Université Assane Seck de Ziguinchor

UFR Sciences et Technologies

Département Informatique

Mémoire de fin d'études

Pour l'obtention du diplôme de Master

Mention : Informatique

Spécialité : Génie Logiciel

Thème : État de l'art sur les architectures de Deep Learning : Perceptron, CNN et RNN

Présenté et soutenu par :

M. Mody BALDE

le :

03/08/2024

Sous la direction de :

Pr Khadim DRAME

Dr Gorgoumack SAMBE

Membres du jury

Pr Youssou FAYE	Professeur Assimilé	Président/Examinateur	UASZ
Dr Mouhamadou GAYE	Maître de conférence titulaire	Rapporteur	UASZ
Pr Khadim DRAME	Professeur Assimilé	Co-encadrant	UASZ
Dr Gorgoumack SAMBE	Maître de conférence titulaire	Co-encadrant	UASZ

Année universitaire 2022-2023

✠ Dédicaces ✠

Par la grâce de DIEU, Le Tout-Puissant, source de guidance et de bénédiction, je dédie ce modeste travail :

À mes chers parents,

Nulle dédicace ne saurait exprimer l'étendue de mes sentiments sincères et de mon éternelle gratitude envers vous. Votre patience illimitée, vos prières constantes et votre soutien indéfectible ont été le socle de ma réussite. Ce travail est le témoignage de mon profond amour et de mon respect pour vos immenses sacrifices.

À mes frères,

Pour l'amour et le soutien dont vous m'avez fait preuve tout au long de mon parcours académique. Votre présence constante et vos encouragements ont été un pilier essentiel de ma réussite. Chaque étape de ce chemin a été marquée par votre bienveillance et de votre appui indéfectible, me donnant la force de persévérer face aux défis. Votre confiance en ma modeste personne a été une source d'inspiration inestimable, me poussant à donner le meilleur de moi-même.

À tous mes amis et camarades de promo

Pour votre amitié précieuse et vos encouragements qui m'ont accompagné et soutenu tout au long de cette aventure académique.

✠ Remerciements ✠

Tout d'abord, nous exprimons notre profonde gratitude en vers Allah le Tout-Puissant de nous avoir donné la force et l'audace de surmonter toutes les difficultés et de mener à bout ce travail.
En signe de profonde reconnaissance, nous souhaitons exprimer nos plus sincères remerciements à toutes les personnes dont la contribution, de près ou de loin, a été déterminante pour le succès de ce mémoire.

Mes encadrants,

• ***Dr Gorgoumack SAMBE**, connu par ces qualités de scientifiques et d'informaticien aguerri, pour la confiance que vous m'avez accordé en acceptant de travailler avec moi sur ce sujet. Votre disponibilité, vos conseils et vos orientations pertinentes et avisées ont constitué un apport considérable pour l'aboutissement de ce travail scientifique. Merci de fond du coeur pour toute l'aide apportée, pour votre enthousiasme et vos encouragements permanents.*

• ***Pr Khadim DRAME**, éminent chercheur à l'Université Assane Seck de Ziguinchor, pour votre soutien tout au long de ce chemin. Vos précieux conseils et orientations scientifiques et techniques ont contribué pleinement à l'élaboration de ce mémoire. Nous sommes particulièrement reconnaissants de la confiance que vous nous avez toujours témoignée tout au long de notre cursus universitaire.*

Les membres du jury,

*Nous exprimons notre gratitude aux différents membres du jury : le président **Pr Youssou FAYE** et le rapporteur **Dr Mouhamadou GAYE**, pour avoir accepté de consacrer de votre temps à l'évaluation de notre travail.*

Le corps professoral du département Informatique,

Nous adressons nos vifs remerciement à l'ensemble des enseignants du département informatique qui ont grandement contribué à la qualité exceptionnelle de notre formation. Nous somme reconnaissant pour les efforts fournis durant toutes ces années dans le seul but de nous offrir une formation de qualité dans un environnement propice.

La famille,

Votre soutien et vos encouragement m'ont permis de faire face aux défis que nous avons rencontré durant toutes ces années universitaires. Un grand merci à mes frères qui ont ménagé aucun effort pour me mettre dans de bonnes conditions.

✠ Résumé ✠

Le Deep Learning (DL) s'est imposé comme un paradigme révolutionnaire dans le domaine de l'intelligence artificielle et du machine learning. Reposant sur des réseaux de neurones artificiels, le DL s'est montré efficace dans plusieurs domaines d'application tels que la reconnaissance vocale (SIRI d'Apple), la traduction automatique (Google Translate) et bien d'autres.

Dans ce mémoire, nous proposons un état de l'art et une étude comparative des trois architectures de base du deep learning : le perceptron multicouche (Multi-Layer Perceptron (MLP)), les réseaux convolutifs (Convolutional Neural Network (CNN)) et les réseaux récurrents (Recurrent Neural Network (RNN)).

Nous présenterons l'origine et l'évolution historique du deep learning et ses fondements théoriques : le neurone formel, les fonctions d'activation, l'évaluation des modèles, les topologies des réseaux de neurones, les techniques d'apprentissage du deep learning ainsi que les algorithmes d'optimisation.

Nous aborderons l'architecture du MLP, son fonctionnement, l'algorithme de la rétropropagation, son domaine d'application et ses limites.

Pour les CNN, nous présenterons le principe de la convolution qui constitue la base des CNN ainsi que leurs architectures avec leurs différentes couches : couche de convolution, couche de pooling et couche entièrement connectée. Nous présenterons aussi leurs domaines d'application et leurs limites

Pour les RNN, en plus de leur architecture, nous présenterons leur mécanisme d'apprentissage avec la rétropropagation à travers le temps (Backpropagation Through Time). Nous parlerons de leurs problèmes d'apprentissage, à savoir la disparition du gradient et l'explosion du gradient. Nous présenterons les variantes des RNN : les mémoires à long et court terme (Long Short-Term Memory (LSTM)) et les unités récurrentes à portes (Gated Recurrent Unit (GRU)). Leurs domaines d'application ainsi que leurs limites seront abordés.

Des travaux antérieurs ont montré que les CNN sont plus adaptés pour les tâches de traitement d'image, tandis que les RNN sont plus aptes pour le traitement des données séquentielles ou des séries temporelles .

Nous avons proposé une étude comparative entre les architectures MLP et CNN sur la classification d'image avec le jeu de données CIFAR-10 et entre les architectures CNN et RNN pour l'analyse de sentiment avec le jeu de données IMDB, ainsi que pour la génération de poèmes avec un recueil de poèmes de Victor Hugo.

Les résultats de cette étude sur les métriques d'exactitude, précision, rappel, score f1 ainsi que sur le temps d'entraînement, confortent les observations de nos prédécesseurs en élargissant le champ d'étude sur d'autres tâches.

Mots-clés : Réseaux de neurones artificiels, Apprentissage profond, Perceptron multicouche, Réseaux de neurones convolutifs, Réseaux de neurones récurrents, Mémoire long à court terme, Unité récurrente à portes.

✠ Abstract ✠

Deep Learning (DL) has emerged as a revolutionary paradigm in the field of artificial intelligence and machine learning. Based on artificial neural networks, DL has proven effective in various application areas such as voice recognition (Apple's SIRI), machine translation (Google Translate), and many others.

In this thesis, we propose a state-of-the-art review and a comparative study of the three basic deep learning architectures : Multi-Layer Perceptron (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN).

We will present the origin and historical evolution of deep learning and its theoretical foundations : the formal neuron, activation functions, model evaluation, neural network topologies, deep learning training techniques, and optimization algorithms.

We will discuss the architecture of the MLP, its operation, the backpropagation algorithm, its application domain, and its limitations.

For CNNs, we will present the principle of convolution, which forms the basis of CNNs, as well as their architectures with their different layers : convolutional layer, pooling layer, and fully connected layer. We will also discuss their application domains and limitations.

For RNNs, in addition to their architecture, we will present their training mechanism with backpropagation through time (BPTT). We will discuss their learning issues, namely the vanishing gradient and exploding gradient problems. We will present the variants of RNNs : Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). Their application domains and limitations will also be covered.

Previous studies have shown that CNNs are more suited for image processing tasks, while RNNs are more apt for processing sequential data or time series.

We proposed a comparative study between the MLP and CNN architectures for image classification using the CIFAR-10 dataset and between the CNN and RNN architectures for sentiment analysis using the IMDB dataset, as well as for poem generation using a collection of poems by Victor Hugo.

The results of this study on accuracy, precision, recall, F1 score metrics, as well as training time, support the observations of our predecessors by expanding the study field to other tasks.

Keywords : Artificial Neural Networks, Deep Learning, Multi-Layer Perceptron, Convolutional Neural Networks, Recurrent Neural Networks, Long Short-Term Memory, Gated Recurrent Unit.

✠ Table des matières ✠

✠ Liste des acronymes ✠

IA Intelligence Artificielle

ML Machine Learning

ANN Artificial Neural Network

DL Deep Learning

MLP Multi Layer Perceptron

CNN Convolutional Neural Network

RNN Recurrent Neural Networm

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

SVM Suport Vector Machine

KNN K-Nearest Neighbors

GAN Generative Adversarial Network

ReLU Rectified Linear Unit

MSE Mean Squared Error

MAE Mean Absolute Error

BCE Binary Cross-Entropy

CCE Categorical Cross-Entropy

AE Auto-Encodeur

RBM Restricted Boltzmann Machines

✠ Liste des Algorithmes ✠

✠ Table des figures ✠

✠ Liste des tableaux ✠

✠ Introduction générale ✠

Depuis son émergence, le Deep Learning (DL) a déclenché une transformation majeure dans le domaine de l' Intelligence Artificielle (IA) et de l'apprentissage automatique (Machine Learning (ML)) [1]. Cette révolution s'est manifestée par une série d'exploits spectaculaires, allant de la reconnaissance d'images à la traduction automatique en passant par la génération de texte et bien d'autres domaines. L'essor fulgurant du deep learning a fait de l'IA une réalité tangible, capable de transformer radicalement notre façon de vivre, de travailler et d'interagir avec le monde qui nous entoure [2]. Mais avant de plonger pleinement dans ce domaine, il convient de jeter un regard rétrospectif sur deux concepts fondamentaux : l'intelligence artificielle et le machine learning.

L'IA, telle que définie par le professeur John McCarthy de l'université de Stanford en 1955, représente la volonté de l'homme de doter les machines d'une intelligence semblable à la nôtre, capable de résoudre des problèmes complexes avec beaucoup d'efficacité et de précision [3]. L'IA est un croisement de plusieurs domaines tels que les mathématiques, la statistique, la neuroscience et l'informatique [4].

Le ML, quant à lui, est une branche de l'IA qui permet aux systèmes d'apprendre à partir de données et de s'améliorer avec l'expérience sans être explicitement programmés [5]. Plutôt que de suivre des instructions spécifiques, les algorithmes de ML utilisent des données pour entraîner des modèles, capturant ainsi des relations complexes entre les variables. Ces modèles peuvent ensuite être utilisés pour effectuer des prédictions, faire des recommandations ou prendre des décisions, même sur des données qu'ils n'ont jamais rencontrées [6]. Parmi les algorithmes de ML les plus courants, nous pouvons citer les séparateurs à vaste marge plus connus sous le nom de machines à vecteurs de support (Suport Vector Machine (SVM)), les forêts aléatoires (Random Forest), les K plus proches voisins (K-Nearest Neighbors (KNN)), les réseaux de neurones artificiels (Artificial Neural Network (ANN)), entre autres [7–10].

Le DL émerge comme une évolution majeure du ML, suscitant beaucoup d'intérêt pour sa capacité à résoudre des problèmes complexes et à modéliser des données de grande dimension [2]. Il repose sur l'utilisation de réseaux de neurones artificiels profonds, inspirés de l'organisation et du fonctionnement du cerveau humain. Ces réseaux, lorsqu'ils sont configurés avec plusieurs couches, peuvent apprendre des représentations de données avec différents niveaux d'abstraction, ce qui les rend aptes à traiter des tâches complexes telles que la reconnaissance d'images, la traduction automatique, la synthèse vocale, la prédiction, la classification de données, pour ne citer que celles-là [11].

L'avènement du DL a été rendu possible par plusieurs facteurs convergents. La masse de données numériques provenant de diverses sources telles que les réseaux sociaux, les capteurs **IoT**, les appareils mobiles et les caméras de surveillance a fourni le carburant nécessaire pour entraîner efficacement des modèles de deep learning [12]. De

plus, l'utilisation de technologies de calcul parallèle telles que les unités de calcul graphique (Graphical Processing Units (GPU)), a permis d'accélérer de manière significative l'entraînement de ces modèles sur de vastes ensembles de données.

Le deep learning a révolutionné de nombreux domaines, notamment la vision par ordinateur, le traitement du langage naturel, la biologie, la robotique, et bien d'autres [13–16].

Malgré ses succès retentissants, le deep learning soulève encore de nombreuses questions fondamentales, tant sur le plan théorique que pratique. La compréhension des concepts sous-jacents du deep learning reste partielle. Du point de vue pratique, les défis liés au choix et à la conception d'architectures performantes pour des tâches spécifiques méritent une attention particulière.

Face à ces défis, notre travail vise à apporter des explications claires et détaillées sur les fondements théoriques et pratiques du deep learning.

Notre travail est axé sur deux parties essentielles : un état de l'art du deep learning et une étude comparative entre les architectures classiques du deep learning. Nous allons comparer le **Perceptron multicouche** (MLP) et les **réseaux de neurones convolutifs** (CNN) pour une tâche de classification d'image, et aussi faire une comparaison entre les CNN et les **réseaux récurrents** (RNN) ainsi que leurs variantes à savoir les mémoires à long et court terme (LSTM) et les unités récurrents à portes (GRU) pour deux tâches distinctes : l'analyse de sentiment sur des critiques de films avec le jeu de données IMDB, et la génération de poèmes en utilisant un recueil de poèmes de Victor Hugo.

Notre mémoire est structuré en cinq chapitres. Le premier chapitre présente les concepts fondamentaux des réseaux de neurones, établissant les bases théoriques. Le deuxième chapitre se focalise sur le perceptron multicouche (MLP) en présentant son architecture et son mécanisme d'entraînement. Le troisième chapitre aborde les réseaux de neurones convolutifs, leur architecture avec ses différentes couches ainsi que leur fonctionnement. Le quatrième chapitre examine les réseaux de neurones récurrents, leur structure fonctionnelle, leur processus de fonctionnement ainsi que leurs variantes, notamment le LSTM et le GRU. Le cinquième et dernier chapitre présente une étude comparative des différentes architectures abordées, analysant leurs performances respectives selon les tâches et les jeux de données.

⋆ ⋆ Chapitre I ⋆ ⋆

Concepts de base des Réseaux de Neurones

Introduction

Le cerveau humain est composé de près de 100 milliards de neurones interconnectés de manière complexe [17]. De par son mécanisme de fonctionnement, il a beaucoup inspiré les chercheurs dans le domaine de l'IA plus particulièrement de l'IA connexionniste.
Les réseaux de neurones artificiels (ANN) sont des systèmes de traitement de l'information inspirés de la structure et du fonctionnement du cerveau humain. Ils sont constitués d'unités élémentaires interconnectées (neurones artificiels) qui coopèrent pour effectuer un traitement distribué et parallèle des données en entrée [18]. À l'instar du cerveau animal, les réseaux de neurones artificiels présentent des capacités remarquables d'apprentissage, de mémorisation et de généralisation à partir d'exemples.

Contrairement aux méthodes d'apprentissage automatique traditionnelles comme les SVM ou les kNN, qui requièrent une importante phase de conception manuelle de caractéristiques, les ANN sont capables d'extraire automatiquement les caractéristiques pertinentes à partir des données brutes, par apprentissage [19]. Cette capacité d'apprentissage en profondeur (deep learning) leur confère un avantage décisif pour aborder des problèmes complexes impliquant des données d'une certaine nature comme les images, le son, du texte, de l'audio, de la video et bien d'autres [2].

Avant d'approfondir l'étude des architectures de base du DL, il est primordial de bien comprendre les fondements théoriques qui sous-tendent leur fonctionnement. Ce chapitre pose les bases techniques et conceptuelles indispensables en explorant l'évolution historique des neurones artificiels, les fonctions d'activations, les fonctions de perte, les topologies standards et les algorithmes d'optimisation des modèles.

1 Histoire et évolution des réseaux de neurones

1.1 Origine

Les premiers concepts de réseaux de neurones remontent vers années 40 lorsque les scientifiques ont commencé à s'intéresser à la modélisation mathématique du fonctionnement du cerveau humain. En 1943, Warren McCulloch un neurophysiologiste et Walter Pitts un jeune mathématicien, ont proposé le premier modèle de neurone artificiel [20]. Ils ont montré que leur modèle pouvait réaliser des fonctions logiques et arithmétiques (tout au moins au niveau théorique) [21].

En 1949, le psychologue Donald Hebb a introduit une théorie sur l'apprentissage dans les réseaux neuronaux biologiques, connue sous le nom de règle de Hebb [22]. Celle-ci stipule que lorsque deux neurones sont activés simultanément de manière répétée, la force de la connexion entre eux augmente. Cette règle a jeté les bases des algorithmes d'apprentissage utilisés plus tard dans les ANN.

1.2 Les premiers succès

La première application concrète des réseaux de neurones artificiels est présentée en 1958 avec le modèle dit «Perceptron» de Frank Rosenblatt [23]. Le perceptron est une machine électronique fonctionnelle construite sur des principes biologiques et qui avait une capacité d'apprentissage. Il a été capable de réaliser des classifications binaires simples et d'apprendre à partir d'exemples. Une application importante du Perceptron était sa capacité à calculer des fonctions logiques de base, telles que la porte AND et la porte OR.

En 1960 Bernard Widrow et Ted Hoff, deux ingénieurs électriciens présentent le modèle ADALINE (ADAptative LInear Element) [24]. La méthode d'apprentissage utilisée avec ce modèle était différente de celle du Perceptron, elle employait la règle d'apprentissage des moindres carrés moyens. Concrètement, cela signifie que l'algorithme évalue l'importance relative de chaque entrée dans le calcul de l'erreur avant de modifier son poids. Ce modèle a été appliqué avec succès à des problèmes comme le filtrage adaptatif de signal.

1.3 La période sombre

Après une période d'enthousiasme, le domaine a connu par la suite des moments de frustration et de disgrâce. Cette période fut marqué par la publication du livre de Minsky et Papert «**Perceptron : An Introduction to Computational Geometry** » en 1969 [25] qui est venu jeter beaucoup d'ombre sur le domaine. Dans ce livre, ils ont présenté le perceptron comme une machine à deux couches qui ne peut traiter que des données qui sont linéairement séparables, en prenant comme illustration, le problème du **OU exclusif** (XOR). Ils ont conclu à tort que le domaine des réseaux de neurones était une voie sans issue et qu'il fallait cesser de s'y intéresser. Ce qui a réduit de façon considérable l'intérêt public et les fonds disponibles devinrent minimes, seuls quelques chercheurs ont continué à travailler sur le domaine.

Malgré la situation, au cours de cette période beaucoup d'idées ont été avancées, qui seront par la suite développées et améliorées.
Paul Werbos met en place le processus d'apprentissage par rétropropagation en 1974 [26], bien que son utilité ne soit pleinement appréciée qu'en 1986. En 1975, Konihiko Fukushima développe un réseau de neurones multicouche formé par étape pour l'interprétation des caractères manuscrits [27]. En 1976, Stephen Grossberg présente un modèle théorique du traitement de l'information cognitive chez l'humain [28].

1.4 Le renouveau

Dans les années 80, les idées nouvelles et les travaux des chercheurs qui n'ont jamais abandonné même pendant les moments les plus critiques, ont suscité un regain d'intérêt pour le domaine.
Teuvo Kohonen présente son architecture de réseau de neurones dite réseau de Kohonen en 1982 [29]. Dans la même année, Hopfield a développé les réseaux de neurones récurrents qui servent de système de mémoire adres-

sable par le contenu [30]. Avec ses travaux, Hopfield a persuadé des centaines de scientifiques hautement qualifiés de rejoindre le domaine des réseaux de neurones.

En 1986, des groupes de chercheurs découvrent parallèlement l'algorithme de la rétropropagation plus connu sous le nom de **backpropagation**, bien que cette découverte fut accordée dans plusieurs articles à Rumelhart et son équipe [31]. L'algorithme de la rétropropagation est une forme d'apprentissage qui utilise la descente de gradient lors de la formation d'un réseau de neurones.

En 1987, Geoffrey Hinton développe le Perceptron Multicouche [32]. Il s'agit d'une combinaison de perceptron simple qui utilise la rétropropagation pour son entraînement.

A partir des années 90, de nouveaux types d'architectures vont apparaître dans les sujets de recherche, on parle de réseaux de neurones profonds. Yann LeCun et son équipe présentent les réseaux de neurones convolutifs capables de reconnaître des chiffres manuscrits [33]. Dans la même période, Sepp Hochreiter et Jürgen Schmidhuber introduisent les mémoires à longue et court terme (Long Short-Term Memory (LSTM)) [34], une variante des réseaux de neurones récurrents simples qui est aujourd'hui beaucoup utilisée dans le traitement des données séquentielles.

1.5 Ère du deep learning

L'ère moderne du deep learning, marquant une étape décisive dans la réalisation du potentiel des réseaux de neurones artificiels, a véritablement débuté au milieu des années 2000. En 2006, Geoffrey Hinton et ses étudiants redécouvrent une technique d'apprentissage non supervisé appelée "**pré-entraînement par machines de Boltzmann restreintes**" [11] pour initialiser efficacement les poids de réseaux profonds. En 2012, l'équipe de Hinton avec le modèle CNN **AlexNet** remporte la compétition **ImageNet** sur la reconnaissance d'images, montrant la supériorité des réseaux neurones profonds sur les méthodes traditionnelles. Depuis ce jour, beaucoup de recherches scientifiques et informatiques se sont orientées dans le domaine du deep learning. Cela a suscité un engouement énorme au tour du domaine de l'intelligence artificielle. L'ère actuelle du deep learning marque une étape décisive dans la réalisation du potentiel longtemps promis par les réseaux de neurones artificiels.

Dans les sections suivantes, nous présenterons les concepts fondamentaux qui sous-tendent le domaine du deep learning.

2 Neurone biologique

Le neurone biologique, appelé cellule nerveuse, est l'unité principale du système nerveux, responsable du traitement et de la transmission de l'information sous forme d'impulsions électriques et chimiques [35]. Bien que les neurones puissent présenter différentes morphologies selon leur type et leur emplacement, ils partagent certains composants de base comme le montre la figure I.1.

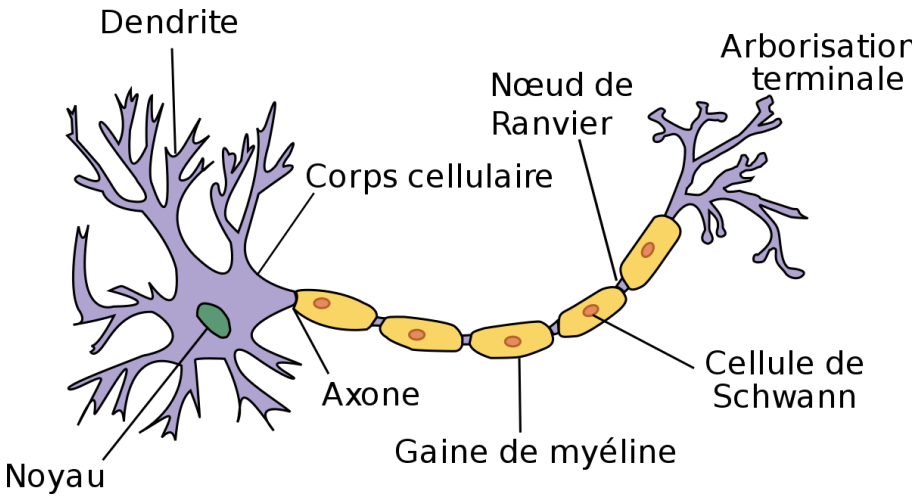

FIGURE I.1 – *Représentation d'un neurone biologique*

- **Corps cellulaires**(soma ou péricaryon) : ils constituent le noyau du neurone, c'est la région dans laquelle les signaux sont traités puis envoyés vers d'autres neurones qui lui sont connectés.
- **Dendrites** : elles se chargent de capturer les signaux extérieurs et leur intensité, puis de les transmettre aux autres neurones qui leurs sont connectés.
- **Synapses** : sont les points d'interconnexion entre les différents neurones.
- **Axone** : il s'agit du canal via lequel le message nerveux est produit puis transmis d'un neurone vers les autres neurones directement connectés.

Les neurones agissent ainsi comme des unités de traitement d'information interconnectées, formant des réseaux complexes au sein du système nerveux. Leur fonctionnement coordonné permet la perception sensorielle, le contrôle moteur, la cognition et de nombreuses autres fonctions vitales [35].

3 Neurone formel ou artificiel

3.1 Structure

Le neurone formel est un modèle mathématique simplifié s'inspirant du fonctionnement du neurone biologique, proposé par McCullogh et Pitts en 1943 [20]. Il constitue l'unité de base des réseaux de neurones artificiels utilisés en deep learning. Le neurone artificiel possède généralement plusieurs entrées et une sortie qui correspondent aux dendrites et au cône d'émergence du neurone biologique (point de départ de l'axone). Chaque entrée est associée à un coefficient numérique (poids synaptique) qui représente les actions excitatrices et inhibitrices des synapses. La figure I.2 montre une représentation comparative entre le neurone biologique et le neurone formel [36].

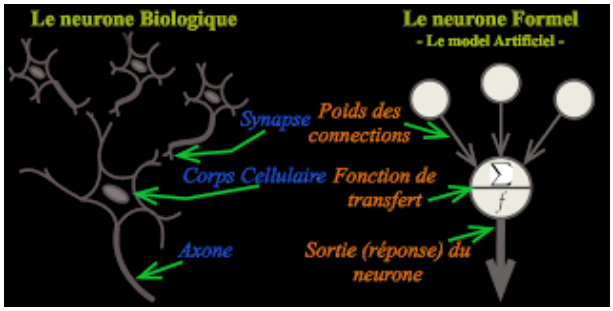

FIGURE I.2 – Neurone formel

Le corps cellulaire du neurone biologique est représenté par une fonction de transfert dans le neurone formel. Les synapses qui définissent les forces de connexion entre deux neurones distincts, sont représentés par les poids des connexions entre les entrées et le neurone. L'axone est à son tour est modélisé par la sortie ou la réponse du neurone formel.

3.2 Fonctionnement du neurone formel

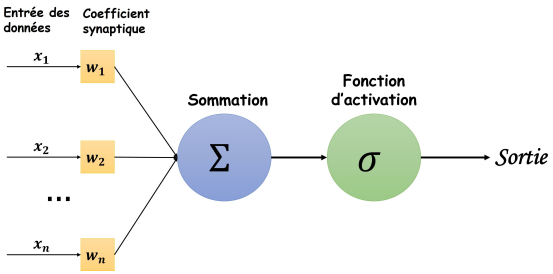

FIGURE I.3 – Représentation d'un neurone formel

Comme illustré dans la figure I.3, le fonctionnement du neurone formel peut être décrit en deux étapes principales :

1. **Agrégation** : le neurone calcule une somme pondérée (z) de ses entrées X_i avec les poids correspondants W_i :

$$z = \sum_{i=1}^{n} W_i X_i \tag{I.1}$$

2. **Activation** : cette somme pondérée z est ensuite passée à travers une fonction d'activation σ, pour produire la sortie finale y :

$$y = \sigma(z) \tag{I.2}$$

La fonction d'activation σ détermine si le neurone est activé ou inhibé en fonction de la valeur de z. Elle introduit une non-linéarité essentielle au modèle, permettant au réseau de neurones d'approximer des fonctions complexes. Ce processus transforme les entrées multidimensionnelles en une sortie unique, constituant ainsi l'unité de base du traitement de l'information dans les réseaux de neurones artificiels [36].

Il existe une variété de fonctions d'activation, chacune ayant ses caractéristiques et applications spécifiques. Dans la section 4, nous présenterons quelques-unes des fonctions d'activation les plus couramment utilisées et leurs propriétés principales.

4 Fonction d'activation

Une fonction d'activation est la fonction qui prend l'entrée combinée z décrite dans la sous-section 3.2, lui applique une transformation et produit une valeur de sortie, cherchant ainsi à reproduire le potentiel d'action observé dans les neurones biologiques [37]. Elle détermine donc l'état d'activation du neurone.
Sans fonctions d'activation, la sortie des neurones serait simplement la somme obtenue dans l'équation I.1, qui est une fonction linéaire. Par conséquent, l'ensemble du réseau de neurones, étant une composition de neurones, deviendrait une composition de fonctions linéaires, ce qui est également une fonction linéaire. Cela signifierait que, même en ajoutant des couches cachées, le réseau resterait équivalent à un simple modèle de régression linéaire, avec toutes ses limitations.
Pour introduire de la non-linéarité dans les neurones, nous utilisons des fonctions d'activation non linéaires. Ces fonctions possèdent des caractéristiques essentielles telles que la dérivabilité et la capacité à atteindre un optimum global sans rester coincées dans un optimum local [38].
Il existe plusieurs choix de fonctions d'activation, les plus couramment utilisées étant la fonction sigmoïde (sous-section 4.1), (ReLU) (sous-section 4.2), tangente hyperbolique (TanH) (sous-section 4.4) et Softmax (soous-section 4.3).

4.1 Fonction sigmoïde ou logistique

Cette fonction est l'une des fonctions les plus couramment utilisée. Elle transforme l'entrée sur une plage comprise entre 0 et 1 [39] comme illustré dans le figure I.4. La fonction sigmoïde, également connue sous le nom de fonction logistique, est un choix classique dans les réseaux de neurones. Elle convient pour des taches de classification binaire [40]. Elle est définie par l'équation (I.3).

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \text{(I.3)}$$

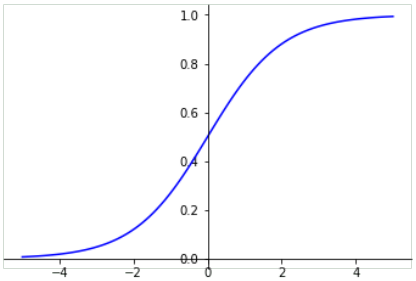

FIGURE I.4 – Fonction Simoide

4.2 Fonction relu (ReLU)

La fonction ReLU renvoie l'entrée telle quelle si elle est positive, et zéro dans le cas contraire comme le montre la figure I.5. Elle favorise la simplicité et améliore l'efficacité des calculs en annulant les valeurs négatives [39]. Dans les tâches de vision par ordinateur telles que la détection et la segmentation d'objets, la fonction d'activation ReLU s'est révélée très efficace en raison de sa capacité à gérer des ensembles de données à grande échelle et à capturer des fonctionnalités complexes [40]. Elle est définie par l'équation (I.4).

$$\text{ReLU}(z) = \max(0, z) \qquad \text{(I.4)}$$

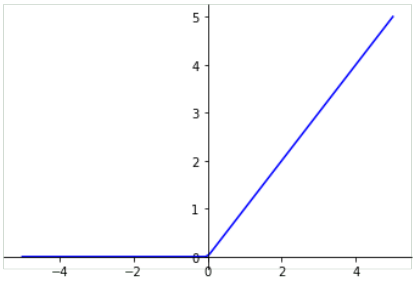

FIGURE I.5 – Fonction ReLu

4.3 Fonction SoftMax

La fonction SoftMax prend un vecteur de nombres réels et les convertit en une distribution de probabilité, où chaque élément du vecteur est transformé en une valeur comprise entre 0 et 1 [39]. Ainsi elle permet de normaliser les sorties d'une couche de réseau de neurones, garantissant qu'elles totalisent 1, ce qui permet de les interpréter comme des probabilités de classe comme illustré dans la figure I.6. Elle est très utile pour des prédictions fiables en produisant des probabilités de classe, aidant ainsi à la prise de décision pour diverses applications de traitement du langage et de vision par ordinateur [40]. Mathématiquement, pour un vecteur d'entrée $\vec{z} = (z_1, z_2, ..., z_K)$ de dimension K (où K est le nombre de classes), la fonction softmax est définie par l'équation (I.5) pour $i = 1, 2, \ldots, K$.

$$\text{softmax}(\vec{z})i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \text{(I.5)}$$

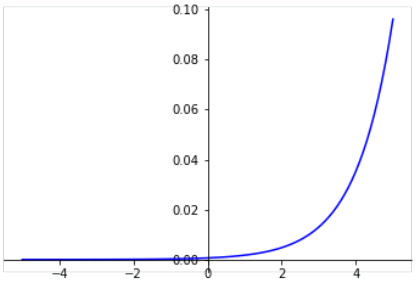

FIGURE I.6 – Fonction SoftMax

4.4 Fonction tangente hyperbolique (Tanh)

Comme illustré dans la figure I.7, la fonction Tanh transforme l'entrée sur une plage comprise entre -1 et 1, présentant des propriétés similaires à celles de la fonction sigmoïde mais avec une transition plus rapide autour de zéro [39]. Elle trouve une application dans les tâches de vision par ordinateur nécessitant des résultats dans une plage spécifique, telles que la reconnaissance des expressions faciales ou la reconnaissance des gestes [40]. Elle est défini par l'équation (I.6).

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \qquad \text{(I.6)}$$

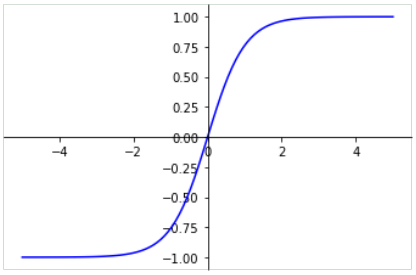

FIGURE I.7 – Fonction TanH

5 Évaluation des modèles

Pour s'assurer de la performance des modèles d'apprentissage profond, nous utilisons deux approches distinctes. La première approche permet d'évaluer l'erreur commise par le modèle lors de son apprentissage, et la seconde consiste à mesurer son efficacité et sa capacité à se généraliser sur les données de validation. Ces deux techniques sont généralement représentées par les concepts respectifs : fonction de coût abordée en sous-section 5.1 et métrique d'évaluation abordée en sous-section 5.2.

5.1 Fonction coût ou perte

La fonction coût, également connue sous le nom de fonction perte ou fonction objective a pour rôle d'évaluer la qualité des prédictions générées par un modèle en mesurant l'écart entre ces prédictions et les valeurs réelles attendues [41]. En d'autres termes, elle permet de quantifier l'erreur du modèle en fournissant une mesure de la différence entre les prédictions et les véritables valeurs cibles. Il existe une diversité de fonctions de perte, chacune adaptée à des types spécifiques de problèmes, notamment la régression et la classification.

1. Problème de régression

   Plusieurs fonctions coûts sont utilisé pour la régression, nous nous limiterons à en présenter deux qui sont très largement utilisées dans la pratique.

   ◇ Erreur quadratique moyenne :

   Erreur quadratique moyenne (Mean Squared Error (MSE)) calcule la moyenne des carrés des différences entre les valeurs prédites par le modèle et les valeurs réelles des données d'entraînement. La formule de la MSE comme indiquée dans l'équation (I.7) reflète cette notion en prenant la moyenne des carrés des écarts entre les prédictions et les valeurs réelles, normalisée par le nombre d'observations [42].

   $$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{(I.7)}$$

   ◇ Erreur absolue moyenne :

   Contrairement à la MSE, l'erreur absolue moyenne (Mean Absolute Error (MAE)) MAE calcule la moyenne des valeurs absolues des différences entre les prédictions et les valeurs réelles. Elle est définie dans l'équation I.8 :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{I.8}$$

2. Problème de classification

   Les fonctions de perte couramment utilisées pour les problèmes de classification comprennent : l'entropie croisée (équation I.9) binaire utilisée pour la classification binaire et l'entropie croisée catégorielle (équation I.10) utilisée pour la classification multiclasse.

   ◇ Entropie croisée binaire(Binary Cross-Entropy (BCE)) :

$$\text{BCE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log(\hat{y}_i) + (1-y_i)\cdot \log(1-\hat{y}_i)] \tag{I.9}$$

   ◇ Entropie croisée catégorielle(Categorical Cross-Entropy (CCE)) :

$$\text{CCE} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \cdot \log(\hat{y}_{ij}) \tag{I.10}$$

5.2 Métriques d'évaluation

Il existe différentes façons de mesurer l'efficacité d'un modèle, notamment sa capacité à confondre les exemples des différentes classes, l'exactitude de ses prédictions, sa précision à classer les différents exemples, sa sensibilité, ainsi que son score. Ces différentes méthodes sont présentées ci-dessous. Dans la pratique, l'utilisation de ces différentes métriques nécessite la séparation du jeu de données en données d'entraînement et en données de test, et dans certains cas, en données de validation.

◇ **Matrice de confusion** (Confusion matix) : Elle fournit une visualisation des prédictions faites par le modèle par rapport aux exemples réels des données de test. Elle fournit également une représentation tabulaire des résultats.

	Predition Positive	**Prediction Negative**
Réel Positif	TP	FN
Réel Negatif	FP	TN

◇ **Exactitude** (Accuracy) : Mesure la proportion des prédictions correctes faites par le modèle sur les données d'entraînement en utilisant la formule :

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

◇ **Précision** (Precision) : permet de mesurer le taux de prédiction correctes sur l'ensemble du jeu de données avec la formule :

$$\text{Precision} = \frac{TP}{TP+FP}$$

◇ **Rappel** (Recall) : également connu sous le nom de **sensibilité** , mesure la proportion de prédictions positives réelles entre tous les cas positifs réels à travers la formule :

$$\text{Recall} = \frac{TP}{TP+FN}$$

◇ **Score F1** (F1-Score) : équilibre la précision et le rappel. Autrement dit, elle est calculée comme la moyenne harmonique de la précision et du rappel. Elle permet de pénaliser les valeurs extrêmes négatives de l'un ou l'autre entre la precision et le rappel, ce qui la rend très utile.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

◇ **Spécificité** (Specifity) : permet de mesure la proportion de vrais négatifs correctement identifiés parmi tous les cas négatifs réels. Elle détermine la capacité du modèle à identifier correctement les cas négatifs. Sa formule est donnée par :

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

Avec :

○ **TP (True Positive)** : le modèle a fait une prédiction positive et elle est bonne.

○ **TN (True Negative)** : le modèle a fait une prédiction négative et elle est bonne.

○ **FP (False Positive)** : le modèle a fait une prédiction positive et s'est trompé.

○ **FN (False Neagative)** : le modèle a fait une prédiction négative et s'est trompé.

6 Topologies des réseaux de neurones

La topologie d'un réseau neurone fait référence à la manière dont les neurones sont connectés, et c'est un facteur important dans le fonctionnement et l'apprentissage du réseau [43]. Elle peut être quelconque, mais le plus souvent, il est possible de distinguer une certaine régularité.

6.1 Réseaux entièrement connectés

Dans cette topologie, les neurones sont arrangés par couches. Il n'y a pas de connexion entre neurones d'une même couche et les connexions ne se font qu'entre neurones de couches proches (qui se succèdent directement). Comme le montre la figure I.8, chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous amène à introduire le concept de sens de parcours de l'information au sein du réseau et à définir donc les concepts de neurone d'entrée et de neurone de sortie. Par extension, on appelle couche d'entrée l'ensemble des neurones d'entrée, et couche de sortie l'ensemble des neurones de sortie. Les couches intermédiaires n'ayant aucun contact avec l'extérieur sont appelées couches cachées.

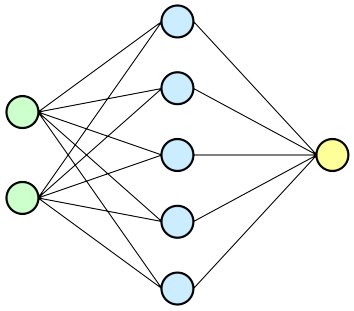

FIGURE I.8 – Topologie de réseau de neurones entièrement connecté

6.2 Réseaux à connexions locales

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Comme illustré dans la figure I.9, chaque neurone entretient des relations avec un nombre réduit et localisé de neurones de la couche d'avant. Les connexions sont donc moins nombreuses que dans le cas d'une topologie de réseau entièrement connecté.

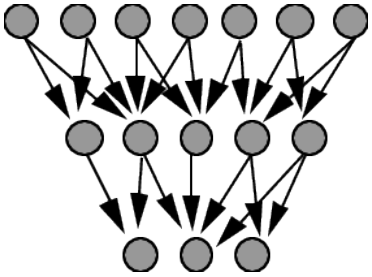

FIGURE I.9 – Topologie de réseau de neurones à connexion local

6.3 Réseaux à connexions récurrents

Les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multicouche. Ces connexions sont le plus souvent locales. Dans cette topologie, chaque neurone reçoit non seulement les entrées de la couche précédente, mais également un retour de sa propre sortie à l'instant précédent.

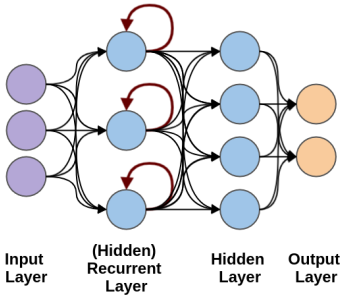

FIGURE I.10 – Topologie de réseau de neurones à connexion récurrente

7 Technique d'apprentissage des réseaux de neurones

Un des grands atouts des réseaux de neurones artificiels est leur capacité à "**apprendre**" à partir de données, de manière automatique. L'apprentissage dans le contexte des réseaux de neurones consiste à entraîner un algorithme (ou modèle) sur un ensemble de données qui peuvent être soit annotées soit non annotées selon la nature du problème. De ce fait, différentes techniques d'apprentissage ont été proposées, chacune offrant différents algorithmes d'apprentissage qui s'adaptent au mieux au type de données à traiter.

On distingue principalement trois grands paradigmes d'apprentissage : l'apprentissage supervisé (sous-section 7.1), semi-supervisé (sous-section 7.3) et non supervisé (sous-section 7.2).

7.1 Apprentissage supervisée

Comme son nom l'indique, l'apprentissage supervisé consiste à utiliser des données étiquetées pour entraîner des algorithmes sur des tâches de prédiction ou de classification [44]. Ces données d'entraînement comprennent des entrées et des sorties correctement annotées, ce qui permet au modèle de s'améliorer au cours du temps d'entraînement. L'algorithme mesure sa précision à travers une fonction de perte, et ajuste ses paramètres jusqu'à ce que l'erreur soit suffisamment réduite.
Parmi les algorithmes d'apprentissage supervisé utilisés dans le DL, on peut citer :
▷ perceptron multicouche,
▷ réseaux de neurones convolutifs,
▷ réseaux de neurones récurrents et ses dérivées(LSTM, GRU).

7.2 Apprentissage non-supervisée

Contrairement à l'apprentissage supervisé, l'apprentissage non supervisé utilise des données qui ne sont pas étiquetées [11]. Avec ces données, il trouve des modèles qui aident à résoudre les problèmes de regroupement ou d'association. Ceci pourrait être utile dans le cas où les spécialistes du domaine ne sont pas sûrs des propriétés communes dans un jeu de données. Plusieurs algorithmes de deep learning sont disponibles pour faire de l'apprentissage non supervisé, dont les plus populaires sont :
▷ réseaux antagonistes génératifs (Generative Adversarial Network (GAN)),
▷ auto-encodeurs (Auto-Encodeur (AE)),
▷ réseaux de boltzmann restreints (Restricted Boltzmann Machines (RBM)).

7.3 Apprentissage Hybride

Aussi appelé apprentissage semi-supervisé, l'apprentissage hybride combine à la fois l'apprentissage supervisé et l'apprentissage non supervisé en utilisant à la fois des données labélisées et des données qui ne le sont pas lors de la formation du modèle [45]. Dans certains cas de figure, il est difficile voire coûteux d'obtenir un volume suffisant de données annotées seulement, alors que les données non étiquetées sont particulièrement accessibles. Dans de tels cas, ni les méthodes d'apprentissage entièrement supervisées ni non supervisées ne fourniront des solutions adéquates. D'où l'utilité de la méthode hybride. Avec le processus de l'apprentissage hybride, nous n'avons pas d'algorithmes spécifiques, mais plutôt des combinaisons d'algorithmes supervisés et non supervisés.
Il existe également une autre forme d'apprentissage hybride, l'apprentissage par renforcement, où un agent apprend par essais/erreurs en interagissant avec un environnement pour maximiser une récompense cumulative [46].

8 Algorithmes d'optimisation

Le rôle d'un algorithme d'optimisation au sein du modèle de réseau de neurones durant la phase d'apprentissage est crucial. Il permet au modèle d'ajuster au mieux les paramètres à travers des calculs pour être plus performant.

Il existe de nombreux algorithmes d'optimisation, donc il est important de faire le bon choix. Nous allons présenter les plus populaires et les plus couramment utilisés dans le domaine du deep learning.

8.1 Descente de gradient

Dans le processus de la descente de gradient, le modèle cherche à minimiser la fonction coût [47]. Autrement dit, il cherche à s'approcher étape par étape du minimum de la fonction d'erreur abordée dans la sous-section II.3. Pour ce faire, comme illustré dans la figure I.11, il part d'un point initial, et évalue la valeur de la dérivée de cette fonction en ce point. Selon le signe de la dérivée, il se déplace vers la droite ou vers la gauche jusqu'à atteindre le niveau le plus proche du minimum [48]..

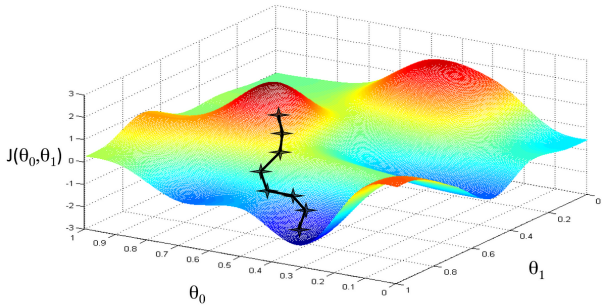

FIGURE I.11 – Algorithme de la descente de gradients

Algorithm 1 Descente de Gradient

Require: α : Taille du pas (taux d'apprentissage)

Require: $f(\theta)$: Fonction objectif avec les paramètres θ

Require: θ_0 : Vecteur de paramètres initial

Ensure: θ_t : Paramètres résultants

1: $t \leftarrow 0$ ▷ Initialiser le compteur de temps

2: $\theta_t \leftarrow \theta_0$ ▷ Initialiser les paramètres

3: **while** θ_t n'a pas convergé **do**

4: $\quad t \leftarrow t+1$

5: $\quad g_t \leftarrow \nabla_\theta f(\theta_{t-1})$ ▷ Calculer le gradient de la fonction objectif

6: $\quad \theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t$ ▷ Mettre à jour les paramètres

7: **end while**

8: **return** θ_t ▷ Paramètres résultants

◇ **Remarques**

Le paramètre α appelé taux d'apprentissage (ou learning rate en anglais), définit la vitesse de convergence de l'algorithme [49]. Autrement dit, il détermine la taille des pas effectués pour mettre à jour les poids à chaque itération. Il est donc évident que son choix sera crucial pour la convergence de l'algorithme. Si α est trop élevé, cela peut faire oscillé l'algorithme, tandis qu'un taux trop faible ralentira considérablement la vitesse de convergence. De plus, sur des jeux de données volumineux, la convergence de l'algorithme peut être ralentie considérablement.

Cela est lié en partie à un coût de calcul élevé et aussi à la difficulté de parallélisation des calculs pour certaines architectures de réseaux de neurones.

8.2 Descente de gradient stochastique

L'algorithme de la descente de gradient stochastique (Stochastic Gradient Descent (SGD)) est une variante de la descente de gradient classique utilisée aussi pour optimiser les modèles de machine learning et de deep learning [50]. Contrairement à l'algorithme de la descente de gradient classique, le SGD permet au modèle de faire face à de grandes quantités de données. En effet, au lieu d'utiliser l'intégralité du jeu de données sur chaque itération, le SGD sélectionne aléatoirement un petit lot sur l'ensemble des données d'entraînement pour calculer les gradients et mettre à jour les paramètres du modèle. Le terme **stochastique** fait référence à l'aspect aléatoire utilisé lors de la sélection des lots. En utilisant un seul exemple ou un petit lot, le coût de calcul par itération peut être considérablement réduit.

Algorithm 2 Descente de Gradient Stochastique (SGD)

Require: α : Taille du pas

Require: $f(\theta)$: Fonction objectif avec les paramètres θ

Require: θ_0 : Vecteur de paramètres initial

Require: $\{x_i, y_i\}_{i=1}^{N}$: Ensemble de données de taille N

Ensure: θ_t : Paramètres résultants

$\theta_t \leftarrow \theta_0$ ▷ Initialiser les paramètres

while θ_t n'a pas convergé **do**

  for chaque échantillon (x_i, y_i) de l'ensemble de données **do**

    $g_t \leftarrow \nabla_\theta f(\theta_{t-1}; x_i, y_i)$ ▷ Calculer le gradient de la fonction objectif pour l'échantillon (x_i, y_i)

    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot g_t$ ▷ Mettre à jour les paramètres

  end for

end while

return θ_t ▷ Paramètres résultants

8.3 Adam (ADAptative du Moment)

L'optimiseur **Adam** est une extension de la descente de gradient stochastique. Avec Adam, les moyennes courantes des gradients et des seconds moments des gradients sont utilisées pour calculer les taux d'apprentissage adaptatifs pour chaque paramètre. Contrairement aux méthodes traditionnelles qui utilisent un taux d'apprentissage fixe, Adam ajuste dynamiquement les taux d'apprentissage pour chaque paramètre et applique une correction de biais pour améliorer l'efficacité dès le départ. De plus, Adam est robuste face aux gradients rares et fonctionne bien dans des environnements non stationnaires, offrant ainsi une solution plus efficace et adaptable pour l'optimisation dans les réseaux de neurones profonds [51].

Algorithm 3 Adam

[51]

Require: α : Taille du pas

Require: $\beta_1, \beta_2 \in [0,1)$: Taux de décroissance exponentielle pour les estimations de moments

Require: $f(\theta)$: Fonction objectif stochastique avec les paramètres θ

Require: θ_0 : Vecteur de paramètres initial

Ensure: θ_t : Paramètres résultants

$m_0 \leftarrow 0$ ▷ Initialiser le vecteur du 1er moment

$v_0 \leftarrow 0$ ▷ Initialiser le vecteur du 2ème moment

$t \leftarrow 0$ ▷ Initialiser le compteur de temps

while θ_t n'a pas convergé **do**

$t \leftarrow t+1$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ ▷ Obtenir les gradients par rapport à l'objectif stochastique au temps t

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$ ▷ Mettre à jour l'estimation biaisée du premier moment

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2$ ▷ Mettre à jour l'estimation biaisée du second moment brut

$\hat{m}_t \leftarrow \dfrac{m_t}{1-\beta_1^t}$ ▷ Calculer l'estimation corrigée du biais du premier moment

$\hat{v}_t \leftarrow \dfrac{v_t}{1-\beta_2^t}$ ▷ Calculer l'estimation corrigée du biais du second moment brut

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$ ▷ Mettre à jour les paramètres

end while

return θ_t ▷ Paramètres résultants

Conclusion

Dans ce chapitre, nous avons exploré en détail les différents concepts fondamentaux des réseaux de neurones artificiels, qui constituent le cœur du deep learning. Pour comprendre le principe de fonctionnement des réseaux de neurones, nous avons brièvement rappelé l'approche biologique, notamment la structure fonctionnelle d'un neurone biologique avec ses différents composants. Cela nous a permis de faire la corrélation entre le neurone biologique et le neurone formel (artificiel) qui constitue l'élément de base des différentes architectures de réseaux de neurones artificiels. Nous avons également présenté les différentes formes de configuration des architectures de réseaux de neurones, notamment les réseaux entièrement connectés, les réseaux à connexion locale et les réseaux à connexion récurrente. De plus, nous avons abordé quelques fonctions d'activation et fonctions de coût qui sont des éléments importants dans l'environnement fonctionnel des réseaux de neurones. L'entraînement ou encore la formation des modèles de réseaux de neurones est assuré par des algorithmes d'apprentissage dont les plus populaires ont été présentés dans ce chapitre, notamment la descente de gradient et ses variantes.

Chapitre II

Perceptron Multicouche

Introduction

Dans le domaine du deep learning (DL), le perceptron multicouches (MLP) occupent une place prépondérante grâce à leur capacité à apprendre des représentations complexes à partir de données brutes [2]. Leur topologie multicouche et leurs mécanismes d'apprentissage sophistiqués en font des outils essentiels dans une multitude d'applications, allant de la prédiction financière à la classification d'images.

Au cœur des MLP se trouve le concept fondamental du perceptron simple, un modèle de neurone artificiel qui forme la base des architectures de réseaux de neurones plus complexes. Pour mieux comprendre les MLP, il est crucial de maîtriser à la fois leur structure globale et leur composant fondamental qui est le perceptron simple. Cette compréhension approfondie permet d'optimiser leur utilisation dans une multitude de domaines.

Ce chapitre se concentrera sur une exploration détaillée des MLP, en commençant par leur composant principal, le perceptron simple. Nous analyserons ensuite leur architecture, leur fonctionnement et leur mécanismes d'apprentissage. Une attention particulière sera portée à leurs domaines d'application ainsi qu'à leurs limites.

1 Perceptron simple

Introduit en 1958 par Frank Rosenblatt [23], le perceptron est le modèle le plus élémentaire parmi les différents réseaux de neurones artificiels. Son importance historique est considérable, ayant influencé et initié la recherche dans le domaine des réseaux de neurones artificiels grâce à son algorithme d'apprentissage intrinsèque et à ses propriétés de classification [36].

À l'origine, le perceptron a été conçu pour effectuer des classifications sur des données linéairement séparables. Sa structure fonctionnelle s'apparente à celle du neurone formel, avec l'ajout notable d'un biais qui définit le seuil d'activation du perceptron, comme illustré dans la figure II.1.

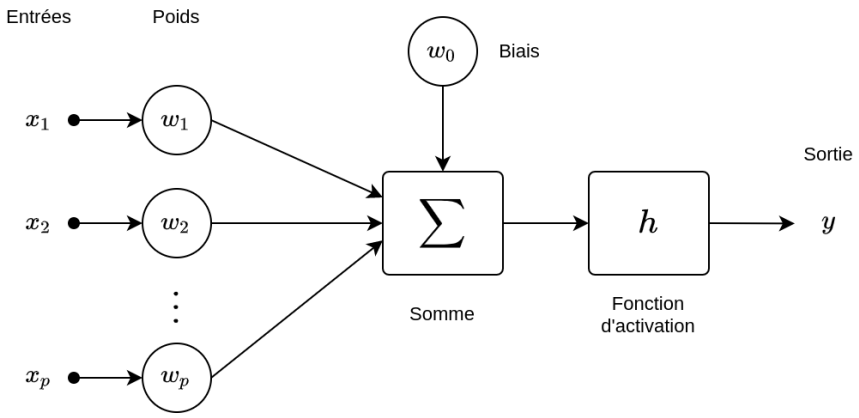

FIGURE II.1 – Structure fonctionnelle du perceptron

Mathématiquement, le perceptron peut être décrit par les équations (II.1) et (II.2) :

$$z = \sum_{i=1}^{n} w_i x_i + w_0 \tag{II.1}$$

$$y = h(z) \tag{II.2}$$

La caractéristique distinctive du perceptron réside dans son algorithme d'apprentissage. Contrairement au neurone formel statique, le perceptron possède la capacité d'ajuster ses paramètres (poids et biais) en fonction de la sortie produite par le modèle [36]. Cette capacité d'apprentissage lui permet de s'adapter aux données d'entrée et d'améliorer ses performances de classification au fil du temps.

Algorithm 4 Algorithme d'apprentissage du perceptron simple
[52]

Require: Données d'entrée $X = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
Require: Taux d'apprentissage α
Require: Nombre maximal d'époques *max_epochs*

Initialiser les poids $w \in \mathbb{R}^d$ aléatoirement
Initialiser le biais $b \in \mathbb{R}$ à 0
for $epoch = 1$ to *max_epochs* **do**
    $erreur \leftarrow 0$
    for chaque (x_i, y_i) dans X **do**
        $z \leftarrow w \cdot x_i + b$
        $\hat{y} \leftarrow \sigma(z)$ ▷ Prédiction
        if $\hat{y} \neq y_i$ **then**
            $w \leftarrow w + \alpha y_i x_i$
            $b \leftarrow b + \alpha y_i$
            $erreur \leftarrow erreur + 1$
        end if
    end for
    if $erreur = 0$ **then**
        break ▷ Convergence atteinte
    end if
end for
    return w, b

Bien qu'étant un modèle pionnier des réseaux de neurones artificiels, le perceptron simple présentait des limites majeures. Sa principale faiblesse résidait dans son incapacité à résoudre des problèmes non linéaires, échouant notamment sur des tâches simples comme la fonction logique XOR [53]. Composé d'un seul neurone sans couches cachées, il ne pouvait capturer que des relations très basiques entre les entrées et les sorties, le rendant inapte à résoudre des problèmes complexes. De plus, ses performances dépendaient fortement de l'initialisation aléatoire des poids, et sa convergence devenait difficile lorsque le nombre de caractéristiques augmentait [25]. Ces limitations significatives ont motivé le développement de modèles multicouches plus puissants et flexibles, ouvrant la voie aux architectures de réseaux de neurones profondes.

2 Architecture et fonctionnement du MLP

Un MLP est un type de réseau de neurones artificiel entièrement connecté [54] contenant une ou plusieurs couches cachées, où chaque couche possède un ou plusieurs neurones. C'est une extension du perceptron simple permettant de modéliser des relations plus complexes [55]. À la base, le MLP ne contient qu'une seule couche cachée, on parle alors de réseau de neurones superficiel. Cependant, avec un nombre suffisant de neurones dans

la couche cachée, ce modèle peut déjà fournir une approximation universelle pour la plupart des problèmes liés aux données tabulaires [56]. Un MLP est qualifié de réseau de neurones profond, lorsqu'il contient plus d'une couche cachée dans sa structure [57]. L'ajout de couches cachées supplémentaires peut accroître les performances du modèle sur certaines tâches complexes. Cependant, cela peut également augmenter le nombre de paramètres ajustables, devenant plus coûteux en termes de calcul et plus susceptible de causer du sur-apprentissage [44].
Comme le montre la figure II.2, dans une architecture MLP, les données sont transférées dans un seul sens, de couche d'entrée vers la couche de sortie en passant par la ou les couche (s) cachée (s). Chaque neurone de la couche d'entrée est connecté à tous les neurones de la couche cachée, et chaque neurone de la couche cachée est à son tour connecté à tous les neurones de la couche suivante qui peut correspondre à une autre couche cachée ou la couche de sortie [55].

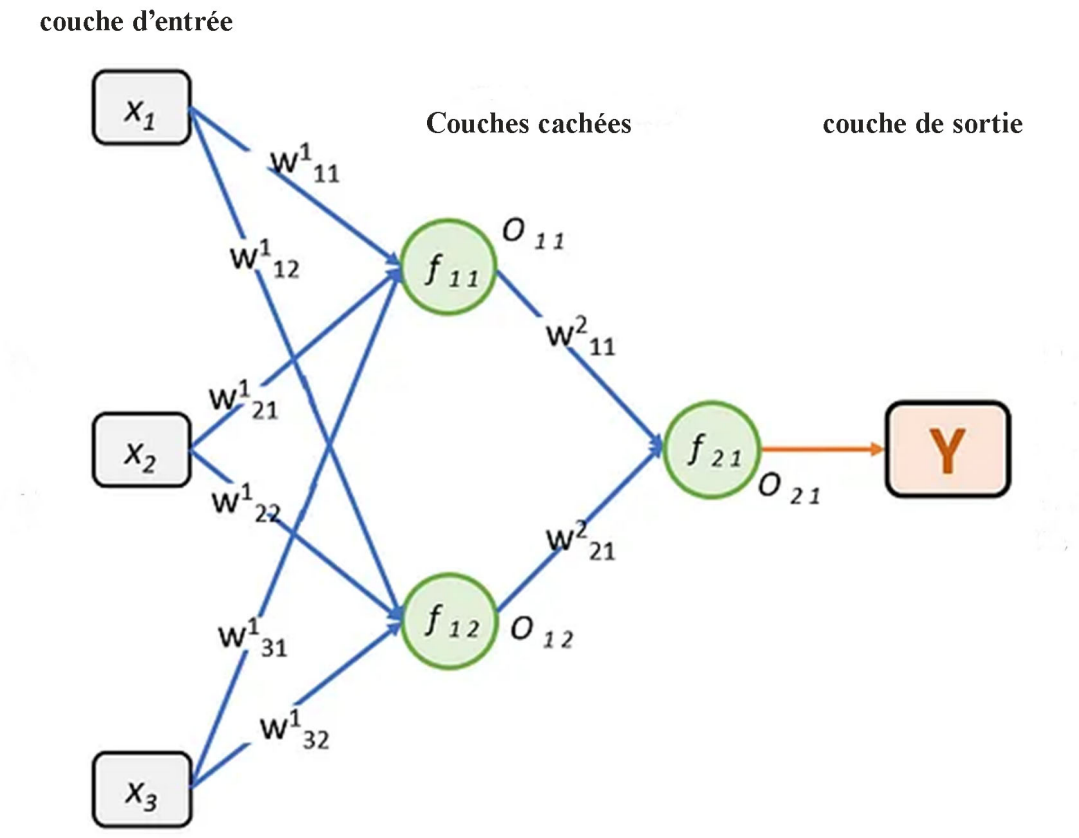

FIGURE II.2 – Perceptron Multicouche

1. **Couche d'entrée :** elle est chargée de recevoir les données d'entrées à partir d'une source externe telle qu'un fichier CSV. Elle nécessite un nœud (neurone) d'entrée par variable (ou caractéristique) [57]. Par exemple, dans le célèbre ensemble de données de classification des fleurs d'iris, il y a quatre caractéristiques (longueur des sépales, largeur des sépales, longueur des pétales et largeur des pétales), donc il y a quatre nœuds dans la couche d'entrée. Si seulement deux caractéristiques sur quatre ont été sélectionnées, la couche d'entrée comprendra alors seulement deux nœuds.
2. **Couches cachées :** Les couches cachées sont ce qui rend les réseaux de neurones artificiels supérieurs à

d'autres algorithmes d'apprentissage automatique. Ce sont des couches intermédiaires entre les couches d'entrée et de sortie. Elles détectent et apprennent des caractéristiques en effectuant des transformations non linéaires des entrées fournies au MLP [58]. Il n'y a pas de règle générale précise pour sélectionner le nombre de neurones dans une couche cachée, cela dépend fortement du problème.

3. **Couche de sortie :** elle est responsable de la production du résultat final du réseau. Le nombre de neurones de la couche de sortie dépend de la nature du problème traité (classification binaire, multi-classes, régression, etc.) [59]. Par exemple, dans un problème de classification binaire (spam/nonspam), la couche de sortie contient un seul neurone. Cependant, dans un problème de classification multiple, le nombre de neurones est généralement égal au nombre de classes (ou catégories). Par exemple, pour la classification des fleures avec avec le jeu de données Iris, la couche de sortie doit disposée trois neurones pour les trois catégories de fleures (Setosa, Versicolor et Virginica).

3 Entraînement d'un MLP par rétropropagation

L'entraînement d'un modèle MLP consiste à ajuster de manière itérative ses paramètres (poids et biais), afin de minimiser son erreur entre ses sorties prédites et les sorties désirées. Pour ce faire, il utilise l'algorithme appelé rétropropagation du gradient (ou Backpropagation) [31].

La rétropropagation ou encore **propagation vers l'arrière** consiste à rétropropager les erreurs produites par le modèle de la couche de sortie vers la couche de d'entrée en passant pas les couches cachées du réseaux. Cela permet au modèle d'ajuster les paramètres en fonction des erreurs qu'il a commises et de s'améliorer en fonction du nombre d'itérations [31]. Elle est aujourd'hui la méthode d'apprentissage la plus utilisée dans le domaine du deep learning [2].

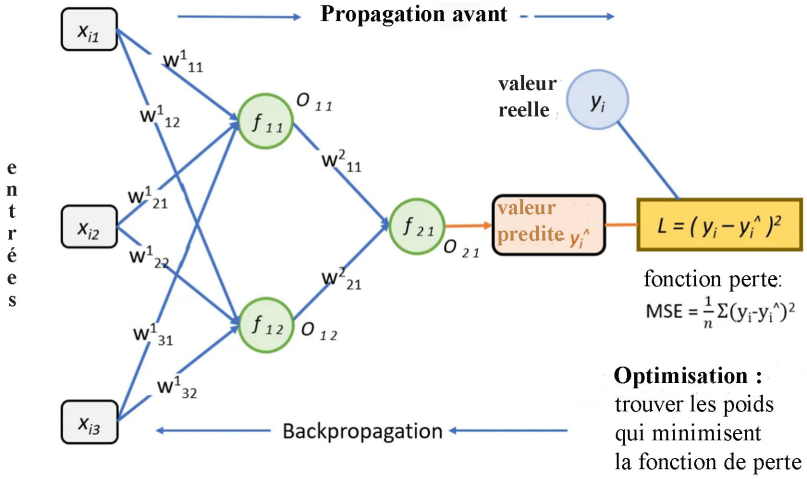

FIGURE II.3 – Apprentissage du MLP

Comme illustré à travers la figure II.3, le processus d'entraînement d'un modèle de réseaux de neurones par rétropropagation implique deux phases :

1. Passe avant (forward pass) :

   Elle consiste à propager les données d'entrée à travers le réseau, de la couche d'entrée à la couche de sortie, en appliquant une transformation linéaire suivie d'une fonction d'activation à chaque couche cachée du réseau. Initialement, les données d'entrée sont pondérées par les poids des connexions synaptiques entre les neurones de la couche d'entrée et ceux de la première couche cachée, puis une fonction d'activation non linéaire est appliquée à la sortie de chaque neurone de la couche cachée. Ce processus est répété pour chaque couche cachée du réseau, permettant ainsi au MLP de capturer des caractéristiques hiérarchiques et des relations non linéaires entre les données. La propagation avant génère finalement des prédictions à partir des données d'entrée, qui sont utilisées pour calculer l'erreur de prédiction et ajuster les poids du réseau lors de la phase de la propagation en arrière (ou backward pass) [31, 44].

   La formulation mathématique du processus à travers un réseau simple de trois couches permet également de mieux comprendre comment les données sont propagées à travers le réseau et comment les prédictions sont générées. Elle sert également de base pour la phase de rétropropagation, où ces calculs seront utilisés pour ajuster les poids et les biais du réseau.

   Nous considérons un réseau composé d'une couche d'entrée avec une seule caractéristique X_1, une couche cachée avec un seul neurone et une couche de sortie avec un seul neurone pour l'illustration mathématique de ce processus.

   - Calcul des sorties des deux couches :
     - ◇ Couche cachée :

$$Z^{[1]} = W^{[1]}x + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$

     - ◇ Couche de sortie :

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$\hat{y} = A^{[2]} = \sigma(Z^{[2]})$$

   - Évaluation de la perte :

     Nous utilisons l'entropie croisée binaire pour définir la fonction coût.

$$\mathscr{L} = -(y\log(A^{[2]}) + (1-y)\log(1-A^{[2]})) \tag{II.3}$$

2. Passe arrière (Backward pass) :

   La passe arrière (backward pass) est l'étape clé de l'algorithme de rétropropagation. Elle consiste à calculer les gradients de la fonction de coût comme définie dans l'équation II.3 par rapport aux paramètres du réseau (poids et biais), puis à mettre à jour ces paramètres.

   Cette phase se déroule comme suit :

- Calcul des gradients :

  Les dérivées partielles de la fonction coût $\mathscr{L}$ par rapport à tous les paramètres du réseau (poids et biais) sont calculées en utilisant la règle de la chaîne (règle de la dérivation partielle). Ce calcul se fait de la couche de sortie vers la couche d'entrée. Pour notre MLP à trois couches, les gradients sont calculés comme suit :

  - Pour la couche de sortie :

$$\frac{\partial \mathscr{L}}{\partial W^{[2]}} = \frac{\partial \mathscr{L}}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}}$$
$$\frac{\partial \mathscr{L}}{\partial b^{[2]}} = \frac{\partial \mathscr{L}}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}}$$

  - Pour la couche cachée :

$$\frac{\partial \mathscr{L}}{\partial W^{[1]}} = \frac{\partial \mathscr{L}}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{(2)}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}]}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}}$$
$$\frac{\partial \mathscr{L}}{\partial b^{[1]}} = \frac{\partial L}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}}$$

- Mise à jour des paramètres :

  Une fois les gradients calculés, les poids et les biais sont mis à jour pour minimiser la fonction coût.

  - Couche cachée :

$$W^{[1]} = W^{[1]} - \alpha \frac{\partial \mathscr{L}}{\partial W^{[1]}}$$
$$b^{[1]} = b^{[1]} - \alpha \frac{\partial \mathscr{L}}{\partial b^{[1]}}$$

  - Couche de sortie :

$$W^{[2]} = W^{[2]} - \alpha \frac{\partial \mathscr{L}}{\partial W^{[2]}}$$
$$b^{[2]} = b^{[2]} - \alpha \frac{\partial \mathscr{L}}{\partial b^{[2]}}$$

  où α est le taux d'apprentissage.

Cette phase de rétropropagation est basée sur la règle de la chaîne et permet au réseau d'ajuster ses paramètres de manière à minimiser l'erreur entre ses prédictions et les valeurs réelles [31, 44].

Après avoir exploré le calcul des gradients à travers le processus de la rétropropagation dans un cas simplifié (un réseau avec deux couches cachées et une couche de sortie, chacune comportant un seul neurone) nous allons maintenant généraliser cette approche avec l'algorithme de rétropropagation dans sa forme complète, applicable à des architectures MLP de taille et de complexité arbitraires.

Algorithm 5 Backpropagation

Require: taux d'apprentissage (learning rate) α

Require: Initialisation des poids W et biais b

Require: données d'entraînement $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Passe avant :

for each couche l from 1 to L **do**

$z^l = W^l a^{l-1} + b^l$

$a^l = \sigma(z^l)$

end for

Evaluer l'erreur de la sortie :

$\delta^L = \nabla_a C \odot \sigma'(z^L)$

Passe arrière :

for each couche l from L-1 to 1 **do**

$\delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(z^l)$

end for

Mise à jour poids et biais :

for each couche l from 1 to L **do**

$W^l \leftarrow W^l - \alpha \frac{\delta^l}{\partial W^l}$

$b^l \leftarrow b^l - \alpha \frac{\delta^l}{\partial b^l}$

end for

4 Domaines d'application

Le MLP est un modèle polyvalent et puissant en apprentissage profond, largement utilisé dans une multitude de domaines pour résoudre des problèmes complexes. Dans cette section, nous allons résumer les différents domaines dans lesquels le MLP a été appliqué avec succès.

1. Dans le domaine de la santé, le MLP a été utilisé pour le diagnostic médical assisté en analysant les images médicales [60, 61] et a permis d'identifier des maladies. Il a également contribué à la découverte de nouveaux médicaments [62].
2. Dans le secteur financier, des modèles MLP ont permis de faire des prévisions financières [63] en analysant des données historiques boursières, des tendances économiques, etc.
3. Dans la prévision météorologique, des modèles MLP ont permis d'effectuer des prévisions à court terme précises en analysant des données climatiques [64].

5 Limites des MLP

Bien que le perceptron multicouches (MLP) soit un modèle puissant et polyvalent en apprentissage profond, il présente également certaines limites qui peuvent affecter ses performances dans certaines situations.

1. Limite en termes de données massive et de capacité de généralisation

   L'une des principales limites des modèles MLP réside dans le traitement de grande quantité de données et à la qualité des données d'entraînement. De par sa structure définie par une topologie de réseau entièrement connecté, la performance du modèle peut être limitée par le nombre de paramètres qui explose avec l'utilisation d'un jeu de données de grande taille. De plus, en raison de sa capacité à modéliser des relations complexes, le MLP est sujet au surajustement, en particulier lorsque les données sont bruitées, incomplètes [65], dans de tels cas, le modèle peut mémoriser le bruit ou les spécificités des données d'entraînement plutôt que de généraliser correctement aux nouvelles données, ce qui entraîne une performance médiocre sur l'ensemble de test.

2. Sensibilité aux hyperparamètres

   Le MLP comporte également un certain nombre d'hyperparamètres, tels que le nombre de couches cachées, le nombre de neurones par couche, le taux d'apprentissage et d'autres paramètres liés à l'optimisation [18]. Le réglage de ces hyperparamètres peut être un processus complexe et intensif en calcul, et un mauvais choix peut entraîner un temps d'entraînement excessif, une convergence lente ou une performance médiocre du modèle.

3. Complexité du modèle et interprétabilité

   En raison de sa structure complexe, le MLP peut être considéré comme une "**boîte noire**", ce qui rend difficile l'interprétation de ses décisions et de son fonctionnement interne [18]. Cela peut poser des défis dans les domaines où la transparence du modèle et l'interprétabilité des résultats sont cruciales, tels que la santé ou le droit.

4. Difficulté à capturer les relations temporelles

   Bien que le MLP puisse traiter des données statiques avec efficacité, il peut rencontrer des difficultés lorsqu'il s'agit de modéliser des séquences temporelles ou des données séquentielles [18].

Conclusion

Ce chapitre a offert une exploration du Perceptron Multicouche (MLP), mettant en lumière leurs fondements théoriques et leur fonctionnement. Nous avons vu comment ces modèles d'apprentissage profond ont su s'imposer comme des outils puissants et polyvalents dans un large éventail de domaines, de la santé à la finance, en passant par de nombreux autres secteurs. Les modèles MLP ont démontré leur efficacité dans de nombreuses applications, offrant des performances remarquables dans la résolution de problèmes variés. Cependant, ils présentent certaines limites, notamment lorsqu'il s'agit de traiter des données massives et structurées, telles que les images RGB ou les séquences temporelles. Ces contraintes ont été le moteur du développement d'architectures plus sophistiquées, conçues pour répondre aux défis spécifiques du monde réel, notamment les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN) qui seront développées dans les chapitres suivants.

Chapitre III

Réseaux de Neurones Convolutifs

Introdcution

Les réseaux de neurones convolutifs (CNN) sont une classe spécifique de réseaux de neurones profonds particulièrement adaptés au traitement des données structurées, en particulier des données avec une structure spatiale telles que les images [66]. Les CNN s'inspirent du fonctionnement du cortex visuel biologique (notamment du fait que certains neurones de notre aire visuelle ne réagissent qu'aux bordures verticales et d'autres aux horizontales ou diagonales). Au lieu de connecter chaque neurone d'une couche à tous les neurones de la couche précédente comme avec les MLP, les neurones d'une couche de convolution sont connectés à une petite région de la couche précédente, appelée champ récepteur [67]. Cela réduit considérablement le nombre de paramètres à apprendre lors de l'entraînement [18]. Cette particularité clé permet aux CNN de mieux gérer les données à haute dimension comme les images, tout en étant plus efficaces en termes de calcul et de stockage.

Dans ce chapitre, nous aborderons en détail le principe du concept de base des CNN, des différentes couches de leur architecture ainsi que leurs paramètres et méthodes d'optimisation. Ensuite, nous ferons un résumé détaillant les domaines d'application des CNN avant d'aborder leurs limites et défis.

1 Principe de convolution

Le principe de base de la convolution consiste à combiner deux signaux pour en produire un troisième [68]. Mathématiquement, cette opération peut être exprimée par la formule présentée dans l'équation III.1. On peut imaginer ce processus comme l'application d'un filtre sur le signal d'origine. Ce filtre, également appelé noyau de convolution, est lui-même un petit signal qui se déplace le long du signal d'entrée.

$$(I * K)(i,j) = \sum_{m=0}^{F-1} \sum_{n=0}^{F-1} I(i+m, j+n) \cdot K(m,n) \tag{III.1}$$

Avec :

- I : Représente l'image d'entrée ou le signal d'entrée.
- K : Représente le noyau de convolution (ou le filtre).
- $*$: Symbole utilisé pour représenter l'opération de convolution.
- (i,j) : Coordonnées du pixel dans l'image de sortie après convolution.

- F : Taille du noyau de convolution (supposée carrée ici, $F \times F$).
- m,n : Variables d'itération pour parcourir le noyau de convolution.
- $I(i+m, j+n)$: Valeur du pixel de l'image d'entrée à la position $(i+m, j+n)$.
- $K(m,n)$: Valeur du coefficient du noyau à la position (m,n).

À chaque position, le filtre interagit avec une portion du signal d'entrée. Cette interaction se traduit par une série de multiplications et d'additions, comme le décrit la figure III.1. Le résultat de ces opérations est ensuite placé dans un nouveau signal de sortie, qu'on appelle souvent le signal convolué.

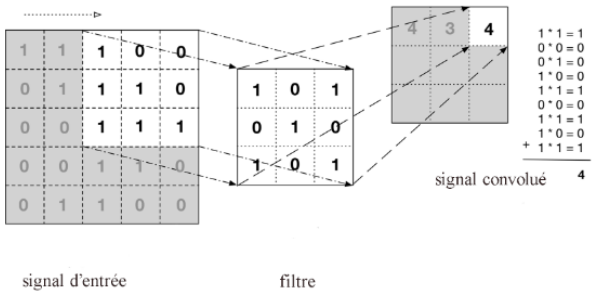

FIGURE III.1 – Opération de convolution

L'effet de cette opération dépend des caractéristiques du filtre utilisé. Certains filtres peuvent mettre en évidence des changements brusques dans le signal, d'autres peuvent atténuer les hautes fréquences pour réduire le bruit, ou encore amplifier certaines fréquences spécifiques. Dans le domaine du traitement du signal numérique, que ce soit pour des images, du son ou d'autres types de données, la convolution joue un rôle crucial. Elle permet d'extraire des caractéristiques importantes du signal, de le modifier de manière contrôlée, ou même de combiner des informations provenant de différentes sources.

2 Architectures et fonctionnement des CNN

L'architecture d'un CNN comprend plusieurs éléments essentiels, notamment les couches de convolution, les couches de pooling et les couches entièrement connectées [69]. Ces composant peuvent être divisés en deux blocs distincts comme illustré dans la figure III.2 : le premier bloc, appelé extraction de caractéristiques, est constitué des couches de convolution et de pooling, dédiées à l'extraction des caractéristiques des données [2]. Le second bloc, composé d'une ou plusieurs couches entièrement connectées, agit en association avec les caractéristiques extraites pour produire le résultat final, tel que la classification [70]. Cette structuration, qui fait la particularité des CNN, permet aux modèles CNN d'apprendre des représentations hiérarchiques des données, en commençant par des caractéristiques simples et locales pour aboutir à des représentations plus abstraites et globales.

Lors de leur formation, les CNN exploitent la structure spatiale de l'image en appliquant des opérations de convolution locale qui partagent les mêmes paramètres d'apprentissage sur toute l'image [33]. Cela permet de détecter efficacement des motifs visuels importants tels que des bords, des textures ou des formes, tout en réduisant considérablement le nombre de paramètres nécessaires.

Les premières couches des CNN extraient des caractéristiques de bas niveau, tandis que les couches suivantes

combinent ces caractéristiques pour former des représentations plus complexes et abstraites. Cette approche hiérarchique capture efficacement les relations spatiales et exploite les propriétés de stationnarité locale des données [35], ce qui est particulièrement bénéfique pour les tâches de vision par ordinateur tel que la reconnaissance d'objet.

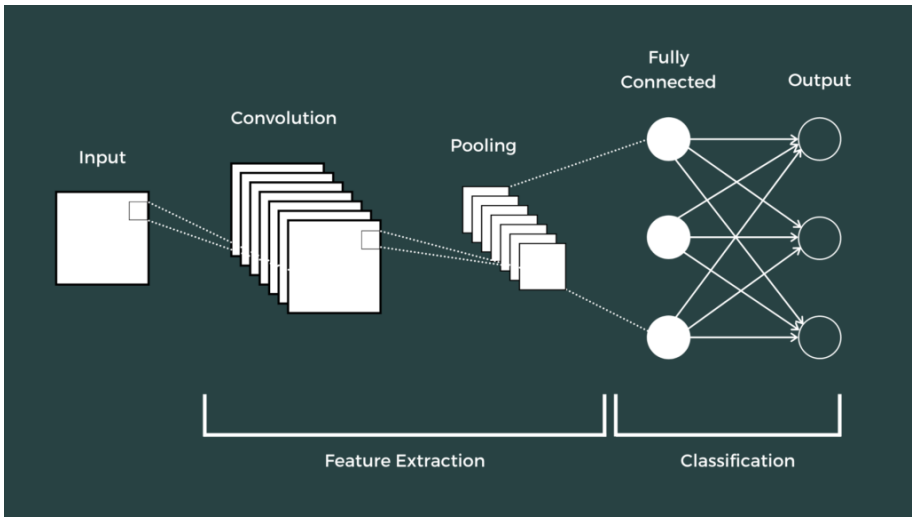

FIGURE III.2 – Architecture des CNN

2.1 Couche de convolution

La couche de convolution est l'élément principal de l'architecture des CNN. Elle est principalement composée d'un ensemble de filtres qui permettent d'extraire les caractéristiques de l'image d'entrée à travers l'opération de convolution [71], décrite dans la section 1.

2.1.1 Paramètres de la couche de convolution

Lors de l'opération de convolution d'un CNN, plusieurs paramètres entrent en jeu, chacun ayant un impact significatif sur le traitement de l'image et les caractéristiques extraites [68]. Parmi ces paramètres, nous retrouvons principalement :

1. **Le filtre (ou noyau de convolution)** : C'est une petite matrice de poids qui glisse sur l'image d'entrée. La taille du filtre est un paramètre important qui détermine la zone locale sur laquelle la convolution est appliquée.
2. **Le padding** : Ce paramètre joue un rôle important dans la préservation des dimensions spatiales et l'utilisation efficace des informations en bordure d'image. Son importance découle de deux observations principales :
   - La réduction de taille : Chaque opération de convolution tend à réduire la taille de l'image de sortie par rapport à l'entrée.
   - La sous-représentation des bords : Les pixels en bordure de l'image sont naturellement moins pris en compte que ceux du centre lors de la convolution.

   Il existe principalement deux types de padding :

- **Valid (par défaut)** : Aucun padding n'est appliqué. La convolution est effectuée uniquement là où le filtre chevauche entièrement l'image d'entrée. Cela entraîne généralement une réduction de la taille de la carte de caractéristiques par rapport à l'image d'entrée. A la sortie, la taille de la carte produite est plus petite que celle de l'image d'entrée.
- **Same** : Des pixels (généralement des zéros) sont ajoutés symétriquement autour de l'image d'entrée de manière à ce que la sortie ait la même dimension spatiale que l'entrée. Cette approche présente deux avantages majeurs :
  - Préservation des dimensions : La taille de la carte de caractéristiques reste identique à celle de l'image d'entrée.
  - Meilleure prise en compte des bords : Les informations en bordure de l'image sont mieux représentées dans le processus de convolution.

Le choix du mode de padding dépend des objectifs spécifiques du modèle et peut significativement influencer la performance du réseau, particulièrement pour les tâches nécessitant une analyse précise des détails en bordure d'image ou une préservation exacte des dimensions spatiales à travers les couches du réseau.

3. **Le stride** : Il s'agit du pas de déplacement du filtre sur l'image. Un stride de 1 signifie que le filtre se déplace d'un pixel à la fois, tandis qu'un stride plus grand permet de réduire la dimension spatiale de la sortie. Le stride offre un contrôle sur la réduction de la taille de l'image et peut aider à capturer des caractéristiques à différentes échelles.

2.2 Couche de Pooling

Pooling est une méthode permettant de prendre une large image et d'en réduire la taille tout en préservant les informations les plus importantes qu'elle contient [71]. Les mathématiques derrière la notion de pooling ne sont une nouvelle fois pas très complexe. En effet, il suffit de faire glisser une petite fenêtre pas à pas sur toutes les parties de l'image et de récupérer soit la valeur maximum soit la moyenne de cette fenêtre à chaque pas [68]. Elle introduit alors une invariance de translation aux petites déformations et par la même occasion, elle réduit de façon considérable le nombre de paramètres à apprendre par la modèle [72]. Ainsi, le modèle préserve les meilleurs caractéristiques de cette fenêtre. Cela signifie qu'il ne se préoccupe pas vraiment d'où a été extraite la caractéristique dans la fenêtre. Il est important de souligner qu'aucun paramètre n'est appris dans les couches de pooling, contrairement aux opérations de convolution.
Il existe différent processus de pooling et les plus couramment utilisées sont le **Maxpolling** et **Averagepolling** comme illustré dans la figure III.3.

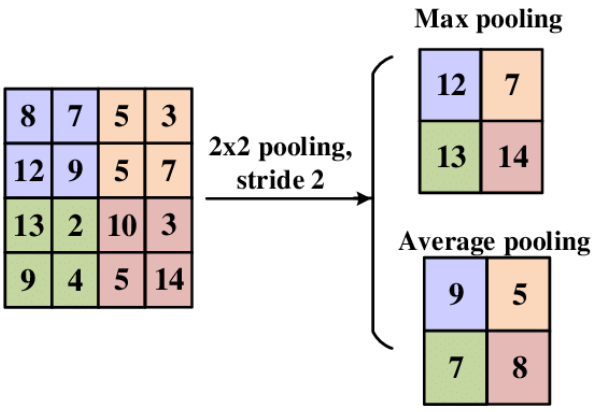

FIGURE III.3 – Présentation MaxPolling et AveragePolling

1. MaxPolling :

   Le MaxPolling consiste à appliquer un noyau (généralement de taille 2x2 ou 3x3) sans valeur, qui se déplace sur la carte en extrayant la valeur maximale pour chaque région spatiale qu'il couvre. Cette valeur maximale devient alors la sortie pour cette région dans la nouvelle carte sous-échantillonnée. En ne conservant que les activations les plus fortes localement, le max-pooling capture les meilleurs caractéristiques tout en diminuant la quantité d'informations à traiter. Cela réduit le risque de sur-apprentissage, au prix d'une perte contrôlée de détails.

2. AveragePolling :

   Contrairement au max-pooling qui sélectionne la valeur maximale, l'average-pooling calcule la moyenne des valeurs dans la région spatiale couverte par le noyau de pooling (typiquement de taille 2x2 ou 3x3). Cette valeur moyenne devient alors la sortie pour cette région dans la nouvelle carte de caractéristiques sous-échantillonnée. En réalisant une moyenne plutôt que de prendre le maximum, l'average-pooling lisse légèrement les représentations et capte une information plus globale sur la région, au détriment d'une moindre sélectivité des activations saillantes. Tout comme le max-pooling, cette opération permet de réduire les dimensions spatiales, diminuant ainsi la quantité de paramètres et introduisant une certaine robustesse aux petites translations, mais de manière généralement moins prononcée.

2.3 Couches entièrement connectées

Les couches entièrement connectées forment un sous-réseau de type MLP qui agit comme un classificateur final. Ce sous-réseau prend en entrée un vecteur créé en aplatissant les cartes de caractéristiques issues des couches de convolution et de pooling précédentes [71]. Chaque neurone dans ces couches est connecté à tous les neurones de la couche précédente, d'où le terme "entièrement connecté" comme le montre la figure III.4. Elles permettent de traiter les caractéristiques extraites par les couches de convolution pour effectuer des transformations non linéaires des informations spatiales et produire la sortie finale. Ainsi le réseau peut passer de la détection de caractéristiques locales à une décision globale sur l'image entière. La sortie finale du réseau représente généralement les probabilités d'appartenance à chaque classe pour une tâche de classification.

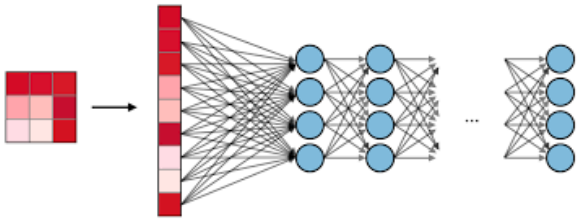

FIGURE III.4 – Couche entièrement connectée

3 Entraînement des CNN

Comme pour les MLP, l'entraînement des réseaux de neurones convolutifs repose principalement sur la rétro-propagation du gradient et l'optimisation par descente de gradient stochastique [33]. Cependant, la présence de couches de convolutions ajoute quelques spécificités supplémentaires. Lors de la propagation avant, les données d'entrée subissent des convolutions avec les noyaux pour capturer les motifs locaux [44]. Pendant la rétropropagation, le gradient de la fonction de perte par rapport aux sorties de la convolution est calculé. Ce gradient sert ensuite de signal d'erreur pour calculer le gradient par rapport aux entrées via une "corrélation croisée" [73], équivalent inverse de la convolution.

Nous allons expliquer ce processus pour une seule couche de convolution, en prenant un cas simple.

L'opération de convolution suivante prend une entrée X de taille (3×3) en utilisant un seul filtre W de taille (2×2) sans aucun remplissage et avec **stride = 1**, générant une sortie H de taille (2×2). La figure III.5 donne une illustration des différentes représentations matricielles de l'image d'entrée, du filtre et de la carte de sortie.

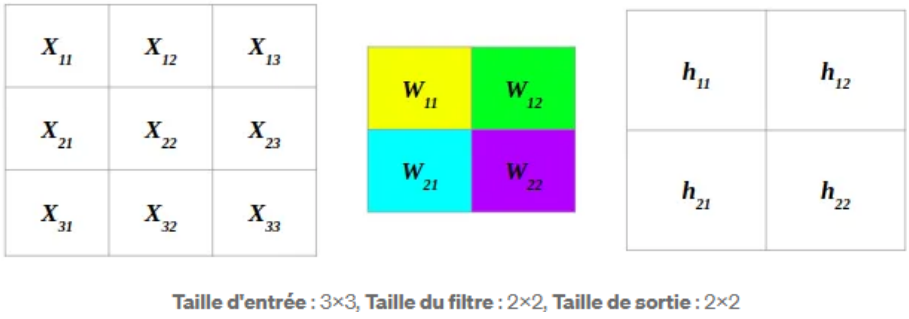

FIGURE III.5 – Représentation matricielle de l'entrée, du filtre et de la carte de sortie

L'écriture mathématique de la propagation des données d'entrée est donnée par les expressions suivantes :

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{13}X_{13}$$

$$h_{12} = W_{21}X_{11} + W_{22}X_{12} + W_{23}X_{13}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{13}X_{23}$$

$$h_{22} = W_{21}X_{21} + W_{22}X_{22} + W_{23}X_{23}$$

Pour implémenter l'étape de la rétropropagation pour la couche actuelle, nous pouvons supposer que le réseau reçoit en entrée ∂h (provenant de la passe en arrière de la couche suivante) et l'objectif sera de calculer ∂w et ∂x.

Il est important de comprendre que ∂x (ou ∂h pour la couche précédente) devient l'entrée de la couche précédente pour la retropropagation.

Chaque poids du filtre influence chaque pixel de la carte de sortie. Par conséquent, toute modification d'un poids du filtre aura un impact sur l'ensemble des pixels de sortie. Ces modifications s'additionnent pour contribuer à la perte finale. Ainsi, nous pouvons calculer facilement les dérivées de la manière suivante.

Pour calculer la dérivée partielle pour tout élément de W_{ij} on utilise la formule :

$$\frac{\partial L}{\partial w_{ij}} = \sum_{ij} \left(\frac{\partial L}{\partial h_{ij}} \times \frac{\partial h_{ij}}{\partial w_{ij}} \right) \tag{III.2}$$

Nous allons poser :

$$\partial h_{ij} => \frac{\partial L}{\partial h_{ij}}$$
$$\partial w_{ij} => \frac{\partial L}{\partial w_{ij}}$$

En utilisant la règle de chaîne pour calculer les différents gradients partiel, nous obtenons à la fin :

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$
$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$
$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$
$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

En forme matricielle cela nous donne :

$$\begin{bmatrix} \partial W_{11} & \partial W_{11} \\ \partial W_{12} & \partial W_{12} \end{bmatrix} = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \otimes \begin{bmatrix} \partial h_{11} & \partial h_{11} \\ \partial h_{12} & \partial h_{12} \end{bmatrix}$$

Donc on a :

$$\partial W = conv\left(X, \frac{\partial L}{\partial H}\right) \tag{III.3}$$

De la même façon, nous pouvons aussi trouver ∂X :

$$\begin{bmatrix} \partial x_{11} & \partial x_{12} & \partial x_{13} \\ \partial x_{21} & \partial x_{22} & \partial x_{23} \\ \partial x_{31} & \partial x_{32} & \partial x_{33} \end{bmatrix} = \begin{bmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{bmatrix} \otimes \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & \partial h_{11} & \partial h_{11} & 0 \\ 0 & \partial h_{12} & \partial h_{12} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\partial X = conv\left(180^\circ rot(W), \frac{\partial L}{\partial H}\right) \tag{III.4}$$

4 Architectures populaires des CNN

Au cours des 10 dernières années, plusieurs architectures CNN ont été présentées [74, 75]. La structure d'un modèle est un facteur essentiel pour améliorer ses performances. Diverses modifications ont été réalisées dans l'architecture CNN de 1989 à aujourd'hui. Ces modifications incluent la reformulation structurelle, la régularisation, l'optimisation des paramètres, etc. À l'inverse, il convient de noter que l'amélioration clé des performances de CNN s'est produite en grande partie grâce à la réorganisation des unités de traitement, ainsi qu'au développement de nouveaux blocs. En particulier, les développements les plus novateurs dans les architectures CNN ont porté sur l'utilisation de la profondeur du réseau.

On peut voir cette progression à travers la figure III.6, montrant le nombre de couches et la précision des meilleurs modèles CNN développés lors du concours **ImageNet**. Il s'agit du concours **ILSVRC** (ImageNet Large Scale Visual Recognition Challenge), dans lequel des modèles sont entraînés pour détecter et classifier correctement des objets et des scènes.

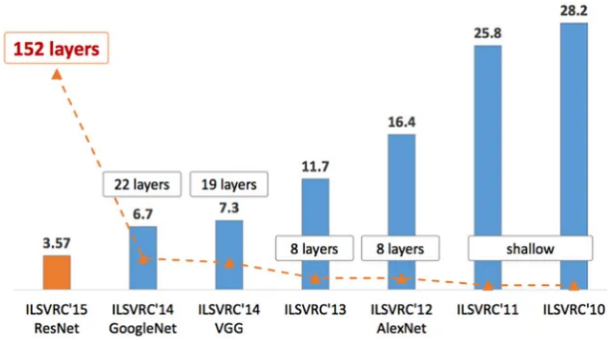

FIGURE III.6 – Progression du nombre de couche et de la précision des modèles CNN lors des compétitions ImageNet

4.1 LeNet-5

LeNet-5, première modèle CNN proposé par Yann LeCun et ses collègues dans [33], pour la reconnaissance des caractères manuscrits. Il est composé de deux ensembles de couches de convolutions et de pooling suivis de deux couches entièrement connectées et une couche de sortie.

4.2 AlexNet

AlexNet a été présenté dans [70]. Il a remporté le défi **ILSVRC** avec un taux d'erreur de 15,3%. Le réseau avait une architecture très similaire à celle de LeNet-5, mais était plus profond, avec plus de filtres par couche [70]. Il a été formé pendant 6 jours simultanément sur deux GPU Nvidia GeForce GTX 580, raison pour laquelle leur réseau est scindé en deux pipelines.

Il est composé de huit couches, les cinq premières étant des couches de convolution, certaines d'entre elles suivies par des couches à pool maximal (MaxPooling), et les trois dernières étant des couches entièrement connectées (FC).

4.3 ZFNet

ZFNet a été introduit dans [76]. Il est une amélioration de l'AlexNet par l'ajustement des hyperparamètres de l'architecture, en particulier en augmentant la taille des couches convolutives intermédiaires et en réduisant la taille de la foulée et du filtre de la première couche. Il a été le champion d'**ILSVRC** en 2013 [77].

4.4 VGGNets

VGGNet a été présenté dans [78]. Il est le finaliste du défi ILSVRC 2014 avec un taux d'erreur de 7,3%. **VGGNet** a été formé sur 4 GPU pendant 2 à 3 semaines. Il est couramment utilisé en tant qu'extracteur de fonctionnalités de base [77]. Bien qu'il soit très puissant, **VGGNet** comprend 138 millions de paramètres, ce qui peut être un peu difficile à gérer.
Sa structure, comprend 13 couches convolutives, 5 couches de Pooling, et 3 couches entièrement connectées et toutes les couches cachées sont équipées de la fonction ReLU.

4.5 GoogleNet/Inception

GoogleNet (ou Inception V1) a été proposé par des chercheurs de Google (avec la collaboration de plusieurs universités) en 2014 dans [79]. Il est le gagnant d'ILSVRC 2014 avec un taux d'erreur 5,67% beaucoup plus petit que ceux produit par les meilleurs modèles des années précédentes. Sa structure est composée de vingt deux couches et cinq millions de paramètres [77].

4.6 ResNet

Aussi appelé réseau résiduel, **ResNet** est proposé par [80] en 2015 et devient la même année le champion de **ILVSRC** avec un taux d'erreur de **3,57%** [80]. Sa structure introduit des **«connexions de saut»** également appelé **«gated units»** ou encore **«gated recurrent units»** et présente ainsi une forte similitude aux **Réseaux de neurones récurrents**.

5 Domaines d'application des CNN

Depuis leur introduction vers les années 80 [81], les CNN ont connu un immense succès, dépassant les performances des méthodes traditionnelles de machine learning et du perceptron multicouche dans de nombreuses applications pratiques. Leur capacité à apprendre des représentations riches à partir de données brutes a contribué à des avancées majeures dans ces domaines. Cette caractéristique les rend extrêmement polyvalents, conduisant à leur utilisation répandue dans de nombreux domaines d'application. Explorons maintenant quelques-uns de ces domaines.

1. Vision par ordinateur

   La vision par ordinateur est un domaine de recherche très vaste qui couvre une grande variété d'approches, non seulement pour traiter les images, mais aussi pour comprendre leur contenu [82]. Il s'agit d'un domaine de recherche actif pour les applications des réseaux neuronaux convolutifs. Les applications les plus courantes sont la classification, la segmentation , la détection et la compréhension de la scène [83–85].

2. Le traitement d'image médicale

   Dans le domaine médical, les CNN sont largement utilisés pour l'analyse d'images médicales, notamment la détection de tumeurs, la segmentation d'organes, le diagnostic de maladies et la prédiction de résultats cliniques [86, 87]. Leur capacité à extraire des caractéristiques discriminantes à partir d'images médicales contribue à améliorer la précision et l'efficacité des diagnostics médicaux. [88] a souligné l'importance des CNN dans l'analyse d'images médicales, mettant en évidence leur capacité à détecter efficacement les anomalies et à aider les cliniciens dans leur prise de décision. Ces travaux de recherche démontrent l'impact positif des CNN sur l'amélioration des soins de santé grâce à une analyse précise et automatisée des images médicales.

3. Traitement automatique du langage

   Bien que les CNN soient principalement utilisés pour le traitement d'images, leur application au traitement automatique du langage a également été couronnée de succès. Comme l'a souligné [89], les architectures CNN spécialement conçues pour cette tâche, comme les "**Convolutional Neural Networks for Text**", ont démontré leur efficacité dans diverses applications, notamment la classification et la génération automatique de texte, la traduction automatique et bien d'autres. Ces travaux ont ouvert de nouvelles perspectives dans le domaine du traitement automatique du langage naturel en exploitant les capacités des CNN pour extraire des caractéristiques pertinentes à partir de données textuelles [90]. De plus, les CNN ont été appliqués avec succès à la reconnaissance automatique de la parole [91]. Les travaux de [92] et de [93] font partis des premiers à appliquer les CNN à la reconnaissance automatique de la parole.

6 Limites des CNN

Malgré leurs performances remarquables, les CNN font encore face à d'importantes limitations qui freinent leur applicabilité à grande échelle. Leur formation reste complexe, nécessitant d'immenses ressources computationnelles et de vastes jeux de données annotées manuellement [94–96]. Leur manque d'interprétabilité pose également des défis cruciaux en termes de confiance et de certification pour des applications critiques[97]. Par ailleurs, ces architectures restent limitées dans leur capacité à modéliser certaines invariances complexes ou à traiter des données séquentielles[98, 99].

1. Complexité de la formation

   Les CNN, en particulier les architectures modernes très profondes, peuvent être extrêmement complexes à entraîner de manière efficace et stable. Le grand nombre de paramètres à optimiser (pouvant atteindre des dizaines de millions), combiné à la non-linéarité de la fonction de perte, rend l'optimisation difficile et sujette aux minima locaux [94]. Des techniques avancées comme l'initialisation appropriée des poids [80], l'utilisation de schémas d'optimisation adaptés[100] et la normalisation des couches [101] sont souvent requises pour assurer la convergence.

2. Besoin en ressource informatique

   L'entraînement des grands CNN modernes nécessite d'immenses ressources computationnelles. Les accélérateurs GPU haut de gamme sont indispensables pour des temps d'entraînement raisonnables, ainsi que des capacités de stockage et de bande passante mémoire conséquentes pour les jeux de données volumineux

[95]. Le déploiement en inférence de ces mêmes CNN sur des systèmes embarqués aux ressources limitées reste également un défi d'optimisation majeur[102].

3. Besoin en données annotées

   Comme mentionné précédemment, les CNN requièrent typiquement d'énormes quantités de données annotées manuellement pour apprendre efficacement, en particulier pour des tâches complexes comme la détection d'objets[96]. Bien que des techniques existent pour réduire ces besoins (transfert d'apprentissage, données synthétiques), elles restent des solutions partielles [103].

4. Manque d'interprétabilité

   À l'instar de nombreux modèles d'apprentissage profond, les CNN souffrent d'un manque d'interprétabilité, rendant difficile la compréhension des motifs et caractéristiques sur lesquels ils s'appuient pour effectuer leurs prédictions [97].

Conclusion

L'exploration des réseaux de neurones convolutifs (CNN) dans ce chapitre nous a permis de développer une compréhension globale de leur architecture, de leur fonctionnement et de leurs divers domaines d'application. Les CNN sont particulièrement adaptés au traitement de données structurées de manière spatiale, comme les images et les vidéos, mais ils peuvent également être appliqués au traitement du langage naturel. Leur capacité à capturer les relations spatiales locales dans les données en fait des outils puissants pour la vision par ordinateur, la reconnaissance de la parole et d'autres tâches liées au traitement des images. Cependant, la complexité de leurs architectures et les exigences en matière de données peuvent influencer la performance des modèles.

Bien que les CNN aient été utilisés avec un certain succès sur des données séquentielles, ils présentent des limites en termes de performance et d'efficacité dans ce contexte. C'est pourquoi les réseaux de neurones récurrents (Recurrent Neural Network (RNN)) ont été introduits avec une configuration architecturale plus adaptée au traitement de données séquentielles et temporelles. Dans le chapitre suivant, nous aborderont en détails les RNN à travers leur architecture et mécanisme de fonctionnement, leurs applications et limites.

Chapitre IV

Réseaux de neurones récurrents

Introduction

Dans les chapitres précédents, nous avons exploré les architectures de deep learning DL tels que le perceptrons multicouche MLP et les réseaux de neurones convolutifs (CNN). Ces modèles se sont révélés particulièrement efficaces pour traiter des données d'entrée et de sortie de taille fixe. Cependant, de nombreux problèmes réels impliquent des données de nature dynamique telles que des données séquentielles ou des séries temporelles, où les entrées et les sorties peuvent avoir des longueurs variables.

La structure des modèles MLP ou encore les CNN ne peut pas s'adapter efficacement à de telles variations de longueur de séquences. Ainsi, les réseaux de neurones récurrent (RNN) ont été introduits avec une architecture qui s'adapte au mieux aux données de nature dynamique, avec une capacité à mieux capturer les dépendances au sein de ces données [104]. Ils présentent un mécanisme de fonctionnement permettant de conserver les informations historiques pour prévoir les valeurs futures.

Dans ce chapitre, nous aborderons dans un premier temps la structure et le fonctionnement de base des RNN simple. Nous explorerons ensuite leurs principales variantes, notamment les LSTM et les GRU. Nous verrons également les spécificités de leur entraînement. Enfin, nous présenterons quelques-unes de leurs applications phares, avant de discuter de leurs limites actuelles et les perspectives de recherches.

Prenons l'exemple de la traduction automatique : pour traduire la phrase : "**Cette année, j'ai décidé d'apprendre la programmation pour développer mes propres applications**" en anglais, le modèle doit pouvoir traiter une séquence de 13 mots en entrée, et générer une séquence de sortie nettement plus courte (9 mots) , comme "**This year, I decided to learn programming to develop my own apps**".

1 Modélisation séquentielle des données

Selon la nature de la tache à réaliser, la modélisation de la séquence peux prendre différentes formes. Allant de celle avec une seule entrée et une seule sortie à celle avec plusieurs entrées et plusieurs sorties comme le montre la figure IV.1.

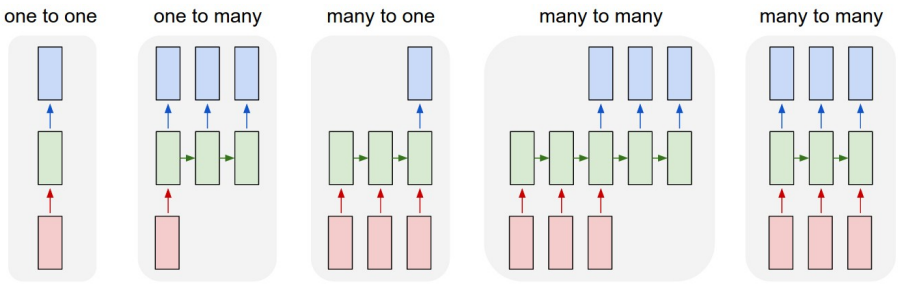

FIGURE IV.1 – Représentation des différentes configurations d'un RNN

Pour mieux comprendre, voyons quelques modélisations de taches qui sont rencontrées le plus souvent.

- **Un à un (One-to-one)** : La modélisation one-to-one est souvent utilisée pour des tâches où les données d'entrée et de sortie sont indépendantes les unes des autres et ne nécessitent pas de traitement séquentiel. Elle peut être utilisée pour faire de la classification d'images (nous avons une image de chat en entrée et la sortie est soit un chat soit un chien).
- **Un à Plusieurs (One-to-many)** : Cette configuration est appliquée dans une situation qui donne plusieurs sorties pour une entrée. Elle peut être appliquée pour générer la légende d'une image (l'entrée est une image et la sortie est une description de l'image).
- **Plusieurs à un (Many-to-one)** : la modélisation many-to-one est utilisée lorsque plusieurs entrées sont nécessaires pour une seule sortie. Le réseau traite une séquence d'entrées, élément par élément, et utilise une mémoire interne pour capturer les dépendances entre les différentes étapes de la séquence. Une fois que toute la séquence a été traitée, le réseau produit une sortie unique basée sur l'ensemble de la séquence d'entrée. Cette topologie peut être appliquée pour faire de l'analyse des sentiments.
- **Plusieurs à plusieurs (Many-to-many)** : La modélisation many-to-many présente en entrée un ensemble d'éléments d'une séquence pour produire une séquence d'éléments à la sortie. Elle peut être retrouvée dans des tâches aux quelles les correspondances entre les éléments d'entrée et de sortie sont prises en compte, notamment dans la traduction automatique ou encore à la reconnaissance vocale.

2 Architecture et fonctionnement d'un RNN

Les RNN sont une catégorie de modèle de deep learning qui possède une mémoire interne, leur permettant de capturer des dépendances séquentielles. Contrairement aux réseaux de neurones traditionnels, notamment les MLP qui traitent les entrées comme des entités indépendantes, les RNN prennent en compte l'ordre temporel des entrées, les rendant adaptés aux tâches impliquant des informations séquentielles [105]. En utilisant une boucle, les RNN appliquent la même opération à chaque élément d'une série, le calcul actuel dépendant à la fois de l'entrée courante et des calculs précédents.

Un RNN est structuré de façon à parcourir successivement les entrées d'une séquence $[X_0,X_1,...,X_t]$, en leur appliquant une fonction pour produire une séquence de sortie $[y_0,y_1,...,y_t]$, en maintenant un état interne $[h_0,h_1,...,h_t]$. Une illustration est données dans la figure IV.2.

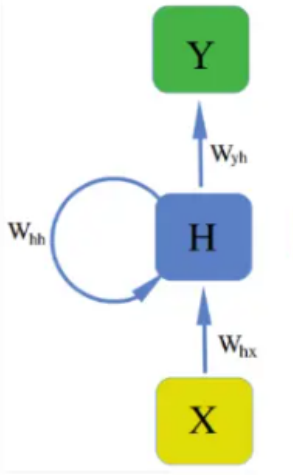

FIGURE IV.2 – Structure d'un RNN

Cette représentation simple peut être déroulée dans le temps sur toute la longueur de la séquence d'entraînement comme illustré dans la figure IV.3.

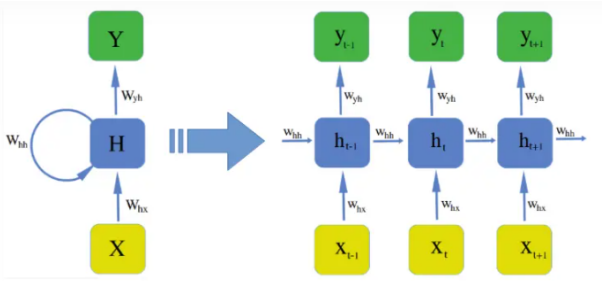

FIGURE IV.3 – RNN déroulé dans le temps

A chaque pas de temps h_t, l'état caché h_t est mis à jour en fonction de l'entrée x_t et de l'état précédent h_{t-1}, capturant ainsi l'information du contexte précédent. La sortie y_t est obtenu en appliquant une fonction d'activation à la sortie de l'état de la cellule h_t à l'instant t.
Ce fonctionnement est régit par les équations mathématiques suivantes :

$$h_t = tanh(w_{xh}X_t + w_{hh}h_{t-1}) \tag{IV.1}$$

$$y_t = f(w_{hy}h_t) \tag{IV.2}$$

Où :

- w_{xh} poids reliant l'entrée X à l'état caché h
- w_{hh} poids des états cachés
- w_{hy} poids de la couche de sortie y
- f fonction d'activation non linéaire

2.1 Entraînement d'un RNN

Comme mentionné précédemment, la structure récurrente des RNN leur permet de capturer des dépendances dans les données séquentielles [106]. Cependant, la présence de connexion récurrente dans la configuration du réseau augmente la complexité lors de la phase d'entraînement par rapport aux réseaux de neurones classiques. En effet, pour entraîner un RNN, il ne serait possible d'appliquer directement l'algorithme de rétropropagation du gradient standard. À chaque pas de temps t, la mise à jour des paramètres dépend non seulement de l'entrée X_t courante, mais également de l'état caché h_{t-1} hérité du passé via les connexions récurrentes. Pour faire face à cette spécificité, une version adaptée de la rétropropagation prenant en compte l'aspect temporel est utilisée : la rétropropagation à travers le temps ou (**Backpropagation Through Time** (BPTT)).

2.2 Retropropagation à travers le temps

La rétropropagation à travers le temps est un algorithme d'apprentissage standard pour entraîner les réseaux de neurones récurrents sur des données séquentielles [107]. Il s'agit d'une extension de la rétropropagation classique, prenant en compte la nature récurrente et partageant les mêmes paramètres. L'idée consiste donc à **"dérouler"** le RNN dans le temps sur toute la longueur de la séquence d'entraînement, ce qui permet d'obtenir une architecture de réseau de neurones profond à l'image du perceptron multicouche avec des connexions partagées entre chaque couche [108]. Le BPTT consiste alors à rétropropager l'erreur à travers le réseau déplié en appliquant l'algorithme de la rétropropagation pour calculer les gradients en remontant dans le temps, comme le montre la figure IV.4.

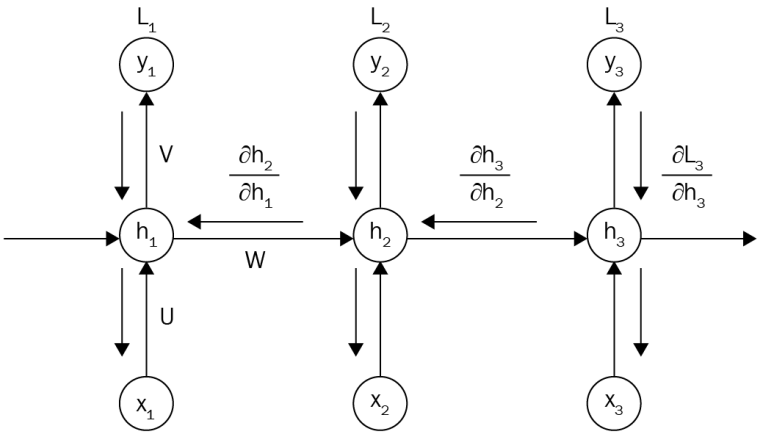

FIGURE IV.4 – Retropropagation à travers le temps : BPTT

• **Calcul des gradients** :

Cela revient à calculer les dérivées partielles de la fonction coût sur chaque pas de temps par rapport aux différents paramètres [108]. Comme le processus reste le même pour chaque paramètre, nous allons juste considérer un paramètre pour expliquer le calcul des gradient.

L'erreur globale du modèle est obtenue par la sommation des erreurs produites sur chaque pas de temps. Ainsi, le gradient $\frac{\partial L}{\partial U}$ est obtenu en additionnant le gradient sur chaque pas de temps comme le montre l'équation IV.3.

$$\frac{\partial L}{\partial U} = \frac{\partial L_1}{\partial U} + \frac{\partial L_2}{\partial U} + \frac{\partial L_3}{\partial U} \tag{IV.3}$$

Nous allons commencer le calcul du gradient de la l'erreur du dernier pas de temps, c'est-à-dire L_3 en utilisant la règle de la chaîne.

$$\frac{\partial L_3}{\partial U} = \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_3} \cdot \frac{\partial h_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U}$$
$$\frac{\partial L_2}{\partial U} = \frac{\partial L_2}{\partial y_2} \cdot \frac{\partial y_2}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_1} \cdot \frac{\partial h_1}{\partial U}$$
$$\frac{\partial L_1}{\partial U} = \frac{\partial L_1}{\partial y_1} \cdot \frac{\partial y_1}{\partial h_1} \cdot \frac{\partial h_1}{\partial U}$$

Le résultat final est donc :

$$\frac{\partial L}{\partial U} = \sum_{i=1}^{t} \frac{\partial L_t}{\partial y_{t-i}} \cdot \frac{\partial y_{t-i}}{\partial h_{t-1}} \left(\prod_{j} \frac{\partial h_{t-j+1}}{\partial h_{t-j}} \right) \frac{\partial h_{t-i-1}}{\partial U} \tag{IV.4}$$

2.3 Problèmes lies à l'entraînement d'un RNN

Le problème fondamental de l'apprentissage d'un modèle RNN classique se situe dans les dépendances à long terme [109]. La nature même de leur architecture fait qu'ils ont tendance à être biaisés vers les informations les plus récentes de la séquence, au détriment des éléments plus anciens. Cela est principalement dû au fait que, lors de la propagation avant dans le réseau, l'état caché ne peut transmettre qu'une quantité limitée d'informations contextuelles des étapes précédentes. En conséquence, plus la séquence est longue, plus il devient difficile pour un RNN standard de relier les événements actuels aux informations pertinentes apparues bien avant dans la séquence. Ce phénomène restreint considérablement les performances du modèle sur de nombreuses tâches où les dépendances à très long terme sont cruciales, comme la traduction automatique et l'analyse de textes longs. Ce phénomène est souvent matérialisé par le problème de l'explosion du gradient (ou exploding gradient problem) [110] et le problème de la disparition du gradient (ou vanishing gradient problem) [111]. Pour pallier ce problème, de nouvelles architectures dérivées des RNN ont été proposées, notamment les mémoires à long et court terme (LSTM) [112] et les GRU [113].

2.3.1 Explosion du gradient (Exploding gradient)

Le phénomène de l'explosion du gradient se produit lorsque les gradients s'accumulent de manière exponentielle pendant la rétropropagation à travers le temps, conduisant ainsi à des valeurs de gradient extrêmement élevées [18]. L'explosion des gradients se réfère au fait que les gradients à long terme peuvent croître exponentiellement en magnitude lorsqu'on les propage en arrière à travers le temps [114]. Cette croissance exponentielle peut entraîner une instabilité numérique, rendant l'apprentissage inefficace ou impossible.

2.3.2 Disparition du gradient (Vanishing gradient)

Contrairement au problème de l'explosion du gradient, le phénomène de la disparition du gradient se produit lorsque les gradients deviennent considérablement petits lors de la rétropropagation à travers le temps [111, 114], rendant ainsi l'apprentissage du modèle RNN difficile sur de longues séquences d'éléments.

3 Mémoire à long et court terme (LSTM)

Les LSTM sont une architecture de réseaux de neurones récurrents, considérée comme une évolution des RNN classiques [18]. Ils ont été introduits dans [34] dans le but de résoudre le problème des dépendances à long terme dont souffraient les RNN traditionnels.

De par leur structure, les LSTM présentent des capacités remarquables à capturer et à maintenir des dépendances à long terme dans des séquences de données étendues. Cette caractéristique leur permet de traiter efficacement des séquences temporelles complexes où l'information contextuelle peut s'étendre sur de longues périodes [115]. Ils parviennent ainsi à atténuer le problème de la disparition du gradient (vanishing gradient) [111, 116] en utilisant un système de portes (gates) pour contrôler le flux d'information.

3.1 Structure d'une cellule LSTM

La structure d'un LSTM peut être représentée par une cellule, appelée **Cellule LSTM**, composée de trois portes qui sont des zones de calcul permettant de traiter le flot d'information (en réalisant des opérations de calcul spécifiques) [34]. Elle présente également deux types de sorties appelées **états** (states).

- Forget gate : porte d'oubli
- Input gate : porte d'entrée
- Output gate : porte de sorti
- Hidden state : état caché
- Cell state : état de la cellule

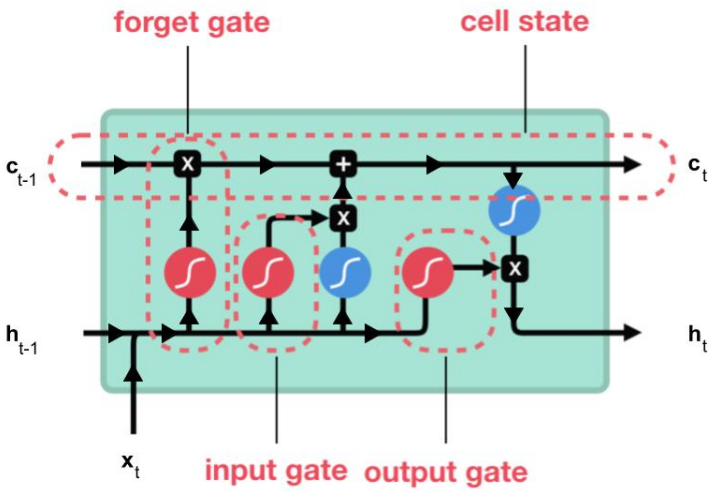

FIGURE IV.5 – Représentation d'une cellule LSTM

3.2 Fonctionnement d'une cellule LSTM

La **porte d'oubli** (équation IV.5) permet de supprimer l'information qui n'est plus utile. Pour ce faire, elle reçoit deux entrées, l'état caché précédent h_{t-1} et l'entrée actuelle x_t, multipliées par des matrices de poids, suivies de l'ajout d'un biais. Le résultat est passé à travers une fonction d'activation qui donne une sortie binaire. Si, pour un état de cellule particulier, la sortie est 0, l'information est oubliée et pour une sortie de 1, l'information est conservée pour une utilisation future.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{IV.5}$$

La **porte d'entrée** offre à la cellule la capacité d'intégrer une nouvelle information à l'instant t, même si celle-ci était inexistante ou peu pertinente à l'instant $t-1$ [117]. Pour ce faire, on applique parallèlement une fonction

sigmoïde (équation IV.6) et une fonction TanH au résultat de la concaténation de l'état précédent et de l'entrée courante (équation IV.7).

La fonction 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. Inversement, si elle est proche de 1, elle est jugée importante (c'est-à-dire utile pour la prédiction du modèle LSTM). La fonction TanH va simplement normaliser les valeurs entre -1 et 1 pour éviter les problèmes de saturation de l'ordinateur. Le produit des deux sorties permettra donc de ne garder que les informations jugées importantes et les autres étant quasiment remplacées par 0.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{IV.6}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{IV.7}$$

L'**état de la cellule** (équation IV.8) est déterminé à partir de la porte d'oubli et de la porte d'entrée en multipliant d'abord, élément par élément, la sortie de la porte d'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 par coordonnée) avec la sortie de la porte d'entrée, ce qui permet d'enregistrer dans l'état de la cellule les données que le LSTM a jugées pertinentes.

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{IV.8}$$

La **porte de sortie** (équation IV.10) contrôle l'information qui sera transmise au temps $t+1$ en fonction de l'état de la mémoire **C** et des fonctions d'activation [118].

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{IV.9}$$

$$h_t = o_t \odot \tanh(C_t) \tag{IV.10}$$

Où :

- σ est la fonction d'activation sigmoïde
- tanh est la fonction d'activation tangente hyperbolique
- W_f, W_i, W_C, W_o sont les matrices de poids pour chaque porte
- b_f, b_i, b_C, b_o sont les biais pour chaque porte
- h_{t-1} est l'état caché précédent
- x_t est l'entrée au temps t
- $\odot$ représente le produit de Hadamard (multiplication élément par élément)
- o_t est le vecteur des valeurs de la porte de sortie pour l'étape de temps actuelle.
- h_t est l'état de la mémoire actuelle.

Compte tenu de la complexité de la structure d'une cellule LSTM, qui s'explique par la présence de trois portes distinctes (oubli, entrée et sortie) avec leurs matrices de poids respectives, il est logique que leur phase d'apprentissage requiert plus de temps que celle d'un réseau de neurones conventionnel ou d'un RNN classique [118]. Cependant, cette complexité accrue se traduit par des performances nettement supérieures [119].

Dans les faits, la majorité des résultats remarquables obtenus aujourd'hui par les réseaux récurrents sont attribuables aux LSTM [117]. Cette prédominance s'explique principalement par leur capacité exceptionnelle à gérer les dépendances à long terme dans les séquences de données [34]. Cette aptitude les rend particulièrement efficaces dans des domaines tels que le traitement du langage naturel, la reconnaissance vocale et l'analyse de séries temporelles complexes [120].

4 Unité récurrente à portes (GRU)

GRU est une autre variante de l'architecture RNN qui aborde également le problème de dépendance à long terme ou encore le problème de mémoire à court terme [113], offrant une structure plus simple par rapport au LSTM [121]. Il est conçu en combinant la porte d'entrée et la porte d'oubli de la cellule LSTM en une seule porte appelée porte de mise à jour (update gate), ce qui donne lieu à une structure plus rationalisée [122]. Contrairement au LSTM, GRU ne considère pas un état de cellule séparé.

4.1 Structure d'une cellule GRU

Une cellule GRU est principalement composée de trois éléments principaux : la porte de mise à jour (update gate), une porte de réinitialisation (reset gate) et le contenu de l'état actuel de la mémoire [113]. Ces portes permettent à la cellule GRU de mettre à jour et d'utiliser de manière sélective les informations précédentes de la séquence, facilitant ainsi la capture des dépendances à long terme [123]. La figure IV.6 montre une illustration de la structure d'une cellule LSTM.

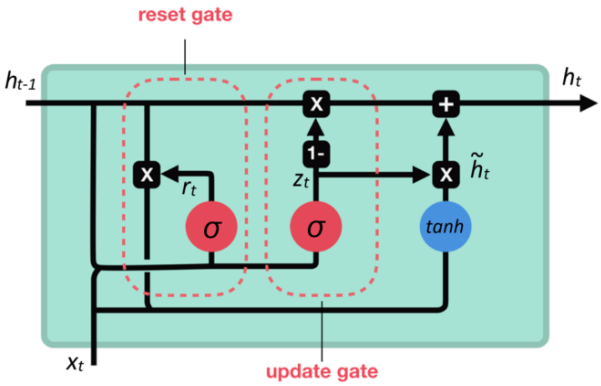

FIGURE IV.6 – Représentation d'une cellule LSTM

4.2 Fonctionnement de la cellule LSTM

Comme nous le constatons avec l'illustration dans la figure IV.6, le fonctionnement d'une cellule LSTM repose sur ses deux portes en plus de l'état actuel de la mémoire.

La **porte de mise à jour** (équation IV.11) détermine quelle quantité d'informations passées doit être conservée et combinée avec l'entrée actuelle à un pas de temps spécifique. Elle est calculée en se basant sur la concaténation de l'état caché précédent h_{t-1} et de l'entrée actuelle x_t, suivie d'une transformation linéaire et d'une fonction

d'activation sigmoïde.

$$z_t = \sigma(W_z[h_{t-1}, x_t] + b_z) \tag{IV.11}$$

La **porte de réinitialisation** (équation IV.12) décide quelle quantité d'informations passées doit être oubliée. Elle est calculée de manière similaire à la porte de mise à jour, en utilisant la concaténation de l'état caché précédent h_{t-1} et de l'entrée actuelle x_t.

$$r_t = \sigma(W_r[h_{t-1}, x_t] + b_r) \tag{IV.12}$$

Le **contenu de la mémoire actuelle** (équation IV.13) est calculé en se basant sur la porte de réinitialisation et la concaténation de l'état caché précédent transformé et de l'entrée actuelle. Le résultat est passé à travers une fonction d'activation tangente hyperbolique pour produire l'activation candidate.

$$\tilde{h}_t = \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) \tag{IV.13}$$

L'**état de la mémoire final** h_t est obtenu par une combinaison de l'état caché précédent h_{t-1} et de l'activation candidate (équation IV.14). La porte de mise à jour détermine l'équilibre entre l'état caché précédent et l'activation candidate. De plus, une porte de sortie o_t peut être introduite pour contrôler le flux d'informations du contenu de mémoire actuel vers la sortie (équation IV.15). La porte de sortie est calculée en utilisant l'état de la mémoire actuel h_t et est typiquement suivie d'une fonction d'activation, telle que la fonction sigmoïde.

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{IV.14}$$

$$o_t = \sigma(W_o h_t + b_o) \tag{IV.15}$$

Avec :

- o_t est la porte de sortie au temps t
- σ est la fonction d'activation sigmoïde
- W_o est la matrice de poids pour la porte de sortie
- h_t est l'état de mémoire actuel
- b_o est le biais pour la porte de sortie

La principale différence entre LSTM et LSTM réside dans la manière dont ils gèrent l'état des cellules mémoires. Dans LSTM, l'état de la cellule mémoire est conservé séparément de l'état caché et est mis à jour à l'aide de ses trois portes [34]. Dans LSTM, l'état de la cellule mémoire est remplacé par un vecteur d'activation candidat, qui est mis à jour à l'aide de ses deux portes [124]. Ainsi, LSTM offre une alternative plus simple au LSTM avec moins d'opérations, permettant un entraînement plus rapide. Cependant, le choix entre LSTM et LSTM dépend du cas d'utilisation spécifique et du problème en question. Les deux architectures ont leurs avantages et inconvénients, et leurs performances peuvent varier selon la nature de la tâche [121].

5 Domaines d'application des RNN

En raison de leur capacité à modéliser des données séquentielles ou encore des séries temporelles, les réseaux de neurones récurrents (RNN) ont démontré leur polyvalence et leur efficacité dans un large éventail d'applications.

1. Traitement du langage naturel (NLP) :

   Les modèles RNN excellent dans la compréhension du contexte linguistique, permettant de prédire avec précision les mots suivants dans une séquence [125]. En s'appuyant sur leur capacité de modélisation du langage, les modèles RNN peuvent générer du texte cohérent et faciliter la saisie en proposant des compléments pertinents [125]. Ils sont capables de capturer efficacement les nuances émotionnelles dans les textes, rendant possible une analyse fine des sentiments dans divers contextes, des critiques de films aux tweets [126, 127].

2. Finance :

   Les LSTM ont montré une grande efficacité dans la prévision des mouvements de prix des actions et des taux de change, en capturant les tendances temporelles complexes des données financières [128]. Leur capacité à traiter des séries temporelles les rend particulièrement utiles pour l'évaluation et la gestion des risques financiers.

3. Prévision météorologique :

   Les RNN ont considérablement amélioré la précision des prévisions météorologiques en intégrant efficacement les données historiques et les modèles climatiques complexes [129].

4. Musique :

   Les RNN peuvent générer des séquences musicales originales, ouvrant de nouvelles possibilités pour la création artistique assistée par ordinateur [130]. Aussi, en analysant les signaux audio, les RNN facilitent la conversion de la musique en partitions, améliorant ainsi les processus de notation musicale [131].

5. Santé :

   En fonction des informations qu'ils contiennent, les dossiers médicaux électroniques peuvent être analysé avec des modèles RNN pour aider à la prédiction et au diagnostic précoce de diverses conditions médicales [132]. En traitant les séries temporelles de données physiologiques, les RNN contribuent également à une surveillance plus précise et personnalisée des patients, notamment dans les unités de soins intensifs [132].

Cette diversité d'applications démontre la polyvalence et l'efficacité des RNN dans le traitement de données séquentielles et temporelles à travers de nombreux domaines, de la linguistique à la santé, en passant par la finance et les sciences.

6 Limites des RNN

Malgré leur efficacité dans de nombreux domaines, les RNN font face à plusieurs limites :

1. Problème du gradient qui disparaît ou explose :

   Sur de longues séquences, le gradient peut devenir très petit (disparition du gradient), rendant l'apprentissage inefficace pour les dépendances à long terme [111]. À l'inverse, le gradient peut devenir excessivement grand (explosion du gradient), déstabilisant l'apprentissage [133]. Ces problèmes limitent considérablement la capacité des RNN standard à apprendre des dépendances à long terme dans les séquences, affectant leur performance dans de nombreuses applications pratiques.

2. Difficulté à capturer les dépendances à long terme :

   Les RNN standard ont du mal à maintenir des informations sur de longues périodes [34]. Cette limitation de la mémoire affecte leur performance dans des tâches nécessitant la compréhension de contextes étendus, comme la génération de mot sur de très longs phrases ou encore l'analyse de séries temporelles complexes.

3. Coût computationnel élevé :

   Le traitement pas à pas des RNN peut être lent, surtout pour de longues séquences [134]. De plus, la nature séquentielle des RNN rend difficile leur parallélisation efficace sur des GPU. Ce coût computationnel élevé limite l'utilisation des RNN dans des applications nécessitant un traitement en temps réel ou impliquant de très grands ensembles de données.

4. Surapprentissage (Overfitting) :

   Les RNN, en particulier les variantes comme LSTM, ont de nombreux paramètres, augmentant le risque de surapprentissage [135]. Des techniques de régularisation spécifiques sont nécessaires pour prévenir ce problème dans les RNN. Le surapprentissage peut conduire à une performance médiocre sur des données non vues, limitant la généralisation du modèle.

5. Difficulté d'interprétation :

   Comme d'autres réseaux neuronaux profonds, les RNN sont souvent considérés comme des "boîtes noires", rendant difficile l'interprétation de leurs décisions [114]. Cet enjeu d'explicabilité pose des défis dans des domaines nécessitant des décisions transparentes, comme la médecine ou la finance, où la compréhension du processus décisionnel est cruciale.

Conclusion

Dans ce chapitre, nous avons exploré en détail les différents concepts et principes de fonctionnement des réseaux de neurones récurrents, en nous concentrant sur les RNN simples et leurs variantes plus avancées, à savoir les LSTM et les GRU. Cette étude met en lumière l'apport considérable des réseaux de neurones récurrents dans l'essor du deep learning.

Les RNN se distinguent par leur capacité remarquable à modéliser des données de nature séquentielle ou des séries chronologiques. Cette caractéristique leur a permis de révolutionner de nombreux domaines d'application, notamment le traitement du langage naturel, la traduction automatique et bien d'autres.

Malgré leurs nombreux avantages et leurs applications réussies, les RNN présentent certains défis. Ils sont notamment exigeants en termes de ressources informatiques et sensibles au choix des hyperparamètres, ce qui peut compliquer leur mise en œuvre et leur optimisation.

Chapitre V

Étude comparative des architectures de deep learning

Introduction

Les chapitres précédents ont présenté en détail les architectures fondamentales du deep learning : les perceptrons multicouches (MLP), les réseaux de neurones convolutionnels (CNN) et les réseaux de neurones récurrents (RNN). Bien que partageant des concepts de base, ces modèles diffèrent par leur structure et leur fonctionnement, ce qui influence leur efficacité selon les types de tâches. Le choix de l'architecture la plus appropriée pour une tâche spécifique demeure un défi majeur en deep learning. C'est pourquoi de nombreuses recherches se sont concentrées sur la comparaison de ces modèles [136–143].

Notre travail consistera, dans un premier temps à évaluer de manière approfondie les travaux comparatifs existants et leurs résultats dans le domaine du deep learning et en second lieu, présenter notre propre étude comparative, articulée autour de deux axes principaux : une comparaison entre les MLP et les CNN, ainsi qu'une étude comparative des CNN et des différentes architectures RNN, incluant les RNN simples, les LSTM et les GRU.
Pour la comparaison MLP-CNN, nous utiliserons le jeu de données CIFAR, une référence reconnue pour la classification d'images. Pour l'évaluation comparative CNN-RNN, nous nous concentrerons sur deux tâches distinctes : l'analyse de sentiment, avec le jeu de données IMDB (Internet Movie Database), un vaste corpus de critiques de films, et la génération automatique de poésie, pour laquelle nous constituerons un dataset bruts à partir d'un recueil de poèmes de Victor Hugo.

1 Revue des travaux comparatifs existants

Dans le cadre de notre étude, nous avons examiné un éventail de travaux de recherche pertinents qui ont contribué à l'évaluation comparative des architectures MLP, CNN et RNN. Ces travaux couvrent plusieurs domaines d'applications, offrant ainsi un aperçu précieux des performances relatives de ces différentes architectures de réseaux neuronaux dans divers contextes.

1.1 Présentation des travaux connexes

Une étude comparative des modèles d'apprentissage automatique (MLP, CNN, RNN) a été menée pour la détection automatique du cancer du sein [137]. Le jeu de données Wisconsin Breast Cancer Database (WBCD) a été utilisé. Il s'agit d'un ensemble de données médicales contenant des caractéristiques numériques extraites à partir d'images numériques de ponction à l'aiguille fine de masse mammaires. Ils ont utilisé 683 enregistrements soient 239 cas malins et 444 cas bénins avec chacun 9 attributs telles que la texture, le périmètre, l'aire, le rayon, la compacité, la concavité, la symétrie, la dimension fractale et le lissage [144].
Les modèles ont été implémentés avec MATLAB et la bibliothèque Neural Network ToolBox, utilisant l'algorithme Levenberg-Marquardt pour l'entraînement [145]. L'évaluation a été réalisée sur les mêmes jeux de données d'entraînement et de test pour tous les modèles. Les performances ont été mesurées en termes d'exactitude (accuracy), de spécificité et de sensibilité (rappel). Les résultats comparatifs sont présentés dans le tableau V.1, offrant une base pour évaluer l'efficacité relative de chaque architecture dans la classification des tumeurs mammaires.

Modèles	**Accuracy (%)**	**Spécificité (%)**	**Sensibilité (%)**
MLP	91,92	92,34	91,19
CNN	97,46	97,81	96,86
RNN	**98,61**	**98,91**	**98,11**

TABLE V.1 – Présentation des résultats WBCD

L'analyse des résultats présentés dans le tableau V.1 révèle des performances notables pour les modèles RNN et CNN, avec un léger avantage pour le RNN qui affiche les meilleurs résultats. En revanche, le MLP montre des performances moins satisfaisantes comparées aux deux autres architectures.
L'auteur attribue ces différences de performance à plusieurs facteurs potentiels. Premièrement, l'utilisation de l'algorithme d'apprentissage de Levenberg-Marquardt qui pourrait avoir un impact différent sur chaque architecture. Deuxièmement, l'estimation des paramètres du réseau qui peut varier en efficacité selon le type de modèle. Enfin, la nature dispersée et mélangée des données peut être mieux gérée par certaines architectures que d'autres. Ces éléments soulignent la complexité de l'optimisation des modèles de deep learning et l'importance de considérer l'interaction entre l'algorithme d'apprentissage, l'architecture du réseau et la nature des données dans l'analyse des performances.

Les auteurs de [138] ont proposé une étude comparative des modèles MLP et CNN pour la prédiction financière. Leur expérience s'est appuyée sur deux jeux de données : le German Credit Dataset et l'Australian Credit Dataset [146, 147]. Ce sont des données provenant respectivement des clients d'une banque Allemande des clients d'une banque Australienne incluant différentes caractéristiques telles que l'âge, le genre, le statut professionnel, l'état civil, la durée du crédit, le montant du crédit, l'historique des paiements, le types de logement et le nombre de dépendant. Pour l'entraînement, ils ont utilisé l'algorithme quasi-Newton de Broyden-Fletcher-Goldfarb-Shannon (BFGS) pour les MLP, et la descente de gradient stochastique du momentum pour les CNN [148, 149]. L'évaluation des performances a été réalisée à l'aide de trois métriques : le Taux de Précision Global (TPG), le Taux de Faux Alarme (TFA) qui permet de mesurer la proportion de faux positifs parmi tous les exemples négatifs et le Taux de Manque d'Alarme (TMA) qui est la proportion de faux négatifs parmi tous les exemples positifs. Les auteurs ont

testé diverses configurations pour chaque modèle, et les meilleurs résultats obtenus sont présentés dans le tableau V.2.

Modèles	TPG(%)	TFA(%)	TMA(%)	Dataset
MLP	81,20	13,98	32,81	GCD
CNN	**90,85**	33,00	2,90	
MLP	90,75	8,91	9,72	ACD
CNN	**99,74**	0,63	0	

TABLE V.2 – Présentation des résultats GCD et ACD

L'analyse des résultats expérimentaux présentés dans le tableau V.2 révèle une performance supérieure de l'architecture CNN par rapport au modèle MLP. Les CNN offrent des résultats nettement plus satisfaisants sur l'ensemble des métriques évaluées.
L'auteur conclut que ces résultats confirment clairement l'efficacité de l'approche utilisée. La différence significative de performance entre les deux architectures met en évidence l'avantage substantiel des CNN sur les MLP dans ce contexte de prédiction financière.

L'étude, menée par les auteurs de [150], se concentre sur l'amélioration des performances des systèmes de reconnaissance optique des caractères (Optical Character Recognition (OCR)) en comparant les modèles MLP et CNN.
Pour le MLP, ils ont utilisé un descripteur de caractères unifié (Unified Character Descriptor (UCD)), tandis que pour le CNN, ils ont adapté le réseau LeNet-5 [33] pour supporter 62 classes de caractères. Les expériences ont été réalisées sur un PC avec un processeur Intel Core i5-7200U et un GPU NVIDIA GeForce 940Mx. Le système DIGITS a été utilisé pour la création et l'entraînement des modèles. Ils ont utilisé le jeu de données **chars74k20** qui comprend des caractères alphabétiques et numériques dont certains sont écrits à la main, d'autres imprimés ou capturés dans des environnement réels [151]. Ils ont utilisé 34658 pour les données d'entraînement, 15748 pour les données de test et 12586 pour les données de validation. L'évaluation des performances a été basée sur l'exactitude (accuracy) des modèles sur les données d'entraînement et de test. Les résultats de cette comparaison sont présentés dans un tableau V.3.

Modèles	Accuracy test(%)	Accuracy train (%)
MLP	43,4	70,72
CNN(Lenet-5)	**85,53**	**86,23**

TABLE V.3 – Présentation des résultats chars74K20

Les résultats présentés dans le tableau V.3 démontrent clairement la supériorité de l'architecture CNN par rapport au modèle MLP dans le contexte de la reconnaissance optique de caractères (OCR).
Malgré l'utilisation de l'extracteur de caractéristiques UCD (Unified Character Descriptor) pour le MLP, qui fournit des informations détaillées sur chaque caractère défini dans le standard Unicode, le CNN a tout de même surpassé ses performances. Cette différence significative de performance met en lumière un avantage clé des mo-

dèles CNN : leur capacité à intégrer directement l'extraction des caractéristiques au sein de leur structure.
L'auteur souligne que cette supériorité du CNN peut être attribuée à sa capacité intrinsèque à apprendre et à extraire automatiquement les caractéristiques pertinentes directement à partir des données brutes. Cette aptitude élimine le besoin d'un extracteur de caractéristiques prédéfini, comme l'UCD utilisé pour le MLP, et permet au réseau d'adapter son apprentissage de manière plus flexible et efficace aux spécificités du jeu de données.

Les auteurs de [143] ont mené une étude comparative des modèles CNN et des variantes RNN (LSTM et GRU) pour la détection et la classification des logiciels malveillants. Pour leurs expériences, ils ont utilisé le jeu de données BIG2015 proposé par Microsoft lors d'un challenge de détection de logiciel mal veillant [152]. Il contient des fichiers exécutables Windows dont chacun est étiqueté avec une classe de logiciel malveillant spécifique parmi 9 classes basées sur le type de menace qu'il représente. Ils ont utilisé 10872 exemple d'entraînement et 10867 données de test avec une partie non définie des données d'entraînement réservée à la validation. Les expériences ont été réalisées avec un GPU NVIDIA Quadro P2000, utilisant le framework Keras sur Theano en Python. Leur approche se décompose en deux parties principales :

1. Une étape d'extraction de caractéristiques utilisant différentes techniques d'embedding (Word2vec, FastText et GloVe).
2. L'optimisation des modèles en variant les tailles de batch et les fonctions d'activation (Sigmoïde, ReLU, TanH).

Les performances des modèles ont été évaluées en utilisant la précision (accuracy) sur les données de validation. Les résultats sont présentés dans le tableau V.4.

Modèles	**Validation Accuracy(%)**
CNN	96
LSTM	**98,54**
GRU	95

TABLE V.4 – Présentation des résultats Big2015

Les résultats présentés dans le tableau V.4 révèlent des performances satisfaisantes pour tous les modèles, avec une supériorité notable de l'architecture LSTM par rapport au CNN. Selon les auteurs, cette supériorité du LSTM souligne leur efficacité particulière dans la capture des dépendances à long terme, un atout crucial pour la détection de logiciels malveillants.

Les auteurs de [153] ont mené une étude comparative des performances des modèles CNN, RNN simple, LSTM et GRU sur trois tâches distinctes avec des jeux de données différents. Pour l'analyse de sentiments sur des critiques de film, ils ont utilisé le jeu de données IMDB [154] contenant 50000 données divisées à part égale en données d'entraînement et en données de test. Pour la reconnaissance d'activités humaines dans un environnement intelligent, ils ont utilisé le jeu de données ARAS [155](Activity Recognition with Ambient Sensing) avec 846000 pour les données d'entraînement et 338400 pour les données de test. Pour la tache de classification de fruits, ils ont utilisé le jeu de données Fruits-360 [156] contenant 20 classes de fruits. Sur 12983 images, ils ont utilisé 20% pour les données de test.

Les performances ont été évaluées à l'aide de plusieurs métriques : précision globale (accuracy), précision (precision), rappel (recall) et score F1.
Le tableau V.5 présente les différents résultats obtenus à travers cette étude.

Modèles	Accuracy(%)	Precision(%)	Recall(%)	F1-Score(%)	Time	Datasets
CNN	80,69	87,01	**86,12**	86,61	0 :01 :59	IMDB
RNN	68,97	74,72	77,14	75,96	0 :16 :56	
LSTM	85,46	85,70	84,32	85,04	1 :45 :44	
GRU	**87,04**	84,92	**88,98**	**86,97**	1 :29 :25	
CNN	89,18	93,78	88,69	91,16	**0 :02 :18**	ARAS
RNN	91,73	93,56	90,74	92,13	0 :10 :00	
LSTM	**92,27**	**93,97**	**f91,75**	**92,85**	1 :03 :27	
GRU	**92,24**	**94,00**	**91,91**	**92,94**	0 :36 :13	
CNN	**99,69**	**99,75**	**98,25**	**98,99**	**0 :06 :28**	Fruit-360
LSTM	70,49	84,08	59,46	69,60	0 :38 :32	

TABLE V.5 – Présentation des résultats IMDB, ARAS et Fruits-360

Les résultats présentés dans le tableau V.5 montrent des performances variées selon les jeux de données :

Pour l'analyse de sentiments (IMDB), les modèles GRU et CNN surpassent le LSTM en termes de précision. Le CNN se distingue par un temps d'entraînement nettement inférieur au GRU. L'auteur conclut que CNN et GRU sont des approches efficaces pour l'analyse de sentiment, chacun offrant des compromis différents entre performance et temps d'entraînement.

Sur le jeu de données ARAS (reconnaissance d'activité), les modèles récurrents LSTM et GRU démontrent une meilleure performance que le CNN. L'auteur attribue cela à la nature temporelle des données de capteurs. Il suggère que pour la reconnaissance d'activités dans des environnements intelligents, les modèles récurrents sont particulièrement adaptés, avec une préférence pour le GRU qui offre un bon équilibre entre performance et temps d'entraînement.

Pour la classification d'images (Fruits-360), seuls CNN et LSTM ont été comparés. Le CNN surpasse nettement le LSTM en performance et en temps d'entraînement. L'auteur explique cette supériorité par la capacité du CNN à capturer efficacement les caractéristiques spatiales des images grâce à ses couches de convolution. Il conclut que pour les tâches de classification d'images, l'architecture CNN est plus efficace et efficiente que les modèles récurrents comme LSTM.

1.2 Discussion

L'examen des diverses études comparatives entre MLP, CNN et RNN a révélé que les performances des modèles varient significativement selon les domaines d'application et les jeux de données utilisés.
Dans le domaine de la classification d'images, les CNN ont généralement montré une supériorité, notamment dans l'étude utilisant le dataset Fruits-360. De plus, pour des tâches comme la prédiction financière, les CNN ont surpassé les MLP traditionnellement utilisés dans ce domaine. Pour l'analyse de sentiments et le traitement de données

séquentielles, les modèles récurrents (LSTM et GRU) ont souvent excellé, comme démontré avec le dataset IMDB. Néanmoins, les CNN ont également montré des performances compétitives dans certains cas, illustrant leur polyvalence. Dans le domaine de la reconnaissance d'activités humaines, les modèles récurrents, en particulier le GRU, se sont distingués, probablement en raison de leur capacité à capturer efficacement les dépendances temporelles et de par leur architecture moins complexe par rapport aux LSTM. Ces résultats soulignent l'importance du contexte dans le choix de l'architecture. Ils suggèrent également que les performances sont influencées par divers facteurs, incluant la nature des données, la complexité de la tâche, et les spécificités de l'implémentation.

Pour approfondir et confirmer ces observations, nous proposons une approche dans laquelle nous explorons trois domaines distincts :

1. Classification d'images : Comparaison MLP vs CNN sur le dataset CIFAR-10,
2. Analyse de sentiments : Confrontation CNN vs RNN avec le dataset IMDB.
3. Génération de poèmes : Évaluation CNN vs RNN sur un corpus de poèmes de Victor Hugo,

2 Proposition d'études comparatives

À travers cette étude, nous cherchons à approfondir d'avantage l'étude comparative des différentes architectures à savoir les MLP, les CNN et les RNN en reprenant certaines expériences proposées dans les travaux connexes, notamment l'analyse de sentiment avec le jeu de données IMDB et aussi la classification d'image en utilisant un jeu de données différent. De plus, nous allons explorer d'autres tâches pour avoir une confirmation générale de l'influence du contexte et de la nature des données sur les performance des modèles.

Pour nos expériences, nous avons utilisé l'environnement **Google Colab** (Google Colaboratory). Il s'agit d'une plateforme fournie par Google permettant d'écrire et d'exécuter du code **Python** (ou des Notebooks Jupyter) via un navigateur. Avec Google Colab, nous n'avons pas à nous soucier des ressources matérielles et logicielles préinstallées sur l'ordinateur. En effet, il fournit un accès à des ressources de calcul à accès limité et également à des bibliothèques d'apprentissage automatique telles que **TensorFlow**, **Keras**, **NumPy**, **Pandas**, **Scikit-learn**, **Matplotlib**, **Seaborn** et **Plotly**. Avec l'environnement **Google Colab**, nous avons accès à **12,7 Go** de RAM et **78,2 Go** de stockage, ainsi qu'à un **GPU T4** avec **15 Go** de RAM. Cependant, il faut bien noter que ces ressource sont très limitées pour entraîner des modèles de réseaux de neurones profond sur des jeu de données de grandes tailles.

2.1 Comparaison MLP vs CNN

Pour comparer les architectures MLP et CNN, nous avons utilisé le jeu de données CIFAR-10 [157].

2.1.1 Présentation du jeux de données

Le jeu de données est composé de 60000 images en couleur de tailles 32×32 réparties en 10 classes, avec 6000 par classe. Sur les 60000 images, 50000 images constituent les données d'entraînement donnant ainsi 5000 exemples pour chaque classe et les 10000 restantes sont réservées pour le test, soit 1000 exemples par classe. Nous avons appliqué une opération de normalisation en divisant les valeurs de pixels par 255.0. Cela va nous permettre de ramener toutes les valeurs dans la plage **[0 1]**.

2.1.2 Expérimentation et résultats

Dans cette expérience, nous utilisons un modèle avec 3 couches cachées composées respectivement de 128, 64 et 64 neurones. Avec le modèle CNN, nous utilisons deux couches de convolution avec 64 et 32 filtres. Nous faisons suivre chaque couche de convolution par une couche de maxpoolling de dimension 2. Les deux modèles sont entraînés avec les même paramètres d'entraînement, 30 époques et la taille des lots (batch-size) égale 128. Nous comparons les deux modèles à travers leur comportement lors de l'entraînement en visualisant l'évolution de la précision de chaque modèle sur les données d'entraînement et de validation comme illustré respectivement avec la figure V.1 et la figure V.2.

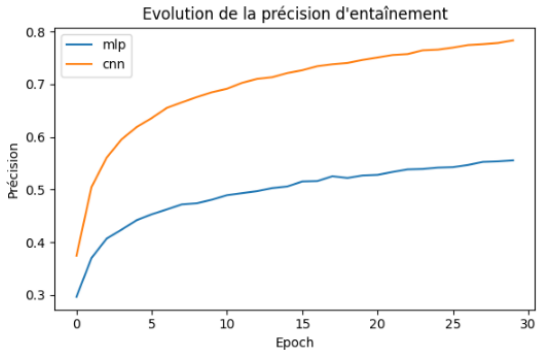

FIGURE V.1 – Précision sur les données d'entraînement

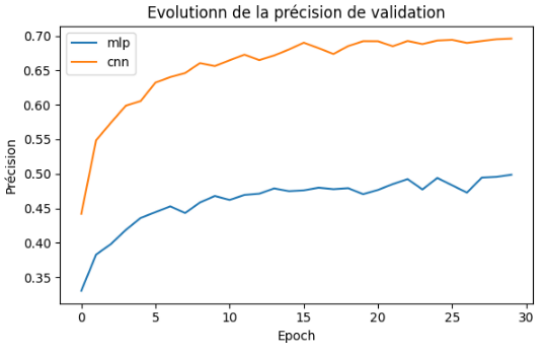

FIGURE V.2 – Précision de la validation

Les figures V.4 et V.3 présentent respectivement les courbes d'évolution des erreurs produites sur les données de validation et sur les données d'entraînement pendant le processus d'entraînement des modèles. Ces deux graphiques permettent également d'évaluer le comportement de chaque modèle en fonction de l'évolution de la perte au cours de son entraînement.

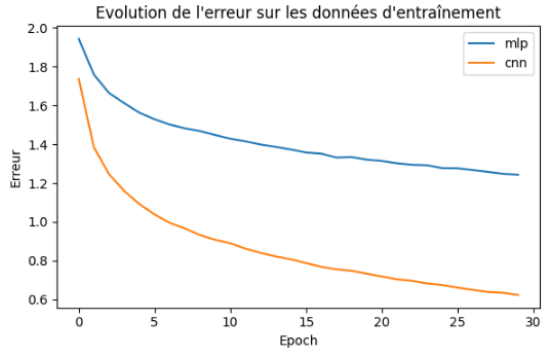

FIGURE V.3 – Perte sur les données d'entraînement

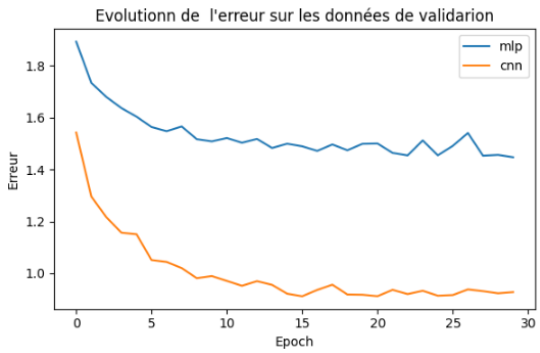

FIGURE V.4 – Perte de la validation

Pour mieux appréhender la performance de chaque modèle, nous utilisons les métriques telles que l'exactitude (accuracy), la précision(precision), le rappel (recall) et le score f1. Nous considérons également le temps de l'entraînement du modèle pour mesurer son efficience. Les résultats obtenus pour chaque modèle sont présentés dans le tableaux V.6.

Modèles	Accuracy(%)	Precision(%)	Recall(%)	F1-Score(%)	Time(s)
MLP	49,21	49,61	49,21	48,24	84,89
CNN	**67,77**	**68,77**	**67,77**	**67,88**	**84,15**

TABLE V.6 – Comparaison MLP et CNN avec CIFAR-10

Au regard des résultats obtenus par ces deux modèles, présentés dans le Tableau V.6, on peut observer une nette dominance du modèle CNN par rapport au MLP sur la classification des images CIFAR-10. Comme le témoignent plusieurs chercheurs, ces résultats pourraient venir du fait que les CNN soient spécifiquement conçus pour le traitement de données d'images, présentant une capacités à extraire des caractéristiques très pertinentes.
De plus, le nombre de paramètres entraînables pour le modèle MLP (406 410) est six fois plus important que celui du modèle CNN (59 176), en considérant les deux architectures utilisées dans cette expérience. Ce surplus de paramètres peut entraîner des redondances, ce qui peut rendre difficile l'entraînement du modèle, cela pourrait également expliquer la différence du temps d'entraînement des modèles, légèrement plus court avec le CNN. Bien que la différence de performance soit considérable, les comportements des modèles observés à travers les courbes d'évolution semblent montrer une cohérence dans l'apprentissage des modèles. Les courbes de précision et de pertes des deux modèles, illustrées dans les figures V.1 - V.4, montrent la même allure tant sur les données de validation que sur les données d'entraînement, ce qui indique que les deux modèles ne font pas de surapprentissage.

2.2 Comparaison CNN vs RNN

Cette étude porte sur la comparaison de l'architecture CNN et les réseaux récurrents, notamment RNN simple, LSTM et GRU. Elle se concentre sur deux taches distinctes, l'analyse de sentiments avec le jeu de données **IMDB** et le génération de poème en utilisant des poèmes de **Victor Hugo**.

2.2.1 Présentation des jeux de données

1. Le jeu de données **IMDB** :

   Il est constitué de critiques de film avec des étiquettes binaires correspondant à la nature de la critique selon qu'elle soit positive ou négative. Ainsi il est largement utilisé pour l'analyse de sentiment. Le dataset contient au total 50000 critiques, réparties de manière équitable en 25000 données d'entraînement et 25000 données de test. Il y a une distribution égale des étiquettes positives et négatives, avec 25000 exemples pour chaque classe.

   Nous utilisons un vocabulaire contenant les 10000 mots les plus fréquemment utilisés. Pour uniformiser la longueur des exemples d'entrée, nous fixons la taille maximale des séquences à 150. Pour l'encodage des mots du vocabulaire, nous utilisons pour chaque architecture la couche **Embedding** de **Keras**. Elle permet de représenter chaque mot par un vecteur de dimension 16 que nous avons choisi.

2. Recueil de poème de Victor Hugo :

   Nous avons utilisé un ensemble de poème de Victor Hugo pour constituer notre jeu de données. Pour nos expériences, nous avons utilisé un vocabulaire contenant 32 caractères présents dans la collection de poème

après pré-traitement. Le jeu de données est divisé en 26828 données (séquences) d'entraînement, 6708 données de validation et 8385 données de test.

2.2.2 Expérimentation et résultats sur IMDB

Pour la configuration des modèles, nous utilisons une couche de convolution avec 64 filtres de taille 5 suivi d'une couche GlobalMaxPooling et une couche dense de 32 neurones. Pour les réseaux récurrents, nous avons utilisé deux couches cachées avec respectivement 64 et 32 unités récurrentes. Les modèles sont entraînés en utilisant les même paramètres, notamment sur 10 époques et des lots de 128.
Les modèles ont été évalués selon plusieurs métriques, l'exactitude, la précision, le rappel, le score F1 et le temps d'entraînement.

Nous pouvons voir à travers les figures V.5 et V.6 l'évolution de la précision sur les données d'entraînement et sur les données de validation de chaque modèle au cours du processus de l'entraînement. A travers ces courbes d'évolution, nous pouvons avoir une idée sur la façon dont les modèles apprennent à partir des données d'entraînement et des données qu'ils n'ont jamais vu.

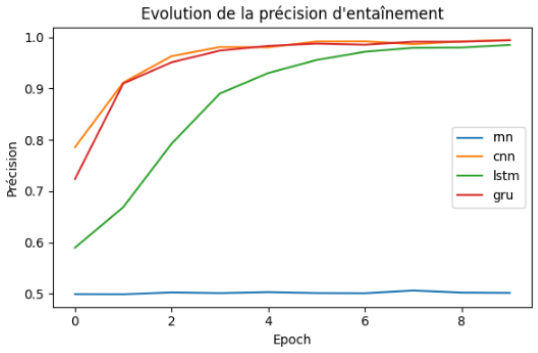

FIGURE V.5 – Précision d'entraînement imdb

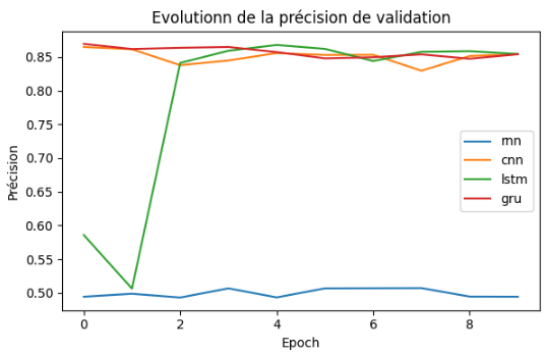

FIGURE V.6 – Précision de validation imdb

La figure V.7 présente l'évolution de la perte sur les données d'entraînement au cours du processus de formation des différents modèles.
La figure V.8 illustre le comportement des modèles à travers l'évolution de la perte sur les données de validation tout au long du processus d'entraînement des modèles.

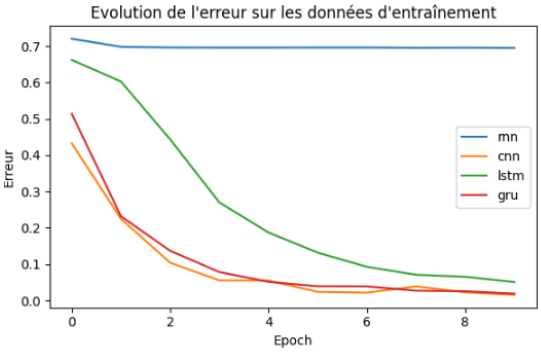

FIGURE V.7 – Perte sur l'entraînement imdb

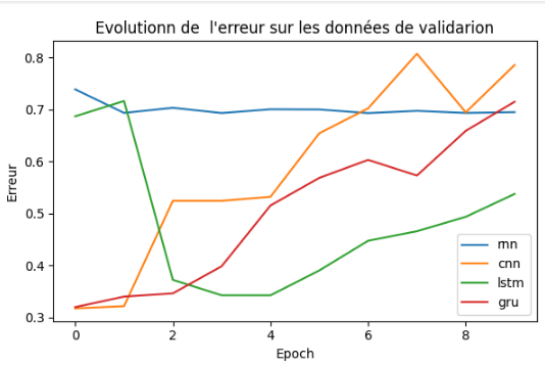

FIGURE V.8 – Perte sur la validation imdb

Les résultats des différents modèles obtenus en utilisant les métriques sont présentés dans le tableau V.7. A travers ce tableau, nous avons un aperçu sur la différence de performance des modèles en terme d'exactitude, de précision et de sensibilité.

Modèles	Accuracy(%)	Precision(%)	Recall(%)	F1-Score(%)	Time(s)
RNN	55,18	55,32	55,31	55,30	207,07
CNN	84,52	84,73	84,52	84,49	46,27
LSTM	83,45	83,45	83,45	83,45	69,42
GRU	**84,56**	**84,60**	**84,56**	**84,56**	64,90

TABLE V.7 – Comparaison CNN, RNN, LSTM et GRU sur IMDB

Sur la base des résultats fournis dans le tableau V.7, nous observons une meilleure performance des modèles CNN et GRU sur le jeu de données IMDB pour l'analyse de sentiment. Les deux modèles ont démontré une grande précision dans la classification des sentiments, avec des résultats presque identiques.
Il convient cependant de noter que le temps d'entraînement du modèle CNN est significativement inférieur à celui du modèle GRU. Autrement dit, le modèle CNN est plus rapide à entraîner tout en maintenant d'excellentes performances. Cette différence de temps pourrait s'expliquer par la structure des CNN, qui les rend hautement parallélisables. Cet avantage devient particulièrement notable lors de l'utilisation de matériel de calcul spécialisé tel que les **GPU**, comme ce fut le cas dans cette expérience.
Cependant, l'examen de l'évolution de la précision et de la perte sur les données d'entraînement et de validation, illustré dans les figures V.5 à V.8, révèle un problème de surapprentissage. Les modèles ne parviennent pas à généraliser efficacement l'apprentissage sur les données de validation. Cet écart entre les performances sur les données d'entraînement et de validation indique clairement un surapprentissage, ce qui constitue une limitation importante à prendre en compte.
Pour améliorer ces résultats et atténuer le problème de surapprentissage, on pourrait envisager plusieurs stratégies tels que le dropout, la régularisation L2, l'augmentation de la taille du jeu de données, l'arrêt précoce (early stopping) pour interrompre l'entraînement avant que le surapprentissage ne devienne trop important.
Malgré cette limitation, les résultats suggèrent que les modèles CNN et GRU sont plus adaptés que les autres architectures testées pour l'analyse de sentiment avec le jeu de données IMDB.

2.2.3 Expérimentation et résultats sur la génération de poème

Nous utilisons une architecture CNN avec une seule couche de convolution de 128 filtres de taille 3 suivi d'une MaxPoolin1D de taille de pooling égale à 2. Les architectures récurrentes ont utilisé la même configuration avec une couche cachée de 128 unités récurrentes. La sortie des différents réseaux est une couche dense de nombre de neurone égale à la taille du vocabulaire avec une activation softmax.
Les modèles sont entraînés avec les même paramètres : 20 époques et des lots de tailles 20. Nous avons évalué les différents modèles avec la précision sur les données d'entraînement et les données de validation.
Les performances des modèles ont été évaluées en utilisant l'exactitude sur les données d'entraînement et les données de validation.

Les figures V.9 et V.10 montrent l'évolution de la précision des modèles au cours du processus de formation sur les données d'entraînement et les données de validation.

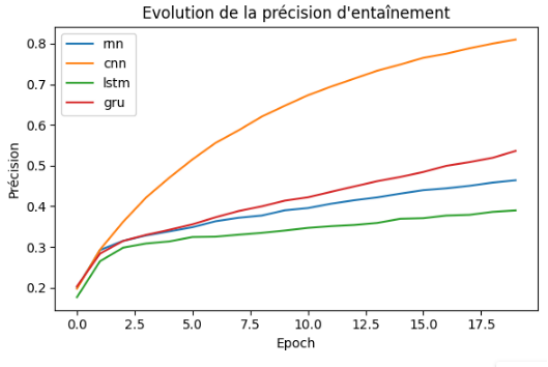

FIGURE V.9 – Précision sur les données d'entraînement

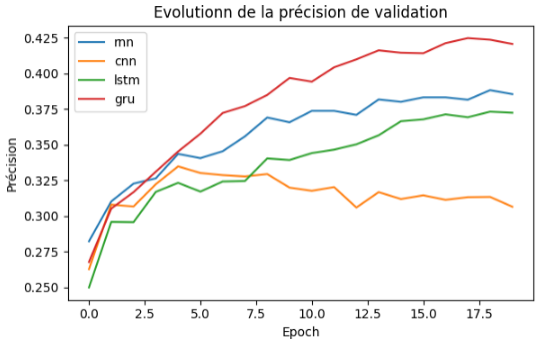

FIGURE V.10 – Précision sur la validation

Nous présentons également le comportement de la fonction de perte des différents modèles lors de la période d'entraînement des modèles à travers les graphiques visualisés dans les figures V.11 et V.12.

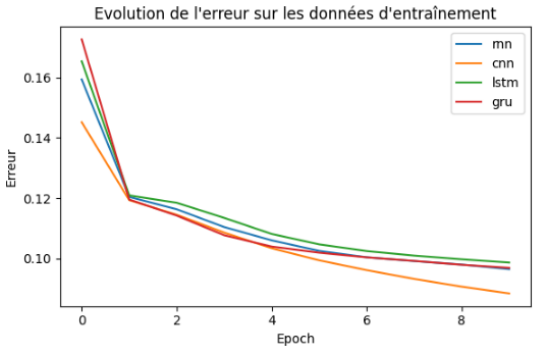

FIGURE V.11 – Perte sur les données d'entraînement

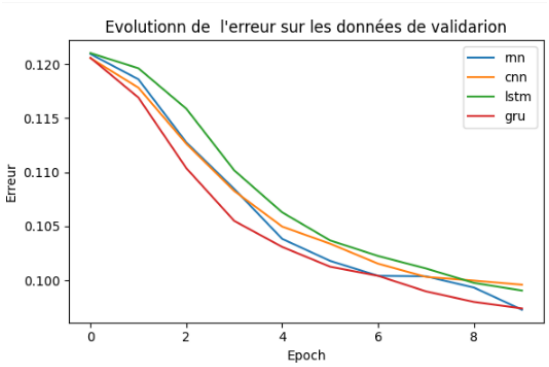

FIGURE V.12 – Perte sur la validation

Le tableau V.8 montre les résultats obtenus avec les différents modèles en terme d'exactitude sur les données d'entraînement et les données de test.

Modèles	Accuracy train(%)	Accuracy test(%)	Time(s)
RNN	47,08	38,44	321,48
CNN	**70,83**	30,34	**202,79**
LSTM	39,16	37,40	1251,46
GRU	52,62	**42,02**	804,34

TABLE V.8 – Comparaison CNN, RNN, LSTM et GRU sur la génération de poème

Au vu des résultats présentés dans le tableau V.8, le modèle GRU fournit de meilleures performances en termes d'exactitude, tant sur les données d'entraînement que sur les données de validation par rapport aux autres modèles

pour la génération de poèmes de Victor Hugo. Cette supériorité peut être attribuée à sa capacité à capturer les dépendances à long terme et à sa structure plus simple comparée au LSTM. Cependant, il est important de noter que le temps d'entraînement du modèle CNN est nettement inférieur à celui du modèle GRU, bien que ce dernier fournisse de meilleures performances. Cette différence de temps d'entraînement pourrait être un facteur à considérer dans des contextes où les ressources computationnelles sont limitées.
L'analyse des courbes d'évolution présentées dans les figures V.9 et V.10 montre que le modèle GRU s'adapte le mieux aux données de validation. De même, les figures V.11 et V.12 illustrent que l'évolution de la perte du modèle GRU suit une allure similaire pour les données d'entraînement et de validation. Ces observations indiquent que le modèle GRU ne fait pas de surapprentissage, ce qui renforce la fiabilité des résultats obtenus.
Il est intéressant de noter que, contrairement à certaines tâches de traitement du langage naturel où les CNN excellent parfois, dans ce cas spécifique de génération de poèmes, le GRU démontre une supériorité claire par rapport au CNN. Cela souligne l'importance de la nature séquentielle et du contexte à long terme dans la poésie de Victor Hugo.

2.2.4 Synthèse des résultats

Dans cette étude, nous avons mené une comparaison approfondie des modèles de deep learning sur différentes tâches. Notre approche s'est concentrée sur deux axes principaux : d'une part, la comparaison entre MLP et CNN pour la classification d'images, et d'autre part, la comparaison entre CNN et réseaux récurrents (RNN, LSTM, GRU) pour l'analyse de sentiment et la génération de texte.

Pour la classification d'images, nos expériences ont démontré une supériorité claire des CNN par rapport aux MLP. Les CNN ont non seulement atteint une précision plus élevée (67,77% contre 49,61% pour les MLP), mais ont également montré une meilleure efficacité en termes de temps d'entraînement et d'inférence.

Pour l'analyse de sentiment avec le jeu de données IMDB, les modèles CNN et GRU ont montré des performances quasi identiques montrant des précisions plus considérables (84,73% et 84,60%). De plus, le CNN s'est distingué par un temps d'entraînement significativement plus court.

En ce qui concerne la génération de texte, les modèles récurrents, en particulier le GRU, ont surpassé le CNN, démontrant une meilleure capacité à capturer les dépendances à long terme essentielles pour cette tâche avec une exactitude de 42,02% contre 30,34% pour le CNN sur les données de test.

Ces résultats démontrent clairement que chaque architecture de deep learning s'adapte au mieux dans des domaines spécifiques. Comme souligné dans beaucoup d'articles de référence, les CNN sont plus adaptés pour la classification d'image par rapport aux MLP. Cependant, sur la tâche d'analyse de sentiment avec le jeu de données IMDB, les performances du modèle CNN montrent une contradiction vis à vis de ce qui est soutenu dans beaucoup d'articles de référence qui concluent que les réseaux récurrent tels que les LSTM et les GRU sont beaucoup plus adaptés pour le traitement des données séquentielles telles que les phrases de textes dans notre cas. Néanmoins, les résultats satisfaisants du modèle GRU sur la génération de texte peuvent témoigner l'utilité des réseaux récurrents sur le traitement du texte à travers leur capacité à capturer les dépendances à long terme dans de longues séquences.

Bien que nous ayons obtenu des résultats significatifs, il n'est pas exclu de reconnaître certaines limites qui

pourraient avoir influencé nos conclusions. Le problème fondamental qui pourrait impacter notre étude comparative est bien évidemment les contraintes de ressources de calcul. Cela a restreint la possibilité d'effectuer des expériences à grande échelle. Nous n'avons pas pu tester nos modèles sur des ensembles de données très volumineux ou réaliser un grand nombre d'itérations pour affiner nos résultats. C'est ce qui pourrait potentiellement limité la généralisation de nos conclusions et la robustesse de nos comparaisons entre les différentes architectures. De plus, ces contraintes nous ont empêché de faire une analyse approfondie de la stabilité des performances des modèles face à la variation des données, un aspect crucial pour évaluer la fiabilité des architectures dans des scénarios réels. Une telle analyse aurait pu fournir des informations précieuses sur la robustesse des différents modèles dans des conditions variables.

2.3 Discussion générale

Après avoir exploré différents articles sur l'étude comparative des architectures traditionnelles de deep learning, notamment le MLP, le CNN et les réseaux récurrents (RNN, LSTM et GRU), nous avons proposé une approche visant à approfondir et confirmer les observations décelées dans ces différentes méthodologies. Les résultats obtenus avec nos différentes expériences semblent corroborer les observations des études antérieures, notamment avec l'analyse de sentiments en utilisant le jeu de données IMDB ainsi que la classification d'image qui montre une nette supériorité des CNN par rapport aux autres modèles. De plus, notre expérience sur la génération de texte témoigne clairement la puissance des réseaux récurrents sur le traitements des données séquentielles et leurs capacités à gérer les dépendances à longue terme.

Dans notre approche, le choix des paramètres a été guidé par les capacités en termes de ressources de calcul de l'environnement sur lequel nous avons travaillé. Nous avons utilisé différentes métriques pour mesurer la performance des modèles en fonction de la nature de la tâche. Pour la génération de texte, qui est une tâche de prédiction, nous n'avons pas utilisé les métriques de classification comme pour les autres tâches, mais nous avons plutôt évalué les modèles en utilisant la précision sur les données d'entraînement et de test.

Conclusion

Dans ce chapitre, nous avons abordé l'étude comparative approfondie des performances des modèles traditionnels de deep learning, incluant le MLP, le CNN et les variantes de RNN (RNN Simple, LSTM et GRU), sur un éventail de tâches variées.

Notre approche s'est articulée autour de deux axes principaux : d'une part, une revue exhaustive des travaux existants sur les études comparatives de ces architectures, et d'autre part, une proposition de notre propre étude comparative.

Les résultats et observations ont montré que les CNN excellent beaucoup plus dans la classification d'images et démontrent une performance remarquable en terme de précision et d'efficience dans l'analyse de sentiments, notamment sur le jeu de données IMDB. Par ailleurs, notre étude a mis en évidence la supériorité des réseaux récurrents, particulièrement les modèles GRU et LSTM, dans les tâches de génération de texte, soulignant leur capacité à capturer efficacement les dépendances à long terme.

✠ Conclusion générale ✠

Dans ce mémoire, nous avons mené une exploration approfondie des concepts fondamentaux et des architectures traditionnelles du deep learning, couvrant le (MLP), les réseaux de neurones convolutifs (CNN) et les réseaux de neurones récurrents (RNN). À travers sept chapitres, nous avons retracé l'évolution historique du domaine, examiné les principes de base des réseaux de neurones artificiels, et analysé en détail la structure et le fonctionnement de chaque architecture. Notre étude comparative a mis en lumière l'importance cruciale du choix de l'architecture en fonction de la nature spécifique de la tâche et des données traitées.

L'objectif de ce travail était de fournir une compréhension intuitive et accessible du deep learning, en proposant des explications claires et concises des concepts théoriques et pratiques. Nous espérons que ce rapport servira de base solide pour ceux qui souhaitent approfondir leur exploration de ce domaine en constante évolution.

Au-delà des architectures traditionnelles abordées, le paysage du deep learning continue de s'enrichir avec l'émergence de nouvelles approches prometteuses. Les réseaux antagonistes génératifs (GAN) [158], par exemple, ouvrent de nouvelles perspectives en génération de données réalistes et en apprentissage non supervisé. Les Auto-Encodeurs offrent des possibilités intéressantes en réduction de dimensionnalité [113], tandis que les Transformers [159] révolutionnent le traitement des dépendances à long terme dans les données séquentielles. Les architectures hybrides [160], combinant différents types de modèles, visent à exploiter les forces de chaque approche pour des tâches complexes .

Ces développements récents promettent des avancées significatives dans le traitement des données complexes, élargissant le champ d'application du deep learning et renforçant son rôle central dans l'intelligence artificielle moderne. Malgré les défis persistants, les recherches actuelles continuent d'améliorer les performances des modèles existants et d'étendre leur applicabilité à des domaines encore peu explorés.

Le domaine du deep learning demeure dynamique et en constante évolution, ouvrant la voie à des innovations passionnantes et à des applications toujours plus sophistiquées. Cette progression continue renforce l'importance du deep learning dans l'avancement de la technologie et de la science des données, promettant un avenir riche en découvertes et en applications révolutionnaires. Alors que nous continuons à repousser les frontières de ce que les réseaux neuronaux peuvent accomplir, il est clair que le deep learning jouera un rôle crucial dans la façon dont nous abordons les défis complexes de notre époque, de l'analyse de données massives à la création de systèmes intelligents capables d'interagir de manière naturelle avec le monde qui nous entoure.

✠ Bibliographie ✠

[1] J. Schmidhuber. Deep learning in neural networks : an overview. *Neural Netw.*, 61, 2014.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553) :436–444, May 2015.

[3] John McCarthy et al. What is artificial intelligence. 2007.

[4] Gheorghe Tecuci. Artificial intelligence. *WIREs Computational Statistics*, 4(2) :168–180, March 2012.

[5] Iqbal H. Sarker. Machine Learning : Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, 2(3) :160, March 2021.

[6] Pramila P. Shinde and Seema Shah. A Review of Machine Learning and Deep Learning Applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, Pune, India, August 2018. IEEE.

[7] M.A. Hearst, S.T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4) :18–28, July 1998. Conference Name : IEEE Intelligent Systems and their Applications.

[8] Steven J. Rigatti. Random Forest. *Journal of Insurance Medicine*, 47(1) :31–39, January 2017.

[9] Eve Mathieu-Dupas. Algorithme des k plus proches voisins pondérés et application en diagnostic. In *42èmes Journées de Statistique*, 2010.

[10] B. YEGNANARAYANA. *ARTIFICIAL NEURAL NETWORKS*. PHI Learning Pvt. Ltd., January 2009. Google-Books-ID : RTtvUVU_xL4C.

[11] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7) :1527–1554, July 2006.

[12] Georgia Koppe, Andreas Meyer-Lindenberg, and Daniel Durstewitz. Deep learning for small and big data in psychiatry. *Neuropsychopharmacology*, 46(1) :176–190, 2021. ISBN : 0893-133X Publisher : Springer International Publishing Cham.

[13] Ahsan Adeel, Mandar Gogate, and Amir Hussain. Contextual deep learning-based audio-visual switching for speech enhancement in real-world environments. *Information Fusion*, 59 :163–170, 2020. ISBN : 1566-2535 Publisher : Elsevier.

[14] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3) :55–75, 2018. ISBN : 1556-603X Publisher : IEEE.

[15] Yuan Chen, Zhiming Ye, Yanfang Zhang, Wenxi Xie, Qingyun Chen, Chunhong Lan, Xiujia Yang, Huikun Zeng, Yan Zhu, Cuiyu Ma, Haipei Tang, Qilong Wang, Junjie Guan, Sen Chen, Fenxiang Li, Wei Yang, Huacheng Yan, Xueqing Yu, and Zhenhai Zhang. A Deep Learning Model for Accurate Diagnosis of Infection Using Antibody Repertoires. *Journal of Immunology (Baltimore, Md. : 1950)*, 208(12) :2675–2685, June 2022.

[16] Swarnendu Ghosh, Nibaran Das, Ishita Das, and Ujjwal Maulik. Understanding Deep Learning Techniques for Image Segmentation. *ACM Computing Surveys*, 52(4) :1–35, July 2020.

[17] Frederico A C Azevedo, Ludmila R B Carvalho, Lea T Grinberg, José Marcelo Farfel, Renata E L Ferretti, Renata E P Leite, Wilson Jacob Filho, Roberto Lent, and Suzana Herculano-Houzel. Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *The Journal of comparative neurology*, 513(5) :532—541, April 2009.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, November 2016. Google-Books-ID : omivDQAAQBAJ.

[19] Jürgen Schmidhuber. Deep learning in neural networks : An overview. *Neural Networks*, 61 :85–117, January 2015.

[20] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Bioph.*, 5, 1943.

[21] Bohdan Macukow. Neural Networks – State of Art, Brief History, Basic Models and Architecture. In Khalid Saeed and Władysław Homenda, editors, *Computer Information Systems and Industrial Management*, Lecture Notes in Computer Science, pages 3–14, Cham, 2016. Springer International Publishing.

[22] D. Hebb. *The Organization of Behavior*. Wiley, New York, 1949.

[23] F. Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6) :386–408, 1958. Place : US Publisher : American Psychological Association.

[24] Ted Hoff Bernard Windrow. Widrow, B., Hoff, M. : Adaptive switching circuits. Technical report 1553-1, Stanford Electron. Labs., Stanford, June 1960. 1960.

[25] M. L. Minsky and S. Papert. *Perceptrons : An Introduction to Computational Geometry*. MIT Press, Cambridge, 1969.

[26] Paul Werbos. *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Science. Thesis (Ph. D.). Appl. Math. Harvard University*. PhD thesis, 01 1974.

[27] Kunihiko Fukushima. Cognitron : A self-organizing multilayered neural network. *Biological Cybernetics*, 20(3-4) :121–136, 1975.

[28] S. Grossberg. Adaptive pattern classification and universal recoding. *Biol. Cyber.*, 23, 1976.

[29] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1) :59–69, 1982.

[30] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. USA*, 79, 1982.

[31] David E Rumelhart, Geoffrey E Hintont, and Ronald J Williams. Learning representations by back-propagating errors. 1986.

[32] Geoffrey E. Hinton. Learning translation invariant recognition in a massively parallel networks. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, J. W. Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, volume 258, pages 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987. Series Title : Lecture Notes in Computer Science.

[33] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Ha. Gradient-Based Learning Applied to Document Recognition. 1998.

[34] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8) :1735–1780, November 1997.

[35] Kunihiko Fukushima. Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4) :193–202, April 1980.

[36] Samandarov Erkaboy Karimboyevich and Abdurakhmonov Olim Nematullayevich. Single Layer Artificial Neural Network : Perceptron. *European Multidisciplinary Journal of Modern Science*, 5 :230–238, April 2022.

[37] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning : A comprehensive survey and benchmark. *Neurocomputing*, 503 :92–108, September 2022.

[38] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech Recognition Using Deep Neural Networks : A Systematic Review. *IEEE Access*, 7 :19143–19165, 2019.

[39] Yuhan Bai. RELU-Function and Derived Function Review. *SHS Web of Conferences*, 144 :02006, 2022.

[40] Serhat Kiliçarslan, Kemal Adem, and Mete ÇeliK. An overview of the activation functions used in deep learning algorithms. *Journal of New Results in Science*, 10(3) :75–88, December 2021.

[41] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *Schedae Informaticae*, 1/2016, 2017.

[42] Tibshiran Friedman, Hastie. *The Elements of Statistical Learning*. 2001.

[43] Risto Miikkulainen. *Topology of a Neural Network*, pages 988–989. Springer US, Boston, MA, 2010.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. L'apprentissage en profondeur. *Presse du MIT*, 2016.

[45] Padmanabha Y C a, Viswanath Pulabaigari, and Eswara B. Semi-supervised learning : a brief review. *International Journal of Engineering Technology*, 7 :81, 02 2018.

[46] R.S. Sutton and A.G. Barto. Reinforcement learning : An introduction. *IEEE Transactions on Neural Networks*, 9(5) :1054–1054, 1998.

[47] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training, 2019.

[48] Saad Hikmat Haji and Adnan Mohsin Abdulazeez. Comparison of optimization techniques based on gradient descent algorithm : A review. *PalArch's Journal of Archaeology of Egypt / Egyptology*, 18(4) :2715–2743, Feb. 2021.

[49] Jiawei Zhang. Gradient descent based optimization algorithms for deep learning models training, 2019.

[50] Shun ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4) :185–196, 1993.

[51] Diederik P. Kingma and Jimmy Ba. Adam : A method for stochastic optimization, 2017.

[52] Simon S Haykin. *Neural networks and learning machines*. Pearson Upper Saddle River, 2009.

[53] Marvin Minsky and Seymour A. Papert. *Perceptrons : An Introduction to Computational Geometry*. The MIT Press, 2017.

[54] Raul Rojas. *Neural Networks : A Systematic Introduction*. Springer Science & Business Media, June 2013. Google-Books-ID : 4rESBwAAQBAJ.

[55] Richardlippmann. Book review : "neural networks, a comprehensive foundation", by simon haykin. *International Journal of Neural Systems*, 05, 11 2011.

[56] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5) :359–366, 1989.

[57] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks : Tricks of the Trade : Second Edition*, pages 9–48. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[58] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning : A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8) :1798–1828, 2013.

[59] Christopher Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. October 2007.

[60] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *nature*, 542(7639) :115–118, 2017.

[61] Hayit Greenspan, Bram Van Ginneken, and Ronald M Summers. Guest editorial deep learning in medical imaging : Overview and future promise of an exciting new technique. *IEEE transactions on medical imaging*, 35(5) :1153–1159, 2016.

[62] Thomas Janssoone, Pierre Rinder, Clémence Bic, Dorra Kanoun, and Pierre Hornus. Modèles d'apprentissage automatique de la persistance aux médicaments : application au cancer du sein. In *Conférence Nationale en Intelligence Artificielle*, 2019.

[63] Mathias Kraus and Stefan Feuerriegel. Decision support from financial disclosures with deep neural networks and transfer learning. *Decision Support Systems*, 104 :38–48, 2017.

[64] Bouab Ghania and Ait Allouche Nouara. *Détection des précipitations en utilisant un réseau de neurones MLP.* PhD thesis, Université Mouloud Mammeri, 2015.

[65] Christopher Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. October 2007.

[66] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10) :1995, 1995.

[67] Yann LeCun, Lawrence D. Jackel, Léon Bottou, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, Urs A. Muller, Eduard Sackinger, and Patrice Simard. Learning algorithms for classification : A comparison on handwritten digit recognition. *Neural networks : the statistical mechanics perspective*, 261(276) :2, 1995.

[68] Maria Vakalopoulou, Stergios Christodoulidis, Ninon Burgos, Olivier Colliot, and Vincent Lepetit. Deep Learning : Basics and Convolutional Neural Networks (CNNs). In Olivier Colliot, editor, *Machine Learning for Brain Disorders*, Neuromethods, pages 77–115. Springer US, New York, NY, 2023.

[69] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks : an overview and application in radiology. *Insights into Imaging*, 9(4) :611–629, August 2018. Number : 4 Publisher : SpringerOpen.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[71] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, Antalya, August 2017. IEEE.

[72] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A Survey of Convolutional Neural Networks : Analysis, Applications, and Prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12) :6999–7019, December 2022.

[73] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Toward real-time indoor semantic segmentation using depth information. *Journal of Machine Learning Research*, 2014.

[74] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE access*, 7 :53040–53065, 2019.

[75] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. A survey of the recent architectures of deep convolutional neural networks. *Artificial intelligence review*, 53 :5455–5516, 2020.

[76] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks, November 2013. arXiv :1311.2901 [cs].

[77] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, January 2015. arXiv :1409.0575 [cs].

[78] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015. arXiv :1409.1556 [cs].

[79] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[80] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition, December 2015. arXiv :1512.03385 [cs] version : 1.

[81] Y. Le Cun, L.D. Jackel, B. Boser, J.S. Denker, H.P. Graf, I. Guyon, D. Henderson, R.E. Howard, and W. Hubbard. Handwritten digit recognition : applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11) :41–46, November 1989.

[82] Sutanu Bhowmick, Satish Nagarajaiah, and Ashok Veeraraghavan. Vision and deep learning-based algorithms to detect and quantify cracks on concrete surfaces from UAV videos. *Sensors*, 20(21) :6299, 2020. ISBN : 1424-8220 Publisher : MDPI.

[83] Swarnendu Ghosh, Nibaran Das, Ishita Das, and Ujjwal Maulik. Understanding Deep Learning Techniques for Image Segmentation. *Enquêtes informatiques ACM*, 52(4) :73 :1–73 :35, August 2019.

[84] T. Ahonen, A. Hadid, and M. Pietikainen. Face Description with Local Binary Patterns : Application to Face Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12) :2037–2041, December 2006.

[85] Guoqing Li, Meng Zhang, Jiaojie Li, Feng Lv, and Guodong Tong. Efficient densely connected convolutional neural networks. *Pattern Recognition*, 109 :107610, 2021.

[86] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen A. W. M. van der Laak, Bram van Ginneken, and Clara I. Sánchez. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42 :60–88, December 2017.

[87] S. Suganyadevi, V. Seethalakshmi, and K. Balasamy. A review on deep learning in medical image analysis. *International Journal of Multimedia Information Retrieval*, 11(1) :19–38, March 2022.

[88] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1) :24–29, January 2019.

[89] Yoon Kim. Convolutional neural networks for sentence classification. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[90] Wei Wang and Jianxun Gang. Application of convolutional neural network in natural language processing. pages 64–70, 07 2018.

[91] Serkan Kiranyaz, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman. 1D convolutional neural networks and applications : A survey. *Mechanical Systems and Signal Processing*, 151 :107398, 2021.

[92] Honglak Lee, Peter Pham, Yan Largman, and Andrew Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. *Advances in neural information processing systems*, 22, 2009.

[93] Darren Hau and Ke Chen. Exploring hierarchical speech representations with a deep convolutional neural network. *UKCI 2011 Accepted Papers*, 37 :31, 2011.

[94] Kenji Kawaguchi. Deep learning without poor local minima. *Advances in neural information processing systems*, 29, 2016.

[95] Ahmad Shawahna, Sadiq Sait, and Aiman El-Maleh. Fpga-based accelerators of deep learning networks for learning and classification : A review. *IEEE Access*, PP :1–1, 12 2018.

[96] K-K Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on pattern analysis and machine intelligence*, 20(1) :39–51, 1998.

[97] Wojciech Samek, Thomas Wiegand, and Klaus-Robert Müller. Explainable artificial intelligence : Understanding, visualizing and interpreting deep learning models. *ArXiv*, abs/1708.08296, 2017.

[98] Taco Cohen and Max Welling. Group equivariant convolutional networks. 01 2016.

[99] Shaojie Bai, J. Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. 03 2018.

[100] Ashia Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. 05 2017.

[101] Sergey Ioffe and Christian Szegedy. Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift, March 2015. arXiv :1502.03167 [cs].

[102] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. Efficient processing of deep neural networks : A tutorial and survey. *Proceedings of the IEEE*, 105(12) :2295–2329, 2017.

[103] Connor Shorten and Taghi M. Khoshgoftaar. A survey on Image Data Augmentation for Deep Learning. *Journal of Big Data*, 6(1) :60, July 2019.

[104] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A Simple Way to Initialize Recurrent Networks of Rectified Linear Units, April 2015. arXiv :1504.00941 [cs].

[105] Ajay Shrestha and Ausif Mahmood. Review of Deep Learning Algorithms and Architectures. *IEEE Access*, 7 :53040–53065, 2019.

[106] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent Neural Networks for Multivariate Time Series with Missing Values. *Scientific Reports*, 8(1) :6085, April 2018. Number : 1 Publisher : Nature Publishing Group.

[107] Giosué Lo Bosco, Giovanni Pilato, and Daniele Schicchi. A Neural Network model for the Evaluation of Text Complexity in Italian Language : a Representation Point of View. *Procedia Computer Science*, 145 :464–470, January 2018.

[108] Gang Chen. A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation, January 2018. arXiv :1610.02583 [cs].

[109] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent Advances in Recurrent Neural Networks, February 2018. arXiv :1801.01078 [cs].

[110] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, March 2010. ISSN : 1938-7228.

[111] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02) :107–116, 1998. ISBN : 0218-4885 Publisher : World Scientific.

[112] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A Review of Recurrent Neural Networks : LSTM Cells and Network Architectures. *Neural Computation*, 31(7) :1235–1270, July 2019. Conference Name : Neural Computation.

[113] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, September 2014. arXiv :1406.1078 [cs, stat] version : 3.

[114] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2) :157–166, March 1994.

[115] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[116] Razvan Pascanu, Tomas Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*, 11 2012.

[117] Alex Graves. Generating sequences with recurrent neural networks. 08 2013.

[118] Klaus Greff, Rupesh Srivastava, Jan Koutník, Bas Steunebrink, and Jürgen Schmidhuber. Lstm : A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28, 03 2015.

[119] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. 09 2012.

[120] Ilya Sutskever, Oriol Vinyals, and Quoc Le. Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 4, 09 2014.

[121] Saedeh Abbaspour, Faranak Fotouhi, Ali Sedaghatbaf, Hossein Fotouhi, Maryam Vahabi, and Maria Linden. A comparative analysis of hybrid deep learning models for human activity recognition. *Sensors*, 20(19), 2020.

[122] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2342–2350, Lille, France, 07–09 Jul 2015. PMLR.

[123] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, December 2014. arXiv :1412.3555 [cs].

[124] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[125] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Proc. Interspeech 2010*, pages 1045–1048, 2010.

[126] Duyu Tang, Bing Qin, Xiaocheng Feng, and Ting Liu. Effective lstms for target-dependent sentiment classification, 2016.

[127] Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. Cross-lingual sentiment classification with bilingual document representation learning. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pages 1403–1412, Berlin, Germany, August 2016. Association for Computational Linguistics.

[128] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270, 12 2017.

[129] Amir Ghaderi, Borhan Sanandaji, and Faezeh Ghaderi. Deep forecast : Deep learning-based spatio-temporal forecasting. 07 2017.

[130] D. Eck and J. Schmidhuber. Finding temporal structure in music : blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[131] Avinash Pawar, Mrs. Mrunal Bewoor, Suhas Patil, Mrs.Sheetal Patil, Rohini Jadhav, and Amol Kadam. Music generation using rnn-lstm with gru. 11 2023.

[132] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning, October 2015. arXiv :1506.00019 [cs].

[133] George Philipp, Dawn Song, and Jaime G. Carbonell. The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions, 2017.

[134] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[135] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958, 2014. ISBN : 1532-4435 Publisher : JMLR. org.

[136] Meha Desai and Manan Shah. An anatomization on breast cancer detection and diagnosis employing multilayer perceptron neural network (MLP) and Convolutional neural network (CNN). *Clinical eHealth*, 4 :1–11, 2021.

[137] Elif Derya Übeyli. Implementing automated diagnostic systems for breast cancer detection. *Expert systems with Applications*, 33(4) :1054–1062, 2007.

[138] Victor-Emil Neagoe, Adrian-Dumitru Ciotec, and George-Sorin Cucu. Deep Convolutional Neural Networks Versus Multilayer Perceptron for Financial Prediction. In *2018 International Conference on Communications (COMM)*, pages 201–206, June 2018.

[139] Xue bo Jin, Xing-Hong Yu, Xiaoyi Wang, Yu ting Bai, Tingli Su, and Jianlei Kong. Prediction for time series with cnn and lstm. 2019.

[140] Fatemehalsadat Madaeni, Karem Chokmani, Rachid Lhissou, Saeid Homayouni, Yves Gauthier, and Simon Tolszczuk-Leclerc. Convolutional neural network and long short-term memory models for ice-jam predictions. *The Cryosphere*, 16(4) :1447–1468, April 2022. Publisher : Copernicus GmbH.

[141] Wei Quan, Zheng Chen, Jianliang Gao, and Xiaohua Tony Hu. Comparative study of cnn and lstm based attention neural networks for aspect-level opinion mining. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 2141–2150, 2018.

[142] Fabio C. Zegarra, Juan Vargas-Machuca, and Alberto M. Coronado. A comparative study of CNN, LSTM, BiLSTM, AND GRU architectures for tool wear prediction in milling processes. *Journal of Machine Engineering*, 23(4) :122–136, 2023.

[143] Rahal Al Orabi Wael Al Safa Haidar, Nassar Mohamed. Analyse des réseaux de neurones convolutifs et récurrents pour la classification des logiciels malveillants. In *2019 15e Conférence internationale sur les communications sans fil et l'informatique mobile (IWCMC)*, 2019.

[144] Street Nick Wolberg William, Mangasarian Olvi. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1995. DOI : https ://doi.org/10.24432/C5DW2B.

[145] Jorge J. More. The levenberg-marquardt algorithm : Implementation and theory. In G. A. Watson, editor, *Numerical Analysis*, pages 105–116, Berlin, Heidelberg, 1978. Springer Berlin Heidelberg.

[146] Ulrike Grömping. South german credit data : Correcting a widely used data set, 11 2019.

[147] Ross Quinlan. Statlog (Australian Credit Approval). UCI Machine Learning Repository. DOI : https ://doi.org/10.24432/C59012.

[148] Sebastian Blauth and Kevin Sturm. Quasi-newton methods for topology optimization using a level-set method. *Structural and Multidisciplinary Optimization*, 66(9), September 2023.

[149] Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum, 2020.

[150] S Ben Driss, Mahmoud Soua, Rostom Kachouri, and Mohamed Akil. A comparison study between mlp and convolutional neural network models for character recognition. In *Real-Time Image and Video Processing 2017*, volume 10223, pages 32–42. SPIE, 2017.

[151] T. E. de Campos, B. R. Babu, and M. Varma. Character recognition in natural images. In *Proceedings of the International Conference on Computer Vision Theory and Applications, Lisbon, Portugal*, February 2009.

[152] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. Microsoft malware classification challenge, 02 2018.

[153] Farhad Mortezapour Shiri, Thinagaran Perumal, Norwati Mustapha, and Raihani Mohamed. A comprehensive overview and comparative analysis on deep learning models : Cnn, rnn, lstm, gru, 2023.

[154] M. Bahraminasr and A. Vafaei Sadr. Imdb data from two generations, from 1979 to 2019 ; part one, dataset introduction and preliminary analysis, 2020.

[155] Hande Alemdar, Halil Ertan, Ozlem Durmaz Incel, and Cem Ersoy. Aras human activity datasets in multiple homes with multiple residents. In *2013 7th International Conference on Pervasive Computing Technologies for Healthcare and Workshops*, pages 232–235, 2013.

[156] Mihai Oltean. Fruits-360 dataset : new research directions, 09 2021.

[157] Raveen Doon, Tarun Kumar Rawat, and Shweta Gautam. Cifar-10 classification using deep convolutional neural network. In *2018 IEEE Punecon*, pages 1–5, 2018.

[158] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[159] Tao Lei, Yu Zhang, Sida I. Wang, Hui Dai, and Yoav Artzi. Simple Recurrent Units for Highly Parallelizable Recurrence, September 2018. arXiv :1709.02755 [cs].

[160] Mohsin Ashraf, Fazeel Abid, Ikram Ud Din, Jawad Rasheed, Mirsat Yesiltepe, Sook Fern Yeo, and Merve Ersoy. A hybrid cnn and rnn variant model for music classification. *Applied Sciences*, 13 :1476, 01 2023.