche naïve de **6,652**. Cependant, l'amélioration semble modérée à cause de la complexité du modèle supplémentaire. seq_RNN et stateful_RNN semblent converger contre une valeur supérieure à cinq (**05**), tandis que seq_RNN n'atteint qu'une valeur proche de six (**06**). La valeur la plus élevée de stateful_RNN était attendue en raison des informations manquantes au début des séquences de l'échantillon.

La propriété avec état pourrait réduire MAE au niveau de notre_RNN mais ne l'améliore pas clairement. En raison de ce résultat et des restrictions supplémentaires lors de la construction d'un modèle avec état (**stateful**), nous continuons à travailler avec la variante "sans état" (**stateless**) et le premier format de données (voir la section **Démarche-8**), Comme nous l'avons déjà mentionné dans le chapitre 3 section «Critères de choix de modèle».

Jusqu'à présent, nous avons utilisé un petit **SEQ_LEN** (longueur de la séquence d'échantillon) relatif de dix (**10**). Une façon plus simple de fournir plus d'informations sur les observations passées que l'option avec état (stateful) qui aurait augmentée la longueur de la séquence d'échantillonnage. Nous n'avons pas considéré cela jusqu'à présent parce que les RNN simples sont difficiles à former sur de longues séquences. La raison de l'apparition du problème de gradient de fuite (vanishing gradient) déjà entamé dans le premier chapitre.

Il est à noter que l'évaluation des résultats de cette démonstration avec l'erreur quadratique moyenne "Mean Square Error" (MSE) c'est presque similaire à MAE (voir Figure 4.2).

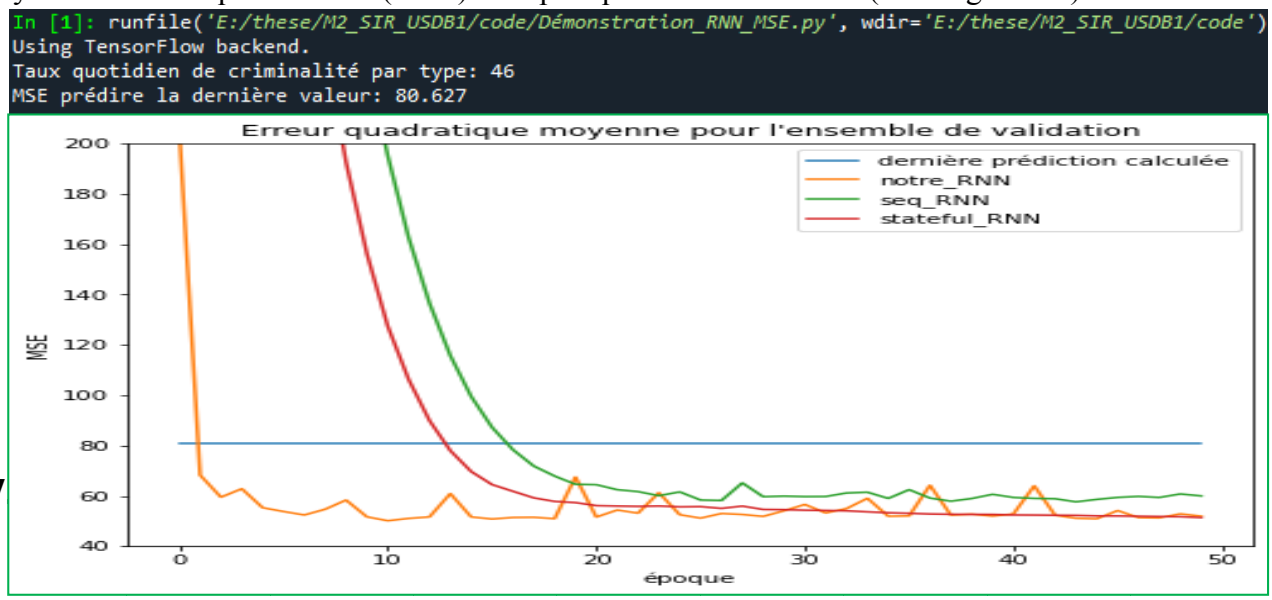

Figure IV.2 - Démonstration avec MSE.

Comme nous l'avons mentionné précédemment dans le troisième chapitre section «Critères de choix de modèle», notre but est de développer une classe pratique qui convient aux modèles récurrents et qui permet de créer des architectures récurrentes avec différentes cellules en se basant sur les facteurs démontrés ci-dessus comme des arguments et des paramètres et aussi sur les bonnes pratiques **[28]**.

Les architectures récurrentes implémentées dans notre projet pour des fins de prédiction de crimes sont: LSTM, GRU et Simple RNN, l'adaptation d'une classe mère ajustée à ces modèles récurrents doit permettre une comparaison neutre entre ce trio LSTM, GRU et Vanilla RNN (Simple RNN).

En conséquence, nous avons opté pour une classe mère personnalisée qu'on a appelée "**Architecture**" héritée du modèle de "**keras.Sequential**" dont les paramètres et les hyper-paramètres issues après une expérience (voir la section **Constat**) et qui seront décrit par la suite.

Nous allons donc commencer par décrire le corps de notre classe modèle "Architecture" récurrente, notant que c'est une architecture non-complexe constituée d'une couche d'entrée, une couche cachée paramétrable [SimpleRNN, LSTM, GRU] et une couche de sortie:

```
class Architecture(Sequential):


    def __init__(self, fichier, max_decalage, type_cellule, neurones_cellule=4, nombre_cellules=1,
                 fonctperte='mean_squared_error', optimizer='adam',
                 arguments_cellule={}, arguments_fit={}, epochs=10, batch_size=1,
                 entrainer_size=0.8, verbose=True, metrics=None):
```

Les paramètres de la classe "Architecture" à savoir max_decalage, type_cellule, neurones_cellule, epochs, batch_size et mertrics,...etc, ainsi que les principales fonctions telles que celles de'apprentissage, de prediction et de tracement des ajustements sont décrites en annexe ci-join.

IV.5.5.1 Hyperopt (ajustement des paramètres):

Maintenant notre objectif est d'affiner nos réseaux de neurones, et puisque nous voulons être en mesure de faire une comparaison équitable entre les trois types de réseaux de neurones, nous devons les ajuster de manière similaire, et pour être précis, nous devons également ajuster les mêmes hyperparamètres pour eux. De plus, pour faciliter l'ajustement de nombreux modèles, nous avons automatisé le processus de génération de données et de rassembler d'autres fonctionnalités dans notre classe "**Architecture**".

Entre outre, pour l'ajustement des hyper-paramètres nous utiliserons un algorithme d'optimisation et d'ajustement qui offre un ensemble optimal des hyper-paramètres grâce à sa fonction optimale.

Pour simplifier les choses, nous utilisons le même ensemble d'hyperparamètres pour chaque modèle et évaluons brièvement le **MAE/MSE** sur l'ensemble de validation à différentes époques. Nous examinerons plus en détail le réglage des hyperparamètres après avoir introduit des RNN plus complexes avec LSTM et GRU.

Nous incluons "max_decalage", "neurones_cellule" et "batch_size" comme hyperparamètres, afin de réduire le temps d'apprentissage. En outre, nous n'avons aucune raison de croire que le nombre d'incidents criminels dans le passé lointain (par exemple une année passée "365 jours"), serait un bon indicateur du nombre d'actes criminels qui seront produit lendemains.

La plupart de nos modèles ont des paramètres similaires que nous voulons ajuster. Pour l'échantillonnage du sous-espace, nous utilisons principalement des distributions uniformes, car elles imposent les hypothèses les moins a priori sur l'emplacement des paramètres optimaux. De plus, principalement en raison de limitations matérielles, nous limitons les valeurs maximales des hyperparamètres.

IV.5.5.2 Holdout:

- **Séparation entre les données et l'ensemble Holdout:**

Parce que nous utilisons une bibliothèque qui ajuste les paramètres par rapport à l'ensemble de test, nous devons séparer l'ensemble Holdout.

Sur cet ensemble, que nous n'exposerons pas à nos modèles, nous pouvons évaluer les performances de nos modèles ajustés.

- **Évaluer sur l'ensemble Holdout:**

Afin d'évaluer correctement nos modèles, nous refaire l'apprentissage, en utilisant les paramètres optimaux ainsi que les données qui jusqu'à présent n'ont pas été touchées. Nous pouvons facilement le faire à partir de nos modèles personnalisés de la classe "Architecture". Nous avons juste besoin de trouver la nouvelle taille de notre données d'apprentisssage (en pourcentage) parmi l'ensemble de données d'origine.

Ensuite, nous pouvons former les modèles finaux et calculer l'erreur moyenne absolue et l'erreur quadratique moyenne sur l'ensemble Holdout.

IV.5.5.3 Fonction d'activation ReLU et fonction de perte MSE:

Nous avons utilisé la fonction d'activation de l'unité de rectification linéaire **ReLU** (Rectified Linear Unit), c'est l'une des méthodes les plus simples et les plus efficaces, elle permettre que notre modèle ne génère pas des prédictions inférieures à zéro. Elle a une convergence six (**06**) fois meilleure que les fonctions de **tanh régulier** ce qui évite également le problème de gradient de fuite **[20]**.

Concernant la fonction de perte, on a utilisé **MSE** (Mean Squared Error), c'est la plus adéquate aux problèmes de régression et de prédiction **[28]**.

L'erreur quadratique moyenne **(MSE)** est une fonction de perte populaire en ML, signifie mathématiquement le carré de la différence entre l'étiquette et la prédiction, c.à.d. [observation - prediction(x)]^2.

Autrement dit, **MSE** correspond à la perte quadratique moyenne pour chaque exemple de test. Pour calculer l'erreur MSE, il faut additionner toutes les pertes quadratiques de chaque exemple de test, puis diviser cette somme par le nombre d'exemples de test selon l'équation suivante:

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prédictions(x))^2$$

Où :

- (x,y) est un exemple de test dans lequel :
  - x est l'ensemble des caractéristiques (par exemple, température, âge et type_crime) que le modèle utilise pour réaliser des prédictions ;
  - y est l'étiquette de l'exemple de test (par exemple, valeur d'observation "code_crime").
- prédiction(x) est une fonction des pondérations et biais en combinaison avec l'ensemble des caractéristiques x.
- D est un ensemble de données contenant de nombreux exemples étiquetés, qui sont des paires (x,y).
- N est le nombre d'exemples de test dans D.

IV.5.6 Simple RNN:

IV.5.6.1 Résultats et tracés:

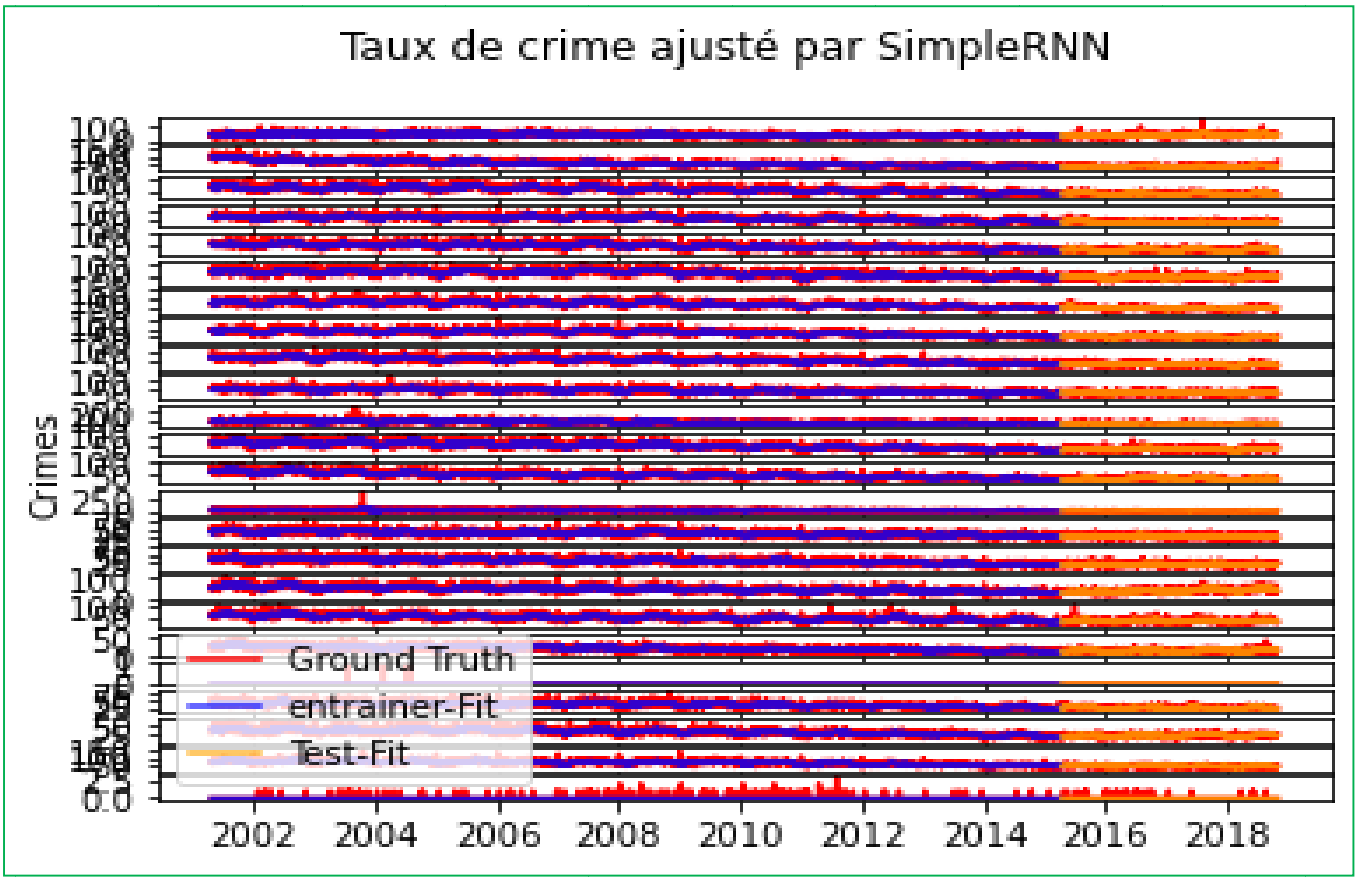

Figure IV.3 - SimpleRNN- taux de crimes par type

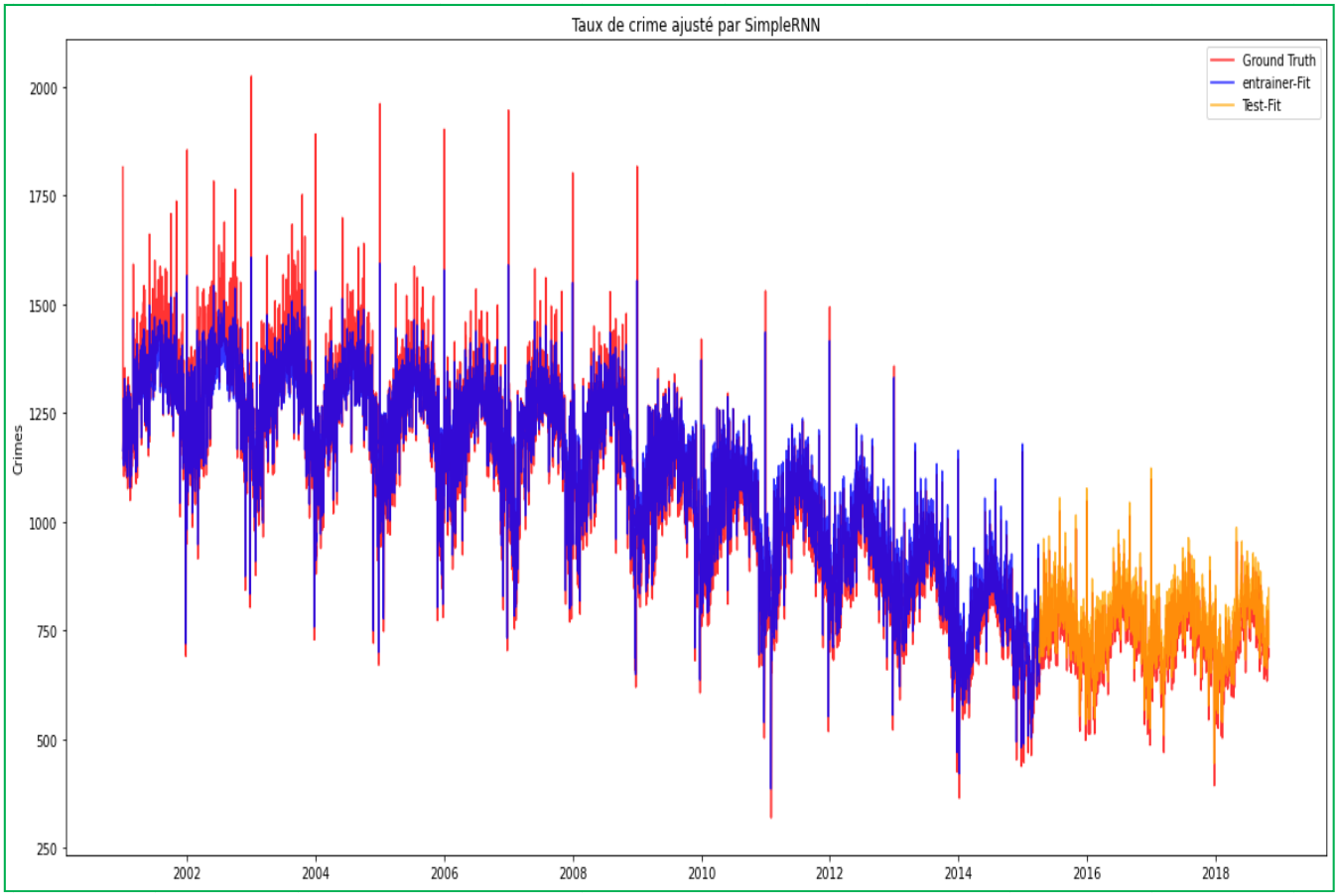

Figure IV.4 - SimpleRNN - taux de crimes total

IV.5.7 LSTM:

IV.5.7.1 Résultats et tracés:

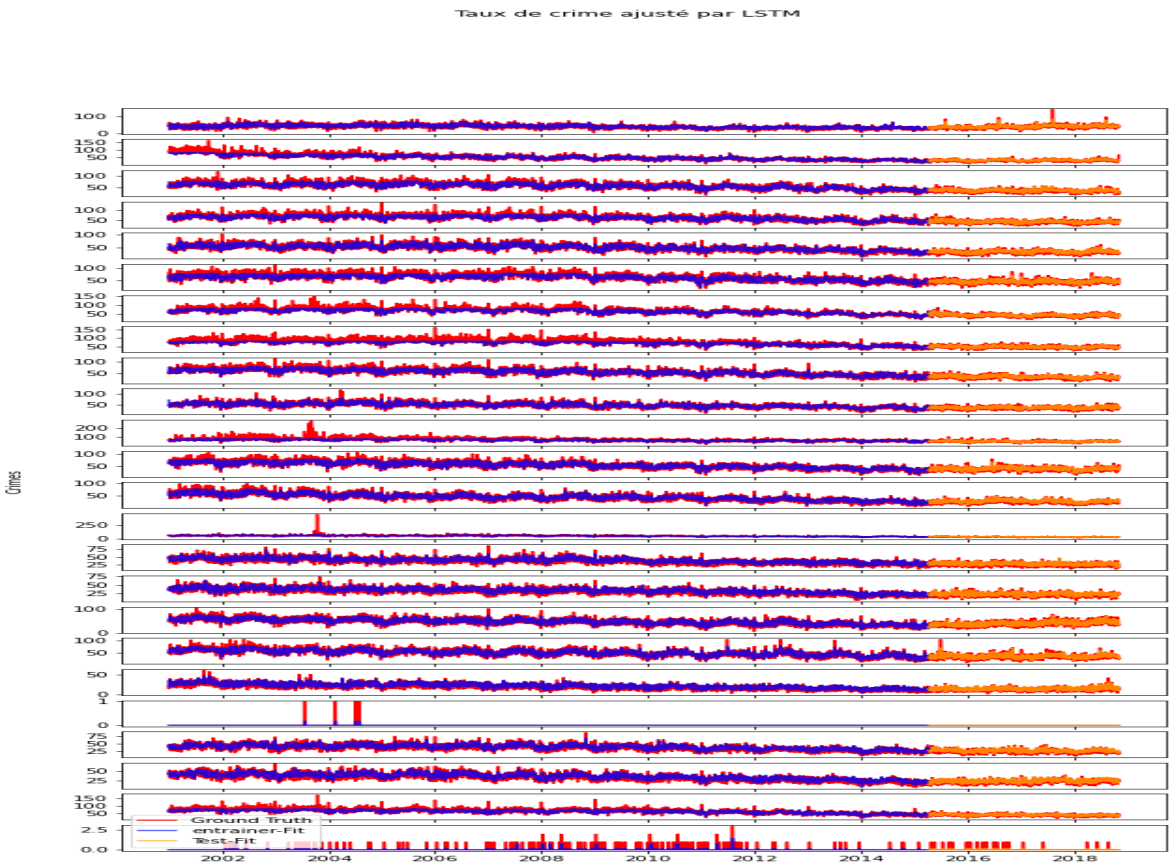

Figure IV.5 - LSTM- taux de crimes par type

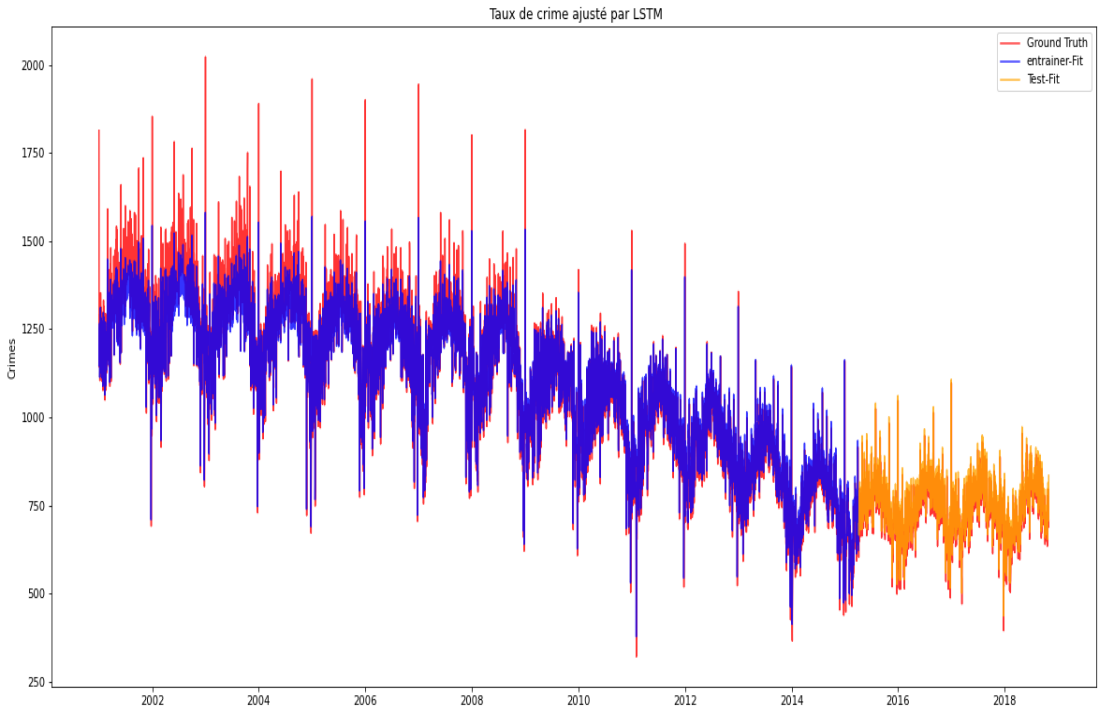

Figure IV.6 - LSTM- taux de crimes total

IV.5.8 GRU:

IV.5.8.1 Résultats et tracés:

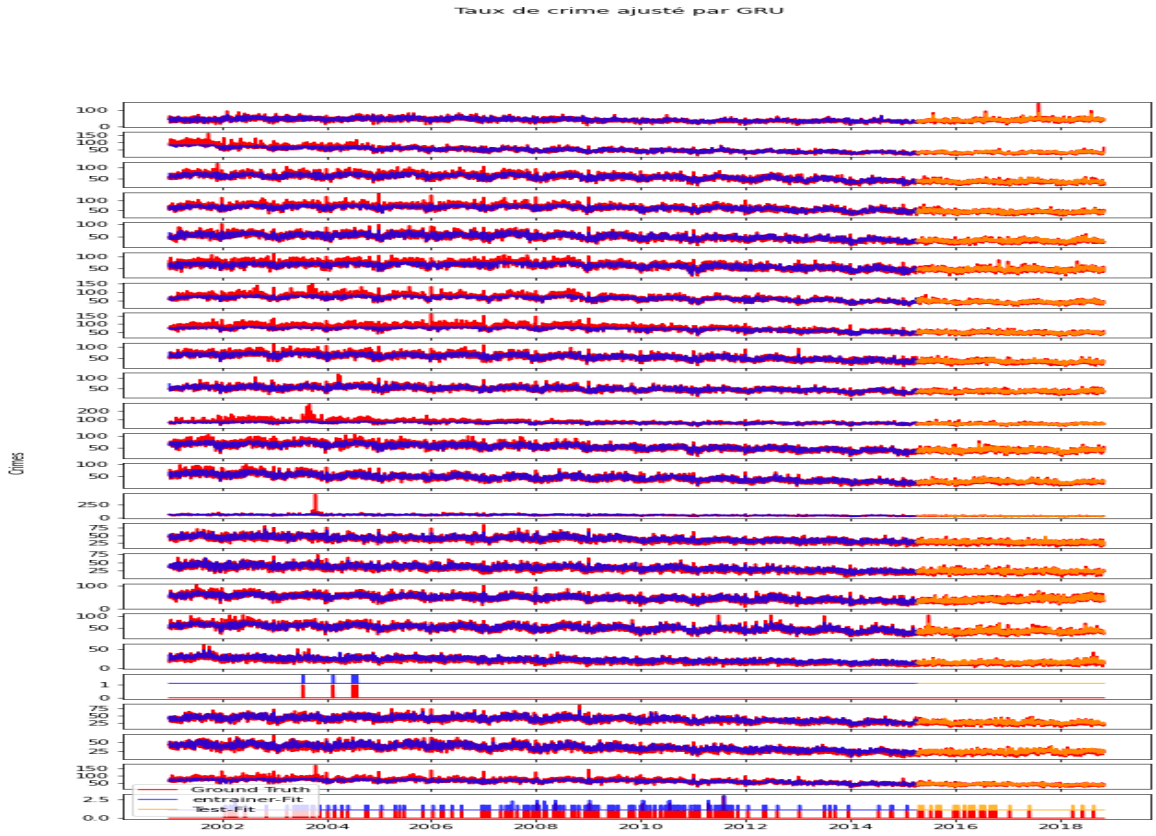

Figure IV.7 - GRU- taux de crimes par type

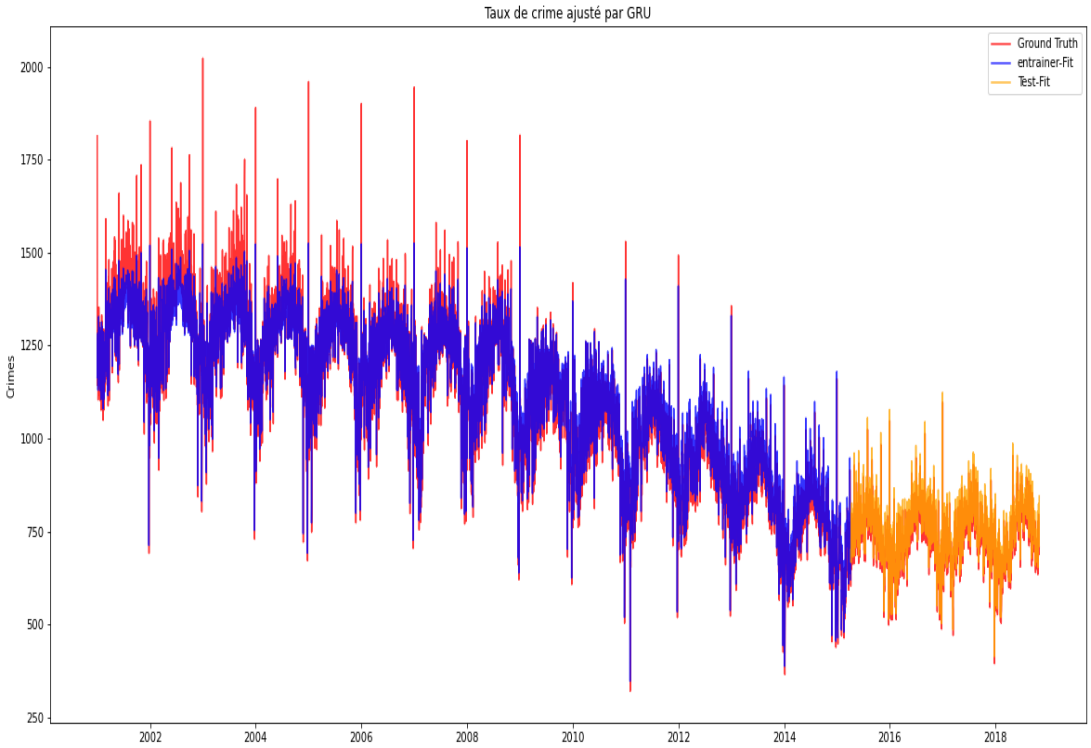

Figure IV.8 - GRU- taux de crimes total

IV.6 Visualisation et évaluation de résultats:

Dans cette section, nous focalisons sur les différents résultats obtenus et nous comparons les modèles supra cité.

Après avoir testé tous nos modèles, nous arrivons aux résultats représentés dans le tableau ci-dessous. Nous évaluons nos modèles en calculant l'erreur absolue moyenne (perte MAE) et l'erreur quadratique moyenne (perte MSE), ces les deux métriques d'évaluatuion les plus significatives dans les problèmes de régression et de prédiction **[28]**. Comme modèle de base, nous utilisons un modèle basé sur l'approche naïve (ne prédit que le dernier jour), c'est-à-dire que le nombre de crimes d'aujourd'hui est égal au nombre de crimes d'hier [déja presenter dans le modèle du démonstration au début du chapitre].

N°	MODÈLE	PERTE MAE	PERTE MSE
1.	MODÈLE DE BASE	6.65	80.63
2.	RNN	5.16	48.72
3.	LSTM	4.97	46.15
4.	GRU	4.96	45.57

Tableau IV.1 - Évaluation des résultats.

Tous nos modèles fonctionnent environ **23%** pour cent mieux que le modèle de base en examinant la perte MAE, c'est-à-dire la perte en nombre absolu d'incidents de crimes. Les modèles fonctionnent de manière similaire sauf que les résultats du simple RNN ne sont pas aussi bons que ceux du LSTM et du GRU. Cela peut être dû au problème de gradient de fuite qui peut être évité avec LSTM et GRU. De plus, ces deux derniers ont des valeurs de perte très similaires.

En général, le GRU peut être considéré comme le modèle le moins complexe et le plus rapide lorsqu'on a beaucoup de données (série trop longue). Le LSTM et le GRU présentent des performances similaires (il y a beaucoup de recherches qui les ont testés pour voir le plus performant d'entre eux selon les données entrées). Le LSTM atteint souvent de meilleures performances que le GRU, on peut affirmer que cela est dû au fait que le LSTM posséde un "état de cellule" "cell state" supplémentaire qui peut conserver des informations qui ne sont pas nécessaires pour la sortie et peut aussi mieux se souvenir des informations concernant les dépendances à long terme (grâce à la coordination "état de cellule-porte d'oubli-porte d'entrée" - voir chapitre 1).

IV.7 Test:

Le développement de logiciels implique une série d'activités de production où les influences de la faillibilité humaine sont énormes.

L'erreur peut commencer à se produire lors de l'inspection même du processus où l'objectif peut être énormément ou imparfaitement spécifié ainsi qu'au stade de la conception et du développement. En raison de l'incapacité humaine à effectuer et à communiquer avec perfection, les activités et les tests d'assurance qualité du développement logiciel sont nécessaires.

Les tests de logiciels sont un élément crucial des garanties de qualité des logiciels et représentent un examen ultime des spécifications, de la conception et du codage et aussi pour but de détection de fautes différentes.

Il est à noter que lors de passage des paramètres et hyperparamètres que ce soit dans l'instanciation ou dans le sur-ajustement des modèles prédictifs, ils sont contrôlés dans la manière de choix optimale, et ceci en appelant les fonctions **trails()** et **tracer_trails()** lors de l'appel des méthodes de sur-ajustement **hyperopt** et **hyperopt.fmin** qui utilise l'optimiseur "**Tree of Parzen Estimators - TPE**":

```
paramspace = {
    "max_decalage": scope.int(hp.quniform("max_decalage", 0, 365, 1)),
    "neurones_cellule": scope.int(hp.quniform("neurones_cellule", 1, 30, 1)),
    "batch_size": scope.int(hp.quniform("batch_size", 1, 10, 1))}

trials = Trials()
meilleur_param = fmin(fn=objective,
                      space=paramspace,
                      algo=tpe.suggest,
                      trials=trials,
                      max_evals=1000)
```

Voici un exemple de test de paramètre lors de sur-ajustement pour les deux séries chronologiques "all_crime" et "type_crime" en limitant les hyperparamètres max_decalge, neurones_cellule et batch_size à 100, 30 et 100 respectivement:

```
meilleur All_crime
{"All_crime": {"SimpleRNN": {"batch_size": 1.0, "neurones_cellule": 27.0, "max_decalage": 14.0},
               "LSTM": {"batch_size": 1.0, "neurones_cellule": 19.0, "max_decalage": 23.0},
               "GRU": {"batch_size": 1.0, "neurones_cellule": 22.0, "max_decalage": 26.0}},

meilleur type_crime
"type_crime": {"SimpleRNN": {"batch_size": 4.0, "neurones_cellule": 7.0, "max_decalage": 42.0},
               "LSTM": {"batch_size": 1.0, "neurones_cellule": 8.0, "max_decalage": 50.0},
               "GRU": {"batch_size": 4.0, "neurones_cellule": 28.0, "max_decalage": 44.0}}}
```

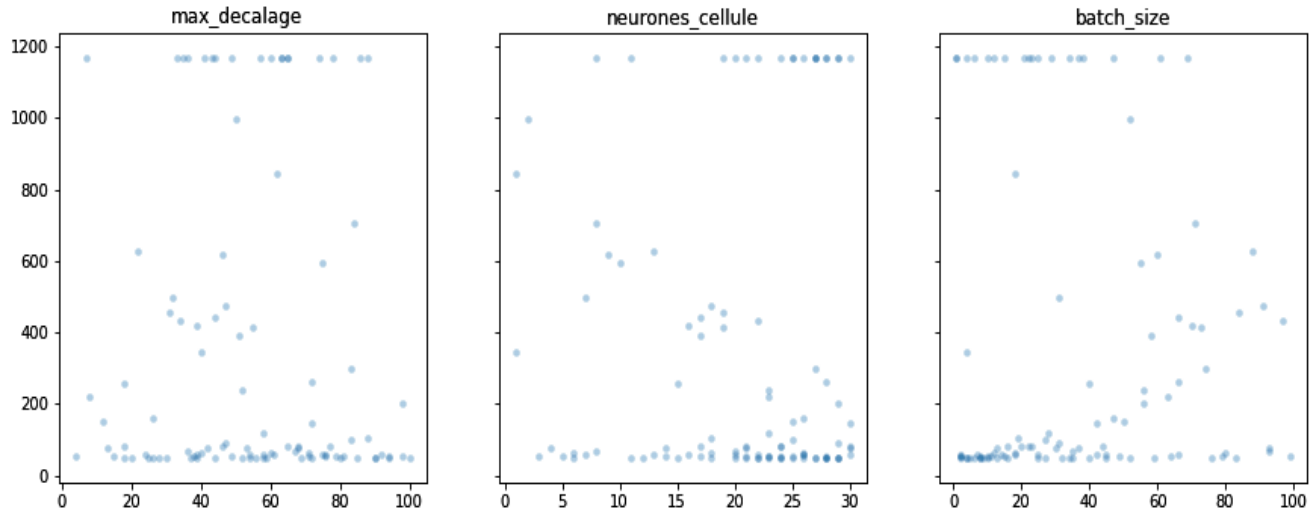

Figure IV.9 - Example de tracer_trials()

IV.8 Conclusion:

Ce chapitre a présenté une architecture pour apprendre trois modèles de prédiction pour les séries chronologiques (temporelles). Nous utilisons l'ajustement (sur-réglage) à travers hyperopt et les hyperparamètres pour obtenir de meilleures prédictions sur des données chronologiques. Trois différentes architectures de réseaux de neurones récurrents sont proposées, la première Vanilla RNN et la deuxième LSTM et la dernière GRU. Les trois modèles sont appris en partageant les mêmes paramètres, celles de la classe "Architecture". Des expériences sont menées sur notre jeux de données avec les trois modèles prédictifs que nous proposons obtient les meilleures performances avec LSTM et GRU puis Vanilla RNN respectivement.

Durant l'étape de développement, nous avons été amenés à écrire un code d'un niveau d'abstraction élevé de point de vue implémentation, surtout concernant les algorithmes d'apprentissage automatique. Les modèles et les classes Python nous ont permis de profiter de la puissance du langage en termes de réseaux de neurones récurrents, d'apprentissage et de métriques d'évaluation. En fin nous avons effectué des tests fonctionnels et structurels sur notre prédicteur de crimes.

CONCLUSION ET PERSPECTIVES

1. Conclusion:

Notre thème est consacré au développement d'un prédicteur de crimes (basé sur les algorithmes de Machine Learning) pour assister les services de sécurité publique lors de la prise de décision et lors de l'analyse des phénomènes criminels. Ce dernier fonctionne en parallèle avec trois (03) types de réseaux de neurones récurrents, Vanilla RNN, LSTM et GRU, via Python. L'utilisation de ce type de réseaux de neurones permet le fonctionnement du prédicteur dans un environnement de données séquentielles (Time Series - Série Chronologique ou Temporelles).

Les tâches de raisonnement du prédicteur sont totalement unidimensionnelles. En d'autre terme, le prédicteur utilise une série chronologique univariée (type de crimes) espacées dans le temps, une comparaison a été réalisée entre les résultats des différents types de réseaux de neurones récurrents utilisés.

En plus, puisque le prédicteur est développé dans un environnement Python qui est multi plate-forme, ceci permet le fonctionnement du prédicteur sur les différents types de machines. En termes de portabilité notre prédicteur est alors performant.

Dans le cadre de la prédiction des crimes en utilisant les réseaux de neurones récurrents et en mesure de faire une comparaison équitable entre les trois types de réseaux de neurones utilisés, une classe mère récurrente a été développée afin qu'elle s'hérite à ses instances les mêmes paramètres et traitements (hyper-paramètres et ajustement).

Le prédicteur est chargé de prévoir les crimes dans le temps en nombre et en type. Ces prédictions font l'objet d'une évaluation en utilisant certains métriques (Mean-Absolute-Error et Mean-Squared-Error).

En effet, notre prédicteur fonctionne de la façon coopérative suivante : le formatage des données, l'instanciation de la classe mère récurrente, l'apprentissage, l'ajustement (sur-réglage) et l'évaluation.

2. Perspectives:

Notre travail peut être à l'initiative d'autres travaux futurs notamment ce qui concerne les systèmes d'aide à la décision et les systèmes de prédiction, tels que:

- L'intégration d'autres facteurs et caractéristiques influant dans l'analyse et la prédiction criminelle, à savoir :
  - La région.
  - La tranche d'âge.
  - La nationalité.
  - La fonction,...etc.
- Faciliter la présentation des résultats en adaptant une analyse graphique et cartographique de la criminalité, en utilisant un logiciel de cartographie ou d'information géographique par exemple QGis, SAGA GIS, ArcGIS,...etc

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Predictive Policing -The Role of Crime Forecasting in Law Enforcement Operations -Walter L. Perry, Brian McInnis, Carter C. Price, Susan C. Smith, John S. Hollywood (2013).

[2] Pat Carlen. Women, crime and poverty. Open University Press Milton Keynes, 1988.

[3] Ellen G Cohn. Weather and crime. British journal of criminology, 30(1):51–64, 1990.

[4] Desmet, P. (1996). Comparaison de la productivité d'un réseau de neurones à rétro-propagation avec celles des méthodes de régression linéaire, logistique et AID pour le calcul des scores en marketing direct. Recherche Et Applications En Marketing,11(2),17-27.

[5] Cerqueira, Vitor & Torgo, Luís & Soares, Carlos. (2019). Machine Learning vs Statistical Methods for Time Series Forecasting: Size Matters.

[6] M.Serrurier Apprentissage artificiel IRIT, Toulouse, France December 8,2015.

[7] Randrianarivony, Détection de concepts et annotation automatique d'images médicales par apprentissage profond, 09 mars 2018.

[8] Hardy, C. (2019). Contribution au développement de l'apprentissage profond dans les systèmes distribués (Doctoral dissertation, Rennes 1).

[9] RAPHAËL, M. J. B. (2017). Machine learning: Introduction à l'apprentissage automatique.

[10] Djeriri, Youcef. (2017). Les Réseaux de Neurones Artificiels.

[11] Lauly, S. Exploration des réseaux de neurones à base d'autoencodeur dans le cadre de la modélisation des données textuelles.

[12] Lorrain, V. (2018). Etude et conception de circuits innovants exploitant les caractéristiques des nouvelles technologies mémoires résistives (Doctoral dissertation).

[13] Science des données – cours au Collège de France, Stéphane Mallat – janvier 2018.

[14] https://deepai.org/machine-learning-glossary-and-terms/vanishing-gradient-problem.

[15] CS 209B: Advanced Topics in Data Science Protopapas, Glickman Recurrent Neural Networks: Exploding, Vanishing Gradients & Reservoir Computing Authors: M. Mattheakis, P. Protopapas, Last Modified: March 6, 2019.

[16] Durand, S., & Alexandre, F. (1995). Learning speech as acoustic sequences with the unsupervised model, tom. In Neural Networks and Their Applications. (pp. 267-73).

[17] Cleeremans, A., & McClelland, J. L. (1991). Learning the structure of event sequences. Journal of Experimental Psychology: General, 120(3), 235.

[18] Miclet, L., & Cornuéjols, A. (2003). Avec la participation d'Yves Kodratoff. Apprentissage artificiel (concepts et algorithme), préface de Tom Mitchell, Deuxième tirage.

[19] https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html.

[20] Krishnan, A., Sarguru, A., & Sheela, A. S. (2018). Predicitve analysis of crime data using deep learning. International Journal of Pure and Applied Mathematics, 118(20), 4023-4031.

[21] Baboo, S. S. (2011). An enhanced algorithm to predict a future crime using data mining. International Journal of Computer Applications, 975, 8887.

[22] Shingleton, J. S. (2012). Crime trend prediction using regression models for salinas, california. NAVAL POSTGRADUATE SCHOOL MONTEREY CA.

[23] Stalidis, P., Semertzidis, T., & Daras, P. (2018). Examining Deep Learning Architectures for Crime Classification and Prediction. arXiv preprint arXiv:1812.00602.

[24] Benbouzid, B. (2017). Des crimes et des séismes La police prédictive entre science, technique et divination.

[25] Suggested Citation: Fischer, Thomas; Krauss, Christopher (2017): Deep learning with long short-term memory networks for financial market predictions, FAU Discussion Papers in Economics, No. 11/2017, Friedrich-Alexander-Universität Erlangen-Nürnberg, Institute for Economics, Nürnberg.

[26] Forecasting Crime with Deep Learning, Alexander Stec and Diego Klabjan, arXiv:1806.01486v1 [stat.ML] 5 Jun 2018.

[27] Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation, Kyunghyun Cho Bartvan Merrienboer Caglar Gulcehre Université de Montréal. Dzmitry Bahdanau Jacobs University, Germany. Fethi Bougares Holger Schwenk Université du Maine, France. Yoshua Bengio Université de Montréal, CIFAR Senior Fellow. arXiv:1406.1078v3 [cs.CL] 3 sep 2014.

[28] Brownlee, J. (2016). Deep learning with Python: develop deep learning models on Theano and TensorFlow using Keras. Machine Learning Mastery.

[29] Guinard, S. (2018). Cours d'introduction Python-Stéphane Guinard.

ANNEXE

LA CLASSE ARCHITECTURE

1. Paramètres de la classe Architecture:

N°	PARAMETRE	DESCRIPTION
1.	fichier	"pandas.DataFrame" avec une colonne. L'index doit être un "pandas.DatetimeIndex" ou un "pandas.MultiIndex" où le premier niveau est un "pandas.DatetimeIndex" et le deuxième niveau un indicateur de groupe. Le dataframe sert pour présenter les séries "type_crimes.csv" et "all_crimes.csv".
2.	max_decalage	("type: int") Le nombre maximum de décalage a généré par rapport aux données (fichier).
3.	type_cellule	Type cellule soit [LSTM, GRU, SimpleRNN] de " keras.layers".
4.	neurones_cellule	("type: int") Le nombre de neurones dans la couche cachée de "type_cellule".
5.	nombre_cellules	("type: int") Le nombre de cellules.
6.	fonctperte	Fonction de perte à utiliser.
7.	optimizer	Optimiseur à utiliser.
8.	arguments_cellule	(type "dict") Les arguments "kwargs" pour la méthode "keras.Sequential.fit".
9.	arguments_fit	(type "dict") Les arguments mots-clés pour "type_cellule".
10.	epochs	("type: int") Le nombre d'époques d'apprentissage (train).
11.	batch_size	("type: int") Le nombre des échantillons par lot.
12.	entrainer_size	("type: int") La proportion des échantillons d'apprentissage (train). (0 < entrainer_size <= 1).
13.	verbose	(type "bool") Le niveau de verbosité pendant l'apprentissage (train).
14.	metrics	"list" de métriques d'évaluation. [Ils ne sont pas utilisés pendant l'apprentissage (train)].

Tableau des Paramètres de la classe "Architecture".

2. Train (apprentissage / formation):

Architecture.entrainer(self)

Former le modèle en fonction des paramètres transmis à __init__.

3. Prédiction:

Architecture.prediction(self, X)

Calculez les prédictions.

- Paramètre:

✓ **"X":** "numpy.ndarray" sous la forme ("n", "max_decalage"), "n" est arbitraire.

- Retour:

✓ **"out":** prédictions du modèle sous la forme ("n", 1).

4. Tracer l'ajustement (tracé):

Architecture.tracer_fit(self, savepath=None, **kwargs)

Visualisez l'ajustement des données d'apprentissage et de test.

- Paramètre:

✓ **"savepath":** Nom de fichier pour enregistrer la figure, non enregistré si "None".
✓ **"**kwargs":** Arguments mot-clé pour "pyplot.subplots".

- Retour:

✓ **"(fig, ax)":** La figure et l'axe dépendent de l'index "self.data".

  - Si "pandas.DatetimeIndex": un seule tracé pour toute la séries (all_crimes.csv).
  - Si "pandas.MultiIndex": multiple tracés pour chaque séries (type_crimes.csv).