République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari Boumediene

Faculté d'Electronique et d'Informatique

Département d'Informatique

Mémoire de Licence

Spécialité :

Ingénierie des Systèmes d'Informations et des Logiciels

Thème

Un modèle de réseaux de neurones efficace pour la classification de données simulées pour l'identification du signal du Higgs au collisionneur hadronique du CERN

Sujet Proposé par :
Pr. ATIF Karim
Soutenu le : 14 Juillet 2019

Présenté par :
M. BOUAYED Aymene Mohammed

Devant le jury composé de :
Président du jury : Pr. GUESSOUM A.
Membre du jury : Pr. KHENNAK I.

Projet N° 097/2019

Remerciement

Je tiens tout d'abord à exprimer le plaisir que j'ai eu lors des longues séances de discussions avec mon promoteur Pr. K. Atif qui a toujours su trouver les issues de sorties ainsi que les orientations adéquates. Ses conseils m'étaient d'une grande importance. Merci Monsieur.

Aussi, je remercie les membres du jury Pr. A. Guessoum et Pr. I. Khannak qui ont bien voulu me faire l'honneur de juger mon travail.

Je n'oublierai jamais tous les enseignants qui ont assuré ma formation durant mon parcours universitaire, pour l'ensemble des connaissances que j'ai consenti à leur égard.

J'adresse aussi mes plus sincères remerciement à tous les membres de ma famille, qui m'ont aidé à surmonter tous les obstacles au cours de la réalisation de ce mémoire par leurs prières et encouragements.

Enfin, je dis "el hamdou li Lah" de m'avoir donné le courage, la foi et surtout la patience pour réaliser de ce travail.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Le Conseil Européen de la Recherche Nucléaire (CERN) est un des plus grand et des plus prestigieux centres de recherche en physique des particules. En 2014, il a lancé une compétition en vue de résoudre un problème dans le domaine de la physique fondamentale en utilisant les algorithmes de l'intelligence artificielle. Il s'agissait de démêler le signal du Higgs du bruit de fond. Beaucoup d'experts dans le domaine s'y sont mis pour essayer de donner les meilleurs résultats en terme de précision. Le lauréat de cette compétition a utilisé une structure de réseau de neurones bien spécifique et il a obtenu une précision de Signification Médiane Approximative (AMS : une fonction proposée par le CERN pour mesurer le performance d'un modèle de classification) de 3.805.

Dans notre projet de fin d'études de licence, nous nous proposons de traiter le même problème de la compétition du CERN 2014 pour essayer d'améliorer le résultat du lauréat de la compétition.

Pour atteindre les objectifs assignés, nous procèdons comme suit. Dans une première étape, nous présentons le CERN, ses missions et ses objectives ainsi que le LHC (l'accélérateur le plus récent de CERN). Ensuite, nous nous intéressons à la particule du Higgs détectée en 2012 par le LHC. Nous introduissons la compétition que le CERN a lancé en 2014, le résultat que le lauréat a obtenu et l'approche qu'il a adopté pour obtenir son résultat. Enfin, nous établissons la problématique et les objectifs de notre projet.

Dans une seconde étape, nous nous intéressons à la théorie des réseaux de neurones. Nous définissons les composants essentiels des réseaux de neurones. Puis, nous traitons de l'algorithme d'apprentissage le plus utilisé (algorithme de rétro-propagation), ainsi que des différents paramètres d'apprentissage. Enfin, nous présentons les algorithmes Adaboost, SAMME.R et Bagging.

Dans une avant dernière étape, nous proposons deux architectures de réseaux de neurones à savoir réseau de neurones multi-couches (MLP) et réseau de neurones convolutionnel (CNN) dédiés à résoudre notre problématique. Pour chaque architecture, nous étudions les paramètres correspondants en vue de les optimiser. Ceci nous mène vers la proposition de notre architecture optimale. Une approche de résolution parallèle est proposée devant l'important volume de données à traiter.

Dans une dernière étape, nous analysons les données fournies par le CERN dans le cadre de la compétition. Nous sélectionnons les caractéristiques pertinentes qui peuvent aider dans la classification en signal et bruit de fond. Nous traitons les valeurs manquantes. Enfin, nous faisons l'apprentissage des réseaux de neurones que nous avons proposé et on mesure leur performance en terme d'AMS.

Nous terminons par une conclusion générale en établissant une synthèse des travaux réalisés avec les éventuelles perspectives.

Chapitre 1

Environnement d'étude

1.1 Introduction

Dans ce chapitre, nous allons présenter le Conseil Européen pour la Recherche Nucléaire (CERN), notamment son organisation, sa mission et ses tâches. Ensuite, nous allons présenter notre projet avec ses principaux objectifs.

1.2 Organisme d'accueil : Conseil Européen de la Recherche Nucléaire

1.2.1 Présentation

L'Organisation Européenne pour la Recherche Nucléaire (aussi appelée Laboratoire Européen pour la Physique des Particules et couramment désignée sous l'acronyme CERN : Conseil Européen pour la Recherche Nucléaire) institué en 1952, est le plus grand centre de physique des particules et un des plus prestigieux laboratoires scientifiques du monde. Il se situe à quelques kilomètres de Genève, en Suisse, à cheval sur la frontière franco-suisse, sur la commune de Meyrin (canton de Genève) [1].

1.2.2 Missions et tâches

Le CERN a pour vocation la physique fondamentale, la découverte des constituants et des lois de l'Univers. Il utilise des instruments scientifiques très complexes pour sonder les constituants ultimes de la matière : les particules fondamentales. En étudiant ce qui se passe lorsque ces particules entrent en collision, les physiciens appréhendent les lois de la nature.

Les instruments qu'utilise le CERN sont des accélérateurs et des détecteurs de particules. Les accélérateurs portent des faisceaux de particules à des énergies élevées pour les faire entrer en collision avec d'autres faisceaux ou avec des cibles fixes. Les détecteurs, eux, observent et enregistrent le résultat de ces collisions [2].

1.3 Large Hadron Collider

1.3.1 Spécification

Le Large Hadron Collider (LHC *Grand collisionneur des hadrons*) est l'accélérateur de particules le plus grand et le plus puissant du monde. Il a démarré le 10 septembre 2008 et est le dernier maillon du complexe d'accélérateurs du CERN. Le LHC a atteint une puissance

d'accélération de particules de 13 TeV (Téra électron Volt) en 2015. Ce qui est 7×10^6 fois plus que l'accélérateur de Draria à Alger qui a une puissance de 2 MeV (Méga électron Volt). Le prochain but du CERN est d'arriver à booster l'énergie du LHC jusqu'à 100 TeV.

Le LHC consiste en un anneau de 27 kilomètres de circonférence, situé à 100 mètres sous terre, formé d'aimants supraconducteurs et de structures accélératrices qui augmentent l'énergie des particules qui y circulent. Il est composé de quatre principaux détecteurs ATLAS, CMS, ALICE et LHCb (voir figure 1.1) [2][3] .

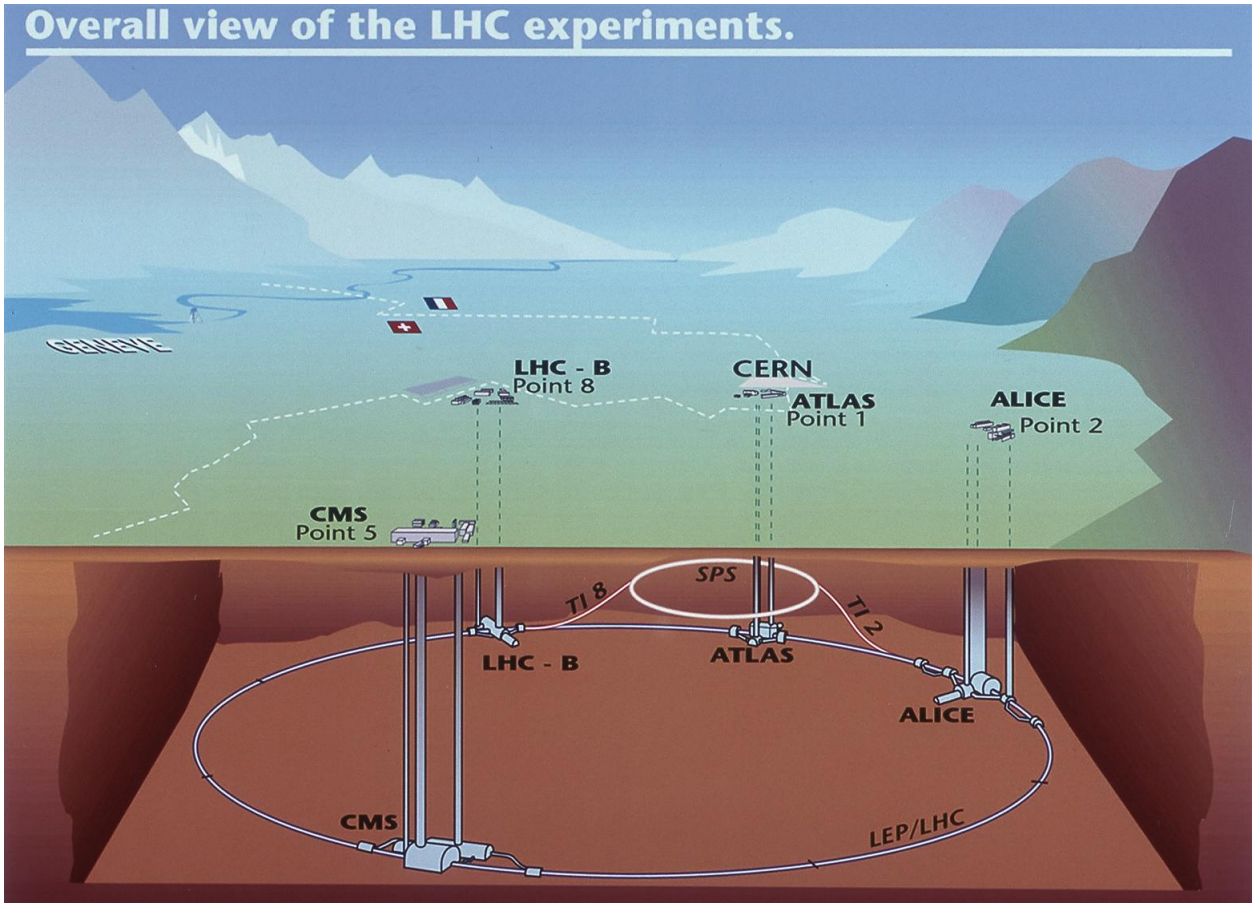

FIGURE 1.1 – Large Hadron Collider [4]

1.3.2 Missions et tâches

Chaque détecteur du LHC a un sous laboratoire associé. Ce dernier a une mission bien spécifique, déterminée par les capacités et l'architecture du détecteur. Les détecteurs ATLAS et CMS, du CERN, ont été construits en vue de découvrir le boson du Higgs ainsi que les particules super-symétriques par deux approches différentes permettant de confirmer les résultats trouvés [5][6]. Le détecteur ALICE a été conçu pour étudier les propriétés physiques de la matière soumise à l'interaction forte, à des densités d'énergie extrêmes auxquelles une phase de la matière appelée plasma quarks-gluons se forme [7]. Le détecteur LHCb explore les légères différences qui existent entre matière et anti-matière grâce à l'étude d'un type de particule appelé "quark beauté" ou "quark b" [8].

1.3.3 Le détecteur ATLAS

Mesurant 46 mètres de long, 25 mètres de haut et 25 mètres de large, et pesant 7000 tonnes, le détecteur ATLAS est le détecteur de particules le plus volumineux jamais construit

(voir figure 1.2). Il se situe à 100 mètres sous terre, à proximité du site principal du CERN. ATLAS est l'un des deux détecteurs polyvalents du LHC. Il étudie des domaines de physique très variés, de la recherche du boson de Higgs aux dimensions supplémentaires de l'espace-temps, en passant par les particules qui pourraient former la matière noire [5].

Les faisceaux de particules du LHC entrent en collision au centre du détecteur ATLAS. Les débris de collision ainsi produits forment de nouvelles particules, qui émergent du point de collision dans toutes les directions. Six sous-systèmes de détection différents, disposés en couches autour du point de collision, enregistrent la trajectoire, l'impulsion et l'énergie des particules. Ceci permet d'identifier chacune d'elles. Un énorme système d'aimants permet d'incurver la trajectoire des particules et, ainsi, de mesurer leur impulsion [5].

Les interactions produites dans le détecteur d'ATLAS créent un énorme flux de données. Pour assimiler ces données, ATLAS utilise un système de déclenchement de pointe, qui indique au détecteur les événements devant être enregistrés et ceux devant être ignorés. Des systèmes complexes d'acquisition de données et de calcul sont ensuite utilisés pour analyser les événements enregistrés [5].

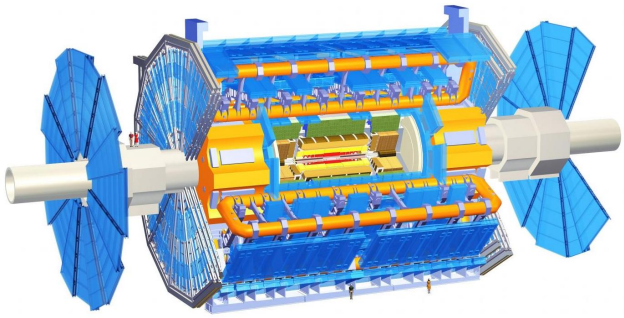

FIGURE 1.2 – Le détecteur ATLAS [9]

1.3.4 Le détecteur CMS

Le Solénoïde Compact pour Muons (CMS *Compact Muon Solenoid*) est un détecteur polyvalent installé sur l'anneau du LHC. Il a été conçu pour explorer un large éventail de domaines de la physique, allant de la recherche du boson de Higgs à celle d'autres dimensions, en passant par la quête des particules qui pourraient constituer la matière noire. Bien que ses buts scientifiques soient les mêmes que ceux de l'expérience ATLAS, la collaboration CMS a opté pour d'autres solutions techniques et un système magnétique de conception différent qui permet de confirmer les résultats entre ces deux laboratoires [6].

Le détecteur CMS est construit autour d'un énorme aimant solénoïde, qui se présente sous la forme d'une bobine cylindrique supra-conductrice, générant un champ magnétique d'environ 100 000 fois le champ magnétique terrestre. Ce champ magnétique créé est confiné par une "culasse" d'acier, qui constitue la pièce la plus lourde de ce détecteur (14 000 tonnes) [6].

Contrairement aux autres détecteurs géants du LHC, dont les éléments ont été construits sous terre, CMS a été construit à la surface, en 15 sections, qui ont ensuite été descendues dans une caverne souterraine située près de Cessy (France), où elles ont été assemblées. Le détecteur dans son ensemble, mesure 21 mètres de long, 15 mètres de large et 15 mètres de haut (voir figure 1.3) [6].

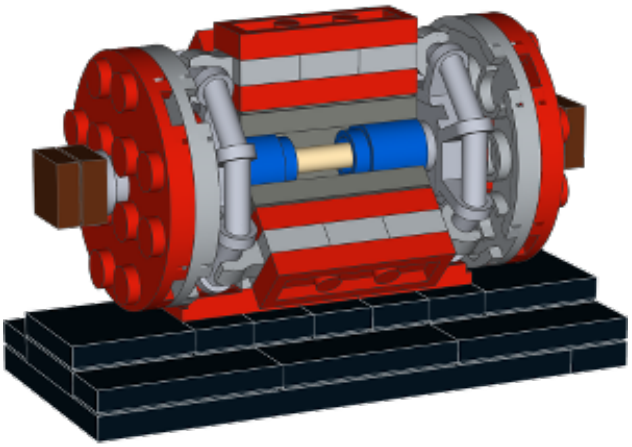

FIGURE 1.3 – Le détecteur CMS [10]

1.4 Higgs

1.4.1 De sa théorie à sa découverte

Dans les années 1970, les physiciens ont compris qu'il y avait des liens étroits entre deux des quatre forces fondamentales, [1] la force nucléaire faible et la force électromagnétique. Ces dernières peuvent être décrites dans le cadre d'une théorie unifiée, qui constitue la base du Modèle Standard de Weinberg-AbduSalam-Glashow. On entend par "unification" le fait que l'électricité, le magnétisme, la lumière et certains types de radioactivité sont toutes des manifestations d'une seule et même force appelée force électro-faible [11].

Les équations fondamentales de la théorie unifiée décrivent de façon correcte l'interaction électro-faible et ses particules médiatrices, à savoir le photon et les bosons W et Z. Mais l'inconvénient est que dans ce modèle, toutes ces particules paraissent dépourvues de masse. Or si le photon a effectivement une masse nulle, nous savons que les particules W et Z ont une masse non nulle. Heureusement, les théoriciens Robert Brout, François Englert et Peter Higgs ont proposé une théorie qui devait résoudre ce problème. Ce que nous appelons à présent mécanisme de Higgs. Ce mécanisme donne une masse au W et au Z ainsi qu'à toutes les autres particules tel que l'électron, ... lorsqu'ils interagissent avec un certain champ invisible, dit « champ de Higgs », présent dans tout l'Univers et auquel est associée une particule dite boson de Higgs [11].

Pendant de nombreuses années, le problème a été qu'aucune expérience n'avait pu observé le boson de Higgs, ce qui aurait permis de confirmer la théorie. Le 4 juillet 2012, les expériences ATLAS et CMS auprès du LHC ont annoncé qu'elles avaient toutes deux observé une nouvelle particule dont la masse se situait dans la région des 125 GeV et qui aurait toutes les propriétés requises de la particule de Higgs. Ainsi, fut découvert le dernier maillon manquant du modèle standard. Le 8 octobre 2013, le prix Nobel de physique a été attribué conjointement à François Englert et Peter Higgs pour la découverte théorique du mécanisme contribuant à notre compréhension de l'origine de la masse des particules subatomiques [11].

1. Les quatre forces fondamentales sont : la force nucléaire faible, la force électromagnétique, la force gravitationnelle et la force nucléaire forte

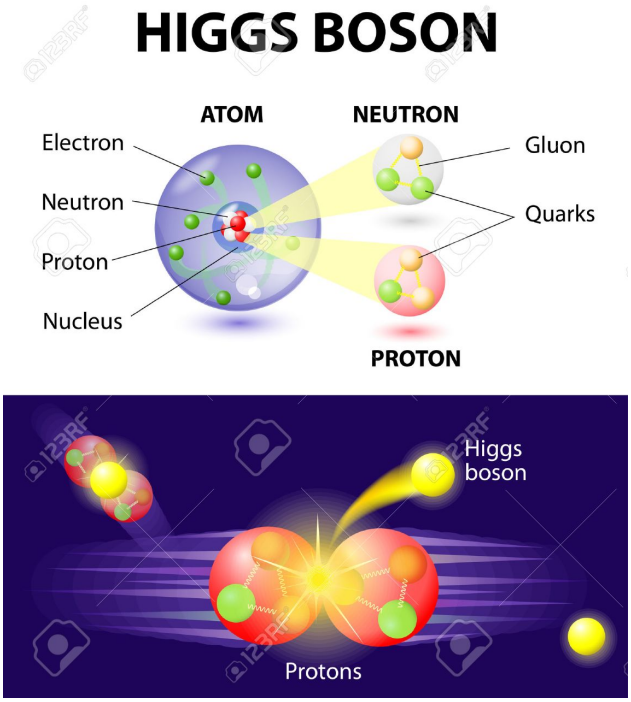

FIGURE 1.4 – Collision donnant lieu aux particules de Higgs [12]

1.4.2 Spécification et utilité

Les exigences de symétrie de l'espace temps font que le modèle standard des interactions électro-faible et forte est obligatoirement formulé avec des particules toutes sans masses. La masse physique observée des particules est alors introduite via le mécanisme de Higgs. Ce mécanisme stipule qu'il existerait une particule massive de charge électrique nulle associée à un champ dit champ de Higgs. Ce dernier, lors de son interaction avec les champs associés avec toutes les autres particules fondamentales connues leur donne leurs masses respectives. Au niveau des détecteurs, la signature du Higgs est faite à travers sa désintégration selon les processus prévus par le modèle standard. Il s'agit du signal que le LHC recherchait. Néanmoins, ce signal ($H^0 \rightarrow \tau + \bar{\tau}$) est entaché d'un bruit de fond provenant de la désintégration d'autres particules tel que le boson Z qui donne le même résultat que la désintégration du Higgs [2] (voir figure 1.5). Donc, démêler le signal provenant du Higgs de celui du bruit de fond est une tâche importante pour mieux identifier toutes les caractéristiques attendues du Higgs et confirmer sa découverte (voir figure 1.6) [11].

2. Il est à remarquer ici que le bruit de fond ayant comme résultat final $\tau + \bar{\tau}$ peut aussi provenir par exemple d'un photon γ ou bien d'autres processus. Mais pour des fins de simplicité, on néglige tous ces processus et on concentre notre attention sur la désintégration $Z \rightarrow \tau + \bar{\tau}$ [13]

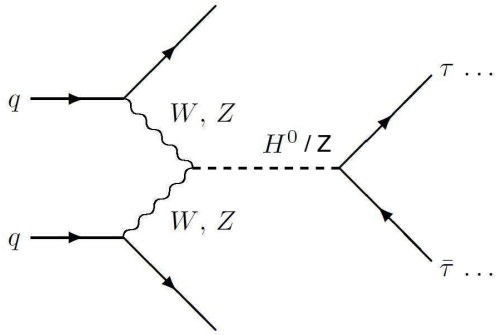

FIGURE 1.5 – Processus de création au LHC de la particule Higgs H^0 (signal) et de la particule Z (bruit de fond) et leur désintégrations en $\tau + \bar{\tau}$[14]

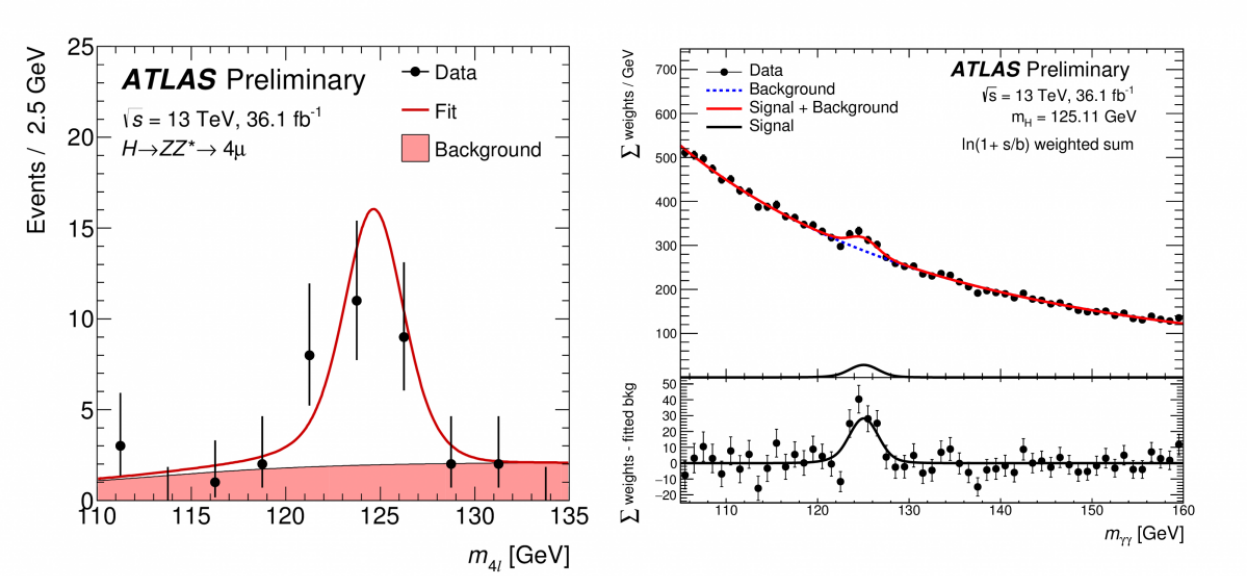

FIGURE 1.6 – Observation au LHC de la particule du Higgs avec une masse de 125GeV (voir pic sur la figure) [15]

1.4.3 Simulation

Jusqu'à présent, le modèle standard construit par les physiciens pour gérer les interactions électro-faible et forte de façon unifiée a réussi à surmonter tous les tests de précision auxquels il a été soumis. C'est donc, ce modèle qui est utilisé pour simuler la réalité physique dont celle de l'interaction avec le boson de Higgs en vue de prédire des résultats à comparer avec les données expérimentales collectées dans des domaines d'énergies inexplorés au paravant. Parmi les logiciels dont le modèle standard est implémenté, on peut citer le logiciel MadGraph 5 [16][17].

1.5 Higgs Boson Machine Learning Challenge

1.5.1 La compétition

En 2014, le CERN a organisé une compétition au nom de "Higgs Boson Machine Learning Challenge" en vue de trouver des idées innovantes et nouvelles s'articulant sur les techniques de l'intelligence artificielle permettant de démêler le signal de Higgs du bruit de fond de la façon la plus optimale et la plus concise possible. Pour cela, le laboratoire ATLAS a

simulé 800000 évènements (maintenant, présent sur le site d'ATLAS avec 818238 évènements) répartis en 30 caractéristiques (voir annexe A). Ces évènements ont été utilisés dans la compétition dont 250000 évènements pour l'apprentissage et 550000 évènements pour les tests et la mesure de la performance du modèle proposé [18][19].

1.5.2 Proposition et résultat du lauréat de la compétition

Le lauréat de la compétition a utilisé un modèle basé sur les réseaux de neurones. Il a obtenu un résultat de Signification Médiane Approximative (AMS, voir paragraphe 4.4) de 3.805. Pour arriver à ce résultat, il a utilisé l'algorithme de Bagging (voir paragraphe 2.13) avec 70 réseaux de neurones. Chacun des réseaux de neurones qu'il a utilisé a trois couches cachées à 600 neurones. Chacune de ces couches a une fonction d'activation "gagnant local emporte le tout" d'un bloc de 3 neurones [20]. En plus des couches cachées, il a aussi utilisé une couche de sortie avec deux neurones et une fonction d'activation Softmax (voir paragraphe 2.5.2.5). Le but de chaque réseau de neurones était de classifier les données comme étant soit signal du Higgs soit bruit de fond. Puis, le lauréat calcule la moyenne de l'ensemble des probabilités résultantes de chaque réseau de neurones. L'apprentissage a été fait pour 200 itérations par réseau de neurones avec l'algorithme d'optimisation "stochastic gradient descent" et un taux l'apprentissage qui était initialement à 1 et qui décrémente à chaque itération de 97%.

En ce qui concerne l'analyse des données, le lauréat a supprimé les caractéristiques se terminant par _phi (*_phi) car il a remarqué qu'elles causent du sur-apprentissage. Les caractéristiques avec de longues queues ont été transformées en leur appliquant la fonction logarithmique. Puis, toutes les caractéristiques ont été normalisées pour avoir une moyenne de 0 et une déviation standard de 1. Toutes les valeurs manquantes ont été remplacées par la valeur 0 [21].

L'apprentissage des 70 réseaux de neurones du lauréat a été fait sur une carte graphique Nvidia de type GTX Titan contenant 2688 CUDA cores et 24Go de RAM.

1.6 Présentation du Sujet

1.6.1 Problématique

Les chercheurs physiciens algériens travaillant dans le domaine de la physique fondamentale veulent se doter d'outils intelligents leur permettant de mieux comprendre les différents mécanismes qui gouvernent la matière. Mais, nous constatons que très peu de recherches ont abordé ce lien entre l'intelligence artificielle et l'exploitation des données de l'expérience en physique fondamentale tel que pour la détection et l'étude du signal du Higgs. Par ailleurs, quoique la proposition et les résultats obtenus par le lauréat de la compétition ont montré une certaine richesse, ces derniers manquent une certaine valorisation par rapport à d'autres modèles de réseaux de neurones.

1.6.2 Objectifs

Notre objectif principal dans ce mémoire, réside dans l'établissement d'un réseau de neurones artificiels dédié à la classification des données en deux classes. Une classe correspondant au signal provenant du Higgs ($H^0 \rightarrow \tau + \bar{\tau}$) et l'autre classe correspondant au bruit de fond provenant du Z ($Z \rightarrow \tau + \bar{\tau}$). Et comme le lauréat de la compétition a obtenu une Signification Médiane Approximative (AMS, voir paragraphe 4.4) de 3.805, notre objectif secondaire

est de s'approcher le plus possible de cette valeur de précision sinon de valoriser les résultats du lauréat. Le modèle réalisé pourrait être une plateforme d'expérience bénéfique aux chercheurs physiciens du pays pour pouvoir maitriser le concept du Higgs.

1.7 Conclusion

Dans ce chapitre, nous avons présenté le CERN, son organisation, ces laboratoires ainsi que la particule du Higgs. Ensuite, nous avons présenté le sujet de ce mémoire et nos objectifs. Dans le chapitre suivant, nous allons présenter les principes et la théorie des réseaux de neurones que nous utiliserons dans les chapitres suivants en vue de résoudre notre problématique.

Chapitre 2

Réseaux de neurones artificiels

2.1 Introduction

Les réseaux de neurones artificiels ont démontré une grande utilité et beaucoup d'avantages dans plusieurs aspects de la vie courante et ils continuent toujours à progresser. Ceci est dû à la très grande activité de la recherche en apprentissage profond qui étudie les réseaux de neurones artificiels.

Vu que nous allons utiliser les réseaux de neurones dans notre projet alors, dans ce chapitre, nous nous intéressons à la théorie des réseaux de neurones artificiels. Nous commençons par définir les réseaux de neurones et leurs composants essentiels. Puis, nous traitons de l'algorithme de rétro-propagation ainsi que des différents paramètres d'apprentissage. Et nous terminons par présenter les réseaux de neurones convolutionnels et les algorithmes Adaboost, SAMME.R et Bagging.

2.2 Définition d'un neurone artificiel

Un neurone artificiel ou un perceptron est un concept inspiré du neurone biologique. C'est l'entité de base dans n'importe quel réseau de neurones. Les perceptrons ont été créés en vue d'avoir les mêmes capacités d'apprentissage que les neurones biologiques. De façon générale, un perceptron fait d'abord la somme pondérée de ses entrées. Puis, il applique à cette somme une certaine fonction mathématique dite fonction d'activation. Ensuite, il envoie le résultat donné par cette fonction aux perceptrons qui lui sont directement reliés. C'est une technique similaire à ce que fait un neurone biologique [22].

2.3 Fonctionnement d'un neurone artificiel

Un neurone artificiel j reçoit en entrée les données à traiter qui sont sous forme d'un vecteur $X_{(j)}$ de dimension $|X_{(j)}| = n_{(j)}$. Ce neurone retourne en résultat un scalaire $o_{(j)}$. Le passage de l'entrée $X_{(j)}$ à la sortie $o_{(j)}$ se fait de la manière suivante [22] :

$$o_{(j)} = \varphi(net_j)$$

où φ est la fonction d'activation qui est une fonction mathématique permettant de différencier si un neurone ne va pas s'activer ($o_{(j)} = 0$), est totalement activé ($o_{(j)} = X_{(j)}.W_{(j)} + B_{(j)}$), ou est partiellement activé ($o = \alpha(X_{(j)}.W_{(j)} + B_{(j)})$ avec $0 < |\alpha| < 1$) ; et

$$net_j = (X_{(j)}.W_{(j)} + B_{(j)}).$$

La somme pondérée $X_{(j)}.W_{(j)}$ est donnée par [22] :

$$\sum_{i=1}^{n} X_{(j),i} W_{(j),i}$$

où $W_{(j)}$ est un vecteur de même dimension que le vecteur des données $X_{(j)}$ ($|W_{(j)}| = |X_{(j)}| = n_{(j)}$). Les composantes $W_{(j),i}$ de ce vecteur représentent les poids des l'entrées $X_{(j),i}$ (voir figure 2.1). $B_{(j)}$ est un biais qui permet d'améliorer l'apprentissage dans les problèmes complexes en ajustant la fonction d'activation φ.

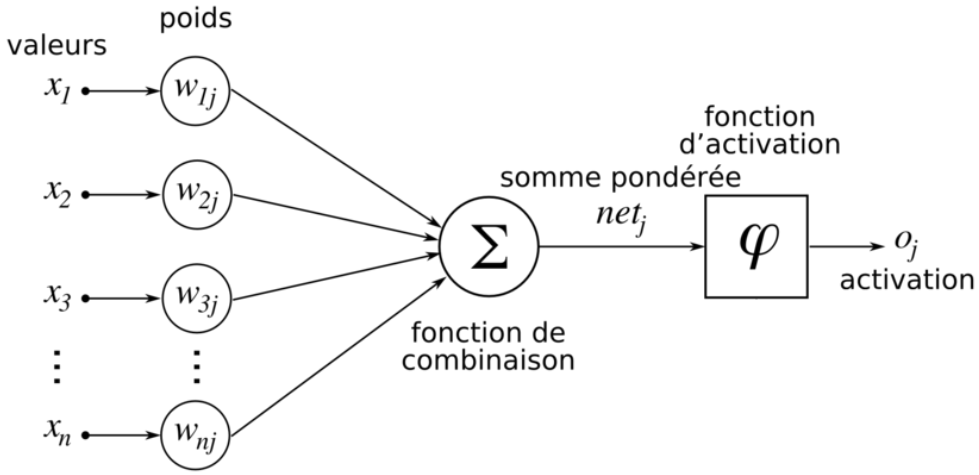

FIGURE 2.1 – Principe de fonctionnement d'un neurone artificiel j [23]

2.4 Définition d'un réseau neurones multi-couches

Un ensemble de N neurones montés de façon parallèle constitue une couche à N neurones. Dans une couche donnée les neurones ont une même fonction d'activation. Le montage en série d'une ou de plusieurs couches de même nombre ou de nombres différents de neurones permet de former un réseau neurones multi-couches (MLP :*Multi-Layer Perceptron* ,voir figure 2.2). Un neurone d'une couche donnée est relié à tous les neurones de la couche suivante. Par conséquent, le résultat d'un neurone donné est transmis à tous les neurones de la couche suivante et cela successivement jusqu'à la couche de sortie. La couche de sortie retourne le résultat du problème en question sous forme d'un vecteur de dimension égale au nombre de neurones de cette couche [22] [24].

2.5 Fonction d'activation

2.5.1 Définition et rôle

Une fonction d'activation est une fonction mathématique. Elle agit directement sur la somme pondérée calculée au sein du neurone et affecte sa sortie. Elle permet de différencier si un neurone ne va pas s'activer ($o_{(j)} = 0$), est totalement activé ($o_{(j)} = X_{(j)}.W_{(j)} + B_{(j)}$), ou est partiellement activé ($o = \alpha(X_{(j)}.W_{(j)} + B_{(j)})$ avec $0 < |\alpha| < 1$). L'activation partielle d'un neurone introduit une certaine forme de non-linéarité au réseau de neurones. Cette non-linéarité est mieux adaptée pour traiter des problèmes où la relation entre l'entrée et la sortie n'est pas linéaire [22] [24].

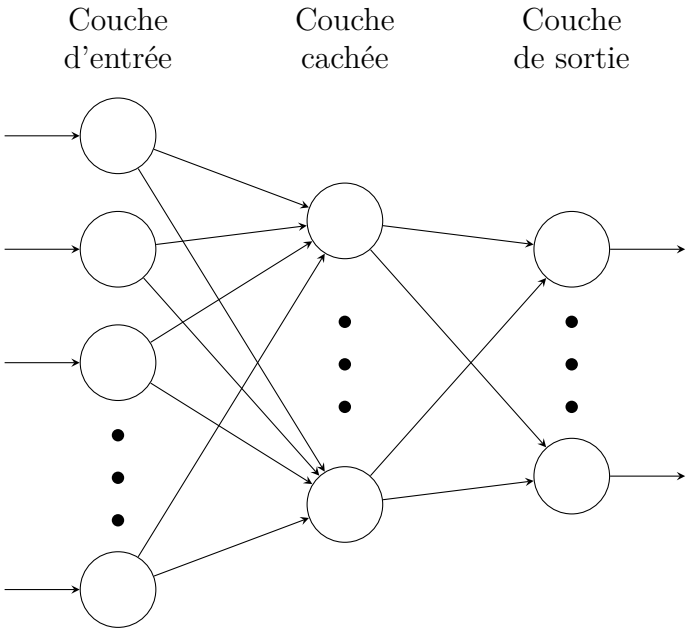

FIGURE 2.2 – Schématisation d'un réseau de neurones à 3 couches

2.5.2 Fonctions d'activation les plus utilisées

1. **La fonction linéaire**

$$\varphi(net_j) = net_j$$

C'est une fonction très simple. Néanmoins elle n'est pas très performante. Son inconvénient est qu'elle ne s'adapte pas pour la traitement de la non-linéarité qui existe entre l'entrée et la sortie du réseau. Par ailleurs, pour un réseau de neurones dédié à la prédiction, elle convient au niveau de la couche de sortie qui correspond à une étape linéaire [25].

2. **La fonction unité de rectification linéaire (appelée RELU)**

$$\varphi(net_j) = max(0, net_j) = \begin{cases} net_j & \text{si } net_j \geq 0 \\ 0 & \text{sinon} \end{cases}$$

C'est une fonction très simple et qui a montré de bonnes performances quand on l'utilise dans les couches cachées des réseaux de neurones. Son inconvénient est qu'elle retourne la valeur 0 lorsque net_j est inférieur à 0. Ceci peut fausser les résultats. On l'utilise alors juste lorsqu'on est sûr que les valeurs de net_j sont positives [26].

3. **La fonction unité de rectification linéaire paramétrique avec paramètre fixe (appelée LeakyRELU)**

$$\varphi(net_j) = \begin{cases} net_j & \text{si } net_j \geq 0 \\ \alpha\ net_j & \text{sinon.} \quad \text{avec } 0 < \alpha < 1 \end{cases}$$

C'est une fonction d'activation basée sur la fonction RELU. Mais, au lieu de retourner la valeur 0 pour toutes les valeurs négatives de net_j elle retourne la valeur $\alpha\ net_j$ où α a une valeur fixe entre 0 et 1 ($0 < \alpha < 1$). Par conséquent, ses résultats sont meilleurs que celle de la fonction RELU dans les problèmes où il y a des valeurs négatives de net_j qui traversent le réseau [26][27]. Son inconvénient est que la valeur de α doit être fixée avant la phase de l'apprentissage. Mais, on ne sait pas toujours quelle valeur optimale choisir.

4. **La fonction unité de rectification linéaire paramétrique (appelée PRELU)**

$$\varphi(net_j) = \begin{cases} net_j & \text{si } net_j \geq 0 \\ \alpha\ net_j & \text{sinon.} \qquad \text{avec } 0 < \alpha < 1 \end{cases}$$

C'est une fonction d'activation basée sur la fonction LeakyRELU. Mais, au lieu que le paramètre α soit fixé avant l'apprentissage il est considéré comme un des poids qu'on met à jour lors de la phase de l'apprentissage [25].

5. **La fonction softmax**

La fonction softmax est utilisée dans les couches de sorties des réseaux de neurones dédiés à la classification. Elle retourne un vecteur contenant la probabilité de véracité de chaque classe [25] [24].

Lorsque cette fonction est employée, le comportement de la couche de sortie change. Au lieu que les neurones de cette couche retournent leurs résultats indépendamment des autres neurones, le résultat de la somme pondérée effectuée par chaque neurones de la couche de sortie est mis dans un vecteur qu'on note $\vec{a}$. Puis, on applique la fonction f dite softmax à chaque composante j du vecteur $\vec{a}$. Ceci retourne un vecteur $\vec{b}$ de la même dimension que le vecteur $\vec{a}$. Ce vecteur $\vec{b}$ contient les probabilités des classes de chaque neurone. Et la somme de ses composantes est égal à 1 [25].

$$b_j = f(a_j) = \frac{e^{a_j}}{\sum_i e^{a_i}}, \qquad \sum_i b_i = 1$$

6. **La fonction sigmoïde**

$$\varphi(net_j) = \frac{1}{1 + e^{-net_j}}$$

C'est une fonction d'activation qui retourne une valeur entre 0 et 1. L'inconvénient de cette fonction est que pour des valeurs très grandes ou très petites de net_j elle retourne une valeur qui se rapproche de 0 (pour les valeur très petites) ou 1 (pour les valeurs très grandes) ce qui fait que sa dérivée est presque 0. Ceci affecte négativement le gradient qui lui aussi va se rapprocher du 0 et le réseau de neurones perd sa capacité d'apprendre [25].

2.6 Apprentissage

La capacité d'apprendre est le concept clé des réseaux de neurones artificiels. Le but de cette phase est de trouver les valeurs optimales du vecteur W des poids des entrées de tous les neurones du réseau [24].

2.6.1 Récupération des données

Avant que la phase d'apprentissage ne commence, les données d'apprentissage doivent être récupérées du fichier des données d'apprentissage et misent dans des vecteurs pour qu'elles soient dans un format compréhensible par le réseau de neurones. Sachant que chaque ligne du fichier de données représente un évènement qui est constitué d'un ensemble de valeurs de caractéristiques et que ces caractéristiques sont communes entre tous les évènements du fichier. Alors, pour faire cette conversion, on isole chaque ligne du fichier et on récupère les valeurs de chaque caractéristique. Puis, on les caste en des nombres réels et on les stocke dans un vecteur en respectant l'ordre des caractéristiques. Ces vecteurs à présent constituent les données d'apprentissage.

2.6.2 Initialisation et apprentissage

Les valeurs initiales des poids et des biais du réseau sont souvent choisies de manière aléatoire. Mais l'utilisation de certaines méthodes heuristiques peut conduire à un ajustement de ces paramètres pour atteindre les valeurs optimales en un nombre minimal d'itérations lors de la phase d'apprentissage. L'apprentissage est ensuite effectué sur plusieurs itérations pour mieux ajuster ces paramètres. Ceci en faisant passer les données d'apprentissage à travers le réseau évènement par évènement, calculer l'erreur commise puis ajuster les poids des liaisons entre les neurones en vue de minimiser cette erreur. L'algorithme utiliser pour faire l'apprentissage des réseaux de neurones artificiels est **l'algorithme de rétro-propagation (*Backpropagation algorithm*)** (voir paragraphe 2.6.3) [22].

2.6.3 L'algorithme de rétro-propagation

L'algorithme de rétro-propagation est l'algorithme qui a redonné naissance aux réseaux de neurones. Il est structuré de la façon suivante [22][24] :

Soit e le nombre d'iterations

Pour $it \in \{1...e\}$:

Pour chaque $i \in \{1...n\}$ (n est le nombre d'évènements) :

1. Propagation vers l'avant :
   - Pour tout les neurones dans la couche de l'entrée on leur donnent un évènement en entrée.
   - Executer le fonctionnement normale du neurone (voir paragraphe 2.3).
   - Utiliser les résultats retournés par chaque neurones de la couche d'entrée comme entrée pour tous les neurones de première couche cachée.
   - Continuer cette propagation de couche en couche succésivement jusqu'à la couche de sortie.
2. Propagation vers l'arrière :
   - Comparer le résultat obtenu par chaque neurone avec le résultat attendu de lui et calculer l'erreur à travers une fonction d'erreur (voir paragraphe 2.7).
   - Utiliser la valeur de l'erreur pour calculer le gradient (voir paragraphe 2.6.4).
   - Utiliser le gradient pour mettre à jour les poids des liaisons entre les neurones ainsi que les biais.

fait ;

fait ;

2.6.4 Calcul du gradient

Soit φ une fonction d'activation, X_i est l'entrée i du j^{eme} neurone (voir figure 2.1). E_j est l'erreur obtenu au niveau de la sortie du neurone j. Elle est donnée par la fonction d'erreur suivante [22][24] :

$$E_j = \frac{1}{2}(o_j - Y_j)^2$$

où Y_j est la valeur que le neurone est censé retourner et o_j est la valeur retournée par le neurone j.

Pour calculer l'effet du poids $W_{(j),i}$ de l'entrée X_i dans le neurone j sur l'erreur E_j, on utilise la formule suivante :

$$\frac{\partial E_j}{\partial W_{(j),i}} = \frac{\partial E_j}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial W_{(j),i}} \tag{2.1}$$

Calculons chaque terme :

$$\left(\frac{\partial E}{\partial o}\right)_j = \frac{\partial E_j}{\partial o_j} = o_j - Y_j$$

$$\left(\frac{\partial o}{\partial net}\right)_j = \frac{\partial o_j}{\partial net_j} = \varphi'(net_j)$$

$$\left(\frac{\partial net}{\partial W_i}\right)_j = \frac{\partial net_j}{\partial W_{(j),i}} = X_{(j),i}$$

On obtient alors :

$$\frac{\partial E_j}{\partial W_{(j),i}} = (o_j - Y_j)\varphi'(net_j)X_{(j),i}$$

Donc la mise à jours du poids $W_{(j),i}$ se fait de la manière suivante :

$$W_{(j),i} = W_{(j),i} - \eta\frac{\partial E_j}{\partial W_{(j),i}} \tag{2.2}$$

où η est une constante dite taux d'apprentissage. Si la mise à jour des poids se fait de façon directe sans l'utilisation du taux d'apprentissage le réseau de neurones serai instable. Le signe moins $(-)$ est mis avant le taux d'apprentissage η car on doit mettre à jour les poids dans le sense contraire de la dérivée pour pouvoir atteindre le minimum de la fonction d'erreur E_j [22].

La formule 2.2 s'applique à tous les neurones de la couche de sortie. Mais pour les neurones des autres couche le terme $\frac{\partial E_j}{\partial o_j}$ de l'equation 2.1 prend la forme suivante [22] :

$$\left(\frac{\partial E}{\partial o}\right)_j = \frac{\partial E_j}{\partial o_j} = \sum_{k=1}^{t}\left(\frac{\partial E}{\partial net}\right)_k W_{(k),j}$$

ou t est le nombre de neurones de la couche suivante.
Donc, on a :

$$\boxed{W_{(j),i} = W_{(j),i} - \eta\frac{\partial E_j}{\partial W_{(j),i}} = W_{(j),i} - \eta\frac{\partial E_j}{\partial o_j}\frac{\partial o_j}{\partial net_j}\frac{\partial net_j}{\partial W_{(j),i}} = W_{(j),i} - \eta\frac{\partial E_j}{\partial o_j}\varphi'(net_j)X_{(j),i}}$$

où

$$\frac{\partial E_j}{\partial o_j} = \begin{cases}(o_j - Y_j) & \text{si le neurone appartient à la couche de sortie}\\ \sum_{k=1}^{t}\left(\frac{\partial E}{\partial net}\right)_k W_{(k),j} & \text{sinon}\end{cases}$$

et le gradient ∇W_i est le vecteur dont les éléments sont les différents $\frac{\partial E_j}{\partial W_{i,j}}$ [22].

2.7 Fonctions d'erreur

2.7.1 Rôle

Durant la phase d'apprentissage, le réseau de neurones apprend la relation entre les données en entrées et le résultat de sortie. Pour mesurer l'erreur commise par le réseau lors de cette phase on utilise des fonctions mathématiques appelées **fonction d'erreur**. Le but de la phase de l'apprentissage est de mettre à jour les poids et les biais du réseau à fin de minimiser cette erreur et augmenter la précision du réseau de neurones [24].

2.7.2 Fonctions d'erreur les plus utilisées

Il existe plusieurs fonctions d'erreurs, chacune d'entre elles est utilisée pour différentes fins. Parmi les fonctions d'erreur les plus utilisées nous présentons :

1. **La fonction d'erreur quadratique moyenne (appelée *Mean Squared Error MSE*)**
   Elle a comme formule [28][24] :

$$\frac{1}{n}\sum_{i=1}^{n}(o_{(j),i} - Y_{(j),i})^2$$

   C'est une fonction d'erreur non linéaire est très utilisée dans les réseaux de neurones artificiels dédié à la prédiction. Elle permet d'apprendre des relations complexes entre les donnée en entrée et la sortie en pénalisant les valeurs les plus fausses plus que les valeurs moins fausses [28].
2. **La fonction d'erreur absolue moyenne (appelée *Mean Absolute Error MAE*)**
   Elle a comme formule [28][24] :

$$\frac{1}{n}\sum_{i=1}^{n}|o_{(j),i} - Y_{(j),i}|$$

   C'est une fonction d'erreur linéaire dédiée au problèmes de prédiction mais elle est peu utilisée vu ses limitations face aux problèmes complexes. Elle permet d'apprendre des relations relativement simple entre les données en entrée et la sortie [28].
3. **La fonction entropie croisée catégorique (appelée *Categorical Cross Entropy*)**
   Cette fonction qu'on note $D(S, L)$ est utilisée pour les problèmes de classification quand l'étiquette est sous forme d'un vecteur. Sa formule est la suivante [29] :

$$D(S, L) = \sum_{i=0}^{m} L_i * log(S_i)$$

   où S est le vecteur des probabilités prédites par le réseau de neurones pour chaque classe, et L est le vecteur correspondant à la probabilité 1 dans la position de la classe voulue et 0 ailleurs [29].
4. **La fonction entropie croisée catégorique clairsemée (appelée *Sparse Categorical Cross Entropy*)**
   Cette fonction est basée sur la fonction de l'entropie croisée catégorique. La difference entre ces deux fonctions est dans l'implementation. Au lieu que le L soit un vecteur, c'est une valeur entière qui représente la classe que le réseau de neurones doit prédire. Quand on a plusieurs classes un vecteur peut prendre beaucoup d'espace mémoire à l'opposé d'un entier non. Ceci est l'avantage principale de cette fonction [29] [24].

2.8 Algorithmes d'optimisation

Le but des algorithmes d'optimisation dans le contexte de l'apprentissage des réseaux de neurones est de minimiser l'erreur E en mettant à jour les poids des liaisons ainsi que les biais. Parmi les algorithmes d'optimisation les plus utilisés, on cite :

2.8.1 Gradient Descent

C'est le plus basique les algorithmes d'optimisation. Il fonctionne comme décrit dans l'algorithme de rétro-propagation (voir paragraphe 2.6.3) sauf qu'il ne met pas à jour les poids pour chaque entrée X_i. Mais il calcule la moyenne de l'erreur de tous les entrées. Puis il met à jour les poids et les biais la fin d'une itération. Ceci coûte très cher en temps car on fait passer tous nos données à travers le réseau de neurones pour mettre à jour nos poids et nos biais une seule fois. Ceci peut prendre beaucoup de temps pour atteindre un taux d'erreur acceptable [30] [31].

2.8.2 Mini-bach Stochastic Gradient Descent avec élan

Cet algorithme est basé sur l'algorithme Gradient Descent. Il permet d'accélérer l'apprentissage en utilisant la formule suivante pour la mise à jour des poids.

$$v_t = \beta v_{t-1} + (1-\beta)\frac{\partial E}{\partial W_{i,j}}$$

$$W_{i,j} = W_{i,j} - \eta v_t$$

où β est le terme de l'élan qui a une valeur qui est dans les alentour de 0.9 et $v_0 = 0$.

Cette formule permet de donner une certaine vitesse à la minimisation de l'erreur. Aussi, la mise à jour des poids ne ce fait pas à la fin de chaque itération mais après un certain nombre défini au préalable (nommé "*batch size*"). Ceci permet de gagner du temps ainsi qu'avoir une bonne précision [30] [31].

2.8.3 Adaptive Moment Estimation (appelé Adam)

Cette algorithme est une combinaison de l'algortihme RMSprop [1] ainsi que l'algortihme mini-batch Stochastic Gradient Descent avec élan. Il permet d'avoir un taux d'apprentissage s_t variant et adapté à chaque poids ainsi qu'un élan v_t pour accélérer l'apprentissage [30] [31] :

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)\frac{\partial E}{\partial W_{i,j}}$$

$$s_t = \beta_2 s_{t-1} + (1-\beta_2)\left(\frac{\partial E}{\partial W_{i,j}}\right)^2$$

Mais, s_t et v_t ne sont pas utilisés directement, on doit les normaliser d'abord pour que l'algorithme donne de bons résultats dès les premières itérations. La normalisation se fait de la manière suivante [30] :

$$\hat{v}_t = \frac{v_t}{(1-(\beta_1)^t)}$$

$$\hat{s}_t = \frac{s_t}{(1-(\beta_2)^t)}$$

Puis, on met à jour les poids de la manière suivante :

$$W_{i,j} = W_{i,j} - \frac{\eta \hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

ϵ qui est de l'ordre de $1e-8$. Son utilité est d'évité la division par 0.

1. L'algorithme RMSprop permet d'accorder un taux d'apprentissage unique à chaque poids. Ce taux d'apprentissage décrémante d'une façon bien spécifique quant on s'approche du minimum de la fonction d'erreur

2.9 Réseaux de neurones convolutionnels

Les réseaux de neurones convolutionnels (CNN : *Convolutional Neural Networks*) ont connu un grand succès dans le domaine de l'imagerie. Ils permettent d'exploiter la corrélation existante entre les pixels d'une image donnée en vue de détecter les caractéristiques essentielles permettant la résolution du problème en question. Le CNN est une combinaison d'un réseau de neurones multi-couches (MLP) dit partie Dense du CNN avec une ou plusieurs couches dites couche(s) de convolution (voir paragraphe 2.9.1) supplémentaires placées avant l'ensemble des couches dense [32] [24].

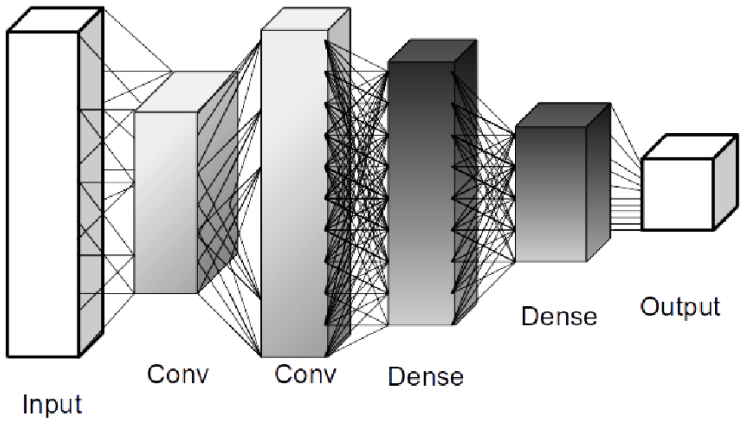

FIGURE 2.3 – Schéma d'un réseau de neurones convolutionel [33]

2.9.1 Couche de convolution

Définition

La couche de convolution permet d'extraire les caractéristiques pertinentes présentent dans les données d'entrée. Ceci est en faisant passer un ou plusieurs filtres sur les données d'entrée. Un filtre est un vecteur contenant des poids $W_{i,j}^{(F)} \equiv F[i][j]$. Ces poids ont initialement des valeurs aléatoires. Puis, ces valeurs sont réajustées lors de la phase d'apprentissage en utilisant un des algorithmes d'optimisation (voir paragraphe 2.8). Un filtre est à une dimension (1D) si les données d'entrées sont un vecteur à une dimension. Il est à deux dimension (2D) si les données d'entrées sont à deux ou plusieurs dimensions. Il a une longueur et une largeur fixes et définis avant la phase de l'apprentissage (dite *kernal size*). Ces dernières doivent impérativement être inférieures à la longueur et la largeur des données en entrée [24].

Opération de convolution

La phase de convolution se déroule de la façon suivante [32] [24] :

Prenant une matrice en entrée I qui est à 2 dimensions $(L \times L)$ et un filtre F aussi à 2 dimensions $(l \times l)$ avec $l < L$.

1. On pose le filtre sur le coin haut gauche de la matrice I telle que tous les poids du filtre F ont une valeur correspondante dans cette matrice.
2. On fait la somme de la multiplication des poids du filtre par leur valeurs correspondantes dans la matrice d'entrée I. On applique une fonction d'activation sur le résultat. Puis, on le sauvegarde dans la première case de la matrice de sortie S (la case $S[0][0]$, voir figure 2.4).

3. On déplace le filtre de s cases à droite (s nombre défini avant la phase d'apprentissage appeler ***stride***) et on refait la même opération en stockant le résultat dans la case $S[0][1]$. Et on continue de faire cette opération en déplaçant le filtre à chaque fois horizontalement vers la droite jusqu'à ce qu'il y ait au moins un poids de la matrice F qui n'a pas de valeur correspondante dans la matrice I.

   Par exemple, la valeur 4 surlignée en vert dans la figure 2.4 correspondant à la position $S[0][3]$ et est calculée de la manière suivante (en supposant une fonction d'activation linéaire) :

$$\begin{aligned} S[0][3] &= F[0][0] \times I[0][3] + F[0][1] \times I[0][4] + F[0][2] \times I[0][5] \\ &\quad + F[1][0] \times I[1][3] + F[1][1] \times I[1][4] + F[1][2] \times I[1][5] \\ &\quad + F[2][0] \times I[2][3] + F[2][1] \times I[2][4] + F[2][2] \times I[2][5] \\ &= 1 \times 1 + 0 \times 0 + 1 \times 0 + 0 \times 1 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 1 + 1 \times 1 \\ &= 4 \end{aligned}$$

4. On descend ensuite de s cases et on revient au côté le plus à gauche de la matrice I. Et on refait les étapes 2 et 3 en sauvegardant le résultat sur la ligne suivante de la matrice S.
5. On refait l'étape 4 jusqu'à ce qu'on parcourt toute la matrice I.

La matrice S ainsi trouvée constitue l'entrée de la couche suivante du CNN.

Remarque :

On remarque bien que la dimension de la matrice S est inférieure à celle de la matrice I. On peut ajouter une marge d'épaisseur p (nommé ***padding***) tout autour de la matrice I ne contenant que la valeur 0 dans chacune de ses cases. Cela est pour que la matrice S ait la même dimension que la matrice I après l'opération de convolution (voir figure 2.5). L'activation du paramètre *padding* se fait en choisissant soit la valeur "*valid*" qui correspond à pas de marge, soit la valeur "*same*" qui correspond à avec marge [24].

La dimension de la matrice S est donnée par la formule suivante :

$$(\frac{L - l + 2p}{s} + 1 \times \frac{L - l + 2p}{s} + 1)$$

Par exemple, pour le cas de la figure 2.4, on a $L = 7$, $l = 3$, $s = 1$ et $p = 0$ (idem "valid"). Ceci donne :

$$dim(S) = \frac{L - l + 2p}{s} + 1 = \frac{7 - 3 + 2 \times 0}{1} + 1 = 5$$

Si, par ailleurs, on prend $p = 1$ (idem "same") alors

$$dim(S) = \frac{L - l + 2p}{s} + 1 = \frac{7 - 3 + 2 \times 1}{1} + 1 = 7 \equiv dim(I) = L$$

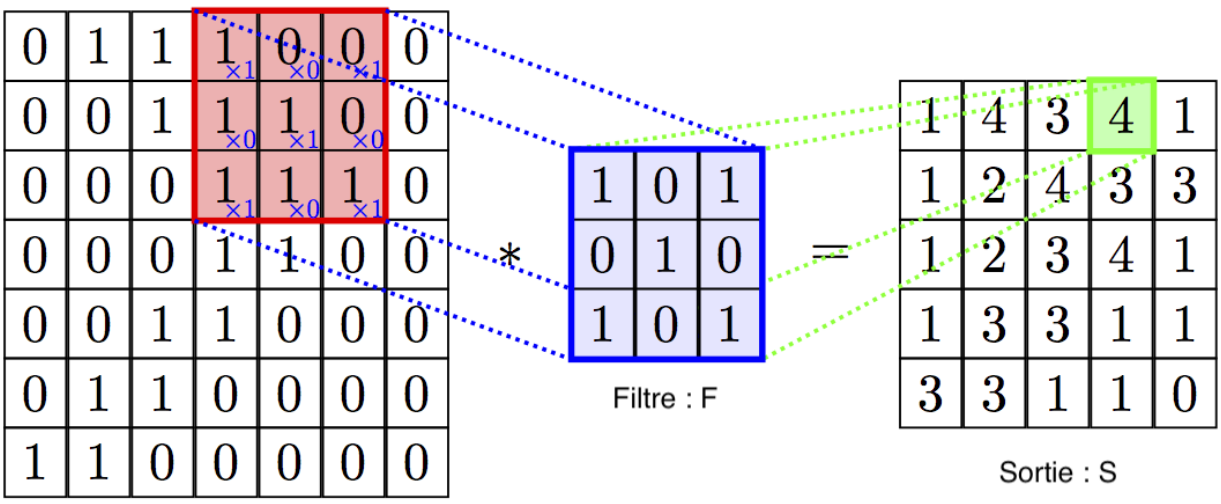

FIGURE 2.4 – Opération de convolution [34].

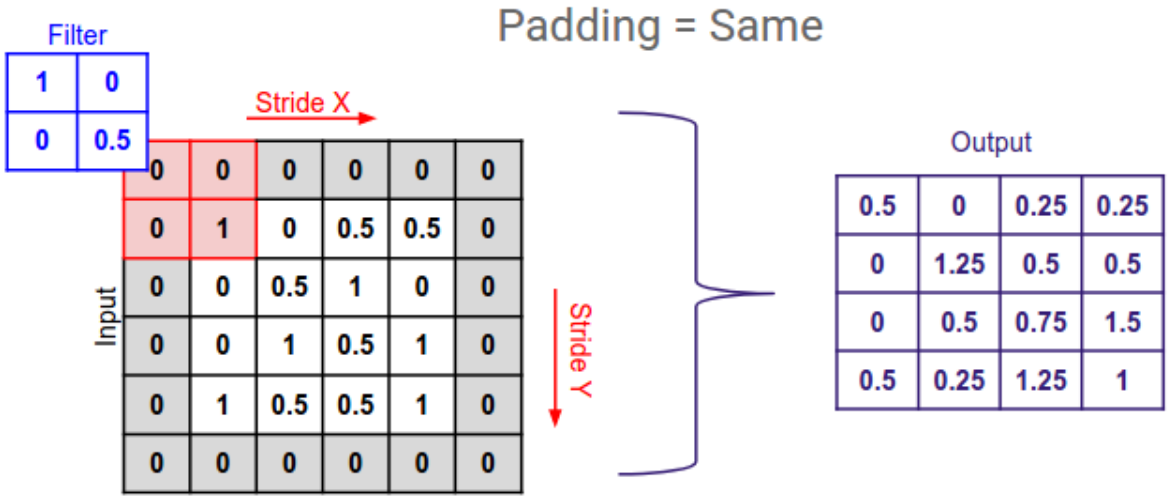

FIGURE 2.5 – Illustration de l'application du padding [35].

2.9.2 Calcul du gradient dans les couches de convolution

L'apprentissage des CNNs se fait aussi à l'aide de l'algorithme de rétro-propagation ou un de ses variants. Donc, l'étape du calcul du gradient est une étape essentielle. Le calcul du gradient dans la partie dense des CNNs se fait de la même façon décrite précédemment (voir paragraphe 2.6.4). Par contre, le calcul du gradient dans les couches de convolution se fait de la manière suivante [36] [24] :

$$\frac{\partial \tilde{E}}{\partial W^{(F)}} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{\partial \tilde{E}}{\partial S_{i,j}} \frac{\partial S_{i,j}}{\partial W^{(F)}} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{\partial \tilde{E}}{\partial S_{i,j}} X_{i,j}$$

où $\tilde{E}$ est l'erreur sur la sortie S, calculée par une des fonction d'erreur précédente (voir 2.7). S est la matrice de sortie de la couche dont on est en train de calculer le gradient de ses poids. $W^{(F)}$ sont les poids de la matrice filtre F de la couche de convolution et $X_{i,j}$ sont les éléments de la matrice d'entrée I.

2.10 Le sur-apprentissage

2.10.1 Définition

Le sur-apprentissage est le fait que les paramètres d'un modèle donné (réseau de neurones ou autre) s'ajuste trop sur les données d'apprentissage ceci donne l'illusion lors de la phase d'apprentissage que la performance du modèle est bonne. Mais, lors du test de ce modèle, on remarque que sa performance n'est pas aussi bonne qu'on le croyait. On dis alors que **le modèle a perdu sa capacité de généralisation** (voir figure 2.6) [31][24].

2.10.2 Origine

Le sur-apprentissage, dans le context des réseaux de neurones, peut avoir plusieurs origines. Parmi eux, on cite [31] [24] :

— Le réseau de neurones est trop profond.
— La phase d'apprentissage a trop duré (beaucoup itérations).
— L'ensemble de données d'apprentissage est trop restreint.
— La présence de beaucoup de caractéristiques corrélées.

2.10.3 Identification

On peut repérer le sur-apprentissage lors de la phase d'apprentissage, en utilisant 15% à 20% des données de l'apprentissage comme données de validation. Le modèle ne fait pas l'apprentissage sur ces données de validation. Elles sont juste utilisées pour tester sa performance après chaque itération. Lorsqu'on remarque que l'erreur pour les données de validation est trop supérieur à l'erreur des données de l'apprentissage et que cette difference continue à augmenter alors on peut trancher qu'on a du sur-apprentissage [31] [24].

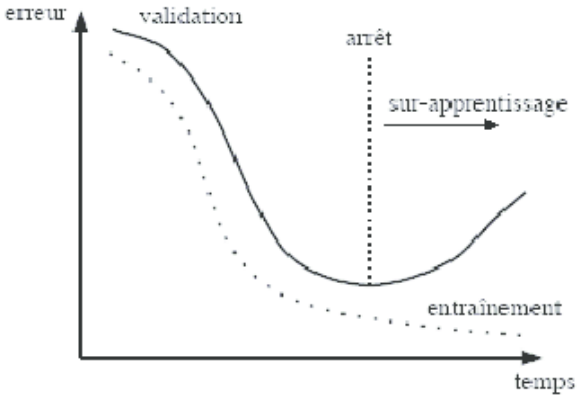

FIGURE 2.6 – Phénomène du sur-apprentissage [37]

2.10.4 Remède

Pour éviter de tomber dans le sur-apprentissage, il y a plusieurs techniques. Parmi eux, on cite [31] :

— Éliminer les caractéristiques corrélées.

— Utiliser la technique du dropout : Qui est le fait de désactiver un certain pourcentage des neurones dans une couche donnée à chaque itérations.
— Réduire la profondeur du réseau de neurones : enlever quelques couches du réseau de neurones.
— Réduire le nombre d'itérations de l'apprentissage.
— Si on travaille avec des images, on applique les algorithmes d'augmentation des données qui permet d'augmenter le nombre de données d'apprentissage.

2.11 Prédiction ou classification après la phase d'apprentissage

Après que la phase d'apprentissage d'un réseau de neurones soit achevée, on a en notre possession, les poids et les biais optimaux de réseau de neurones dédiée à la résolution du problème en question, que ce soit un problème de classification ou de prédiction. Alors, pour faire la prédiction ou la classification de nouvelles données, on les transforme en un vecteur. Puis, on les propage à travers le réseau de neurones en utilisant les paramètres obtenus après l'apprentissage. Chaque résultat retourné par la couche de sortie est la prédiction/classification correspondante à la donnée en entrée.

2.12 Adaptive Boosting

2.12.1 Définition

L'idée derrière l'algorithme Adaptive Boosting (Adaboost) est d'utiliser plusieurs estimateurs (MLPs ou CNNs ou autres), combiner leurs résultats à travers un vote pondéré pour avoir un résultat qui est le plus juste possible [38].

2.12.2 Apprentissage

Les estimateurs de l'algorithmes Adaboost (MLPs ou CNNs ou autres) sont identiques en terme d'architecture et ils font l'apprentissage sur le même ensemble de données. Sauf que leur apprentissage doit se faire de façon séquentielle. Car après qu'un estimateur a fini l'étape de l'apprentissage l'ensemble des estimateurs précédents ainsi que l'estimateur courant sont évalués. Un poids est affecté à l'estimateur courant par rapport à l'erreur qu'il a commis. Puis, les poids des évènements où cet ensemble n'a pas donné de bon résultats sont augmentés pour que l'estimateur qui va faire l'apprentissage dans l'itération suivante se focalise plus sur ces évènements. Ceci permet d'améliorer la précision totale de l'ensemble (voir figure 2.7).

L'inconvénient de l'algorithme d'Adaboost est qu'il est seulement utilisé pour les problèmes de classification en deux classes. De plus, il ne minimise pas l'erreur assez rapidement par rapport à l'algorithme SAMME.R (voir paragraphe 2.12.3) [39].

De façon formelle, d'algorithme Adaboost est le suivant [38][39] :

1. Donner un poids de 1/ n où n est le nombre d'évènements.

$$w_i = 1/n$$

2. Pour $m = 1...t$ (ou t est le nombre totale d'estimateurs prédéfinis) :
   (a) Faire l'apprentissage du $m^{\text{ème}}$ estimateurs E_m.

(b) Calculer l'erreur :

$$err_m = \sum_{i=1}^{n} w_i.\mathbb{I}(Y_i \neq E_m(X_i))/\sum_{i=1}^{n} w_i$$

où $\mathbb{I}$ est une fonction booléenne qui retourne 1 si la condition entre parenthèses est vrai, 0 sinon, et Y_i est la classe de l'évènement X_i. $E_m(X_i)$ retourne la classe prédite de l'évènement X_i par l'estimateur E_m.

(c) Calculer le poids de l'estimateur E_m qu'on note α_m :

$$\alpha_m = log(\frac{1 - err_m}{err_m})$$

(d) Mettre à jour les poids par la formule suivante :

$$w_i = w_i.exp(\alpha_m.\mathbb{I}(C(X_i) \neq E_m(X_i)) \ , \ i = 1...n$$

(e) Normaliser les poids des évènements pour que leur somme soit égal à 1.

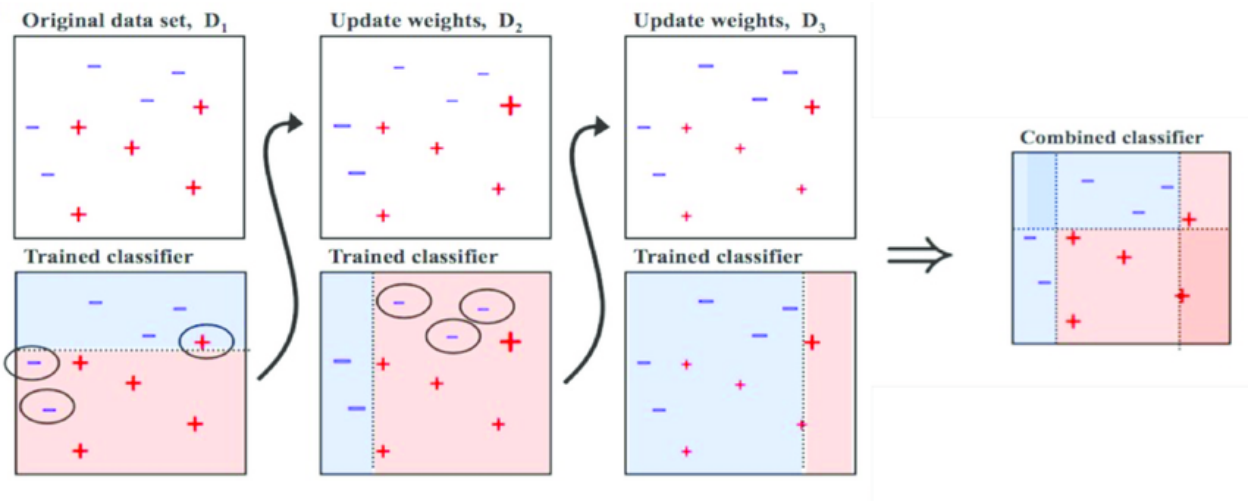

FIGURE 2.7 – Schéma du fonctionnement de l'algorithme Adaboost [40]

2.12.3 Stagewise Additive Modeling using a Multi-class Exponential loss function.Real

L'algorithme Stagewise Additive Modeling using a Multi-class Exponential loss function.Real (SAMME.R) est basé sur l'algorithme Adaboost. Il permet de faire une classification multi-classes. De plus, il est plus performant qu'Adaboost en termes de rapidité et de minimisation de l'erreur. Il fonctionne de la manière suivante [39] :

1. Donner un poids de 1/ n où n est le nombre d'évènements.

$$w_i = 1/n$$

2. Pour $m = 1...t$ (ou t est le nombre totale d'estimateur prédéfini) :
   (a) Faire l'apprentissage du $m^{ème}$ estimateurs E_m.

(b) Calculer :

$$P_k^m(X) = Prob_w(c = k|X), k = 1...K$$

Où K est le nombre de classes. $Prob_w(c = k|X)$ est la probabilité pondérée que l'évènement soit de la classe k. Elle est calculée de la manière suivante :

$$Prob_w(c = k|X) = E(exp(-\frac{1}{K}.Y.H^{m-1}(X)).\mathbb{I}(c(X) = k)|X)$$

Où $H^{m-1}(X)$ est l'ensemble des estimateurs précédents qui nous retournent les classes prédites pour les évènements X. Y sont les classes correctes des évènements X.

(c) Calculer le poids de l'estimateur E_m :

$$h_k^m(X) = (K-1)(logP_k^m(X) - \frac{1}{K}\sum_{k'} logP_k'^m(X)) \ , \ k = 1...K$$

(d) Mettre à jour les poids des évènements par la formule suivante :

$$w_i = w_i.exp(-\frac{K-1}{K}Y_i logP^m(X_i)) \ , \ i = 1...n$$

Où Y_i est la classe de l'évènement X_i.

(e) Normaliser les poids des évènements pour que leur somme soit égal à 1.

2.13 Bootstrap Aggregating

Bootstrap Aggregating (Bagging) est un algorithme qui permet de faire l'apprentissage de n estimateurs (MLPs ou CNNs ou autres) identiques en termes d'architecture, sur un sous ensemble de données différents issue de l'ensemble des données originales. Lors de la prédiction, on prend la moyenne des résultats de tous les n estimateurs. Cet algorithme permet d'améliorer les précisions vu qu'on a le vote de plusieurs modèles à la place d'un seul (voir figure 2.8) [41].

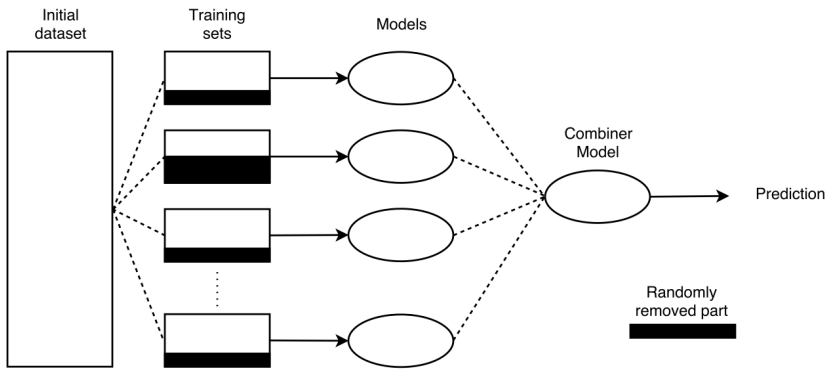

FIGURE 2.8 – Schema du fonctionnement de l'algorithme Bootstrap aggregating [42]

2.14 Conclusion

Dans ce chapitre, nous avons définit les neurones, les réseaux de neurones multi-couches et convolutionnels. Nous avons présenté l'algorithme de rétro-propagation et ses variants. De

plus, nous avons définit les différents paramètres, les différentes fonctions (fonction d'activation et fonction d'erreur) qui permettent d'ajuster l'apprentissage et d'améliorer la précision. Enfin, nous avons introduit les algorithmes Adaboost, SAMME.R et Bagging qui permettent de booster la precision. Ces outils nous seront d'une grande utilité dans les chapitres suivants.

Chapitre 3

Architectures proposée des réseaux de neurones artificiels utilisés

3.1 Introduction

Un réseau de neurones performant doit avoir une structure adéquate et des paramètres bien définis qui sont dédiés à la résolution du problème en question. Pour cela, dans ce chapitre nous allons s'intéresser aux différents paramètres régis et non régis par des règles entrant dans la structure des réseaux de neurones multi-couches et convolutionnels. Ceci, en vue de bien structurer nos réseaux de neurones proposés et maximiser leurs performance dans la détection du signal du Higgs.

3.2 Architecture proposée du réseau de neurones multi-couches utilisé

3.2.1 Proposition

L'architecture du réseau de neurones multi-couches que nous proposons pour adresser notre problématique et atteindre nos objectifs est la suivante :

— Couche 1 : nombre de neurones est égal au nombre de caractéristiques en entrée, Batch Normalization, fonction d'activation : PRelu, Dropout = *Drp*.

— n couches cachées (intermédiaires). Chacune possède un nombre de neurones bien défini *NbN*, Batch Normalization, fonction d'activation : PRelu, Dropout = *Drp*.

— Couche de sortie : 2 neurones, fonction d'activation : Softmax.

3.2.2 Choix des paramètres obéissant à des règles

Les paramètres suivants sont régis par des règles ou bien ont un domaine de valeur limité. Alors, nous avons choisi la valeur qui est la plus performante comme suit :

1. **Couche d'entrée et couche de sortie :**
   D'après le livre de Jeff Heaton - *Introduction to Neural Networks for Java* - [43] la première couche (couche d'entrée) comporte un nombre de neurones égal au nombre de caractéristiques en entrées et la couche de sortie comporte un nombre de neurones égal au nombre de classes à prédire dans les problèmes de classification qui est notre cas [44].

2. **Fonction d'activation :**
   La fonction d'activation PRelu a été utilisé dans toutes les couches sauf la couche de sortie. Car c'est la fonction d'activation la plus performante dans les couches cachées (voir paragraphe 2.5.2.4). Dans la couche de sortie, nous avons utilisé la fonction d'activation Softmax car c'est la fonction d'activation dédié aux problèmes de classification et qui retourne dans chaque neurone la probabilité que la classe qui lui est assignée est vrai (voir paragraphe 2.5.2.5).
3. **Batch normalization :**
   Batch Normalization a été utilisé en vue de normaliser la somme pondérée dans chaque neurone avant l'application de la fonction d'activation. Cela permet de rétrécir la courbe d'erreur et d'atteindre le minimum de la fonction d'erreur plus rapidement (voir figure 3.1) [31]. La normalisation de la somme pondérée se fait à l'aide de la formule suivante :

$$(data - mean)/std$$

   ou *data* est la somme pondérée qu'on veux normaliser, *mean* et *std* sont respectivement la moyenne et l'écart type de toutes les sommes pondérées de la couche du neurone dont on est en train de normaliser sa somme pondérée.

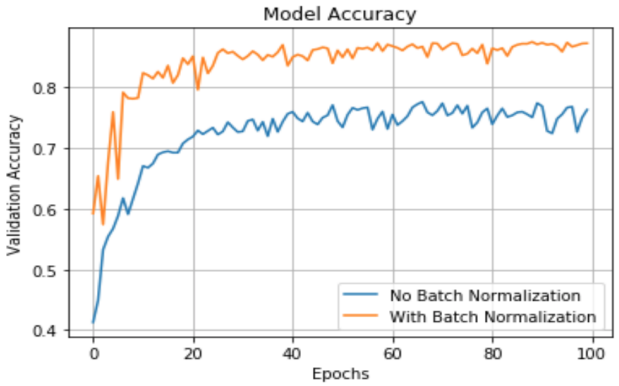

FIGURE 3.1 – Courbe de précision montrant l'effet de Batch Normalization [45]

3.2.3 Choix des paramètres non régis par des règles

Les valeurs des paramètres suivants seront fixés en fonction de la valeur qui donne les meilleurs résultats par rapport au critère de performance, ainsi qu'au matériel disponible pouvant supporté ces valeurs.

1. **Nombre de couches cachées (n) :**
   Partant du fait que les neurones sont les briques fondamentales qui contribuent à l'amélioration de la performance du réseau de neurones et que la majorité de ces neurones résident dans les couches cachées, alors le choix du nombre "n" de ces couches est un paramètre important qui est à optimiser de sorte que le réseau qui en découle donne de bonnes performances et échappe au sur-apprentissage.
2. **Nombre de neurones dans chaque couche cachée (NbN) :**
   Les neurones sont le noyau de le capacité d'apprentissage des réseaux de neurones. Par suite, leur nombre est un paramètre très important à déterminer pour une performance optimale.

3. **Taux du dropout (*Drp*) :**
   Le dropout est le fait de désactiver un certain poucentage de neurones dans une couche donnée. Il permet d'éviter le sur-apprentissage. Sa valeur doit être ajustée méticuleusement de façon à éviter le sur-apprentissage et ne pas entraver la phase d'apprentissage [24].

3.3 Architecture proposée du réseau de neurones convolutionnel utilisé

3.3.1 Proposition

L'architecture du réseau de neurones convolutionnel que nous proposons pour adresser notre problématique et atteindre nos objectifs est la suivante :

— *Nc* couches de convolution convolution 1D, nombre de filtres = *NbF*, dimension du filtre (kernal size) = 3, stride = 1, padding = "valid", fonction d'activation : PRelu.

— Couche Flatten.

— *Nb* couches dense à *NbN* neurones, Batch Normalization, fonction d'activation : PRelu. régularisation l1 = *L1*, l2 = *L2*, Dropout=*Drp*.

— Couche de sortie : 2 neurones, fonction d'activation : Softmax.

3.3.2 Choix des paramètres obéissant à des règles

Les paramètres suivants sont régis par des règles ou bien ont un domaine de valeur limité. Alors, nous avons choisi les valeurs de ces paramètres à cause des raisons suivantes :

1. **Type des couches de convolution :**
   Notre CNN proposé contient *Nc* couches de convolution a une dimension (1D). Ceci est dû au fait que nos données d'entrée sont à une dimension (1D).
2. **Paramètres des couches de convolution :**
   Nous utilisons un décalage (stride) de $s = 1$, pas de marge et une dimension de filtre de 1×3 qui est assez petite pour extraire le plus d'informations possible sans aucune interférance [32].
3. **Couches MaxPooling :**
   Les couches MaxPooling n'ont pas été utilisé. Elles sont utilisées pour rendre l'emplacement des caractéristiques insignifiant dans le vecteur des données en entrée [32][24]. Ce qui n'est pas le cas dans notre problématique (L'ordre des caractéristiques est important).
4. **Couche Flatten :**
   La couche Flatten a été ajouté pour transformer la sortie de la dernière couche de convolution qui est sous forme d'un vecteur multi-dimensionnel en un vecteur à une seule dimension. Ceci est en vue de le faire passer dans la partie dense du CNN.
5. **Couches dense :**
   Celles-ci agissent comme couches intermédiaires entre la couche Flatten et la couche de sortie.
6. **Couche de sortie :**
   La couche de sortie contient 2 neurones : chacun retourne une probabilité. Cette dernière représente la probabilité que la classe assignée à ce neurones est vrai.

7. **Fonction d'activation :**
   La fonction d'activation PRelu a été utilisée dans toutes les couches sauf la couche de sortie. Elle a montré de bonnes performances et une très grande utilité dans les couches cachée (voir paragraphe 2.5.2.4). Dans la couche de sortie, nous avons utilisé la fonction d'activation Softmax. Cette fonction d'activation est dédiée aux problèmes de classification. Elle retourne dans chaque neurone la probabilité que la classe assignée à ce neurone est vrai (voir paragraphe 2.5.2.5).
8. **Batch normalization :**
   Batch Normalization a été utilisé pour normaliser la somme pondérée dans les neurones avant l'utilisation de la fonction d'activation. Cela permet de rétrécir le courbe d'erreur et d'atteindre le minimum de la fonction d'erreur plus rapidement (voir figure 3.1) [31]. La normalisation de la somme pondérée se fait à l'aide de la formule suivante :
   $$(data - mean)/std$$
   ou *data* est la somme pondérée qu'on veux normaliser, *mean* et *std* sont respectivement la moyenne et l'écart type de toutes les sommes pondérées de la couche du neurone dont on est en train de normaliser sa somme pondérée.

3.3.3 Choix des paramètres non régis par des règles

Les valeurs des paramètres suivants seront fixés en fonction de la valeur qui donne les meilleurs résultats par rapport au critère de performance, ainsi qu'au matériel disponible pouvant supporté ces valeurs.

1. **Nombre de couches de convolution (*Nc*) :**
   Vu que les filtres de convolution sont les briques fondamentales qui contribuent à l'amélioration de la performance du CNN et que ces filtres résident dans les couches de convolution, alors le nombre "*Nc*" des couches de convolution est un paramètre important à étudier pour atteindre de bonnes performances.
2. **Nombre filtres par couche de convolution (*NbF*) :**
   Les filtres dans les couches de convolution sont très important car ce sont eux qui permettent d'extraire les caractéristiques les plus importantes. Malheureusement, Il n'y pas de règles pour fixer leur nombre dans une couche. Sauf que le nombre de filtre doit augmenter en avançant dans le réseau. Des tests sur ce paramètres sont requis pour approcher sa valeur optimale [32].
3. **Nombre de couches dense (*Nb*) et le nombre de neurones dans chacune de ces couches (*NbN*) :**
   Les couches dense agissent comme couches intermédiaires entre les couches de convolution et la couche de sortie. Elles permettent d'exploiter les caractéristiques identifiées par les couches de convolutions et les utiliser pour traiter le problème en question. Ce sont des couches contenant des neurones qui sont le composant essentiel et le secret de la capacité d'apprentissage. La valeur de ces deux paramètres influe, de façon directe, la performance du CNN et donc ils doivent être étudié pour approcher leurs valeurs optimales.
4. **Taux du Dropout et de la régularisation L1 et L2 :**
   Le dropout est le fait de désactiver un certain nombre de neurones dans une couche donnée, La régularisation L1 et L2 et le fait d'ajouter un terme bien spécifique à la valeur de l'erreur commise par le réseau de neurones [24]. Ce sont des stratégies permettant d'éviter le sur-apprentissage. Leurs valeurs doivent être ajustées méticuleusement de façon à éviter le sur-apprentissage et ne pas entraver la phase d'apprentissage.

3.4 Paramètres d'apprentissage des réseaux de neurones

3.4.1 Choix des paramètres

Pour faire l'apprentissage de nos réseaux de neurones, nous utilisons les paramètres suivants :

— Optimiseur : Adam.

— Fonction d'erreur : entropie croisée catégorique clairsemée.

— Nombre d'itérations : *NbIt*.

— Batch size (*Taille de lot*) : *BS*.

— Partage de validation : 15%.

— Shuffle : True.

3.4.2 Explication des choix

1. **Algorithme d'apprentissage :**
   L'algorithme d'optimisation utilisé est Adam car il est très performant et il adapte le taux d'apprentissage de chaque poids du réseau de neurones automatiquement en fonction de l'erreur (voir paragraphe 2.8.3).
2. **Fonction d'erreur :**
   La fonction d'erreur utilisée est la fonction entropie croisée catégorique clairsemée car c'est la fonction d'erreur la dédiée aux problèmes de classification et elle permet d'optimiser l'utilisation de la mémoire (voir paragraphe 2.7.2.4).
3. **Partage de validation :**
   Le partage de validation (*validation split*) est fixé à 15% pour donner un aperçu fiable sur s'il y a du sur-apprentissage (voir paragraphe 2.10) ou pas lors de l'apprentissage du réseau de neurones. La valeur recommandée de ce paramètre est entre 15% et 20%. Nous avons choisis 15% qui est la borne inférieure. Ceci est pour maximiser le nombre d'évènements dédiés à l'apprentissage [22].
4. **Shuffle :**
   Le paramètre shuffle permet de mélanger les évènements à chaque itération puis d'en prendre 15% pour la validation. Cela nous évite que le réseau de neurones fasse l'apprentissage sur les même évènements et qu'il ne voit pas les évènements de la validation [22].
5. **Nombre du batch size (*BS*) :**
   Le Batch size (*taille de lot*) est le nombre d'évènements qui traverse le réseau de neurones avant la mise à jour des poids. Une grande valeur de ce paramètre peut complètement fausser les résultats et causer le réseau de neurones de diverger ou d'être coincé dans un minimum locale de la fonction d'erreur. Une valeurs très petite peut ralentir la convergence du réseau de neurones et de nécessiter plus d'itérations pour s'approcher du minimum global de la fonction d'erreur. Pour cela, une étude de ce paramètre est nécessaire [31].
6. **Nombre d'itérations (*NbIt*) :**

Le nombre d'itérations est un paramètre qui doit être fixe de façon à ce qu'il permet d'apprendre un nombre maximal d'informations tout en évitant de tomber dans le sur-apprentissage.

3.5 Paramètres de l'algorithme Adaboost

3.5.1 Choix des paramètres

Nous utilisons l'algorithme Adaboost notamment son variant le plus performant SAMME.R en vue d'avoir de bonnes performances. Nous avons choisis les paramètres suivants :

— Estimateur de base : notre réseau de neurone (qui peut être soit le MLP que nous proposons, soit le CNN que nous proposons).

— Nombre d'estimateurs : *NbE*.

— Taux d'apprentissage : *TA*.

— Algorithme : SAMME.R.

3.5.2 Explication des choix

1. **Estimateur de base :**
   Ce paramètre représente l'architecture du modèle de classification à utiliser lors de l'exécution de l'algorithme Adaboost. Il a été naturellement fixé à être l'architecture du réseau de neurones que nous proposons (soit MLP soit CNN).
2. **Nombre d'estimateurs (*NbE*) :**
   C'est le nombre de réseau de neurones à créer lors de l'exécution de l'algorithme Adaboost. Généralement, en augmentant le nombre d'estimateurs, la performance de l'ensemble des estimateurs augmente. Mais, l'inconvénient est qu'un nombre très élevé d'estimateurs prend beaucoup d'espace mémoire. Ce paramètre est donc à étudier en fonction du matériel disponible.
3. **Taux d'apprentissage (*TA*) :**
   Le taux d'apprentissage (*learning rate*) de l'algorithme Adaboost est un paramètre clé. Il permet de spécifier le taux d'augmentation ou décrémentation des poids des évènements suivant la performance de l'ensemble des estimateurs précédent. Il n'y a pas de règles qui le régit. Donc, il dois être étudié.
4. **Algorithme :**
   Ce paramètre signifie "Quel variant de l'algorithme Adaboost ?" Alors, Nous avons choisis l'algorithme SAMME.R. Car c'est le varient le plus performant de l'algorithme Adaboost (voir paragraphe 2.12 et paragraphe 2.12.3).

3.6 Paramètres de l'algorithme Bagging

3.6.1 Choix des paramètres

Les paramètres de l'algorithme Bagging sont fixés de la manière suivante :

— Estimateur de base : notre réseau de neurones (qui peut être soit le MLP que nous proposons, soit le CNN que nous proposons).

— Nombre d'estimateurs : *NbE*

— Nombre maximal d'évènements à choisir parmi les données originales pour faire l'apprentissage d'un estimateur : 50%.

— Nombre maximal de caractéristiques à choisir : 100%.

— Choix des évènements se fait avec replacement ? : False.

— Choix des caractéristiques se fait avec replacement ? : False.

— Utilisation des poids de l'ancien estimateur pour initialiser les poids du nouveau réseau : True.

3.6.2 Explication des choix

1. **Estimateur de base et Nombre d'estimateur :**
   Ce sont des paramètres identiques et communs entre l'algorithme Adaboost et Bagging (voir paragraphe 3.5.2.1).
2. **Nombre maximal d'évènements par estimateur et caractéristiques à choisir des données d'apprentissage :**
   Vu que les évènements sont représentés par un ensemble de caractéristiques bien définies et chaque caractéristique communique une information pertinente alors toutes les caractéristiques ont été prises. En ce qui concerne le nombre de d'évènements à prendre pour faire l'apprentissage d'un estimateur, nous avons pris 50%. Ceci pour varier les données d'apprentissage de chaque estimateur (réseau de neurones).
3. **Choix des évènements et les caractéristiques ce fait-il avec replacement ? :**
   Non, Car lors du choix avec remplacement, nous pouvons avoir les mêmes caractéristiques ou évènements qui existe deux ou plusieurs fois dans les données d'apprentissage. Ceci peut causer du sur-apprentissage.
4. **Utiliser les poids de l'ancien estimateur pour initialiser les poids du nouveau estimateur ? :**
   Oui. Ceci permet d'accélérer l'apprentissage des estimateurs et de minimiser l'erreur plus rapidement.

3.7 Approche de résolution parallèle

L'apprentissage des réseaux de neurones est une phase qui coûte très chère en termes de temps car elle comporte un grand nombre d'opérations sur des matrices. Ceci peut prendre beaucoup de temps surtout lorsqu'on a un nombre de données d'apprentissage qui est énorme. Ce qui est notre cas vu que nous avons 250 000 évènements pour la phase d'apprentissage. A cet effet, la parallélisation de cette phase est plus que nécessaire. Les idées qui suivent sont inspirées de l'algorithme "**Task Parallelism**" [46].

3.7.1 Parallélisation dans une couche

Partant du fait qu'on a n neurones dans une couche donnée et qu'on a aussi n CPUs pour la parallélisation. La parallélisation des opérations au sein d'une couche donnée d'un réseau de neurones multi-couches se fait de la manière suivante :

1. Nous dupliquons le vecteur d'entrée de la couche en question n fois.

2. Sur chacun des CPU, Nous exécutons le fonctionnement normale d'un des neurones de la couche. Ceci est en utilisant un des vecteurs dupliqués et les poids du neurones en question ainsi que la fonction d'activation.
3. Après que l'exécution sur les n CPUs est terminée, les résultats sont regroupés dans un seul vecteur qui servira de vecteur d'entrée pour chacun des neurones de la couche suivante.

3.7.2 Parallélisation dans un neurone

Partant du fait que le vecteur d'entrée du neurone a n composantes et que nous avons aussi n CPUs pour la parallélisation, la parallélisation de l'apprentissage d'un réseau de neurones au sein d'un neurones se fait en exécutant chacune des multiplications $X_i W_i$ (voir paragraphe 2.3) sur un des n CPUs. Ceci réduit le temps du calcul de ce produit de $\mathcal{O}(\mathrm{n})$ à $\mathcal{O}(1)$. Donc, il reste juste le calcul de la somme de ces produits qui a une complexité de $\mathcal{O}(\mathrm{n})$ [46].

3.7.3 Estimation de la complexité

En suivant l'approche de parallélisation au sein d'un neurone présenté précédemment, nous pouvons réduire la complexité de calcule de $2 \times \mathcal{O}(\mathrm{n})$ à $1 \times \mathcal{O}(\mathrm{n})$. Et si aussi, nous parallélisons le fonctionnement au sein d'une couche comme présenté précédemment, nous pouvons aussi réduire la complexité de calcule de $\mathcal{O}(\mathrm{n}^2)$ jusqu'à $\mathcal{O}(\mathrm{n})$. Donc, pourvu nous ayons suffisamment de CPUs, nous pouvons réduire la complexité toutes les opérations d'une couche d'un réseau de neurones multi-couches de $\mathcal{O}(\mathrm{n}^2)$ jusqu'à $\mathcal{O}(\mathrm{n})$.

3.8 Conclusion

Dans ce chapitre, d'une part, nous avons proposé deux architectures de réseaux de neurones : une du type MLP et une autre du type CNN. D'autre part, nous avons étudie leurs paramètres. Par suite, nous avons choisi pour chaque paramètre la valeur correspondante d'un éventuel meilleur résultat. Enfin, nous avons étudié et choisis les paramètres optimaux de l'algorithme Adaboost. D'autre part, nous avons proposé une approche de résolution parallèle vu le nombre important des données à traiter. Dans le chapitre suivant, nous ferons l'apprentissage de ces réseaux de neurones avec différentes valeurs du nombre d'estimateurs dans les algorithmes Adaboost et Bagging.

Chapitre 4

Réalisation

4.1 Introduction

Dans ce chapitre nous commençons par présenter les environnements matériel et logiciels sur lesquels nos propositions vont être concrétisées. Puis, nous analysons des données fournies par le CERN en vue de tirer les caractéristiques les plus pertinentes. Après, nous présentons des tests permettant de fixer les valeurs optimale des deux modèles proposés. Ceci fait, nous montrons une parallélisation pratique de l'apprentissage des réseau de neurones. Nous finissons par tester les performances des propositions et discuter les résultats obtenus.

4.2 Environnement de l'implémentation

4.2.1 Environnement Matériel : Le GPU

Le processeur graphique ou GPU (*Graphical Processing Unit*) est un processeur spécialisé dans les fonctions du calcul de l'affichage. Avec son architecture bien spécifique ressemblant à plusieurs mini-CPU (appeler les cores CUDA : *Compute Unified Device Architecture*) qui travaillent en parallèle (voir figure 4.1), il devient intéressant comme processeur de calculs matricielle. En fait, les GPUs ont été créé en vue de subvenir au besoin de traitement d'images qui est un ensemble d'opérations sur des matrices volumineuses nécessitant beaucoup de resources.

Le GPU que nous avons utilisé dans l'apprentissage de nos réseaux de neurones est le GPU Nvidia Quadro 4000 du cluster IBN-BADIS de CERIST. Ce GPU a 448 coeurs et 6GB de RAM [47].

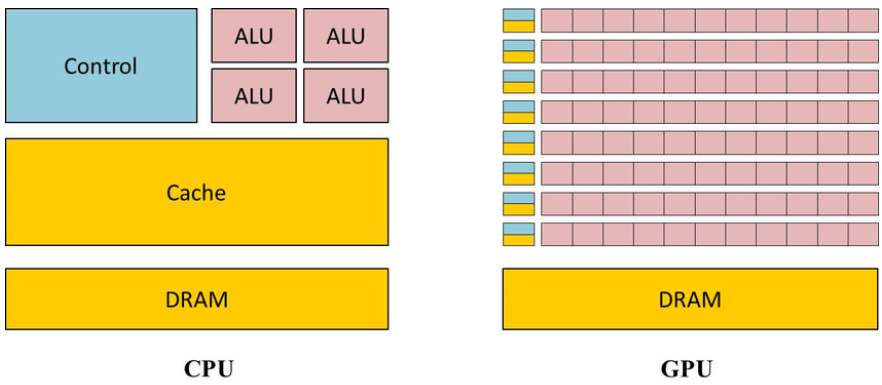

FIGURE 4.1 – Architecture de CPU vs architecture de GPU [48]

4.2.2 Environnement Logiciel

Pour la manipulation des données ainsi que la construction de nos réseaux de neurones nous avons utilisé les bibliothèques suivante :

Tensorflow est une bibliothèque open source développé par Google écrit en C++ et en python, utilisée pour créer l'architecture du réseau de neurones ainsi que faire son apprentissage. Elle permet en particulier la parallélisation de la phase d'apprentissage lors de la détection d'un GPU.

Keras est une bibliothèque open source écrite en python et permettant d'interagir avec les algorithmes de réseaux de neurones profonds plus facilement, notamment ceux de la bibliothèque Tensorflow. C'est un niveau d'abstraction en dessous de Tensorflow qui permet d'avoir un code plus lisible et moins long.

Numpy est une bibliothèque du langage de programmation Python, destinée à manipuler les matrices et les tableaux multidimensionnels ainsi que facilité l'utilisation des fonctions mathématiques opérant sur ces tableaux vue que la plus part d'entre elles sont déjà pré-définies.

Pandas est une bibliothèque écrite pour le langage de programmation Python, permettant la manipulation et l'analyse des données.

Matplotlib est une bibliothèque du langage de programmation Python, destinée à tracer et à visualiser des données en plusieurs formes graphiques tel que Box plot et Heat map.

Scikit-learn est une bibliothèque open source écrite en Python destinée à l'apprentissage automatique. Elle dispose des algorithmes non disponibles dans les bibliothèques Tensorflow et Keras tel que Adaboost, SAMME.R et Bagging.

4.3 Analyse des données de la compétition

4.3.1 L'analyse des données et son importance

L'analyse des données est un processus d'inspection, de nettoyage des données dans le but de découvrir des informations utiles sur les données [49]. Dans notre projet, elle nous permet de connaitre les données (plus précisément les caractéristiques) que nous allons utilisé dans l'apprentissage et dans les tests de nos réseaux de neurones, identifier les anomalies, les points forts et importants présents dans nos données en vue de les régler ou bien de les mettre en valeur.

4.3.2 Visualisation des données fournies

Dans cette partie, nous nous intéressons à analyser les données fournie par le CERN lors de la compétition en 2014 à travers les outils graphiques statistiques Box plot et Heat map. Ceci pour identifier les caractéristiques pertinentes et utiles à la classification des données en signal du Higgs et bruit de fond.

Remarque

D'après [19], la valeur -999.0 dans un évènement représente que la valeur de cette caractéristique est manquante. Alors, pour que cette valeur ne soit pas incluse dans nos représentations graphiques et qu'elle ne fausse pas les représentations, nous l'avons remplacé par la valeur **NaN** (NaN : indique à la bibliothèque que nous utilisons pour faire les représentation graphique que cette valeur est manquante).

1. **Box Plot**

   La Box Plot est une méthode qui consiste à représenter graphiquement les données statistiques sous forme de quartiles. Le premier et troisième quartiles sont représentés par un rectangle. Les valeurs avant le première quartiles ainsi que les valeurs après le troisième quartile sont par contre représentés par des lignes verticale en haut et en bas de ce rectangle. La médiane, qui est le deuxième quartile, est indiquée par une ligne verticale dans le rectangle.

   Dans notre projet, nous avons utilisé cette méthode pour déterminer si une caractéristique peut nous être utile dans la classification des données. Notre critère d'élimination des caractéristiques inutiles est que si une caractéristique présente quasiment la même distribution des données pour les deux classes signal et bruit de fond, alors cette caractéristique n'est pas bénéfique pour la classification. Et, nous devont l'éliminer pour éviter qu'elle n'influe négativement sur les résultats en causant du sur-apprentissage [50] [24] (voir paragraphe 2.10) et de ralentir la phase d'apprentissage.

   Sur la figure 4.2, nous remarquons bien que la distribution des valeurs de la caractéristique DER mass MMC est différente pour les deux classes signal (s) et bruit de font (b). Par contre, pour la caractéristique DER pt tot la distribution est quasiment la même. Par conséquent, la caractéristique DER pt tot est à éliminer.

   L'analyse des box plot de toutes les caractéristiques des données fournies nous a permis d'identifier 7 caractéristiques à savoir : les caractéristiques de terminant par *_phi (PRI tau phi, PRI lep phi, PRI met phi, PRI jet leading phi, PRI jet subleading phi) , DER pt tot, PRI jet subleading pt, qui présentent la même distribution pour les deux classes (signal et bruit de font). Ces 7 caractéristiques sont donc inutiles pour notre classification.

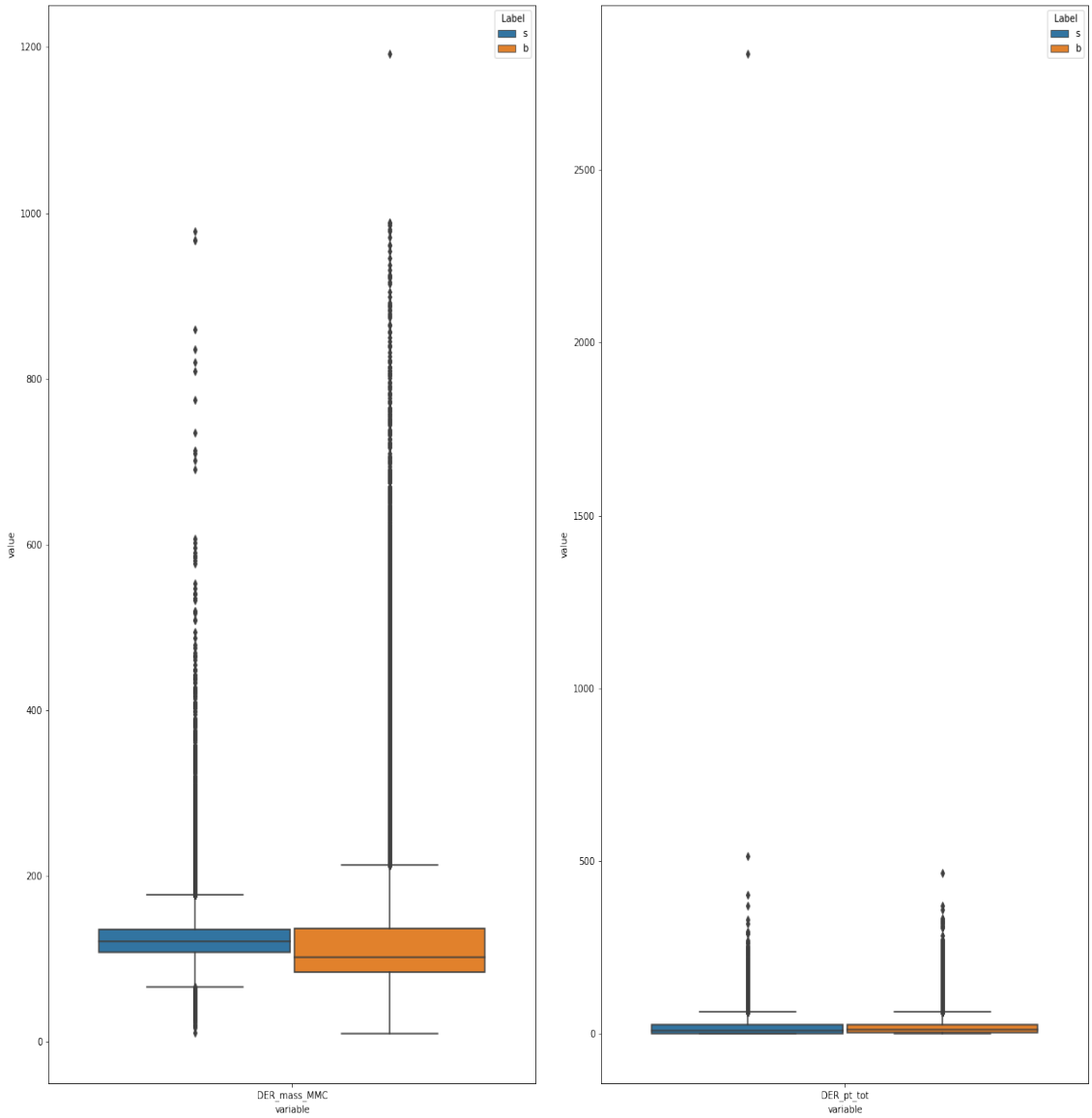

FIGURE 4.2 – Box plot appliquée aux deux caractéristiques DER mass MMC (à gauche) et DER pt tot (à droite)

2. **Heat Map**

   La Heat Map est une représentation des coefficients de corrélation entre les valeurs des données de plusieurs caractéristiques prises deux à deux. C'est une représentation à deux dimensions où les valeurs numériques des coefficients de corrélation sont représentées respectivement par des couleurs. Ces coefficients de corrélation obéissent à la formule de Pearson [51] suivante :

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

   où r est le coefficient de corrélation, X et Y sont deux caractéristiques à comparer, $\bar{X}$ et $\bar{Y}$ représentent la moyenne des deux caractéristiques X et Y respectivement. Cette formule retourne des valeurs entre -1 et +1.

   Une valeur positive de ce coefficient veut dire que lorsque la variable X augmente, la variable Y augmente aussi ($Y \sim X$). Par contre, une valeur négative veut dire que lorsque la variable X augmente, la variable Y diminue ($Y \sim 1/X$). Le cas $r = 0$

signifie qu'il n'y a pas de corrélation entre X et Y. Le cas $|r| = 1$ représente le maximum de corrélation.

Ainsi, on peut dire qu'une Heat map nous fournit un résumé visuel coloré immédiat et simple à comprendre de l'information véhiculée par les données.

Dans notre projet nous avons utilisé cette méthode pour identifier les caractéristiques corrélées. Vu que les caractéristiques corrélées apportent les mêmes informations alors le réseau de neurones revoie les mêmes informations plusieurs fois lors de la phase d'apprentissage au cours de la même itération et même le même évènement. Ceci cause du sur-apprentissage. Alors, pour y remédier, dans un ensemble de caractéristiques qui sont fortement corrélées entre elles, on dois laisser seulement une seule d'entre elles dans les données d'apprentissage. Les autres doivent être éliminées [24].

L'application de cette méthode aux données de nos 23 caractéristiques issue de l'analyse par la méthode Box plot nous donne la Heat map suivante (voir figure 4.3). Nous remarquons que 6 caractéristiques sont fortement corrélées. Donc, nous pouvons éliminer 3 d'entre elles à savoir : PRI jet all pt, DER mass vis et DER sum pt. Cette forte corrélation nous permet de réduire encore plus le nombre de caractéristiques à considérer. Les 20 caractéristiques restantes serviront de base essentielle dans notre étude.

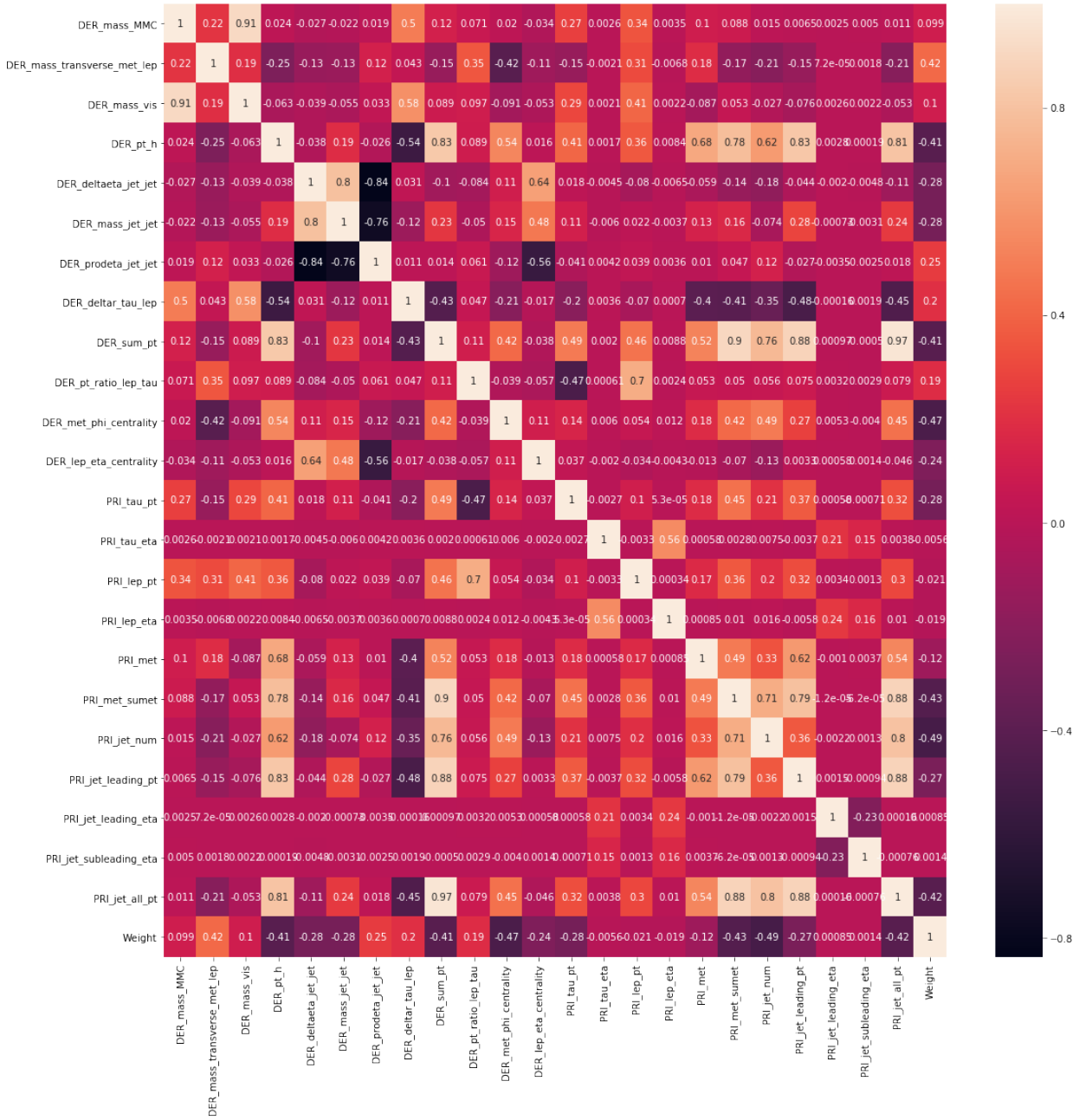

FIGURE 4.3 – Heat Map des caractéristiques utiles : si $r > 0$ alors $Y \sim X$, et si $r < 0$ alors $Y \sim 1/X$. Les différentes couleurs indiquent le taux de la corrélation. La valeur indiquée dans chaque case correspond au taux de corrélation entre les caractéristiques X et Y.

4.3.3 Traitement des valeurs manquantes

En explorant les valeurs de la totalité de nos caractéristiques, nous remarquons qu'il y a des valeurs manquantes dans 11 caractéristiques à savoir (DER mass MMC, DER deltaeta jet jet, DER mass jet jet, DER prodeta jet jet, DER lep eta centrality, PRI jet leading pt, PRI jet leading eta, PRI jet leading phi, PRI jet subleading pt, PRI jet subleading eta, PRI jet subleading phi). Pour y remédier, nous avons trois approches possibles [52] :

1. **Première approche :**

   Soit supprimer les évènements où il y a au moins une données manquante. Mais, cela va causer une perte d'un grand nombre d'évènements qui sont très précieux lors

de l'apprentissage des réseaux de neurones que nous proposons. Cette approche est fortement déconseillée.

2. **Deuxième approche :**
   Soit remplacer toutes les valeurs manquantes d'une caractéristique par une valeur fixe, à savoir le mode, la médiane ou bien la moyenne de cette caractéristique.
   On prend tous les évènements où il n'y a pas de valeurs manquantes. Pour cette caractéristique, on calcule le mode (ou la médiane ou la moyenne voir figure 4.4). Puis, on remplace toutes les valeurs manquantes de cette caractéristique par ce mode (ou la médiane ou la moyenne) qu'on viens de calculer.
   Cette approche peut marcher dans le cadre où l'évènement où il y a une valeur manquante le mode (ou la médiane ou la moyenne) garde la même corrélation avec les autres caractéristiques. Mais, des fois ce n'est pas le cas et donc ceci va faussé les calculs.

3. **Troisième approche :**
   Soit créer un modèle pour la prédiction des valeurs manquantes. Cette approche permet d'avoir des valeurs approximatives de la valeur manquante par rapport aux autres valeurs des caractéristiques de l'événement en question. Puisque nous avons des caractéristiques qui sont corrélées alors cela nous aidera à faire la prédiction.

Approche choisie :

Pour choisir la bonne approche nous avons d'abord explorer les solutions des trois premiers lauréats de la compétition du CERN en 2014. Nous avons remarqué que chacun d'eux a essayé de faire la prédiction des valeurs manquantes (qui est la troisième approche), car c'est normalement l'approche qui donne les résultats les plus précis. Ils ont remarqué que cela influe négativement la précision de la classification des données en signal et bruit de fond (chose que nous avons effectivement vérifier sur un nombre restreint de paramètres). Ceci les a poussé à délaisser cette approche et remplacer les valeurs manquantes par une valeur fixe et c'est ce que nous allons aussi faire. Nous allons donc adopter la deuxième approche de traitement des valeurs manquantes. Nous allons remplacer un tiers des valeurs manquantes par le mode, le deuxième tiers par la moyenne et le troisième tiers par la médiane. La première approche a été aussi délaissée car elle nous fait perdre un nombre énorme d'évènements. De plus, les données de tests eux même contiennent des valeurs manquantes et nous ne pouvons pas éliminer des évènements des données de tests.

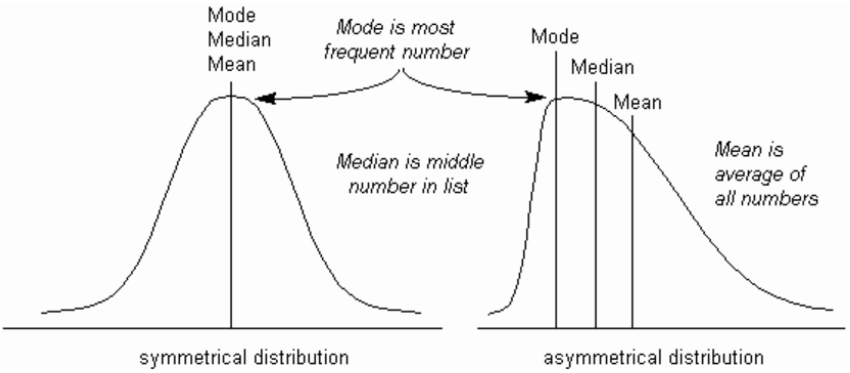

FIGURE 4.4 – Mesures de location [53]

4.4 Signification Médiane Approximative comme critère de performance

Dans le but de mesurer la performance de nos réseaux de neurones, nous utilisons la Signification Médiane Approximative (AMS : *Approximate Median Significance*) tel que exigé par le CERN lors de la compétition en 2014 et dont la formule est [19] :

$$\text{AMS} = \sqrt{2\left((s+b+b_{reg})\ln\left(1+\frac{s}{b+b_{reg}}\right)-s\right)} \tag{4.1}$$

où s est la somme des poids des évènements "signal" prédits correctement par notre réseau de neurones. b est la somme des poids des évènements "bruit de fond". Mais, notre réseau de neurones les prédit incorrectement comme étant "signal". b_{reg} est un terme de régularisation qui a une signification physique. Mais, pour les fins de la compétition, il a été considéré par le CERN comme étant une constante qui vaut 10 [19].

4.5 Les valeurs optimales des paramètres vis à vis des architectures proposées

En vue de bien structuré nos réseaux de neurones et atteindre de bonnes performances, dans cette partie nous allons étudier les différents paramètres non régis par des règles des réseaux de neurones que nous avons proposé dans le chapitre précédent afin de les optimisés un par un.

4.5.1 Architecture finale du réseau de neurones multi-couches proposé

L'architecture finale du réseau de neurones multi-couches que nous proposons est la suivante (voir figure 4.5) :

- Couche 1 : nombre de neurones est égal au nombre de caractéristiques en entrée, Batch Normalization, fonction d'activation : PRelu, Dropout = 0.3.
- Couche 2 : 5 neurones, Batch Normalization, fonction d'activation : PRelu, Dropout = 0.3.
- Couche 3 : 10 neurones, Batch Normalization, fonction d'activation : PRelu, Dropout = 0.3.
- Couche 4 : 16 neurones, Batch Normalization, fonction d'activation : PRelu, Dropout = 0.3.
- Couche 5 : 16 neurones, Batch Normalization, fonction d'activation : PRelu, Dropout = 0.3.
- Couche de sortie : 2 neurones, fonction d'activation : Softmax.

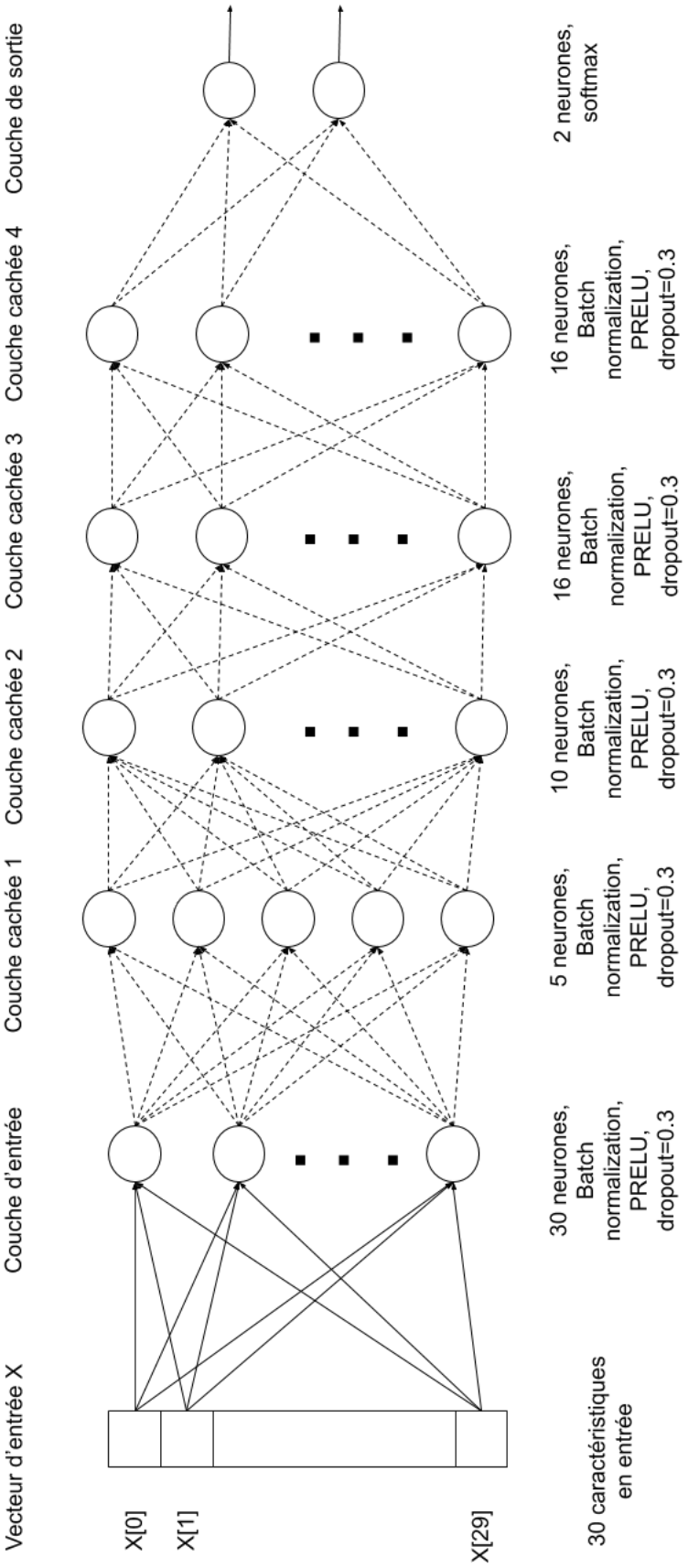

FIGURE 4.5 – Architecture finale du réseau de neurones multi-couches proposé pour 30 caractéristiques en entrée

Étude et choix du nombre de neurones dans chaque couche cachée du réseau de neurones multi-couches

Pour étudier l'effet du nombre de neurones dans chaque couche cachée (NbN) sur la performance et choisir sa valeurs optimale, nous varions ce paramètre et nous observons son impact sur la performance.

• Premièrement : nous choisissons un nombre de neurones qui est le même dans toutes les couches cachées dans tous les réseaux de neurones. Les résultats sont représentés dans le tableau 4.1 et la figure 4.6.

Nombre de neurones	5	9	13	16	23	50	100	200
AMS	2.441	2.421	2.425	2.431	2.342	2.320	2.340	2.300
Nombre de neurones	300	400	500	600	700	800	900	
AMS	2.339	2.275	2.239	2.290	2.311	2.242	2.267	

TABLE 4.1 – AMS du réseau de neurones multi-couches proposé pour différents nombres de neurones par couche (nombre de neurones identique dans les couches cachées)

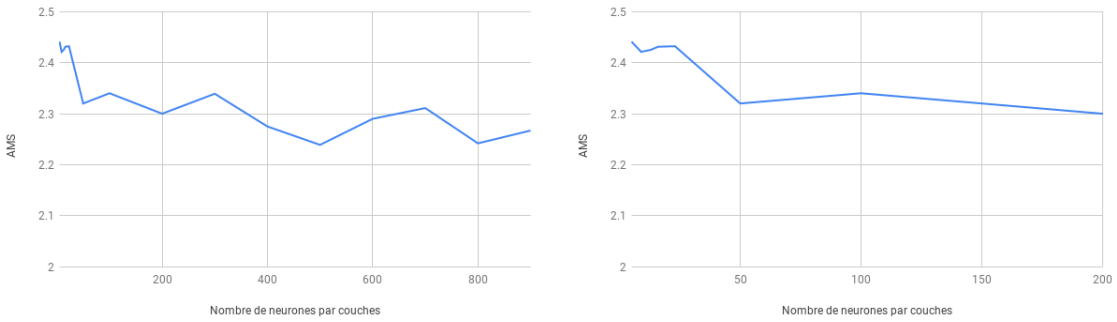

FIGURE 4.6 – AMS du réseau de neurones multi-couches proposé pour différents nombre de neurones par couche (nombre de neurones identique dans les couches cachées)

Nous remarquons bien que il y a une relation entre le nombre de neurones par couche cachée et la performance de l'ensemble des réseaux de neurones. De plus, nous remarquons que de grandes valeurs de neurones dans chaque couche affectent négativement la performance. Pour cela, nous avons choisi des valeurs petite dans l'étude qui suit.

$$\boxed{\textbf{Nombre de neurones dans chaque couche cachée} \in]\mathbf{2, 30}[}$$

• Deuxièmement : nous choisissons un nombre de neurones qui n'est pas le même dans toutes les couches cachées d'un réseau de neurones donné. Nous prenons des valeurs qui sont comprises entre le nombre de neurones dans la couche d'entrée à savoir 30 et le nombre de neurones dans la couche de sortie à savoir 2. Les résultats obtenus sont représentés dans le tableau 4.2.

Nombre de neurones (couche 2 - couche 3 - couche 4)	23-16-9	9-16-23	16-10-5	5-10-16
AMS	2.362	2.360	2.355	2.466

TABLE 4.2 – AMS du réseau de neurones multi-couches proposé pour différents nombre de neurones par couche (nombre de neurones non identique dans les couches cachées)

Nous remarquons bien d'après le tableau 4.2 que l'architecture 5-10-16 donne les meilleurs résultats.

Nombre de neurones par couche : $5 - 10 - 16$

Étude et choix du nombre de couches cachées dans le réseau de neurones multi-couches

Dans cette partie nous étudions l'effet du nombre de couches cachées du réseau de neurones mutli-couches proposé (n) dans le but de trouver la valeur optimale. Pour cela, nous avons testé la performance du réseau de neurones multi-couches avec plusieurs valeurs de nombre de couches cachées. Pour ce qui est du nombre de neurones dans chaque couche nous nous somme inspiré des résultats précédents quant nous avons étudie le nombre de neurones par couche. Les résultats sont représentés dans le tableau 4.3.

Nombre de couches cachées	2 (5-16)	3 (5-10-16)	4 (5-10-16-16)	5 (5-10-16-16-16)
AMS	2.352	2.525	2.553	2.386

TABLE 4.3 – AMS du réseau de neurones multi-couches proposé pour différents nombre de couches cachées

Nous remarquons bien que le meilleur nombre de couches cachées est 4 avec l'architecture 5-10-16-16.

Nombre de couches cachées : 4

Répartition du nombre de neurones par couche : $5 - 10 - 16 - 16$

Étude et choix du taux du dropout dans le réseau de neurones multi-couches

Dans cette partie nous étudions l'impact du taux du dropout (Drp) sur la performance de notre modèle, ainsi que nous cherchons sa valeur optimale pour notre problématique en terme d'évité le sur-apprentissage et ne pas entraver le processus d'apprentissage. Pour cela nous considérons un taux du dropout identique dans toutes les couches du réseau de neurones multi-couches proposé et nous testons plusieurs valeurs de ce paramètre. Les résultats sont représentés dans le tableau 4.4 et la figure 4.7.

Taux du dropout	0	0.1	0.2	0.3	0.4	0.5
AMS	2.398	2.40	2.466	2.52	2.35	2.31

TABLE 4.4 – AMS du réseau de neurones multi-couches proposés pour différentes valeurs du taux de dropout

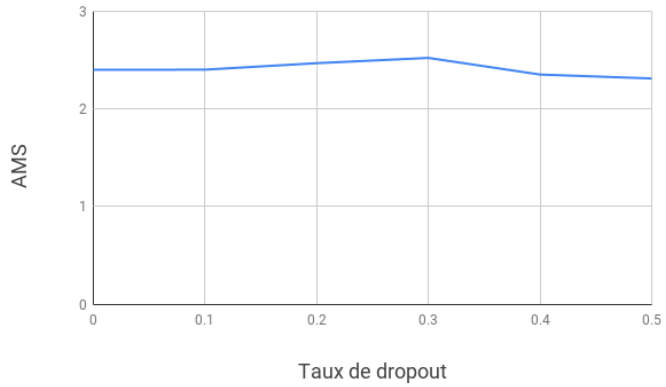

FIGURE 4.7 – AMS des réseaux de neurones proposés pour différents taux de dropout

Nous remarquons que la meilleure valeur du taux de dropout est 0.3.

Taux de dropout : 0.3

4.5.2 Architecture finale du réseau de neurones convolutionnel proposé

L'architecture du réseau de neurones convolutionnel que nous proposons pour adresser notre problématique et atteindre nos objectifs est la suivante (voir figure 4.8) :

- Couche 1 : convolution 1D, filtres = 5, kernal size = 3, stride = 1, padding = "valid", fonction d'activation : PRelu.
- Couche 2 : convolution 1D, filtres = 10, kernal size = 3, stride = 1, padding = "valid", fonction d'activation : PRelu.
- Couche 3 : Flatten.
- Couche 4 : 60 neurones, Batch Normalization, fonction d'activation : PRelu. régularisation l1=0.00001,l2=0.0001, Dropout=0.4.
- Couche de sortie : 2 neurones, fonction d'activation : Softmax.

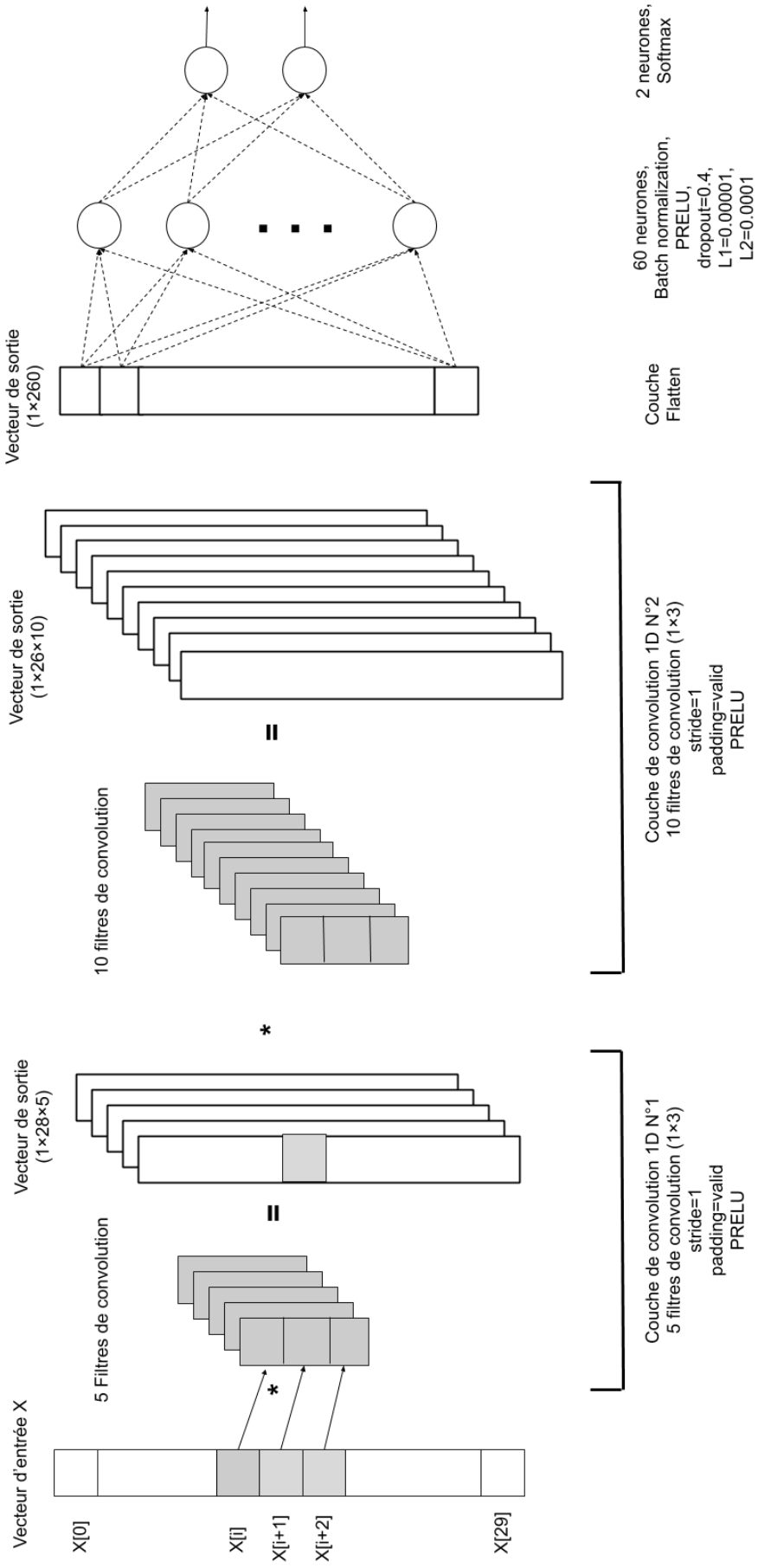

FIGURE 4.8 – Architecture finale du réseau de neurones convolutionnel proposé pour 30 caractéristiques en entrée

Choix du nombre de couches de convolution

Notre réseau de neurones convolutionnel contient deux couches de convolution (Nc=2) inspiré de CNN VGG-16 [54] qui a toujours des pairs de couches de convolution et il donne de bon résultats en terme de performance. Mais contrairement au CNN VGG-16 qui est très profond car il traite des images extrêmement volumineuses, notre nombre caractéristiques de données est très petit par rapport à ceux du CNN VGG-16 donc nous n'avons pas utilisé autant de couche de convolution.

Choix du nombre de filtres

Le nombre de filtre (NbF) a été fixé à 5 dans la première couche et à 10 dans la deuxième couche de convolution. Car après que nous avons testé quelques autres valeurs, il c'est avéré que ces valeurs donnent les meilleurs résultats en terme d'AMS.

Choix du nombre de couches dense de neurones et le nombre de neurones

Concernant le nombre de couches dense (Nb), nous avons mis une seule couche car le nombre de paramètres sortant de la couche Flatten (qui est de 260 paramètres) n'est pas vraiment grand pour un CNN. Cette couche contiens un nombre de neurones (NbN) égal à 60. Car c'est la valeur qui a donnée les meilleurs résultats et qui est entre le nombre de paramètres de la couche précédente et suivante [32].

Choix du taux du dropout et la régularisation *L1* et *L2*

La valeur de dropout a été fixé a 40%, L1 a été fixé a 0.00001 et L2 a été fixé a 0.0001 car après tests sur un ensemble restreint de valeurs, ces valeurs donnent les meilleurs résultats en terme d'AMS.

4.5.3 Révision des paramètres d'apprentissage

Pour faire l'apprentissage des réseaux de neurones (MLP et CNN) que nous proposons nous utilisons les paramètres suivants :

— Optimiseur : Adam.

— Fonction d'erreur : entropie croisée catégorique clairsemée.

— Nombre d'itérations : 40.

— Batch size (*Taille de lot*) : 90.

— Partage de validation : 15%.

— Shuffle : True.

Étude et choix du nombre du batch size

Le paramètre batch size (BS) est un paramètre critique qui dois être étudier méticuleusement pour que notre réseau de neurones converge de façon rapide et efficace. Pour cela nous avons considéré plusieurs valeurs de ce paramètre, nous les avons testé et les résultats sont représentés dans le tableau 4.5 et la figure 4.9.

Nous remarquons bien que la meilleure valeur du batch size est 90.

Batch size : 90

Batch size	40	50	60	70	80	90	100
AMS	2.416	2.410	2.374	2.458	2.51	2.52	2.35

TABLE 4.5 – AMS du réseau de neurones multi-couches proposé pour différentes valeurs du batch size

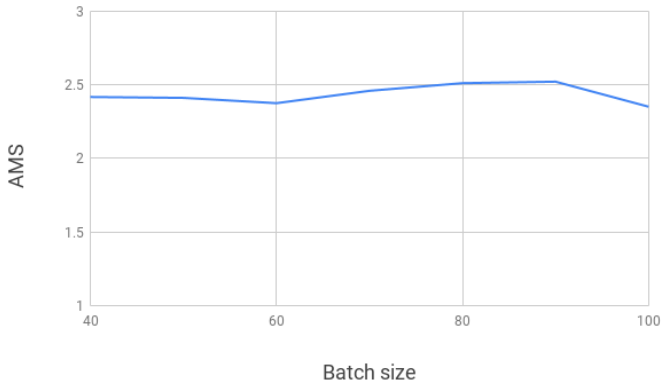

FIGURE 4.9 – AMS des réseaux de neurones proposés pour différentes valeurs du batch size

Étude et choix du nombre d'itérations

Dans cette partie nous étudions l'effet du nombre du nombre d'itérations (*NbIt*) sur la performance du réseau de neurones multi-couches. Pour cela nous faisons l'apprentissage de notre réseau de neurones multi-couches avec différentes valeur du nombre d'itérations et nous observons les résultats. Les résultats de nos tests sont représentés dans sur le tableau 4.6 et la figure 4.10.

Nombre d'itérations	20	30	40	50	60	100
AMS	2.389	2.400	2.410	2.391	2.358	2.314

TABLE 4.6 – AMS du réseau de neurones multi-couches proposé pour différents nombres d'itérations

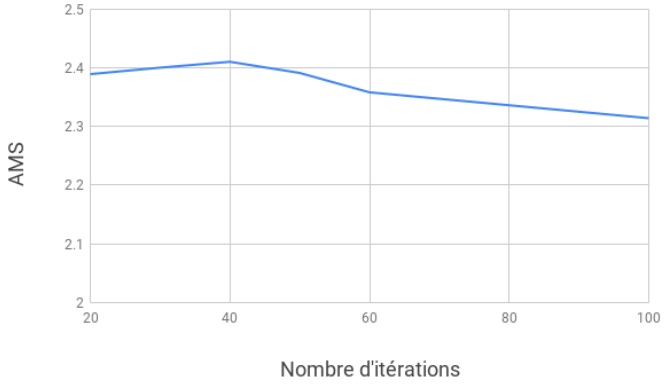

FIGURE 4.10 – AMS du réseau de neurones multi-couches proposé pour différents nombres d'itérations

Nous remarquons bien que le meilleur nombre d'itérations est 40.

Nombre d'itérations : 40

4.5.4 Révision des paramètres de l'algorithme Adaboost

Au sujets des paramètres de l'algorithme Adaboost nous utilisons les paramètres suivants :

— Estimateur de base : notre réseau de neurone.

— Nombre d'estimateur : *NbE*.

— Taux d'apprentissage : 0.3.

— Algorithme : SAMME.R.

Étude et choix du taux d'apprentissage de l'algorithme Adaboost

Dans cette partie nous étudions l'effet du taux d'apprentissage de l'algorithme Adaboost (*TA*) sur la performance de nos réseaux proposés dans le but de tirer la valeur optimale de ce paramètre. Pour cela nous faisons l'apprentissage de plusieurs estimateurs de notre réseau de neurones multi-couches avec différentes valeurs du TA et nous observons les résultats (voir tableau 4.7 et la figure 4.11).

Taux d'apprentissage	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
AMS	2.432	2.335	2.592	2.4	2.37	2.373	2.278	2.465	2.482	2.41

TABLE 4.7 – AMS du réseau de neurones multi-couches proposé pour différentes valeurs du taux d'apprentissage

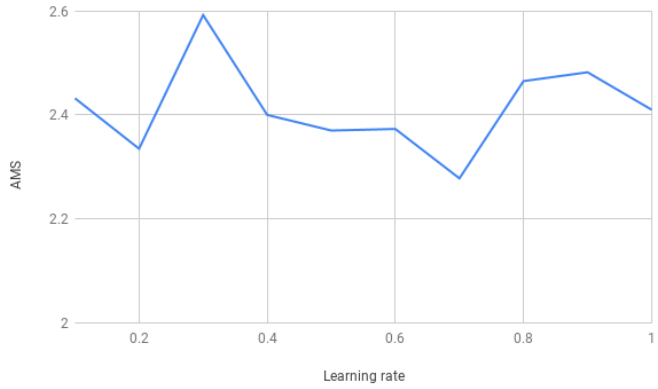

FIGURE 4.11 – AMS du réseau de neurones multi-couches proposé pour différentes valeurs du taux d'apprentissage

Nous remarquons que le meilleur taux d'apprentissage de l'algorithme Adaboost est 0.3.

Taux d'apprentissage de l'algorithme Adaboost : 0.3

4.6 Concrétisation de la parallélisation de l'apprentissage des réseaux de neurones

L'apprentissage des réseaux de neurones est une phase qui coûte très chère en termes de temps car elle comporte un grand nombre d'opérations sur des matrices. A cet effet les GPUs sont utilisés pour accélérer l'apprentissage en partageant la tâche sur les cores CUDA. Ceci en considérant les CUDA cores comme des CPUs et nous parallélisons le traitement sur eux comme décrit dans le paragraphe 3.7 (voir figure 4.12 et figure 4.13).

Cette façon de parallélisation à été implémenté à l'aide de la bibliothèque Tensorflow qui en détectant qu'un GPU est disponible elle parallélise automatiquement l'apprentissage sur ce GPU.

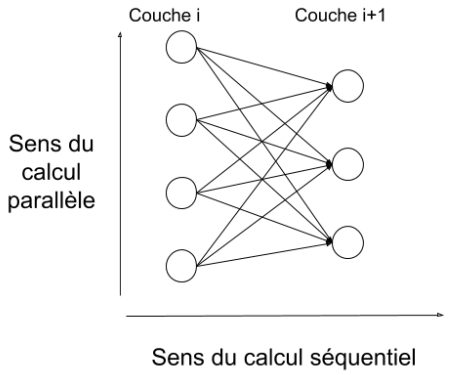

FIGURE 4.12 – Parallélisation au sein d'une couche d'un réseau de neurones donné

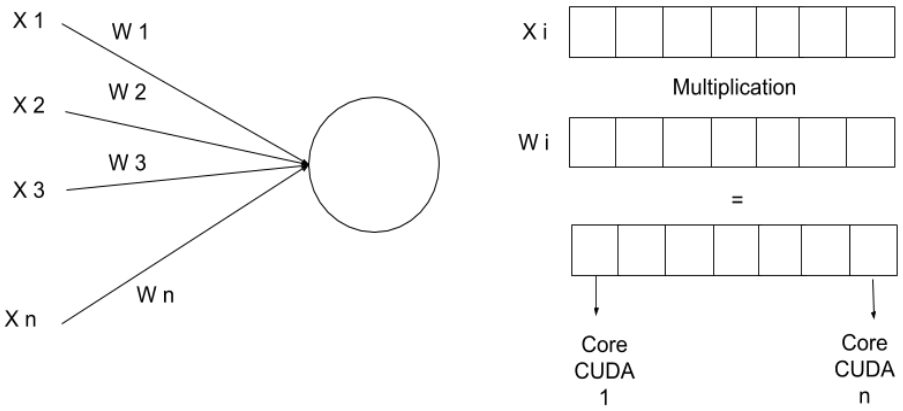

FIGURE 4.13 – Parallélisation au sein d'un neurone

4.7 Mesure de la performance des réseaux de neurones artificiels proposés

Après avoir fait l'apprentissage de nos réseaux de neurones avec les architectures et les paramètres présentées dans le chapitre précédent et un nombre d'estimateurs qui varie dans les algorithmes Adaboost et Bagging, nous avons testé la performance nos réseaux de neurones. Les résultats obtenus sont représentés dans les tableaux 4.8 et 4.9.

Nombre d'estimateurs	2	4	10	20
réseau de neurones multi-couches avec toutes les caractéristiques	2.644	2.463	2.415	/
réseau de neurones multi-couches sans les caractéristiques se terminant par _phi	2.398	2.423	/	/
réseau de neurones multi-couches sans les caractéristiques se terminant par _phi DER pt tot, PRI jet subleading pt	2.429	2.448	/	2.355
Réseau de neurones convolutionel avec toutes les caractéristiques	2.444	2.455	2.462	/

TABLE 4.8 – AMS des réseaux de neurones proposés pour différents nombre d'estimateurs avec l'algorithme Adaboost

Nombre d'estimateurs	2	4	10	20
réseau de neurones multi-couches avec toutes les caractéristiques	2.468	2.371	/	2.452
réseau de neurones multi-couches sans les caractéristiques se terminant par _phi	2.450	2.378	/	/
réseau de neurones multi-couches sans les caractéristiques se terminant par _phi DER pt tot, PRI jet subleading pt	2.442	2.452	/	/
Réseau de neurones convolutionel avec toutes les caractéristiques	2.563	2.499	2.468	/

TABLE 4.9 – AMS des réseaux de neurones proposés pour différents nombre d'estimateurs avec l'algorithme Bagging

4.8 Discussion

4.8.1 Discussion des résultats obtenus

D'après les tableaux 4.8 et 4.9, nous remarquons que :

• La majorité des valeurs de l'AMS des réseaux de neurones que nous avons proposé est dans les alentours de 2.4.

• La meilleure valeur d'AMS que nous avons pu obtenir est 2.644.

• Les valeurs d'AMS obtenue avec l'algorithme Adaboost décrémente après un certain moment. Ceci est peut être dû a une certaine forme de sur-apprentissage que nous n'avons pas pu identifier.

• L'AMS des réseau de neurones multi-couches qui ont fait l'apprentissage avec les données sans les caractéristiques se terminant par _phi, DER pt tot et PRI jet subleading pt est meilleure que celle des réseau de neurones multi-couches qui ont fait l'apprentissage avec les données seulement sans les caractéristiques se terminant par _phi. Donc, les deux caractéristiques DER pt tot et PRI jet subleading pt sont inutiles pour la classification en plus des caractéristiques se terminant par _phi identifiée par le lauréat. Ce qui confirme notre étude statistique sur les données fournies.

• L'AMS des réseaux de neurones avec les différentes données d'apprentissages est presque la même. Donc, l'élimination des caractéristiques jugées inutiles est non nécessaire et n'affecte pas la valeur de l'AMS de façon majeure.

• Malgré que les réseaux de neurones de type CNN soient généralement utilisés dans le domaine de l'imagerie. Dans notre projet, ils donnent de bons résultats qui sont même meilleurs que ceux obtenus par les réseau de neurones multi-couches. Cela est dû au fait que les CNNs profitent de la corrélation existante entre les caractéristiques de nos données.

• L'algorithme Bagging s'adapte mieux avec le CNN et l'algorithme Adaboost s'adapte mieux avec le MLP.

4.8.2 Discussion sur les problèmes de l'exécution

Lors de l'exécution de nos programmes, nous avons été confronté à plusieurs problèmes qui ont limité le champ de choix de nos paramètres. Aussi, ils ont ralenti le rythme d'avancement de notre projet. Ces problèmes sont :

— **Problème de temps d'attente :** Avant de faire l'apprentissage de nos réseaux de neurones sur le cluster IBN-BADIS du CERIST, nos programmes devaient faire longuement la queue. Et cela suite au fait que ce cluster est trop sollicité. Ceci consomme beaucoup de temps surtout lors de la phase de test des différents paramètres.

— **Problème de temps d'exécution :** Faire l'apprentissage d'un seul réseau de neurones multi-couches ou CNN avec 40 itérations nous prenait 1 heure sur CPU. Pour cela qu'on a opté d'utiliser le GPU du cluster de CERIST. L'apprentissage de nos réseaux neurones sur ce cluster prenait 25 minutes par réseau. Alors, pour faire l'apprentissage de plusieurs estimateurs, même avec GPU, nous prenait un temps de calcul énorme. Ceci, malgré que le nombre de paramètres de nos réseaux de neurones est petit (1858 paramètres par réseau de neurones multi-couches et 16542 paramètres par réseau de neurones convolutionel) par rapport au nombre de paramètres du réseau de neurones du lauréat qui été de 1.1 million de paramètres.

— **Problème de mémoire (RAM) :** Lors de la phase d'apprentissage de nos réseaux de neurones sur le cluster IBN-BADIS du CERIST, plusieurs programmes s'arrêtaient après un certain temps avant la fin du programme. Cela est dû au fait que l'ensemble des estimateurs de l'algorithme Adaboost et Bagging occupent un grand espace mémoire et saturent la RAM. Cette saturation arrive toujours malgré que nous avons essayé d'optimiser l'utilisation de la RAM en réutilisant les variables du programme.

Le lauréat de la compétition a pu traiter toutes les données en les modélisant d'une façon bien spécifique et en faisant l'apprentissage de 70 réseaux de neurones avec l'algorithme Bagging. Où chaque réseau de neurones contenait 3 couche cachée à 600 neurones. Chacun des réseaux a un nombre total de paramètres de 1.1 millions. Sa solution été implémentée sur une plateforme CUDA (GPU Nvidia Titan avec 24Go de RAM) dédiée à l'exécution de ses programmes tout en étant seul sur cette plateforme. Avec ces conditions, de données et l'environnement d'implémentation, le lauréat à obtenu une AMS de 3.8.

Par contre dans notre travail avec un maximum de 20 réseaux de neurones en utilisant l'algorithme Bagging ou Adaboost, 16 neurones dans le réseau de neurones multi-couches et 60 dans le CNN. Chaque réseau de neurones contient 1858 paramètres pour le réseau de neurones multi-couches et 16542 paramètres pour le CNN. Les ressources matérielles qui nous ont été allouées étant très limités au niveau du CERIST et surtout partagées avec d'autres utilisateurs du pays (GPU Nvidia Quadro 4000 avec 6Go de RAM). Nous avons atteint une AMS de 2.644.

La meilleure valeurs d'AMS a été obtenu avec 2 réseaux de neurones. Ceci est dû au fait que notre étude et optimisation des paramètres des réseaux de neurones que nous avons proposé été avec 2 estimateurs vu les contraintes matérielles.

4.9 Conclusion

Dans ce chapitre, nous avons analysé nos données en les visualisant par la Box Plot ainsi que la Heat Map. Nous avons éliminé les caractéristiques inutiles. Nous avons remplit les valeurs manquantes dans nos données. Ensuite, nous avons fait l'apprentissage de nos réseaux de neurones avec différentes valeurs de nombre d'estimateurs. Le meilleur résultat en terme d'AMS que nous avons pu obtenir est de 2.644.

Conclusion générale

Dans ce mémoire, nous avons traité le problème de la classification des données en signal et bruit de fond qui était le sujet de la compétition "Higgs boson machine learning challenge" du CERN en 2014.

Pour faire face à ce problème, nous avons effectué, d'une part, une étude statistique sur les données fournies. Cette étude statistique nous a permis d'analyser les données et d'identifier les caractéristiques les plus pertinentes dans la classification et celles permettant de démêler le signal du Higgs du bruit de fond. Durant notre étude, nous avons constaté que parmi les caractéristiques sélectionnées, il y en a celles qui présentent des valeurs manquantes. Alors pour y remédier, nous avons remplacé un tiers de ces valeurs par la moyenne, un deuxième tiers par le mode, et le dernier tiers par la médiane. D'autre part, nous avons proposé deux architectures de réseaux de neurones à savoir multi-couches (MLP) et convolutionnel (CNN). De plus, vue le grand nombre de données à traiter une approche de résolution parallèle a été proposée. Enfin, pour optimiser les paramètres de chaque architecture ainsi que les paramètres des algorithmes Adaboost et Bagging, nous avons étudié chaque paramètre en le faisant varier. Puis, nous avons choisit la valeur qui donne le meilleur résultat en terme de Signification Médiane Approximative (AMS).

Lors de l'exécution de nos programmes, nous avons remarqué que la phase de l'apprentissage des réseaux de neurones que nous avons proposé demande beaucoup de ressources matérielles informatiques. Ce fut une contrainte majeure qui a limité le champs de choix des valeurs de nos paramètres. Par suite, le meilleur que nous avons pu obtenir en terme d'AMS est 2.644. Ce qui est inférieur au 3.8 d'AMS obtenu par le lauréat de la compétition qui a adopté une architecture différente de la nôtre et qui requiert beaucoup de ressources matérielles. Le réseau de neurones du lauréat possède 1.1 millions de paramètres contrairement aux réseaux de neurones que nous avons proposé qui utilise uniquement 1858 paramètres pour le réseau de neurones multi-couches et 16542 paramètres pour le CNN. Les résultats que nous avons obtenu peuvent être améliorés si l'environnement matériel aurait été conséquent.

Afin de gagner encore plus en terme de performances, quelques améliorations dont certaines déjà identifiées peuvent être apportées. Ces améliorations constituent l'objet de nos perspectives tel que :

— Approfondir l'affinement des paramètres du réseau de neurones de type CNN.

— Éliminer les caractéristiques fortement corrélées des données d'apprentissage.

— Remplacer les valeurs manquantes par une seule variante à la fois : soit la médiane, soit le mode, soit la moyenne.

Enfin, il est important de dire que ce projet nous a permis d'apprendre le traitement des données, le fonctionnement des réseaux de neurones multi-couches et convolutionnel ainsi que les algorithmes qui permettent d'améliorer la précision notamment les algorithmes Adaboost, SAMME.R et Bagging, et de mettre en application de façon harmonieuse toutes ces connaissances dans un sujet d'intérêt pratique.

Bibliographie

[1] Organisation europeenne pour la recherche nucleaire. https://fr.wikipedia.org/wiki/Organisation_europ%C3%A9enne_pour_la_recherche_nucl%C3%A9aire, (2019). [En ligne] consulté le 2019/06/02.

[2] Le cern. https://home.cern/fr/about, (2019). [En ligne] consulté le 2019/06/02.

[3] Grand collisionneur de hadrons. https://home.cern/fr/science/accelerators/large-hadron-collider, (2019). [En ligne] consulté le 2019/06/02.

[4] Atlas expirement. http://cds.cern.ch/journal/CERNBulletin/2008/38/News%20Articles/1125888, (2008). [En ligne] consulté le 2019/06/02.

[5] Atlas. https://home.cern/science/experiments/atlas, (2019). [En ligne] consulté le 2019/06/03.

[6] Cms. https://home.cern/fr/science/cms, (2019). [En ligne] consulté le 2019/06/03.

[7] Alice. https://home.cern/fr/science/experiments/alice. [En ligne] consulté le 2019/06/02.

[8] Lhcb. https://home.cern/fr/science/experiments/lhcb. [En ligne] consulté le 2019/06/02.

[9] A toroidal lhc apparatus (atlas). https://www.researchgate.net/figure/A-Toroidal-LHC-ApparatuS-ATLAS_fig5_30513470, (2008). [En ligne] consulté le 2019/06/02.

[10] Lhc micro models. https://build-your-own-particle-detector.org/models/lhc-micro-models/. [En ligne] consulté le 2019/06/02.

[11] Le boson du higgs. https://home.cern/fr/science/physics/higgs-boson, (2019). [En ligne] consulté le 2019/06/04.

[12] Higgs boson or what is the god particle. https://www.123rf.com/photo_24954432_higgs-boson-or-what-is-the-god-particle-the-elusive-higgs-boson-thought-to-be-responsible-for-giving.html. [En ligne] consulté le 2019/06/03.

[13] D. Jeans and G. W. Wilson. Measuring the cp state of tau lepton pairs from higgs decay at the ilc. *Phys. Rev. D 98, 013007 (2018), arXiv preprint arXiv :1804.01241*, (2018).

[14] Laurent Sacco. Boson de higgs : l'origine de la masse des quarks se precise. https://www.futura-sciences.com/sciences/actualites/physique-boson-higgs-origine-masse-quarks-precise-50592/, (2018). [En ligne] consulté le 2019/06/03.

[15] New atlas measurement of the higgs boson mass. https://atlas.cern/updates/physics-briefing/new-atlas-measurement-higgs-boson-mass, (2017). [En ligne] consulté le 2019/06/03.

[16] Madgraph5. https://launchpad.net/mg5amcnlo. [En ligne] consulté le 2019/06/09.

[17] J. Alwall, R. Frederix, S. Frixione, V. Hirschi, F. Maltoni, O. Mattelaer, H.-S. Shao, T. Stelzer, P. Torrielli, and M. Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *JHEP07(2014)079, arXiv preprint arXiv :1405.030*, (2014).

[18] Higgs boson. `https://www.kaggle.com/c/higgs-boson`, (2014). [En ligne] consulté le 2019/02/04.

[19] Claire Adam-Bourdarios, Glen Cowan, Cecile Germain, Isabelle Guyon, Balazs Kegl, and David Rousseau. Learning to discover : the higgs boson machine learning challenge. `http://opendata.cern.ch/record/329/files/atlas-higgs-challenge-2014.pdf`, (2015). [En ligne] consulté le 2019/02/15.

[20] Rupesh Kumar Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jurgen Schmidhuber. Compete to compute. *Advances in Neural Information Processing Systems 26*, (2013).

[21] Gabor Melis. Dissecting the winning solution of the higgsml challenge. *Conference on Neural Information Processing Systems*, (2014).

[22] Bc. Jan Vojt. Deep neural networks and their implementation. *Master thesis, Charles University in Prague*, (2016).

[23] Reseau de neurones artificiels. `https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels`, (2019). [En ligne] consulté le 2019/05/22.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] 7 types of neural network activation functions : How to choose ? `https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/`. [En ligne] consulté le 2019/04/10.

[26] Dabal Pedamonti. Comparison of non-linear activation functions for deep neural networks on mnist classification task. *arXiv preprint arXiv :1804.02763*, (2018).

[27] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolution network. *arXiv preprint arXiv :1505.00853*, (2015).

[28] Ravindra Parmar. Common loss functions in machine learning. `https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23`, (2018). [En ligne] consulté le 2019/04/03.

[29] Kishan Maladkar. Multi-hot sparse categorical cross-entropy. `https://cwiki.apache.org/confluence/display/MXNET/Multi-hot+Sparse+Categorical+Cross-entropy`, (2018). [En ligne] consulté le 2019/04/03.

[30] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv :1609.04747*, (2016).

[31] Andrew Ng. Improving deep neural networks : hyperparameter tuning, regularization and optimization (course 2 of the deep learning specialization). `https://www.youtube.com/playlist?list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc`, (2017). [En ligne] consulté le 2019/04/11.

[32] Andrew Ng. Convolutional neural networks (course 4 of the deep learning specialization). `https://www.youtube.com/watch?v=ArPaAX_PhIs&list=PLkDaE6sCZn6Gl29AoE31iwdVwSG-KnDzF`, (2017). [En ligne] consulté le 2019/03/21.

[33] Timothy J OShea, Johnathan Corgan, and T. Charles Clancy. Convolutional radio modulation recognition networks. *Engineering Applications of Neural Networks*, (2016).

[34] 2d convolution. https://github.com/PetarV-/TikZ/tree/master/2D%20Convolution, (2016). [En ligne] consulté le 2019/06/10.

[35] Ayeshmantha Perera. What is padding in convolutional neural network ?s(cnn ?s) padding. https://medium.com/@ayeshmanthaperera/what-is-padding-in-cnns-71b21fb0dd7, (2018). [En ligne] consulté le 2019/06/11.

[36] Mayank Agarwal. Back propagation in convolutional neural networks ? intuition and code. https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199, (2017). [En ligne] consulté le 2019/04/10.

[37] Mustapha Hatti. https://www.researchgate.net/figure/Illustration-du-phenomene-de-sur-apprentissage-pour-le-cas-dune-approximation-de_fig14_228094570, (2012). [En ligne] consulté le 2019/04/14.

[38] Boosting algorithm : Adaboost. https://towardsdatascience.com/boosting-algorithm-adaboost-b6737a9ee60c, (2017). [En ligne] consulté le 2019/05/13.

[39] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Statistics and Its Interface*, (2009).

[40] Rajesh S. Brid. Introduction to boosting. https://medium.com/greyatom/boosting-ce84639a805d, (2018). [En ligne] consulté le 2019/05/15.

[41] Leo Breiman. Bagging predictors. *Machine learning, volume 24, pp 123-140*, (1996).

[42] Arnaud Miribel. Enrich and clean image collections using deep learning. https://arnaudmiribel.github.io/thesis/thesis.html#bagging, (2017). [En ligne] consulté le 2019/05/23.

[43] Jeff Heaton. *Introduction to Neural Networks for Java*. Heaton Research, Inc., (2008).

[44] How to choose the number of hidden layers and nodes in a feedforward neural network ? https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw, (2016). [En ligne] consulté le 2019/04/27.

[45] Sunita Nayak. Batch normalization in deep networks. https://www.learnopencv.com/batch-normalization-in-deep-networks/, (2018). [En ligne] consulté le 2019/06/03.

[46] Dounia Khaldi, Pierre Jouvelot, Corinne Ancourt, and François Irigoin. Task parallelism and data distribution : An overview of explicit parallel programming languages. *Languages and Compilers for Parallel Computing*, Lecture Notes in Computer Science, vol 7760, pp 174-189. Springer, Berlin, Heidelberg, 2012.

[47] Cluster ibnbadis. http://www.rx-racim.cerist.dz/?page_id=207. [En ligne] consulté le 2019/04/20.

[48] Nader Ben Amor. Cpu vs gpu architecture. https://www.researchgate.net/profile/Nader_Ben_Amor/publication/297734079/figure/fig10/AS:669424134680581@1536614559506/CPU-vs-GPU-architecture.ppm, (2016). [En ligne] consulté le 2019/05/29.

[49] Data analysis. https://en.wikipedia.org/wiki/Data_analysis. [En ligne] consulté le 2019/06/10.

[50] DATAI. Feature selection and data visualization. https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization, (2018). [En ligne] consulté le 2019/03/10.

[51] Association between variables. http://uregina.ca/~gingrich/corr.pdf, (2015). [En ligne] consulté le 2019/03/07.

[52] Kishan Maladkar. 5 ways to handle missing values in machine learning datasets. `https://www.analyticsindiamag.com/5-ways-handle-missing-values-machine-learning-datasets/`, (2018). [En ligne] consulté le 2019/03/12.

[53] Andy Day. `https://www.tutor2u.net/geography/reference/mean-median-and-mode`. [En ligne] consulté le 2019/03/14.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv :1409.1556v6*, (2015).

Annexe A

Liste des caractéristiques fournies

Le fichier fournie par le CERN lors de la compétition "Higgs boson machine learning challenge" contient 818238 évènements. Chaque évènement est composé des 30 caractéristiques suivantes :

DER mass MMC Masse estimée du boson du Higgs.

DER mass transverse met lep Masse transverse entre l'énergie transverse manquante et celle du lepton.

DER mass vis Masse invariante du tau hadronique et du lepton.

DER pt h Module de la somme vectorielle des quantités de mouvements transversales du tau hadronique, du lepton et de l'énergie transverse manquante.

DER deltaeta jet jet Valeur absolue de la pseudo-rapidité de la séparation entre les deux jets.

DER mass jet jet Masse invariante des deux jets.

DER prodeta jet jet Produit de la pseudo-repidité des deux jets.

DER deltar tau lep La séparation R entre le tau hydronique et le lepton.

DER pt tot Module de la somme vectorielle des impulsions transversales manquantes et des impulsions transversales du tau hadronique, du lepton, du jet principal (si PRI jet num ≥ 1) et du jet secondaire (si PRI jet num $= 2$).

DER sum pt Somme des modules des impulsions transversales du tau hadronique, du lepton, du jet principal (si PRI jet num ≥ 1) et du jet secondaire (si PRI jet num $= 2$) et des autres jets (si PRI jet num $= 3$)

DER pt ratio lep tau Rapport entre les moments transversaux du lepton et du tau hadronique.

DER met phi centrality Centralité de l'angle azimutal de l'enveloppe transverse manquante vecteur d'énergie par rapport au tau hadronique et au lepton

DER lep eta centrality Centralité de la pseudorapidité du lepton par rapport aux deux jets (non défini si PRI jet num ≤ 1)

PRI tau pt Moment transversal du tau hadronique.

PRI tau eta Pseudo-rapidité du tau hydronique.

PRI tau phi Angle azimutal du tau hydronique.

PRI lep pt Moment transversal du lepton (electron ou muon).

PRI lep eta Pseudo-rapidité du lepton.

PRI lep phi Angle azimutal du lepton.

PRI met Energie transversale manquante.

PRI met phi Angle azimutal de l'énergie transversale manquante.

PRI met sumet Energie totale transversale dans le détecteur.

PRI jet num Nombre de jets (valeur entière de 0, 1, 2, 3)

PRI jet leading pt Moment transversal du jet principal, c'est le jet avec le plus grand moment transversal (non défini si PRI jet num = 0).

PRI jet leading eta Pseudo-rapidité du jet principal.

PRI jet leading phi Angle azimutal du jet principal.

PRI jet subleading pt Moment transversal du jet secondaire, c'est le jet avec le second plus grand moment transversale (non défini si PRI jet num ≤ 1).

PRI jet subleading eta Pseudo-rapidité du jet secondaire.

PRI jet subleading phi Angle azimutal du jet secondaire.

PRI jet all pt Somme scalaire des moments transversaux de tous les jets des événements.

Weight Poids de l'expérience (le degré d'importance de l'expérience).

Label Un caractère qui prend la valeur (s) s'il s'agit d'un évènement signal, et la valeur (b) s'il s'agit d'un évènement bruit de fond.

Résumé

Le Conseil Européen pour la Recherche Nucléaire (CERN) est un des plus grands laboratoires de recherche dans le domaine de la physique des particules. Il produit des collisions proton proton dans un accélérateur circulaire appelé le Large Hardon Collider (LHC) en vue d'étudier la matière à un niveau approchant l'infiniment petit. Ces collisions entrainent un grand nombre de données qu'il faudra traiter très rapidement et de façon très concise en vue de détecter de nouvelles particules et d'étudier leurs propriétés. Alors pour faire face à ce problème, le CERN a lancé en 2014 une compétition dans le but était d'exploiter les techniques de l'intelligence artificielle pour démêler la signature de la particule du Higgs du bruit de fond dans un amas d'évènements.

Dans ce mémoire de fin d'étude de licence, nous abordons la même problématique de la compétition en utilisant les réseaux de neurones avec une architecture bien spécifique et nous essayons de nous rapprocher le plus possible des résultats obtenus par le lauréat de la compétition. Le modèle créé servira d'une plateforme d'expérience pour les chercheurs physiciens intéressés par le signal Higgs.

Notre démarche passe d'abord par une étude statistiques des données fournies par le CERN. Puis, nous considérons deux types de réseaux de neurones (MLP et CNN). Pour chacun de ces deux types de réseaux, nous identifions les paramètres les plus performants. Ensuite, en vue de booster l'AMS (*Approximate Mediane Significance*), qui est le critère de performance, nous exploitons les algorithmes Adaboost et Bagging. Vu le volume de données à traiter, nous proposant une approche de résolution parallèle. Cette dernière a été implémentée dans un environnement CUDA du centre de recherche CERIST.

Mots clés : CERN, AMS, Réseau de Neurones, MLP, CNN, Apprentissage Profond, Apprentissage Automatique, Analyse des Données, Adaboost, SAMME.R, Bagging.