République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mouloud Mammeri de Tizi-Ouzou

Faculté de : Génie électrique et d'informatique
Département : Informatique

Mémoire de fin d'études

En vue de l'obtention du diplôme de
Master en Informatique

Spécialité : Systèmes Informatiques

Réalisé et présenté par :
MEHALLI Nassim

Sujet : Mise en œuvre de l'apprentissage automatique pour l'évaluation des positions échiquéennes

Proposé et dirigé par : SADI Samy

Soutenu le 11 juillet 2019 devant le jury composé de:

Mr. RAMDANI Mohammed	Président du Jury
M^{lle}. YASLI Yasmine	Membre du Jury
Mr. SADI Samy	Directeur de mémoire

Dédicaces

Je dédie ce travail à mes chers parents qui ne cessent de me donner l'amour nécessaire pour que je puisse arriver à ce que je suis aujourd'hui. Que dieux vous protège et que la réussite soit toujours à ma portée pour que je puisse vous combler de bonheur et d'argent le jour où je serai milliardaire.

Je dédie aussi ce modèste travail

À mes chers oncles Omar et Mahdi

À mon frère et ma sœur

À Mima ma meilleure

À Sousou et mon meilleur ami Mamed

À Racim et Milou

À Ine Boula3rass

À Smail

Remerciements

Je remercie Allah de m’avoir donné le courage et la force afin d’accomplir ce modeste projet.

Je remercie mon encadreur, monsieur Sadi, pour tout ce qu’il m’a appris dans le monde de l’intelligence artificielle ainsi que toutes ses remarques pertinentes.

Je remercie l'ensemble des membres du jury, qui m'ont fait l'honneur de bien vouloir étudier avec attention mon travail.

Je remercie Jason Brownlee pour ses conseils, une personne formidable qui m’a beaucoup appris à travers son expérience dans le domaine.

Résumé

Ce mémoire s'inscrit dans le cadre de l'application de l'apprentissage automatique dans l'évaluation des positions échiquéennes. Nous avons implémenté les algorithmes et les heuristiques les plus utilisées dans la plupart des moteurs d'échecs. Aussi, nous avons mis en pratique une évaluation de positions échiquéennes dotée d'apprentissage automatique. Ainsi, nous avons conçu quatre modèles basés sur les réseaux de neurones artificiels sélectionnés pour leurs performances lors de la phase d'entrainement. Enfin, nous avons testé la qualité de ces modèles en réalisant un championnat entre les modèles lors duquel nous avons eu de bons résultats. L'un des succès est la distinction du moteur utilisant une évaluation à l'aide du modèle RNA1 et qui a confirmé ses statistiques lors de l'entrainement.

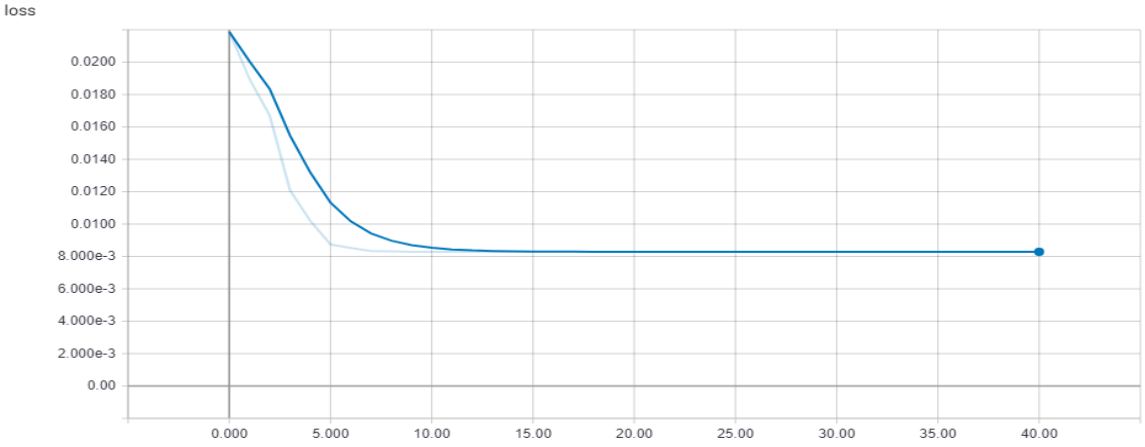
Mots clés: jeux d'échecs, évaluation, intelligence artificielle, programme, informatique, apprentissage automatique, apprentissage profond

Abstract

This thesis is part of the application of machine learning to the evaluation of chess positions. We have implemented the most algorithms and heuristics always used in chess engines. In addition, we designed four models based on artificial neural networks selected for their performance during the training phase. Finally, we tested the quality of these models by realizing a championship between the models during which we had good results. One of the successes is the distinction of the engine which used an evaluation using the model RNA1 and which confirmed its statistics during the training phase.

Key-words: chess, evaluation, artificial intelligence, program, informatics, machine learning, deep learning…

Table des matières



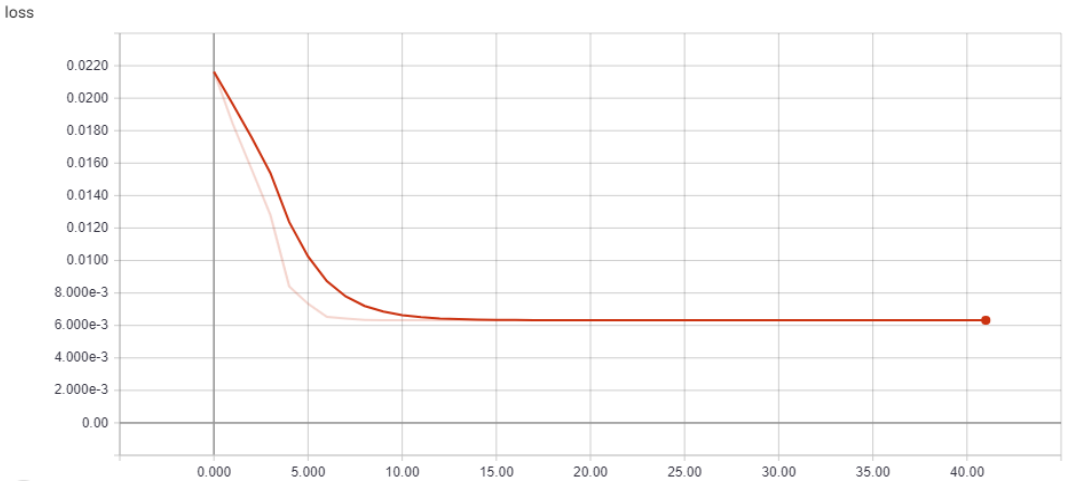


Table des tableaux, des figures et des équations

TABLEAUX

FIGURES

Table des figures

EQUATIONS

Table des figures

Abréviations, sigles, et acronymes

AA : Apprentissage Automatique

CNN : Convolutional Neural Network

RNN : Reccurent Neural Network

IA : Intelligence Artificielle

LC0 : Leela Chess Zero

RNA : Réseau de Neurones Artificiels

ANN : Artificial Neural Network

ResNet : Residual Network

ConvNet : Convolutional Network

UCI : Universal Chess Interface

GUI : GraphicalUser Interface

CPU : Central Processing Unit

BDD : Base De Données

Val_x : L'erreur x sur l'ensemble de données de validation

Introduction générale

1 Contexte du travail

Notre travail s'inscrit dans le cadre de l'informatisation des jeux d'échecs et plus précisément dans le cadre de l'évaluation de positions échiquéennes.

En effet, il existe de nos jours plusieurs moteurs qui sont équipés de fonctions d'évaluations explicitement programmées afin d'évaluer des positions échiquéennes. Ces fonctions statiques nécessitent un développement constant de la part des programmeurs afin de mieux décrire la position et d'augmenter la précision de l'évaluation. Une bonne fonction d'évaluation considère généralement un grand nombre de propriétés en plus de divers paramètres liés aux pièces, tels que la sécurité du roi, les pions passés, les pions doublés, la centralisation des pièces, etc. Le score est ainsi une combinaison linéaire de toutes les fonctionnalités sélectionnées. Plus ces caractéristiques et leurs valeurs associées correspondent aux propriétés inhérentes de la position, plus le programme d'échecs devient puissant.

D'autre part, le développement du domaine de l'apprentissage automatique et notamment celui des réseaux de neurones a permis de réaliser des programmes plus performants que ceux qui existent auparavant. Deux programmes se distinguent parmi tous, à savoir Leela chess Zero et Alpha Zero qui utilisent un apprentissage par renforcement afin d'apprendre tous seuls les règles du jeu et de progresser au fur et à mesure. Ainsi, en utilisant la technologie des réseaux de neurones, l'évaluation de la position devient plus importante et plus performante. Elle est représentée par une fonction non linéaire qui offre plus de généralisation que l'évaluation statique.

2 Problématique

L'évaluation des positions échiquéennes en utilisant l'approche classique avec une fonction statique linéaire s'avère limitée. Cela s'explique par le nombre et la qualité des différents critères d'évaluations inclus dans la même fonction. Ces propriétés qui diffèrent d'un programme à un autre ne peuvent pas être généralisées pour toutes les positions (avec la fonction linéaire). En outre, plusieurs propriétés ne peuvent pas être explicitement programmées dans la fonction d'évaluation vue leurs complexités et leurs multiplicités comme les sacrifices matérielles au détriment de l'amélioration de la position (les différents gambis), la prise en compte du temps de la pendule du joueur, l'avantage spatiale, etc.

Le domaine de l'apprentissage automatique, vient nous apporter plus de rigueur à ce problème. En effet, les réseaux de neurones artificiels (un sous domaine de l'apprentissage

automatique) tirent leur principe de celui de la réflexion humaine en utilisant un système d'automates formels qui nous permet de définir une fonction non linéaire afin de généraliser l'évaluation et d'apprendre d'avantage d'une position échiquéenne. Le but est de réduire les erreurs de la fonction d'apprentissage d'un modèle d'évaluation après l'avoir conçu, entrainé , validé et testé afin d'avoir une meilleure généralisation du problème et pour mieux interpréter les relations existantes sur l'échiquier.

La problématique qui se pose alors est celle du choix des modèles de réseaux de neurones à utiliser pour l'évaluation des positions. Il faut pouvoir choisir des modèles performants, puis les évaluer et enfin les intégrer dans un moteur de jeux d'échecs.

3 Contribution

Notre contribution dans le cadre de ce projet se situe à plusieurs niveaux:

1. Nous avons réalisé un état de l'art sur l'informatisation des jeux d'échecs et de l'apprentissage automatique. En particulier, nous avons décrit les algorithmes de recherche les plus récents utilisés dans les moteurs de jeux d'échecs
2. Nous avons réalisé un moteur complet de jeux d'échecs intégrant le protocole UCI et implémentant les différentes méthodes et algorithmes comme : La table de transposition, l'élagage alpha béta, la recherche à base de fenêtre d'aspiration, la recherche à base de variations principales etc.
3. Nous avons proposé quatre nouveaux modèles d'apprentissage automatique que nous avons sélectionnés parmi une centaine d'autres par rapport à leurs bonnes statistiques d'entrainement.
4. Nous avons entrainé et évalué les performances des modèles en utilisant une collection contenant une centaine de milliers de positions différentes jouées par les meilleurs joueurs au monde.
5. Nous avons testé les modèles en réalisant un mini-tournoi afin de comparer les performances de ces derniers en la présence d'un moteur doté d'une évaluation statique.

4 Organisation du mémoire

En plus de l'introduction générale, ce mémoire est organisé en deux parties, une conclusion générale et une annexe.

Dans la première partie, nous avons fait un état de l'art sur l'informatisation des jeux d'échecs dans le premier chapitre, et sur l'apprentissage automatique dans le second chapitre.

Dans la deuxième partie, nous avons présenté nos contributions. Tout d'abord, dans le troisième chapitre, nous avons décrit les modèles proposés après avoir présenté les caractéristiques du modèle d'un moteur open source à savoir Leela Chess Zero.

Dans le quatrième, chapitre nous avons présenté la conception et l'implémentation de notre application ainsi que son déploiement où nous avons présenté les résultats de nos modèles sous forme de confrontations entre l'approche non linaire de nos modèles et l'approche classique statique.

Enfin, nous avons réalisé une synthèse générale de notre mémoire ainsi qu'une liste de perspectives de notre application que nous comptons implémenter à l'avenir. Par la suite, nous avons conclue avec une bibliographie qui rassemble toutes nos références théorique et une annexe composée de quatre parties qui font référence à des concepts avancés intentionnellement omis dans le mémoire pour des soucis de longueur.

Chapitre 1: Les jeux d’échecs en informatique

1 Introduction

Le jeu d'échecs est l'un des jeux les plus anciens et les plus populaires de l'histoire. Décrit comme étant « le roi des jeux et le jeu des rois » et où le hasard n'a aucune place, cette création indienne ne cesse d'attirer l'attention et l'amour du grand public ainsi que l'intérêt des chercheurs en informatique. Mélange de tactique et de stratégie, ce jeu nous cache toujours des mystères.

Dans ce chapitre nous présentons les principes de bases des jeux d'échecs ainsi que les méthodes et les algorithmes développés afin d'informatiser ce jeu. Pour ce faire, nous commençons par donner un bref historique retraçant l'introduction de l'informatique dans le monde des échecs jusqu'à sa démocratisation actuelle. Puis, nous donnons un aperçu des jeux d'échecs depuis la perspective d'un domaine proche et qui connaît un essor important en informatique, nommément la théorie des jeux. Après quoi, nous exposons les trois composants de base d'un programme d'échecs. Enfin, nous concluons avec une analyse sur l'avenir des jeux d'échecs en informatique notamment dans le monde de l'intelligence artificielle (IA).

2 Historique des jeux d'échecs en informatique

Depuis sa création, le jeu d'échecs a subi plusieurs modifications de règles et beaucoup de théories ont été mises en place afin d'organiser et de développer le jeu à l'image de la Philidor[1] , partie de l'Opéra[2]ou même les théories de Wilhelm Steinitzqui était le premier champion du monde officiel et le premier à étudier le jeu d'échecs avec un esprit scientifique en développant plusieurs thèmes stratégiques optimisés par la suite par ses successeurs.

L'idée de l'informatisation des jeux d'échecs est apparue en 1770 sous forme d'un canular par Johan Wofgang von Kempelen[3] avec son fameux automate « Le turc mécanique ». Ensuite, depuis la fin de la seconde guerre mondiale, les pères de l'informatique à l'instar d'Alain Turing ont commencé à imaginer des algorithmes qui pourraient jouer une partie complète. Cela s'est réalisé dans les années 1950 avec l'apparition des premiers programmes de jeux d'échecs commençant par celui de Von Neumann et le programme de Dietrich Prinz qui était le premier à résoudre les problèmes d'échecs. En 1966, avec le développement technologique, les premières parties entre les programmes sont devenues possibles où s'est distingué le programme Kaissa, qui par la suite deviendra le premier champion du monde en

[1]Ouverture obtenue par les coups : 1-e4 e5 2- Cf3 d6, créée par François-André Philidorau 18éme siècle (la défense Philidor)
[2]Variante créée par Paul Morphy.
[3]Inventeur hongrois prétendait avoir inventé un automate jouant aux échecs alors que c'étais unhomme qui se cachait dansson boitier

1974. Le tournant de l'histoire était en 1997, quand l'ordinateur Deep Blue d'IBM défiait Garry Kasparov[4] et remporte le challenge marquant ainsi l'une des premières victoires de l'informatique qui était vue comme le moment où les programmes prenaient le pouvoir sur les humains.

Depuis l'an 2000, les avancées dans le cadre de l'informatisation des jeux d'échecs ne cessent d'avoir lieu. Cependant, ce n'est que tout récemment qu'une avancée critique a eu lieu avec l'introduction de l'apprentissage automatique dans les jeux d'échecs. En effet, en 2017 DeepMind[5] avait organisé un tournoi opposant « Stockfish » le plus performant au monde avec « AlphaZero ». Le challengeur, AlphaZero, est un programme nouvellement développé avec une nouveauté et non des moindres : Il intègre l'apprentissage automatique dans l'analyse des parties de jeux d'échecs. Le résultat du tournoi fut comme un coup de tonnerre dans la sphère des jeux d'échecs avec une écrasante victoire d'AlphaZero ouvrant ainsi une nouvelle piste de recherche pour l'informatisation de ce jeu (1).

3 Le Jeu d'échecs selon théorie des jeux

La logique des jeux d'échecs et son sens stratégique et réglementaire lui ont permis d'avoir une place importante au sein de la théorie des jeux. Mais qu'en est-il de cette théorie et comment interprète-t-elle ce jeu ? (2).

3.1 Définition de la théorie des jeux

La théorie des jeux (3) et l'une des branches mathématiques qui apporte un langage formel afin de représenter des situations dites « interactives». Autrement dit, des situations où des joueurs s'opposent et où chacun d'eux prends des décisions qui affectent les choix de l'autre. Cette théorie est appliquée à plusieurs domaines tels que les jeux, l'économie, la politique et sciences sociales. Elle classifie les jeux selon plusieurs motifs basés sur l'approche de résolution comme la coopération, la mémoire, la sommation, les informations et la répétition.

3.1.1 Classification

Selon le jargon de la théorie des jeux, le jeu d'échecs est classé dans la catégorie des jeux déterministes[6], séquentiels, non coopératifs, à somme nulle[7], fini, à informations complète[8], non répétés et à mémoire parfaite.[9]

[4] 13ème champion du monde des échecs (1984-2000)
[5] Filiale de Google spécialisée en intelligence artificielle et développeursd'Alpha Zero.
[6] Qui ne répond pas à la loi de probabilité ou du hasard.

D'autre part, la représentation des possibilités stratégiques de ce jeu, se fait en forme extensive sous forme d'un arbre. Chaque branche de l'arbre définie une option ou une décision d'un joueur (déplacement de pièces). Chaque nœud représente un état (position) qui peut être intermédiaire ou final (feuille) (4)**.** Claude Shannon a estimé la complexité de l'arbre des échecs à 10^{120} parties possibles, ce qui rend le parcoure de tous l'arbre totalement impossible. C'est ainsi que plusieurs algorithmes sont implémentés pour étudier les meilleurs chemins à parcourir. D'autre part, il existe des heuristiques qui sont mises en place pour accélérer la recherche en éliminant des nœuds ou des sous arbres afin de réduire la complexité de l'arbre et d'accélérer la recherche.

3.1.2 La résolution du jeu d'échecs

L'informatisation des jeux d'échecs permet principalement de gérer et de maitriser sa complexité. Toutefois, de nos jours et avec l'explosion de la technologie et la puissance des supercalculateurs nous parlons même de sa résolution.

3.1.2.1 Principe du jeu résolu

En théorie des jeux (5), un jeu à somme nulle est dit «résolu » si et seulement si à partir de n'importe quelle position le résultat final peut être prédit en supposant que les deux joueurs jouent parfaitement leur coups.

3.1.2.2 Le Jeu d'échecs est-t- il résolu ?

Le jeu d'échecs est l'un des meilleurs sujets d'études dans la théorie des jeux combinatoire de par sa structure mathématique et ses règles logiques. D'autre part, il s'avère que c'est un jeu très complexe[10], ce qui le rendirrésolu de nos jours. Par contre selon la théorie des jeux, il est classé « partiellement résolu». En effet, dans des positions particulières tel que les finales, il existe de nos jours des logiciels qui fournissent toutes les suites et variantes principales afin d'assurer le résultat final (table des finales).

4 Aperçu des règles de la variante classique des échecs

Depuis sa création, les règles des jeux d'échecs (voir annexe A.1) ont connus plusieurs modifications jusqu'à avoir actuellement leur formes finales. Au cours de toutes ces années, beaucoup de variantes ont aussi vu le jour ayant chacune ses propres règles ce qui a enrichi d'avantage ce jeu (voir annexe A.2).

[7] Jeux où les intérêts s'opposent une victoire vaut 1 point une défaite 0 points et un nul ½ point
[8] A tous moment l'un des joueurs peut se rappeler des coups précédemment joués
[9] A tout instant chaque joueur peut se situer sur l'arbre des décisions
[10] Estimé à 10^{120} parties possibles dont 10^{40} parties n'incluent pas des coups absurdes

Le jeu d'échecs classique se joue entre deux joueurs, qui chacun à son tour réalise une succession de différents types de déplacement de pièces commençant par les blancs sur un échiquier, suivant des règles spécifiques et appliquant des thèmes tactiques et stratégiques (voir annexe A.5 et A.6).

Il existe deux sortes de pièces : blanches et noires qui se décomposent à leur tour en6 types distincts (voir annexe A.3). L'objectif principal d'un joueur est d'emprisonner le roi adverse en lui appliquant un « échec et mat ». La partie peut être alors conclue par le gain de l'un des deux joueurs, autrement elle est déclarée comme étant une partie nulle (6).

5 Notation et représentation d'une partie aux jeux d'échecs

Afin d'analyser ou reconstituer une partie d'échecs jouée, il est indispensable de maitriser la notion de notation avec les conventions qui ont été mises en place. Par la suite, après l'informatisation des jeux d'échecs un autre défie est apparu, celui de la représentation de la partie par les ordinateurs.

5.1 Localisation des cases

Afin de localiser une case sur un échiquier, nous considérons ce dernier sous sa forme matricielle (8x8) où les rangées sont énumérées de « 1- 8 » et les colonnes de « a -h ». Conséquemment, pour localiser une pièce donnée on prend l'intersection de la colonne et de la rangée correspondante à la case associée (voir figure ci-dessous).

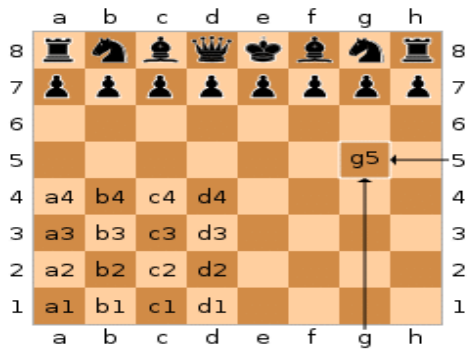


Figure I:1 Localisation des cases

5.2 Notation des coups

La notation de la partie (7) est un système créé par Philip Stamma[11]. Il permet de reproduire la partie jouée en décrivant formellement et sans ambiguïté tous les coups et types

[11] L'un des pionniers des échecs modernes et auteur du fameux livre : « The Nobel Game Of Chess » en 1755

de déplacements (8). Il existe plusieurs types de notations normalisées par la FIDE (voir annexe A.7), dont la plus connue et la plus utilisée est la Notation Algébrique Standard (NAS)

5.2.1 Notations Algébrique Standard

NAS est appliquée dans la plupart des compétitions nationales ou internationales. Elle suit des règles et symboles spécifiques (voir tableau ci-dessous). On retrouve ce type de notation en deux versions : française et anglaise.

+ *or* **ch**	Check
# *or* ++	Checkmate
O-O	Castle Kingside
O-O-O	Castle Queenside
x	Capture
e.p.	En Passant
=	Promotion
1-0	White Wins
0-1	Black Wins

Tableau I-1 Régles de "NAS"

5.3 Représentation des positions

Plusieurs méthodes sont utilisées afin de représenter un échiquier ainsi que les pièces sur un ordinateur (9). Pour ce faire, des informations de la position sont requises[12]. Dans ce qui suit, nous présentons les méthodes les plus connues et les plus implémentées dans les logiciels de jeux d'échecs (10).

5.3.1 La notation de Forsyth – Edwards

La notation de Forsyth – Edwards (FEN) (voir exemple ci-dessous) permet de reproduire la localisation des pièces d'une position jouéeen incluant d'autres informations supplémentaires. Elle présente un seul inconvénient celui de ne pas pouvoir détecter la nullité à trois répétitions consécutives d'une position (11).

La notation FEN se présente sous forme d'une chaine de caractère qui contient 6 champs principaux:

1) **Le positionnement des pièces des joueurs :** les lettres en majuscules (K,Q,R,B,N,P) respectivement roi , dame ,tour, fou, cavalier, pion , représentent les pièces blanches et les lettres en minuscules (k,q,r,b,n,p) représentent celles des noirs.
2) **La couleur des pièces du joueur ayant le trait :** pour cela « b » (black) représente le tour des noirs et « w » (white) celui des blancs.

[12] Les mêmes informations que porte la notation FEN (voir section V.4.2.A : FEN).

3) **La possibilité du roque** : si les blancs ont toujours le droit du roque ; la combinaison« K » représente la possibilité du petit roque, « Q » représente la possibilité du grand roque (respectivement en lettre minuscules pour les noirs), enfin si aucune permission n'existe, ce champs est noté par un tiret.
4) **La possibilité de prise en passant** : dans le cas d'une prise en passant la case respective est notée, dans le cas inverse ce champ est noté par un tiret.
5) **Le nombre de demi-coups consécutifs après un déplacement de pions ou une capture**.
6) **Nombre de coups**.

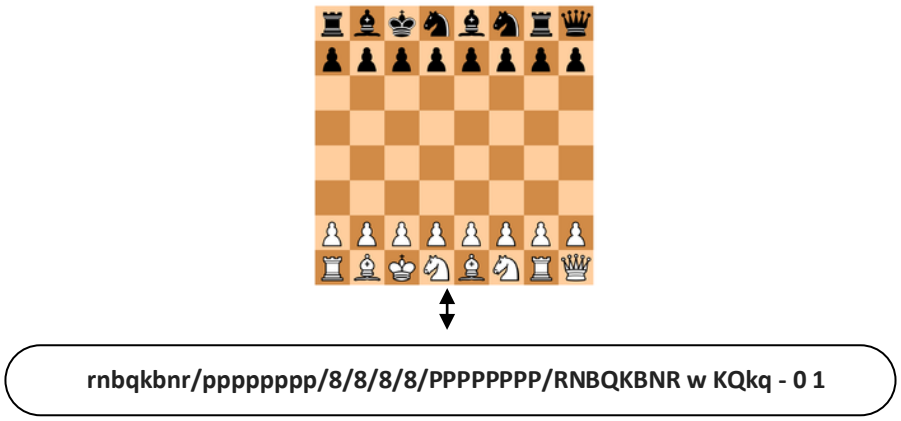


Figure I:2 Exemple de "FEN"

5.3.2 Représentation par des listes de positions

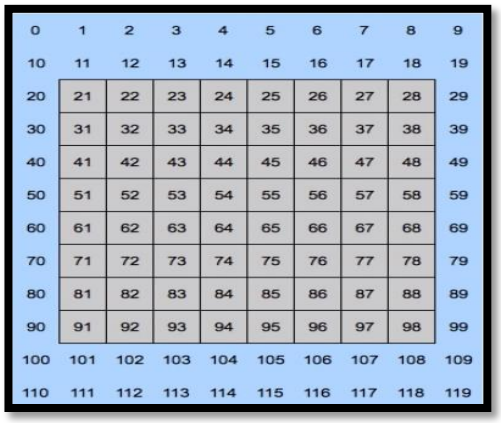
L'objectif principal de cette méthode est d'optimiser le temps d'accès aux listes afin de localiser les pièces sur l'échiquier. Pour cela, au lieu de parcourir tout un tableau de 64 entrées à la recherche d'une pièce, une liste est créée dont la fonction est d'associer à chacune des 12 pièces noires et blanches sa case sur l'échiquier.

5.3.3 Représentation par tableau

C'est la méthode la plus intuitive. Elle permet de représenter l'échiquier par une matrice (8x8) ou sous forme d'un tableau de 64 cases où le premier indice fait référence à la case « a8 » et le dernier à la case « h1 ».

La prochaine étape consiste à attribuer des identifiants pour les 6 types de pièces des deux couleurs dans les entrées du tableau selon leurs positions sur l'échiquier. Cependant, un problème est aperçu lors de la génération des coups où une boucle se met en marche et parcoure toutes les entrées du tableau en prenant encomptes parfois des cases de déplacement illégales qu'il considère comme étant légales. Par conséquence, non seulement la génération des coups sera lente mais aussi donnera des résultats incorrects.

La solution à ce problème est de redimensionner la matrice initiale et de rajouter un cadre « padding» afin de limiter la recherche et de l'arrêter quand la case recherchée est incluse dans ce cadre. De ce fait, la nouvelle dimension utilisée sera (10x12) (voir figure ci-dessous).



NB : les deux rangées de 0 à 9 et de 110 à 119 sont ajoutées pour la vérification des coups légaux des cavaliers

Figure I:3 Méthode de tableau

5.3.4 Méthodede 0x88

C'est une méthode qui permet de représenter une position en utilisant un tableau de 128 entrées équivalent à une matrice de (16x8) représentée généralement par deux matrices de (8x8) l'une à côté de l'autre (voir figure ci-dessous). La matrice de droite (M1) représente les cases de l'échiquier et celle de gauche (M2) est utilisée pour vérifier les cases illégales qui sont hors de la matrice M1.

L'idée principale de cette méthode vient du fait que si on représente sur 8 bits les indices de la matrice M1, on remarque que le 4^{éme} (ou8^{éme}) bit est activé contrairement aux indices de la matrice M2 où ces positions spécifiques de bits sont toujours désactivées. En utilisant ce principe, et le nombre 136 (0x88en hexadécimale) on pourra vérifier si une case donnée appartient à M1 en utilisant un ET logique entre son indexe et 0x88. Ainsi, si le résultat est différent de zéro cela impliquerait que la case est hors de l'échiquier.

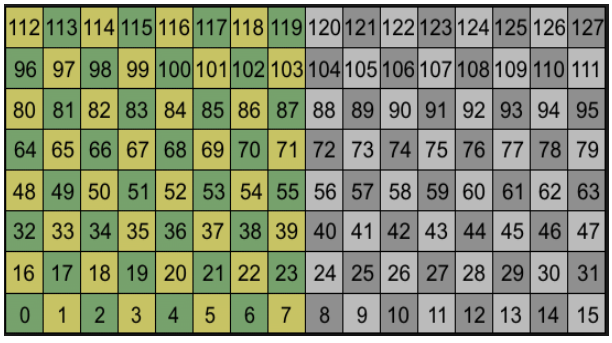


Figure I:4 L'indexation par la méthode 0x88

5.3.5 Représentation par Bitboard

Il s'agit de la méthode la plus utilisée dans les programmes de jeux d'échecs. Elle permet de représenter non seulement la localisation des pièces, mais aussi d'autres informations importantes tel que : les menaces, les défenses, pièces attaquantes et les pièces menacées. Tous cela avec une structure de bas niveau qui la rend plus efficace que les autres méthodes. Elle tire profit des capacités des ordinateurs à représenter un entier long[13] sur 64 bits qui est le même nombre de cases de l'échiquier (voir figure ci-dessous).

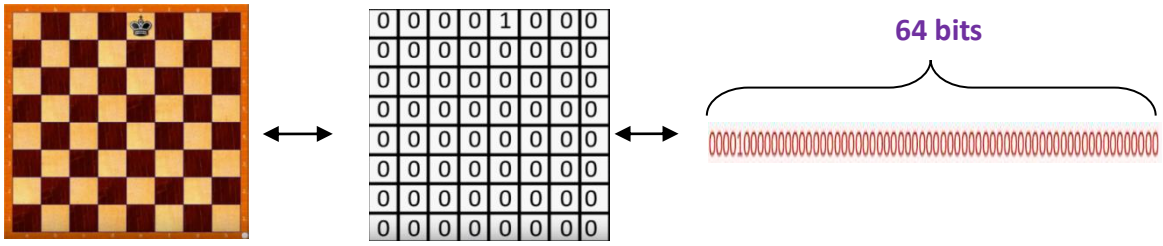
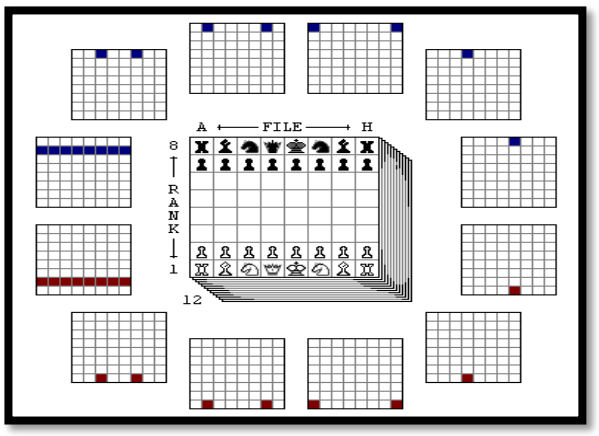


Figure I:5 -Bitboard- conversion d'une position en un entier long.

Ainsi pour chaque position, Bitboard génère douze nombres qui représentent respectivement les types de pièces de chaque joueur comme la montre la figure ci-dessous. Cependant, un seul inconvénient pour cette méthode qui est le coût des mises à jour des positions qui est très élevé.

Il existe d'autres sous variantes complexes de cette méthode pour représenter d'autres informations outre que la localisation, nous parlons alors de rotation de bitboards.



[13] Utilisé pour représenter un entier qui prend plus de mémoire sur une machine qu'un entier standard.

Figure I:6Représentation par bitboard.

5.3.6 La représentation de Huffman

Cette représentationest l'application du « codage de Huffman [14]» sur les programmes des jeux d'échecs. Elle permet principalement de représenter l'échiquier avec le moins de bits possibles. Une table de correspondance est mise en œuvre pour normaliser le codage des pièces (voir tableau ci-dessous).

Cette méthode est la plus utilisée pour des représentations réutilisables à long termes comme les positions d'ouvertures et de finales, car elle s'avère très couteuse en termes de CPU.

Tableau I-2 Table de correspondance –codage Huffman-

Type de case	Représentation
Couleur (c)	1 bit
Roi	11111c
Dame	11110c
Tour	1110c
Fou	1100c
Cavalier	1101c
Pion	10c
Case vide	0
Petit roque	4 bits
colonne de la prise en passant	3bits
Règle des 50 coups	7 bits
Trait à jouer	1bit

5.4 Représentation de parties

Le format PGN (de l'anglais *Portable Game Notation*) (12) est le format le plus utilisé pour la notation des parties pour ordinateurs. Elle est constituée de deux parties : l'entête « *Header* » et le corps « *Body* ».

L'entête contient des informations globales sur la partie comme : nom du tournois, nom des joueurs, la date, numéro de la partie et son résultat final. Il est suivi par un corps qui contient la suite des coups utilisant la notation SAN.

[14] Code qui permet de compresser les données créé par David Albert Huffman.

6 Au cœur d'un programme de jeux d'échecs

La figure ci-dessous représente les trois étapes fondamentales et primordiales pour le bon fonctionnement d'un programme de jeux d'échecs à savoir : la génération de coups, la recherche et l'évaluation. Autrement dit, c'est un système coopératif qui reçoit une entrée (position) et génère une sortie (meilleur(s) coups).

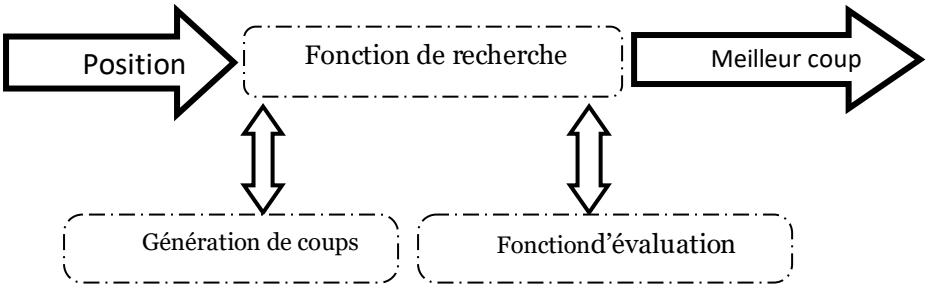


Figure I:7 Fonctions Principales d'un programme d'échecs.

6.1 Fonction de génération de coups

La génération de coups est une des fonctions les plus importantes notamment pour la rapidité du programme. Elle prend en paramètre une position communiquée par la fonction de recherche (voir section 6.2) où elle utilise principalement sa représentation et les règles de déplacement de pièces, afin de générer une liste des coups possibles qui peuvent être joués par l'un des joueurs ayant le trait.

Le jeu d'échecs est un jeu très complexe qui contient plusieurs contraintes et règles à respecter. Ainsi, la liste finale des coups générée s'accroit exponentiellement au cours du jeu. Ainsi, la génération doit être organisée et rigoureuse afin d'éviter un débordement ou des bugs lors de la recherche du générateur des coups.

6.1.1 Types de générateur

Le choix du type de générateur diffère selon la représentation échiquéenne utilisée. Nous distinguons principalement deux types de générateurs de coups : selon la légalité et selon les coups générés. (13)

6.1.1.1 Selon la légalité

1) Coups pseudo-légaux

Ce type de génération est le plus simple à réaliser et est utilisé notamment par Stockfish. Par contre, il ne se soumet pas aux règles de jeux d'échecs vu que ça consiste à générer la liste de tous les coups possibles (déplacements élémentaires) pour chacune despièces en incluant un sous ensemble de « faux coups » comme des roques illégaux (roque en passant par une case attaquée, roque sur échec).

Avec l'utilisation de ce type il faut impérativement tester la validité de la position de l'échiquier après avoir exécuter un des coups de la liste résultante et ne maintenir que les positions légales en éliminant les coups qui mettent la position dans un état invalide. Cependant ce processus se révèle être très couteux en mémoire ainsi que le temps d'exécution CPU.

2) Coups légaux

La génération basée sur les coups légaux consiste à générer directement la liste de tous les coups possibles qui gardent la validité de la position après leurs exécutions.

6.1.1.2 Selon les coups générés

Cette approche est une optimisation de l'approche qui génère les coups légaux. Son principe consiste à réordonner et filtrer les coups de la liste finale résultante, principe qui sera par la suite très bénéfique lors de l'étape de la recherche. L'une des applications de cette méthode, est celle de « la recherche quiescence » (voir section 6.2).

Dans d'autres cas, cette méthode utilise une approche sélective en choisissant des coups qui sont les plus susceptibles à être jouéset les indexer en premier, comme les captures de pions, « killer move » (voir section 6.2), les promotions et aussi favoriser la génération des coups qui mettent le roi adverse en échec, et rajouter à la fin le reste des coups normaux. Une heuristique qui minimise le temps d'exécution et est considérée comme la meilleure approche de génération de coups.

6.1.2 Génération de coups pour les pièces simples

La génération de coups pour les pièces simples suit deux règles fondamentales:

1. Si la case d'arrivée est vide alors sa validité et la permission du déplacement sont vérifiées.
2. Si la case contient une pièce de même couleur, il s'agit d'une capture autrement la génération s'achève.

6.1.3 Génération de coups pour les coups spéciaux

Un moteur de jeu d'échecs doit impérativement être capable de générer des coups spéciaux comme le roque, la prise en passant et la promotion. Au premier lieu, le roque est l'un des coups les plus complexes à générer. Il réunit le déplacement de deux pièces à la fois : la tour et le roi. Ainsi, avant la génération de ce coup il est indispensable de vérifier les permissions suivantes : le roi n'a pas été déplacé auparavant, le roi n'est pas en position d'échec et enfin le roi ne doit pas traverser une case attaquée.

Au deuxième lieu, la légalité de la génération de la prise en passant est vérifiée en fonction du coup précédemment joué.

Enfin pour la promotion, certains programmes utilisent simplement l'heuristique disant que généralement, la dame est la plus choisie. Dans le cas inverse, un autre tableau est généré où les 4 pièces possibles sont sauvegardées à savoir promotion : cavalier, fou, tour ou dame.

6.1.4 Génération des coups pour les pièces glissantes

La génération de coups (14) dans le cas des pièces glissante (fou, tour, dame) se fait généralement en utilisant la méthode nommée « deltas » (voir annexe1.7). Cette dernière, s'avère très couteuse vu le nombre de calcule à exécuter pour chaque pièce pour déterminer les cases de déplacements possibles et surveiller constamment les pièces adverses. Pour cela, les meilleurs programmes du monde comme Stockfish, Houdini ou même Komodo ont mis en place deux méthodes afin d'accélérer la génération pour ce type de pièces.

6.1.4.1 Génération à base de bitboards magiques

En utilisant le bitboard magique, qui est un algorithme de hachage appliqué à la représentation bitboard, on pourra exécuter des décalages à droit en suivant un algorithme bien précis et ainsi enregistrer les cases d'attaques des pièces dans une base de données indexée par le hash calculé.

Ce processus est appliqué pour toutes les cases de l'échiquier et pour n'importe quelles pièces. De ce fait, après avoir enregistré ces données, cela augmenterait selon les développeurs la rapidité de la génération de 20-25%. Par conséquence, la ligne d'attaque est exécutée qu'une seule fois. Cette rapidité de génération implique aussi d'avantage de coups générés.

Citons l'exemple du programme shallow-blue qui a connu, après l'implémentation de bitboards magiques, un gain en rapidité allant jusqu'à 30% lors du calcul de perf(5)[15]. Aussi pour les mêmes taches, le nombre d'instruction s'est diminué de 47 à 15 instructions. Enfin, la vitesse de la génération de coups est passée de 15 millions de nœuds par secondes à 20 millions.

[15]Il s'agit d'une fonction de débogage qui prend en paramètre la profondeur de recherche et parcoure l'arbre de génération de coup en calculant le nombre de feuilles et en mettant de coté les bugs pour qu'à la fin renvoyer les statistiques de performances.

6.1.4.2 Génération à base de bitboards magiques noirs:

Cette technique récemment développée, représente une optimisation de la technique précédente. Elle permet principalement de réduire la taille de la table de hachage stockée.

6.2 Fonction de recherche

La recherche (15) est l'un des processus les plus importants qui font la différence entre la qualité d'un programme à un autre. C'est une étape où les sorties de la génération des coups seront utilisés pour parcourir un arbre de recherche à la quête du meilleur coup et donc d'une évaluation favorable.

6.2.1 Algorithme de Monte-Carlo

Monte Carlo Tree Search (MCTS) (16) est un algorithme de recherche utilisant une méthode heuristique pour accélérer l'exploration de l'arbre de recherche. Sa spécificité réside dans sa méthode implémentée lors de l'exploration de l'espace de recherche qui se fait aléatoirement en tenant compte des recherches antérieures qui sont sauvegardées en mémoire. Ainsi, il pourra deviner le coup le plus susceptible d'être le meilleur dans une position donnée.

L'un des avantages de cette méthode est qu'il n'a pas besoin d'une fonction d'évaluation pour fonctionner mais juste des règles du jeu. Aussi, ses résultats sont satisfaisants et sont similaires à ceux d'alpha-béta (voir section 6.2.3). D'autre part, il présente des inconvénients comme des interruptions soudaines et aussi des sous-estimations de coups, ce qui élimine des sous arbres contenant la meilleure suite de coups.

Le processus de fonctionnement de MCTS (voir annexe A.9) se présente sous forme de 4 étapes qui se répètent jusqu'à avoir une estimation de la meilleure valeur. Ces étapes sont les suivantes :

1) **Sélection :** les nœuds parcourus sont choisi initialement de telle sorte à maximiser une fonction spécifique.
2) **Expansion :** exploration d'un nouveau nœud.
3) **Simulation :** jouer une variante jusqu'àla fin du jeu et ainsi avoir un score.
4) **Propagation en arrière :** retourner le score trouvé vers le nœud père et additionner le score avec ceux de tous les nœuds prédécesseurs et ancêtres.

6.2.2 Algorithme Minimax

L'algorithme Minimax (17) est un algorithme de recherche de type exploration en profondeur, idéal pour les jeux à somme nulle. Il se base sur l'idée que les deux joueurs jouent

leurs meilleurs coups. On distingue deux types de nœuds qui se succèdent: un nœud « Max » et un nœud «Min» de telle sorte que chacun essaye de maximiser ses gains et donc minimiser ceux de son adversaire et où l'évaluation est réalisée par rapport à un adversaire choisis.

L'Algorithme Negamax est une simplification et une variante d'optimisation de Minimax qui utilise une évaluation de point de vue du joueur ayant le trait. (voir annexe A.9)

6.2.3 Algorithme Alpha-Beta

L'algorithme Alpha-Beta (18) est une optimisation de Minimax qui prend en considération deux paramètres additionnels :

- **Alpha (α)** : appelé aussi « *lower bound* ». Il représente la meilleure perspective (score) du nœud Max. Sa valeur est initialisée avec le pire score possible (-∞) et ainsi l'objectif est de le maximiser.
- **Beta (β) :** appelé aussi « *upper bound* ». Il représente la meilleure perspective (score) du nœud Min. Sa valeur est initialisée avec le pire score possible (+∞) et ainsi l'objectif est de le minimiser.

La spécificité de cet algorithme est qu'il réduit l'espace de recherche de l'arbre avec un système d'élagage appelé aussi : « coupure ». Ainsi, des branches ou des sous arbres sont ingnorés, ce qui réduit le temps de recherche. La coupure survient quand on sait que le nœud courant ne peut pas retourner une valeur supérieure (inférieur dans le cas Min) que les résultats trouvés auparavant.

Il existe deux types de coupures : coupures Alpha (au niveau du nœud Max) et des coupures Beta (au niveaudu nœud Min). Comme l'algorithme Minimax, une version de Negamax existe pour Alpha béta (voir annexe A.10)

6.2.4 Optimisation de la recherche

Plusieurs heuristiques et méthodes sont appliquées à l'arbre de recherche afin d'optimiser le temps d'exploration. Cela permet d'accroitre son espace de recherche et de découvrir d'autres branches inexplorées, chose qui fait la différence entre la qualité des programmes de jeux d'échecs. Nous présentons dans ce qui suit quelques-unes de ces méthodes.

6.2.4.1 Algorithme Negascout

NegaScout (19) est un algorithme de type negamax utilisé pour optimiser l'algorithme alphabeta. L'idée derrière cette méthode réside dans le fait que dans la plupart des cas, après le premier coup, le reste entraineront des coupures ce qui permet d'ignorer leurs évaluation.

De ce fait avec NegaScout, la recherche du premier coup ce fait avec une fenêtre ouverte et le reste des coups se feront avec une fenêtre nulle en faisant abstraction des mises à jour d'alpha. (voir annexe A.11)

Principal Variation Search (PVS) est pratiquement similaire avec NegaScout à la différence que certaines implémentations de PVS attendent une amélioration d'alpha avant d'utiliser la fenêtre nulle sur les nœuds.

6.2.4.2 La table de transposition

La table de transposition (20) est une méthode qui tire son principe du fait qu'aux échecs plusieurs positions identiques sont résultantes de différentes suites de coups. Ainsi pour minimiser le temps de recherche, il suffit d'enregistrer une position dans la table de transposition après avoir calculéson évaluation. Par conséquence, avant chaque évaluation, une vérification est réalisée afin de tester son existence dans la table de transposition. Outre que l'évaluation, plusieurs autres informations sur la parties sont enregistrées comme : la profondeur, type de coupure, la suite de coups etc.

Les parties sont identifiées en utilisant « une clé zobrist » calculée avec une fonction de hashage : *zobrist hashing function* (voir annexe A.8). Vu le plus nombre de positions existantes, et pour éviter des problèmes de mémoires, il est indispensable de limiter la taille de la table de transposition. Pour cela étant donnée une table de transposition avec une capacité (c) et une partie avec un hash (h) alors l'indexation (i) de cette partie dans la table est calculée comme suit : **i = h modulo c.**

Dans le cas de l'insertion d'une nouvelle partie à la $i^{ème}$ position coïncide avec l'existence d'une position **(collusion)**, l'une des règles suivante est appliquée pour résoudre ce conflit :

- **Toujours remplacer :** l'ancienne partie est toujours remplacée par la nouvelle.
- **Ne jamais remplacer :** l'ancienne partie reste toujours dans la table.
- **Remplacement par profondeur :** ne pas remplacer que si la nouvelle profondeur est supérieure à l'ancienne.
- **Remplacement par (Profondeur + toujours) :** ce cas est le plus efficace mais le plus dure à mettre en œuvre. Il consiste à sauvegarder deux entrées pour chaque position. Ainsi, si la condition de la profondeur est vérifiée on la remplace dans la première entrée, dans le cas inverse on la remplace toujours dans la seconde entrée.

6.2.4.3 Table des finales

Il s'agit d'une grande base de donné contenant des positions (où le maximum de pièces est 5 voir 7 pièces) tirées des finales en leur associant leur évaluations ou la distance qui les séparentde l'échec et mat. Ainsi, en utilisant les schémas de ses positions, le programme se comportera comme un grand maitre lors des finales en essayant d'évité ou à atteindre les positions sauvegardées. L'avantage avec cette méthode est qu'elle réduit le temps de la recherche mais détient un inconvénient qui est le besoin d'un large espace mémoire vu sa grande taille. C'est pour cette raison qu'elles sont devenues compressée à l'image des plus répandues de toutes : la tablesbase de Nalimov où aussi Syzygy. (21)

Tous comme les tables de finales, il existe aussi des tables d'ouvertures qui contiennent les ouvertures les plus connues avec leurs suites et leurs évaluations.

6.2.4.4 Sélectivité

Il s'agit d'une heuristique qui consiste à chercher les branches qui sont les plus susceptibles de faire partie de la variation principale. Selon les résultats, cela aide à éliminer beaucoup de branches sans intérêts.

6.2.4.5 Extension

Généralement, lors de la recherche à une profondeur donnée, on trouve une position qui n'est pas stable où il existe plusieurs échanges de pièces, des menaces ou des promotions (problème d'horizon). Pour cela, pour bien comprendre la position, la recherche est ainsi étendue généralement de 1 à 5 profondeurs de plus.

6.2.4.6 Heuristique à mouvement nul

Il s'agit de l'une des heuristiques les plus importantes vu les grands nombres de nœuds qu'on peut ignorer avec. Elle utilise un principe qui contredit les règles de jeux d'échecs celui de passé son tour pour l'adversaire. Ainsi, si l'évaluation après le passage du tour est largement inférieureà sa valeur précédente, la position principale est évaluée, sinon si elle reste égale à l'originale dans ce cas aucune évaluation ne sera réalisée. Malheureusement cette idée est risquée parfois, car une mauvaise compréhension de la position peut induire à l'ignorance de son évaluation. (22)

6.2.4.7 Approfondissement itératif

C'est une stratégie de recherche qui consiste à essayer la variation principale de l'itération précédente pour la prochaineen tant que chemin le plus à gauche. Ainsi, on n'a pas besoin de spécifier une profondeur donnée. Aussi, lors de la recherche profonde, cette dernière utilise

les positions de la table de transposition afin de simplifier l'analyse etla valeur finale trouvée sera utilisée pour ajuster la fenêtre d'aspiration dela prochaine recherche.

6.2.4.8 Fenêtre d'aspiration

Cette méthode (23) est une optimisation d'alphabeta afin de réduire le temps de recherche. Elle consiste à deviner l'évaluation d'un nœud à partir de la dernière itération selon l'approfondissement itératif. Ainsi, il va redéfinir la fenêtre de recherche par défaut à savoir (-∞, +∞) en choisissant une fenêtre de type (valeur-fenêtre, valeur+fenêtre) où valeur fait référence à la valeur attendue, et la fenêtre pour la déviation attendue de cette valeur.

De ce fait avec cette heuristique, si la valeur attendue est trouvée dans cette dernière fenêtre alors beaucoup de temps d'exécution sera réduit. Dans le cas inverse, deux approches sont utilisées : soi la recherche est refaite avec la fenêtre par défaut, dans ce cas on sera sur d'avoir la valeur de l'évaluation, ou alors à partir de la dernière fenêtre on essaye d'élargir petit à petit ce domaine de définition jusqu'à avoir un résultat. Elle peut être couteuse parfois mais s'avère intéressante dans d'autres cas.

6.2.4.9 Ordonnancement des coups

C'est une méthode (24) très importante qui permet de simplifier l'arbre de recherche. Elle consiste à réordonner la liste des coups légaux générés en plaçant les meilleurs coups en premiers. La notion des meilleurs coups dans ce cas n'est pas ceux qui maximisent les gains mais principalement ceux qui réduisent la taille de l'arbre ou facilitent la suite de la recherche. Conséquemment, ils sont classés selon le nombre de bonus qu'on leur a affecté selon leurs importances. Ainsi, les coups sont triés par rapport à leur bonus (voir tableau ci-dessous).

Type de coups	Bonus
Le même coup à la même profondeur dans la meilleure variation de l'itération précédente	+20000-profondeur
Killer move	+10000-profondeur
Pièce dans une capture	Si Pièce attaquante :-VP /10[16] Si pièce capturée : +VP
Qui capture la dernière pièce déplacée	+1010
Coups issu de la table de transposition	+1000

Tableau I-3 les priorités dans l'ordonnancement des coups

[16] VP : valeur de la pièce

6.2.4.10 L'heuristique des coups tueurs

C'est une méthode qui améliore le fonctionnement de l'ordonnancement des coups. Elle se base sur la probabilité que si un coup tueur cause une coupure pour une branche d'une profondeur donnée, il serait alors susceptible de causer une coupure dans une autre branche de la même profondeur. C'est ainsi qu'on garde dans un tableau un à trois coups tueurs de chaque profondeur de l'arbre déjà explorée pour les réutilisés en premier avant les autres coups lors d'une recherche à la même profondeur.

6.3 Fonction d'évaluation statique

La fonction d'évaluation (25) est une étape très importante qui intervient sur les feuilles de l'arbre de recherche. Elle permet d'analyser une position selon plusieurs critères et lui attribuer une valeur finale approximative qui définira son appréciation vue de l'un des deux joueurs (les blancs pour le cas de notre projet) sous forme d'une fonction linéaire définit par la somme d'une évaluation matérielle et d'une évaluation positionnelle.

6.3.1.1 Evaluation matérielle

C'est une évaluation qui permet de calculer la différence entre le total de pièces des deux joueurs avec l'unité appelée « centipawn » qui vaut la valeur du pion (1) multiplié par 100. Plus formellement nous pouvons l'exprimer ainsi :

$$\text{Évalmat}=\sum VPBlanches - \sum VPNoires.$$

Équation I-1 Evaluation matérielle

6.3.1.2 Evaluationpositionnelle

En plus de l'évaluation matérielle, il existe une évaluation positionnelle qui rassemble plusieurs informations supplémentaires afin de cerner et de bien comprendre la position et où nous attribuons des bonus et des malus à chacun des joueurs par rapport à leur respect pour les critères suivants :

1) Evaluation spatiale

L'espace est considéré comme étant un avantage vu que le joueur ayant plus d'espace aura plus de mobilité de pièceset de liberté sur l'échiquier. Aussi, il aura moins de chance d'être en position d'échecs et mat dans les coups suivants.

2) Sécurité du roi

On attribut un bonus pour le joueur ayant un roi bien sécurisé, de même un malus est attribué pour le côté ayant un roi déroqué où mal défendu.

3) Structure de pions

Des malus attribués pour les pions doublés ou isolés ou arriérés vu que les pions seront facile à capturer

4) Position des pièces

Des bonus pour les pièces qui contrôlent le centre où des cases stratégiques. Dans le cas inverse, des point seront retirés pour des pièces mal positionnées comme l'exemple des cavaliers où l'on dit : « cavalier au bord, cavalier mort ».

5) Avantage de temps

Il s'agit de l'avantage d'avoir l'initiative d'attaque et de développement.

6.3.1.3 Evaluation Quiescence et problème d'horizon

Cette évaluation (26) est une fonction complémentaire à l'évaluation statique connue précédemment. Elle permet de résoudre le problème de l'horizon et donc d'avoir une meilleure estimation de l'évaluation de la position. Elle est inspirée de la réflexion humaine qui cherche à étendre l'analyse pour avoir des positions sans menaces.

- **Problème d'horizon et solution**

Parfois lors d'une exploration de l'arbre de recherche à une profondeur spécifique (p), on s'arrête sur une position qui n'est pas très claire tactiquement. Pour une meilleure compréhension de la position, on étend la recherche jusqu'à ce que les échanges tactiques et combinaisons (les captures) se terminent. Outre que les combinaisons, dans quelques programmes, tous les coups (ou suite de coups) qui suivent sont même acceptés qui peuvent changer l'évaluation de la position finale comme les promotions ou les échecs.

7 L'avenir des jeux d'échecs en informatique

Depuis sa découverte, le jeu d'échecs a gravé plusieurs échelles et s'est vu développé par les grands pionniers des siècles passés. Depuis ces dernières années et avec l'arrivée de l'informatique et son application à la théorie des jeux, des programmes performants se sont développés utilisant plusieurs algorithmes et méthodes de recherches et d'évaluations statiques. Ces programmes ont vu surpassés les meilleurs joueurs d'échecs notamment Stockfish, le plus puissant de tous. Mais on ne voyait à travers ces programmes que des algorithmes et des instructions qui n'ont pas un sens de créativité ou de gestion d'ambigüités. De ce fait qu'en est-il del'avenir des échecs ? Peut-il être un jour totalement résolu ainsi que tous les autres jeux combinatoires ?

En effet, avec cette nouvelle aire technologique de pointe à savoir l'intelligence artificielle et avec notamment l'apprentissage profond et ses réseaux de neurones, la quête à la résolution de ce jeu et d'autres à repris espoir.

Un espoir qui a commencé par AlphaGo une création de deepMind basée sur l'IA et qui a terrassée Lee sedol[17] où ce dernier était fasciné par son jeu et son « intelligence», même si qu'en plus de la partie IA ils ont inclus une large base de données de parties déjà jouées.

L'année 2017, a connu la plus grande création de Google qui est Alpha Zero (Zéro pour zéro prérequis intégrés) et qui était totalement autodidacte. Elle a réussi à « écrasé » Stockfish ainsi qu'AlphaGo. C'est un programme qui fait preuve de créativité et d'une vision de jeu différente de celle qu'on connait. Aussi, son jeu contredit parfois les règles des échecs même les plus basiques et cela se présente dans ces estimations de positions lors de l'évaluation où il sous-estime les valeurs des pièces et ne leur offre pas les mêmes valeurs standards. Ainsi, il n'hésite pas dans beaucoup de ses parties à faire des sacrifices imprévisibles pour l'humain et celajuste pour améliorer sa position. Il tire cette force de l'apprentissage automatique qui se base sur trois principes : « self-play », algorithme MCTS et les réseaux de neurones.

Enfin nous constatons que l'application de l'IA sur les jeux d'échecs et la théorie des jeux généralement, permettra de redécouvrir les jeux et de résoudre d'autres, mais aussi cela permet de développer le domaine de la théorie des jeux.

8 Conclusion

Dans ce premier chapitre, nous avons vu les principes du jeu d'échecs notamment sa représentation dans les programmes informatiques, auxquels nous avons présenté leurs trois constituants principaux en l'occurrence : la fonction de recherche, la génération de coups et la fonction d'évaluation.

Enfin, nous avons introduit le principe de l'intelligence artificielle appliquée aux échecs et plus précisément le domaine de l'apprentissage automatique qui fait fasciner tous les informaticiens de nos jours.

Nous avons vu en particulier, que l'une des principales tâches d'un moteur de jeux d'échecs est l'évaluation statique des positions. Celle-ci est réalisée en générale avec un algorithme explicitement programmé. Cependant, de nos jours, l'avènement de

[17] Champion du monde du jeu de Go

l'apprentissage automatique offre de nouvelles perspectives. En effet, il est possible d'utiliser un algorithme, cette fois-ci non explicitement programmé, pour réaliser l'évaluation dynamique de la position.

Ainsi, dans le prochain chapitre nous exposerons beaucoup plus en détail l'apprentissage automatique, ses principes, son fonctionnement et aussi celui des réseaux de neurones artificiels.

Chapitre 2: Apprentissage automatique

1 Introduction

Les informaticiens ont longtemps été attirés par l'intelligence humaine et ont toujours été tentés de reproduire sa créativité et de développer des programmes faisant face à des situations ambigües.

La théorie des jeux et notamment les jeux d'échecs sont des domaines qui suscitent l'intérêt des chercheurs en intelligence artificielle à travers le monde. Ces dernières années ont vu le développement de plusieurs moteurs d'échecs extrêmement performant, dotés des dernières technologies de pointe en l'occurrence : l'apprentissage automatique.

Dans ce chapitre nous expliquons la notion d'apprentissage pour les machines. Par la suite, nous présentons les différents principes et fonctions de bases dans le domaine de l'apprentissage automatique ainsi que les réseaux de neurones artificiels. Enfin et avant de conclure, nous présentons quelques défis et solutions à des problèmes communs de programmation rencontrés lors de la mise en œuvre de l'apprentissage automatique.

2 Généralités sur l'apprentissage automatique

2.1 Origines

L'origine de l'apprentissage automatique (27) remonte aux temps du pionnier de l'informatique Alain Turing avec son article « Ordinateur et Intelligence » (28) où il aborde pour la première fois des problèmes liés à l'intelligence des machines en particulier le test de Turing[1].

Avant même les travaux de Turing, plusieurs informaticiens ont essayé de découvrir les concepts liés à l'IA à l'instar des premiers développeurs de la théorie des réseaux de neurones. Or, le premier à avoir utilisé le terme « apprentissage automatique » est bel et bien Arthur Samuel qui l'a appliqué à la théorie des jeux (jeu de Dames).

Outre la théorie de l'apprentissage automatique, la mathématique représente le cœur battant de cette branche de l'IA (probabilité, statistique, algèbre linéaire, calculus). En parallèle la puissance de calcul des ordinateurs était très limitée l'ors de sa découverte ce qui a arrêté sa progression. Depuis le l'an 2000, plusieurs avancées ont été apportées, notamment en réseaux de neurones, ce qui a diversifié même les domaines d'applications et de testes comme en bioinformatique, chimo-informatique, Armé, médecine, cosmologie, robotique,

[1] Un teste qui vise à évaluer la performance et la similitude de la machine intelligente avec l'humain en les mettant en coversation à l'aveugle.

économie, vision (classification d'objet, reconnaissance d'image, etc.) et aussi intégrées dans nos Smartphones ce qui à pousser la GAFAM (Google, Apple, Facebook, Amazon, Microsoft) à s'investir dans ce domaine.

Le Big Data[2] a donné une autre dimension à l'apprentissage automatique et a beaucoup contribué dans son développement depuis sa découverte vu le nombre et la qualité de données fournies contribuant à l'amélioration des performances de ces machines intelligentes.

2.2 Définition

L'apprentissage automatique est une branche de l'intelligence artificielle qui permet de donner aux machines et aux programmes développés la capacité d'affronter des situations ambigües auxquelles ils n'ont pas été explicitement programmés. Cela est réalisé en utilisant les différentes méthodes d'apprentissage se basant généralement sur des données (graphes, images, matrices…) et des méthodes statistiques afin de tirer profit de leur expérience l'ors des entrainements. Par conséquent, le modèle sera capable de généraliser le problème et de faire des prédictions sur de nouvelles données et ainsi acquérir une adaptabilité face au changement de l'environnement.

2.3 Fonctionnement global

Le processus d'apprentissage (voir figure ci-dessous) vise à produire et à optimiser la performance du model conceptuel de la machine. Il se présente en un nombre d'étapes séquentielles qui, misà part la collecte des données initiales, serépètent indéfiniment jusqu'à avoir la performance optimale souhaitée avant l'étape de l'exploitation. Ces étapes sont les suivantes :

1. **Collecte de données :** elle consiste à préparer une large base de données organisée selon le type d'apprentissage (voir types d'apprentissages). Par convention, ces données sont organisées généralement de la façon suivante (29) : 20% pour les tests, le reste est décomposéainsi : 80% pour les entrainements, 20% pour la validation.
2. **Entrainement** : cette étape est la plus cruciale. Elle représente l'étape où les paramètres du modèle sont ajustés afin d'optimiser ses résultats et réduire l'erreur en utilisant principalement l'algorithme du gradient descendant et d'autres méthodes (détaillées dans les sections suivantes).

[2] Domaine qui permet de collecter et de manipuler des données volumineuses et en extraire les informations importante afin d'être exploiter par la suite.

3. **Validation :** c'est l'étape qui permet d'évaluer les performances de l'étape d'entrainement. Elle consiste à calculer la différence entre les résultats de prédictions et les valeurs respectives attendue, ce qui donne une vue sur l'évolution et la progression du modèle.
4. **Test** : cela consiste à mettre le modèle face à des données sur lesquelles il ne s'est jamais entrainé auparavant. De ce fait, les performances du modèle sont évaluées par rapport aux prédictions réalisées lors de cette étape.

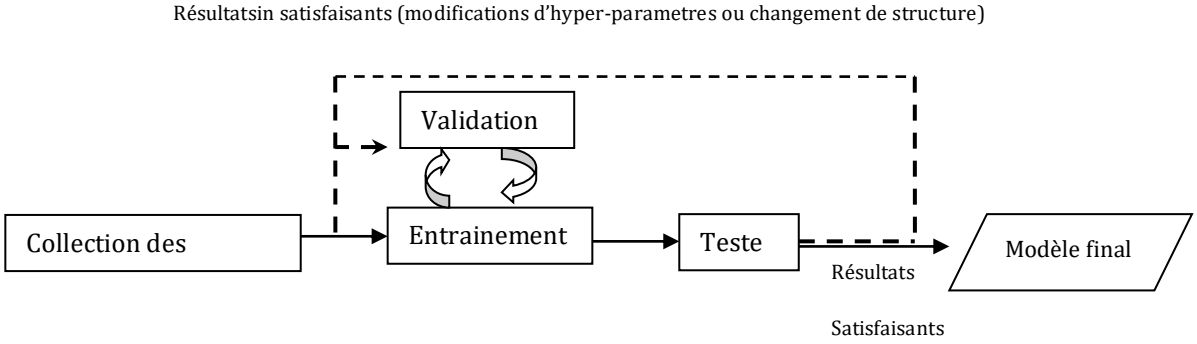


Figure II:1 Etapes pour la construction d'un modèled'apprentissage automatique

2.4 Types d'apprentissage automatique

Il existe plusieurs types d'apprentissage (30), nous présentons dans ce qui suit les plus connus et les plus utilisés :

2.4.1 Apprentissage supervisé

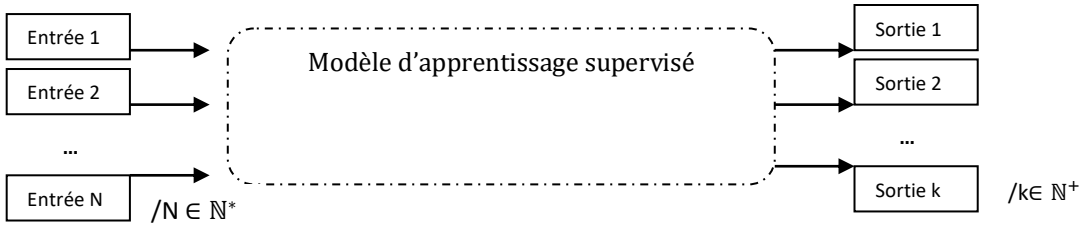


Figure II:2 Schéma global de l'apprentissage supervisé

Ce type d'apprentissage est de loin le plus connu et le plus utilisé. Il est caractérisé par un ensemble de données« X_1, X2,…, Xn/ $X_i \in \Re^t$» auxquelles sont associés respectivement des étiquettes de sortie « Y_1, Y2,….,Yn/ $Y_i \in \Re^{t'}$». Un couple (X_i, Y_i) est appelé exemple et K=$\bigcup_{i=1}^{n}(Xi, Yi)$ est appelé ensemble d'entrainement. De ce faite, l'objectif est de pouvoir

trouver et optimiserune fonction (F) tel que : $Yi=F(X_i)$ + bruit[3] et cela passe par la réduction de la fonction de couts (voir section fonction de coût + annexe B.5).

L'apprentissage supervisé se décompose en deux catégories : Classification et Régression

2.4.1.1 *Classification*

La classification supervisée se caractérise par des sorties finies sous forme de catégories (discret) allant de binaires jusqu'à n-aires. La fonction (F) est appelée « classificateur » et consiste à trouver la probabilité qu'une donnée en entré corresponde à l'une des classes finales en trouvant la meilleure « frontières de décision » possible qui existe pour la séparation de données (voir figure ci-dessous).

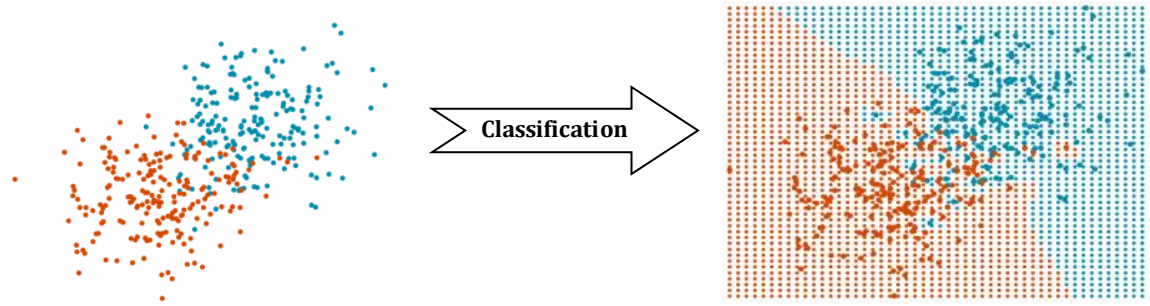


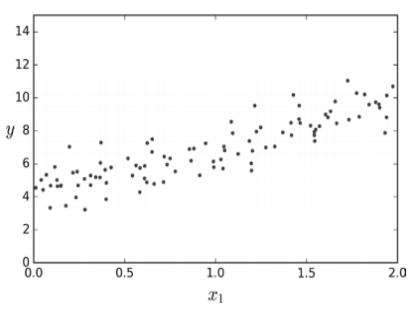
Figure II:3 Classification supervisée

Elle est utilisée dans des problèmes tels : catégorisation des textes et voix, reconnaissance de formes, diagnostics médicales, prédictions météorologiques etc. Plusieurs logiciels qui appliquent cette méthode existent comme : Rapidminer (31), Package R (32), Tanagra (33).

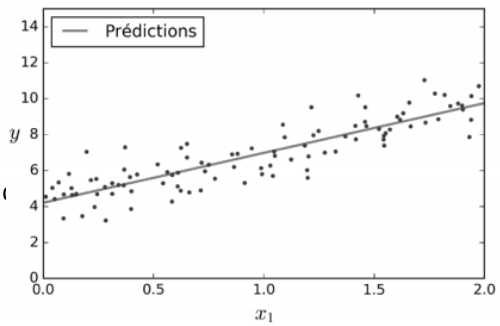
2.4.1.2 *Régression*

La régression se caractérise par des sorties indéfinies sous formes de valeurs numériques (continues). Il existe plusieurs types de régressions dont les plus connus sont :

1) **Régression linéaire :** c'est la plus intuitive. Elle présente une relation sous forme d'une fonction linéaire « Y= Ax+B » (d'où son nom) entre toutes les données d'entrés et les prédictions (voir figure ci-dessous). Par conséquence, le principe étant dechoisir les meilleurs paramètres A et B pour trouver la meilleure courbe qui généralise le problème pour toutes les données.



ont soi par défaut sur des



le

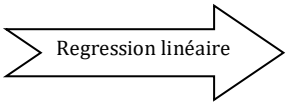


Figure II:4 Régression linéaire

2) **Régression polynomiale :** appelée aussi régression linéaire multiple. Cette méthode est un cas spécial de la régression linéaire en utilisant une fonction polynômiale d'un degré spécifique « $Y= Ax^n+Bx^{n-1}+\ldots. +C$ » (voir figure ci-dessous)

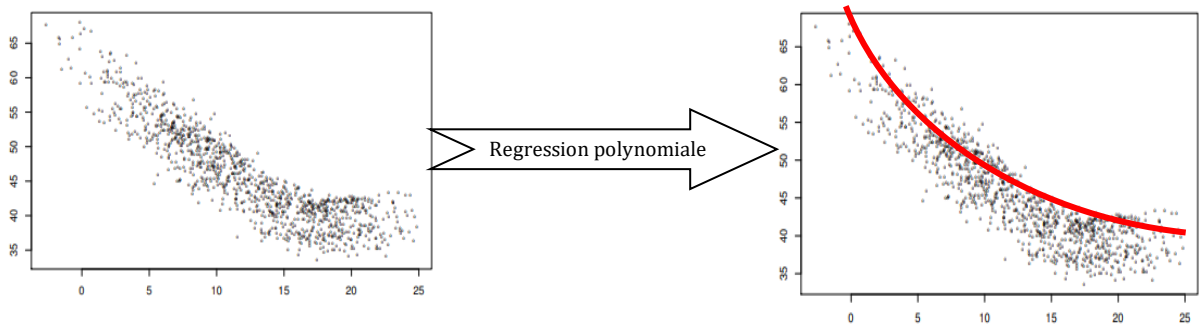


Figure II:5Régression polynomiale.

2.4.2 Apprentissage non supervisé

Figure II:6 Shéma global de l'apprentissage non supervisé

L'apprentissage non supervisé utilise des données non étiquetées et spécifié par le manque de superviseur. Son objectif étantde classifier et d'inférer des connaissances à partir de ces données selon des similarités, des différences ou d'autres motifs qu'il découvre au fur et à mesure à partir de nouvelles données, sans entrainements préalables (voir figure ci-dessus).

Il existe deux types d'apprentissage non supervisé : Parmise en cluster ou par association.

1. **Mise en cluster :** le principe de cette méthode consiste à décomposer un ensemble de donnéesen plusieurs sous-ensembles homogènes (voir figure ci-dessous). Plusieurs

typesde clustering existent à l'exemple de : clustering spectral , co-clustering ,clustering graphique , clustering flou etc.

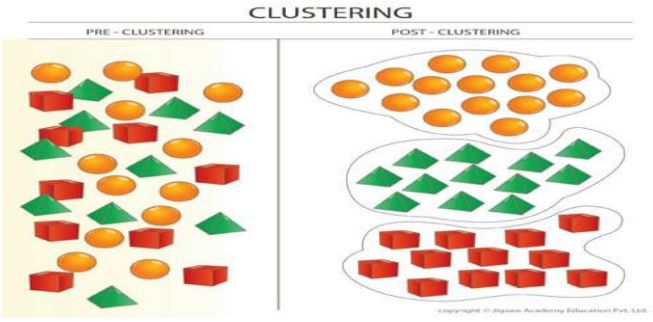


Figure II:7 Exemple de Clustering

2. **Association :** c'est une méthode très similaire au Clustering or dans ce cas, elle utilise principalement des relations entre les données, qu'elle tire au fur et à mesure de les recevoir, afin de décider sur leurs partitionnements (voir figure ci-dessous).

ID	Liste de boissons
1	{Ifri, Lait, Selim, Coca cola, Star}
2	{Ifri, lait}
3	{Ifri, Coca cola, Star, lait}
4	{Selim, Star, Lait}

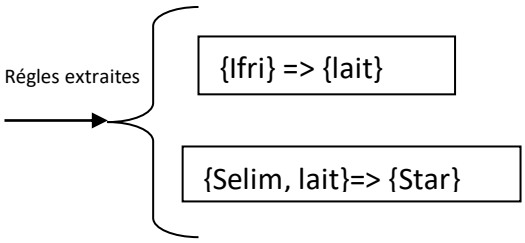


Figure II:8 Exemple d'association

2.4.3 Apprentissage semi-supervisé

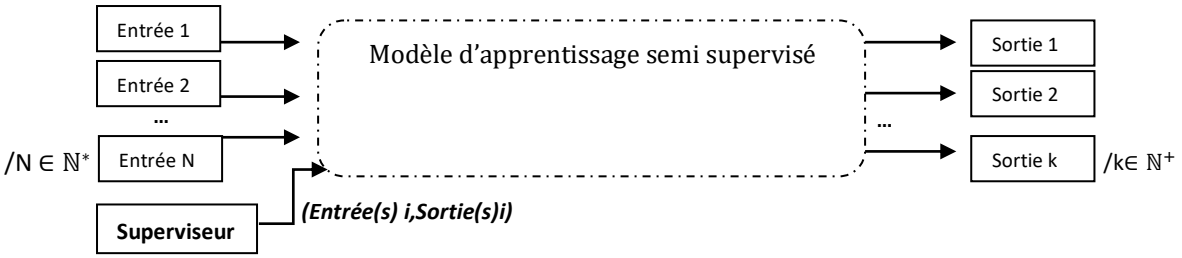


Figure II:9 Apprentissage semi-supervisé

Il s'agit d'une combinaisonde l'apprentissage supervisé et non supervisé où dans les données d'entrainement on retrouve des données étiquetées et non étiquetées. Ainsi, en se basant sur les données libellées fournies par le superviseur (partiellement disponible), elle va tenter de catégoriser ce type (supervisé) et d'extraire des relations entre les données restantes (non supervisé). Ce type d'apprentissage s'avère très intéressant et très performant notamment dans le domaine médical.

2.4.4 Apprentissage par renforcement

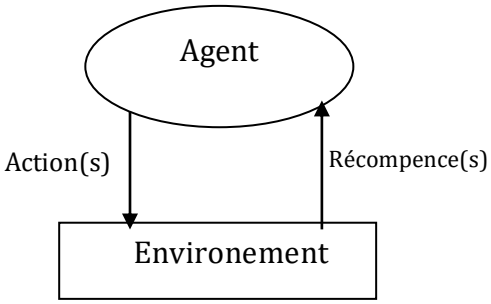


Figure II:10 Fonctionnement de l'apprentissage par renforcement

L'objectif de ce type est de permettre à un agent (machine ou robot etc.) d'apprendre à l'aide de ses interaction avec son environnement et de tirer profit des expériences passées afin d'optimiser sa performance et de réduire ses erreurs suivant un algorithme bien précis (voirannexe B.6). L'environnement quant à lui, fournit des récompenses (négatives ou positives) à l'agent pour chaque action entreprise.

Les méthodes les plus utilisées dans ce type d'apprentissages sont :

- **Le Q-learning :** elle s'agit d'une technique statique où la notion du temps est inexistante. Elle permet de garder la trace de la relation d'un état avec l'action entreprise « Q(e, a) [4]» et la mettre à joursdès qu'une action « a » est exécutée et qu'une récompenses « r » est reçue en utilisant la fonction suivante :

$$Q^{n+1}(e,a) = (1-\alpha)Q^{n-1}(e,a) + \alpha(r + \Upsilon max_{a'}Q(e',a'))$$

Équation II-1 Fonction Qlearning

Où e', a' sont respectivement les nouvelles états et actions, α est le facteur d'apprentissage et enfinΥ représente le facteur d'actualisation.

- **Le TD learning** (34)**:** cette technique est caractérisée par sa dynamique (existence du temps). Aussi, son fonctionnement est basé sur le principe de *bootstrap* [5]et repose sur les avantages de MCTS[6] et la programmation dynamique. Elle permet de mettre à jour la valeur d'un état à un moment donnée en utilisant l'équation suivante :

$$V(E_t) = V(E_t) + \alpha[R_{t+1} + \Upsilon V(E_{t+1}) - V(E_t)]$$

Où E_t représente l'état précédant et E_{t+1} l'état courant

Équation II-2 Fonction TD learning

[4]« Q » pour qualité d'une action dans un état spécifique.

[5] Méthode statistique basée sur des simulations stochastiques itérative en créant de nouveaux échantillons à partir d'un seul et cela par tirage et remise.

[6] Monte carlo tree search.

3 Techniques d'apprentissage automatique

Plusieurs techniques ont été mises en place afin d'entrainer et de faciliter l'apprentissage d'un modèle. Nous présentons dans cette section les méthodes les plus utilisées et les plus prometteuses dans l'apprentissage automatique.

3.1.1 Arbre de décision

L'arbre de décision (de régression ou de classification) (35) est la technique la plus intuitive de toutes. Elle tire son idée du raisonnement humain à faire des choix. Elle se présente sous forme d'arbre construit avec une induction descendante. Il intervient généralement sur des données qualitatives et a pour objectif de prédire des variables connues à partir d'un ensemble de données en entrée. Il contient des nœuds qui représentent des tests et des feuilles qui représentent des classes prédites. Aussi, chaque arête désigne une valeur de variable ou une réponse au test du nœud (voir l'exemple de la figure ci-dessous).

Les algorithmes les plus connus pour la génération des arbres de décisions sont :

- **CART (Classification And Régression Tree) :** il permet de générer des arbres de décision sous forme d'arbres binaires où pour chaque nœud (test) deux réponses possible sont attribuées.
- **C4.5 (optimisation d'ID3) :** c'est l'algorithme le plus utilisé. Il permet de construire un arbre de décision qui contient autant de nombre d'arêtes que les réponses au teste des nœuds.

Cette technique est utilisée dans plusieurs domaines en particulier : en fouille de données notamment par Facebook dans les suggestions d'amis, ainsi que les diagnostics médicaux et dans la théorie des jeux comme le génie du web « Akinator».

D'une part, il présente des avantages comme sa simplicité et la facilité d'implémentation et sa robustesse face aux données aberrantes. Aussi il est qualifié de « White-box » qui fait référence à sa transparence et sa conception intuitive facile à comprendre.

D'autre part, on risque d'avoir des arbres très complexes ainsi difficile voir impossible à parcourir et à entrainer. Aussi, il existe certains problèmes auxquels on ne peut pas appliquer cette technique comme les problèmes de multiplexages

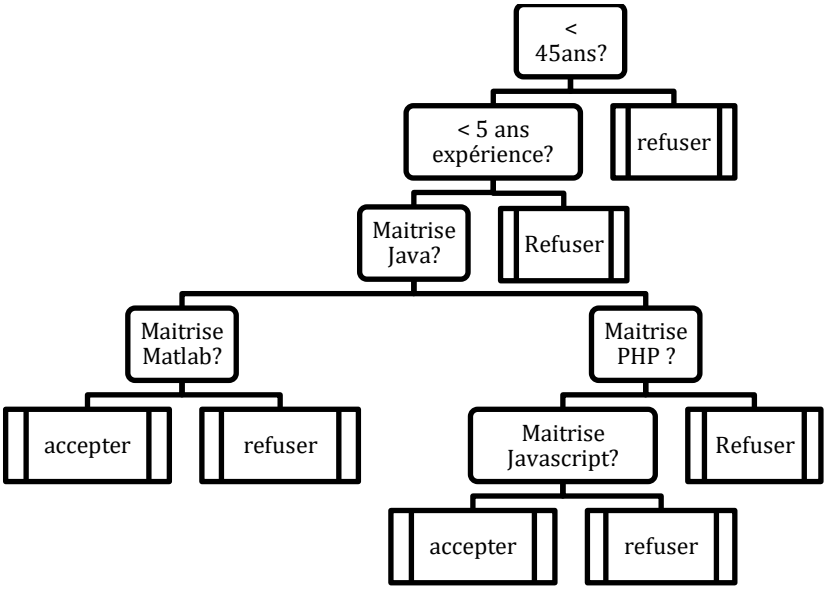


Figure II:11 Exemple d'arbre de décision (CART)

3.1.2 Forêt d'arbres décisionnels

Cette technique utilise des sous-ensembles de données et d'attributs choisis aléatoirement, auxquels sont appliqués les arbres de décisions. Cette méthode permet de limiter le suraprentissage, et produit un apprentissage rapide vu le nombre de paramètres réduit. Par contrela complexité de la conception du modèle risque de rendre les réseaux incompréhensibles et ainsi peut être vu comme étant une boite noire (36).

3.1.3 K plus proches voisins (K-PPV)

Il s'agit d'un algorithme appliqué à l'apprentissage supervisé utilisé pour sa facilité d'interprétation et sa rapidité. Il consiste généralement à classer un échantillon de données dans des classes en fonction du paramètre K (K>=1) où il agit en fonction des données présentes dans le territoire limité. Il représente le nombre d'instances prises en compte pour la détermination de l'affinité avec les classes.

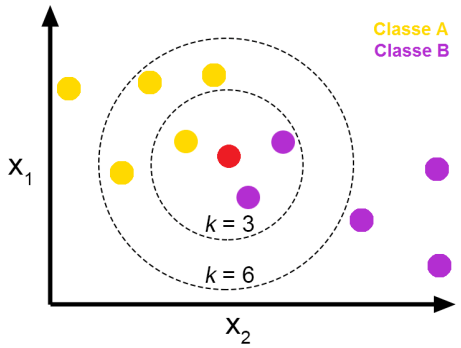


Figure II:12 Exemple de K-PPV

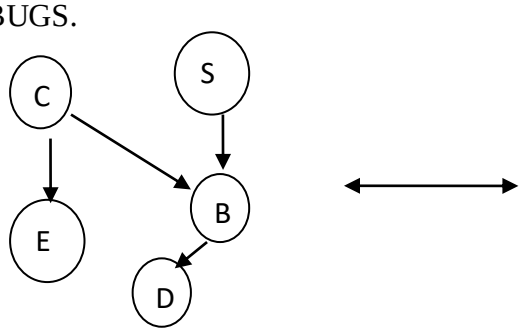
D'après l'exemple de la figure ci-dessus, nous constatons que dans le cas où k=3 la donnée en rouge sera classée dans la classe B (Card[7] (classe B)>Card (classe A)). D'autre part, si k=6, elle va être classée dans la classe A.

K-PPV est très populaire dans le domaine de la reconnaissance de texte ou d'objets, mais aussi dans le domaine médical. Malgré sa popularité dans ces domaines, il présente un inconvénient dans l'étape de l'entrainement où il n'apprend pas des données mais les utilise à chaque requête de prédiction pour la satisfaire ce qui n'est pas un processus optimal et s'avère couteux dans le cas d'une masse de données d'apprentissage mais aussi risque de ne pas généraliser le problème pour les nouvelle données. En outre, une modification de la valeur de K peut changer la classification des nouvelles données, ce qui n'est pas efficace et falsifie les résultats. (37)

3.1.4 Réseaux bayésiens

Un réseau bayésien (38) est un graphe orienté acyclique basé sur une approche probabiliste (généralement conditionnelle) et développé àpartir des principes de la théorie des graphes. Il est équipé d'algorithmes d'inférence (associe un degré de probabilité à une proposition en liaison avecune autre) et a pour objectif de détecter les instanciations[8]. Il est constitué d'un ensemble de nœuds, chaque nœud est identifié par un nom de variable pour laquelle est associée une table de probabilités qui change en fonction des nœuds parents. Outre que les nœuds, il existe aussi des arcs orientés qui relient les nœuds entre eux (voir figure ci-dessous).

Cette technique est un outil utilisé dans des problèmes qui suscitent une prise de décisions conditionnée par plusieurs facteurs complexes. Ainsi, le réseau bayésien représente toutes les relations entre les attributs. Contrairement aux arbres de décisions, cette technique a l'avantage d'être multidirectionnelle et assume la complexité des données. Il existe de nos jours plusieurs logiciels qui permettent d'utiliser les réseaux bayésiens :Elvira, OpenBUGS, WinBUGS.



Douleurs au dos (D)		
B	P(D=T|B)	P(D=F|B)
V	0,7	0,3
F	0,1	0,9

Collègue souffrant du dos (E)		
C	P(E=T|C)	P(E=F|C)
T	0,9	0,1
F	0,01	0,99

Blessure au dos (B)			
C	S	P(B=T|C,S)	P(B=F|C,S)
T	T	0,9	0,1
T	F	0,2	0,8
F	T	0,9	0,1
F	F	0,01	0,99

Pratique sportive (S)	
P(S=T)	P(S=F)
0,02	0,98

Chaise inconfortable (C)	
P(C=T)	P(C=F)
0,8	0,2

Figure II:13 Exemple de réseau bayésien

[7] Cardinalité
[8] La plus grande valeur de probabilité d'un ensemble de variables

3.1.5 Réseaux de neurones artificiels

Il s'agit d'une technique révolutionnaire (voirsection 4) qui tire son principe du fonctionnement du cerveau humain qui consiste en un nombre de neurones qui communiquent mutuellement des informations afin de résoudre une tache précise. Elle est qualifiée de boite noire (*black box*) car son architecture peut être contrôlée mais jamais on peut prédire ou expliquer son comportement à cause de sa liaison à plusieurs paramètres. Il est constitué de trois couches élémentaires qui sont reliées entre elles par des connexions pondérées. En outre, elles contiennent un nombre définis de neurones artificiels (voir figure 2.15). Ces couches sont :

- **La/les couche(s) d'entrée(s) :** c'est la couche initiale qui permet de communiquer avec l'environnement externe et où les donnée seront propagées.
- **La/les couche(s) cachée(s) :** cette couche intermédiaire traite les données reçus à partir de la couche d'entréeet leur applique plusieurs processus afin d'extraire des informations importantes.
- **La couche de sortie :** c'est la dernière couche qui produira le résultat final et le nombre de neurones de cette couche doit correspondre au type du problème traité.

L'objectif principal des réseaux de neurones est de trouver les meilleures valeurs des poids ce qui revient à réduire les erreurs de la fonction d'apprentissage. (39)

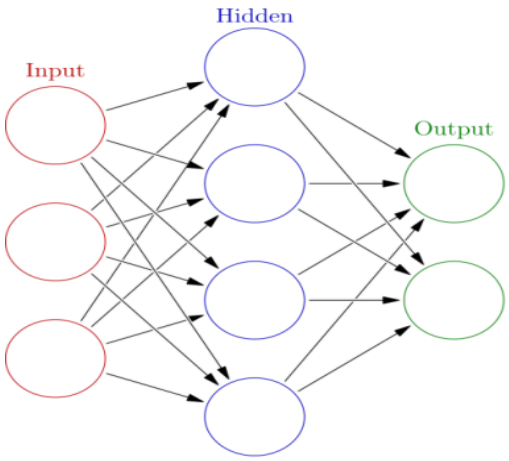


Figure II:14 Réseaux de neurones

4 Réseaux de Neurones Artificielles (RNA)

4.1 Généralités

4.1.1 Neurone formel

Le neurone biologique [9] a inspiré les informaticiens pour le développement du neurone formel (voir figure ci-dessous). Ce dernier peut être défini comme l'unité du réseau de neurones. Il s'agit d'un automate qui applique un ensemble de processus aux entrées (données) et génère des sorties qui sont communiquées pour les autres neurones (40).

Le processus élémentaire du neurone consiste en trois étapes principales:

1. **La pondération :** Elle permet d'associer à chaque entrée un poids (valeur numérique) en réalisant le produit entre les deux entités.
2. **L'ajout du Bias :** Il s'agit de la sommation de l'étape précédente avec le Bias (valeur numérique) et cela afin d'assurer que le résultat de cette linéarisation ne soit pas nul.
3. **Activation:**Une fois la linéarisation est réalisée, le neurone procède à la non-linéarisation de la fonction et cela en appliquant des fonctions mathématiques bien précises appelées : les fonctions d'activations (voir section 4.3).

Formellement, la valeur de sortie du neurone (Y) auquel est appliqué « n » entrés peut être réduite à l'équation ci-dessous :

$$Y = F(Z)$$

Où F estune fonction d'activation et Z définie ainsi :

$$Z = (\sum_{i=1}^{n} X_i W_i) + B$$

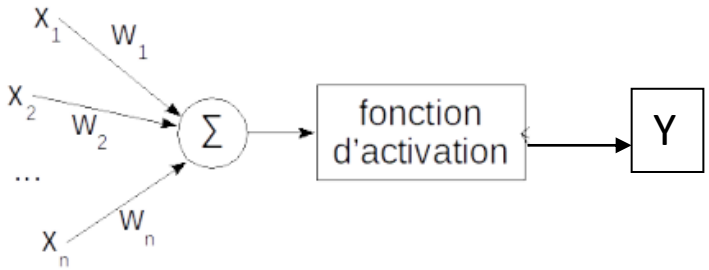


Figure II:15 Neurone Formel

[9] Il s'agit d'une cellule dans le systéme nerveux qui reçoie, traite et communique des informations, il est à l'origine du fonctionnement du cerveau humain

4.1.2 Fonctionnement

4.1.2.1 Paramètres et hyper-paramètres

1. **Paramètre :** il s'agit de la variable principale du réseau de neurones qui est modifiée au cours de l'apprentissage afin de retrouver la meilleure valeur possible qui minimise la fonction de couts.

Nous distinguons deux paramètres principaux : le poids et le bias.

2. **Hyper-paramètres :** représentent des variables secondaires qui seront fixes l'ors de tous l'apprentissage. Ces types de paramètres sont choisis en fonction du nombre des données reçues et selonle problème étudié. Nous distinguons les hyper-paramètres suivant :

- Nombre de couches intermédiaires (couches cachées).
- Nombre de neurones dans une couche.
- Nombre d'époques : il représente le nombre d'itérations exécutées lors de l'entrainement autrement dit le nombre de fois où les paramètres seront modifiés
- Taille du lot (*batch size*) : c'est le nombre de données entrainées à la fois l'ors d'une seule époque.
- Taux d'apprentissage (*learning rate*) : il s'agit d'une valeur incluse entre 0 et 1. Elle représente le taux d'avancement pour atteindre la borne minimale lors de l'utilisation du gradient descent.

4.2 Types d'entrainementsdes réseaux de neurones

Il existe deux types d'entrainements appliqués aux RNA selon le problème étudié : sans rétro-propagation et avec rétro-propagation.

4.2.1 Entrainement Acyclique (FeedForward Propagation)

Dans ce type d'entrainement, les paramètres du réseau ne sont mis à jour qu'une seule fois. De ce fait, la propagation des données se fait dans un seul sens à partir de la couche d'entrée jusqu'aux couches de sorties.

4.2.2 Entrainement cyclique (Backward-propagation)

Ce type d'entrainement est le plus appliqué pour des problèmes complexes. Il se caractérise par deux étapes qui se répètent successivement jusqu'à atteindre le nombre d'époques maximal. Ces étapes représentées par la figure ci-dessous sont :

- **Feed forward propagation :** elle est composée principalement de trois étapes :

- **Initialisation aléatoire :** il s'agit d'une initialisation aléatoire des valeurs des poids et des bias.
- **Propagation des données :** les données sont propagées dans le modèle représentant ainsi le début de l'apprentissage.
- **Calcul de la fonction de coût :** utilisation de la fonction de coût (voir section 4.6.1) afin d'estimer l'erreur des prédictions.

➢ **Back propagation :** après l'application de l'algorithme du gradient (voir section suivante) (étape 4 figure ci-dessous), cette étape permet de retourner l'erreur calculée (étape 5 figure ci-dessous) dans la sortie vers le sens inverse (la couche d'entrée) en recalculant les nouvelles valeurs des poids et des bias (étape 6 figure ci-dessous) en utilisant la formule suivante :

$$\beta_k^{(r+1)} = \beta_k^{(r)} - \tau \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \beta_k^{(r)}} \quad \text{Nouveau Bias}$$

$$\alpha_{kp}^{(r+1)} = \alpha_{kp}^{(r)} - \tau \sum_{i=1}^{n} \frac{\partial Q_i}{\partial \alpha_{kp}^{(r)}} \quad \text{Nouveau Poid}$$

Où τ : taux d'apprentissage, $\frac{\partial Q_i}{\partial \beta_k^{(r)}}$: dérivée de la fonction de couts par rapport au bias (poid)

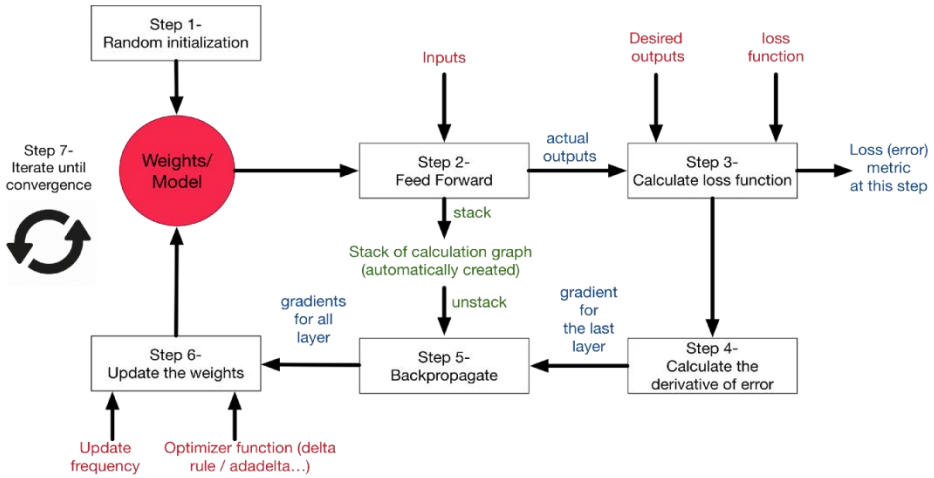


Figure II:16 Entrainement avec retro-propagation

4.3 Principe du gradient descendant

Le gradient descendant (41) est un algorithme d'optimisation qui permet de trouver la meilleure valeur des paramètres d'apprentissage en utilisant la fonction de couts. Il permet d'indiquer à un modèle donné la direction à suivre dans la courbe tracée en fonction de l'un des paramètres d'apprentissage et une fonction objective. Cela se réalise en calculant la dérivée première de cette fonction et en trouvant de ce fait le vecteur jacobien inverse (voir

annexe B.2) résultant pour le guider dans la recherche de tel sorte à avoir la borne minimale locale ou globale (42).

4.4 Architecture des réseaux de neurones artificiels

Les réseaux de neurones (43) sont classifiés en différentes architectures (voir annexe B.1) selon le type du problème défini.Dans ce qui suit, nous présentons les architectures les plus connues.

4.4.1 Perceptron

Le perceptron est l'architecture de base d'un réseau de neurone. Il s'agit d'une architecture appelée monocouche qui ne contient pas de cycle (pas de retro-propagation), ayant un à plusieurs neurones et se terminant par une seule sortie.

4.4.2 Architecture complétement Connectée

Ce type d'architecture est caractérisé par la connexion de chaque neurone d'une couche avec tous les neurones de la couche suivante. C'est une architecture très complexe contenant beaucoup de paramètres à entrainer.

4.4.3 Perceptron multicouches

Cette architecture acyclique, pouvant être de conception dense, contient plusieurs couches intermédiaires, nous parlons alors des réseaux de neurones profonds, utilisé dans des problèmes complexes notamment dans l'apprentissage supervisé où elle est considérée comme un bon classificateur.

4.4.4 Architecture convolutionelle

Le fonctionnement du cortex visuel humain a inspiré les chercheurs pour concevoir une architecture dite « Architecture convolutionelle (CNN) ». Elle s'agit d'une architecture phare utilisée dans le domaine de la vision en intelligence artificielle (reconnaissance d'images ou d'objets) ou même dans l'audio. Elle entre dans la plupart des cas dans le cadre de l'apprentissage par transfert où le résultat de sorite peut être réutilisé comme l'entrés d'un autre modèle. Elle est constituée de couches convolutionelles et de couches de pooling. Elle tire sa performance du nombre de paramètre réduit en appliquant des filtres pour chaque couche ce qui rend l'entrainement plus rapide et donc plus performant.

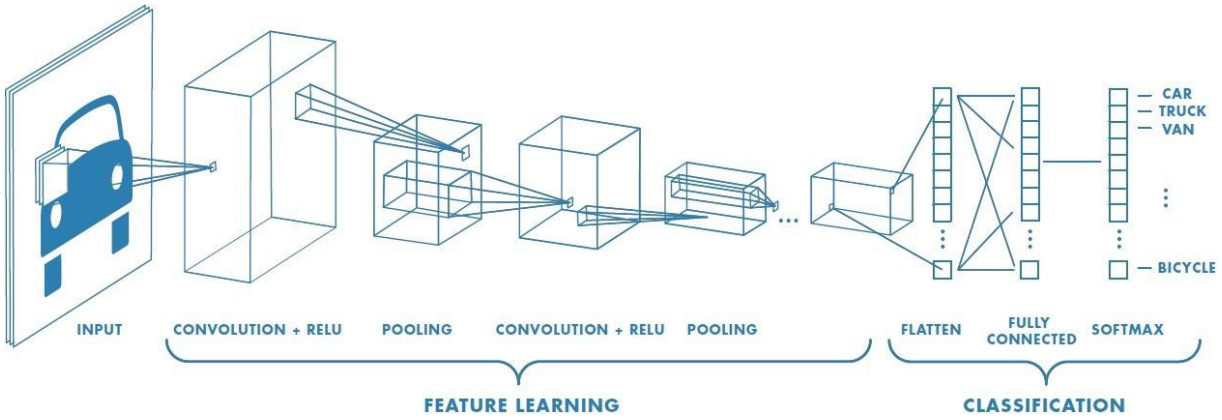


Figure II:17 Architecture de ConvNet

La figure ci-dessus représente l'architecture typique de ConvNet. Elle est constituée généralement d'une succession de trois couches à savoir : une couche convolutionelle, une couches de pooling et enfin une couche d'activation. De ce fait, la sortie finale de cette architecture sera utilisée par une autre architecture (généralement de type dense).

ConvNet offre de nouveaux hyper-paramètres outre que ceux vus précédemment. Ces hyper-paramètres sont : la dimension des matrices filtrantes (de pooling ou de convolution), le nombre des matrices filtrantes, stride, padding.

4.4.4.1 Couche de convolution

Au niveau de cette couche un traitement est effectué (pondération, sommation, activation) en utilisant une matrice de convolution qui, initialement, contient des poids aléatoirement initialisés et quisont par la suite modifiés au cours de l'apprentissage. Ainsi, l'objectif de cette étape est d'identifier les filtres des matrices de convolutions dans l'image ce qui permettra au modèle d'analyser les détails de l'image.

La dimension de la matrice de convolution est laissée au choix du programmeur. Cependant la dimension de la matrice résultante est (N'*N') tel que :

$$N' = \frac{N + 2P - F}{S} + 1$$

Équation II-3 Calcule de la dimension de la matrice covolutionnée

Où (N*N) est la dimension de la matrice initiale. (F*F) la dimension de la matrice convolutionelle (F étant un nombre impair par convention).S (par défaut égale à 1) représente le paramètre« Stride » qui désigne le nombre de case avec lequel est déplacée la matrice de convolution dans la matrice initiale lors de la pondération convolutionelle. P : représente le « padding » ou le cadrage de la matrice initiale de telle sorte que la dimension de la matrice convoluée est égale à la matrice initiale, et aussi recouvre plus de données. Il existe deux valeurs possibles pour le « padding » : « Valide » (où nous appliquons l'équation dernière) ou « Same » (nous rajoutons un nombre de padding de tel sorte que la dimension de la matrice initiale et la matrice convoluée seront égales.

Il existe trois types de couches de convolutions :

1. **Conv-1D :** exécute des convolutions sur un axe à la fois de la matrice initiale.
2. **Conv-2D :** c'est la plus utilisée en pratique. Elle utilise une matrice filtrante multidimensionnelle.
3. **Conv-3D :** elle est utilisée notamment avec des donnée volumineuse (tridimensionnelles) ou contenant des canaux.

4.4.4.2 Couche de « pooling »

Cette couche (44) permet d'appliquer un algorithme spécifique sur une matrice donnée en entrée afin d'éclaircir les résultats de la couche convolutionelle. Elle utilise pour cela une fenêtre de pooling et appliquant une des deux méthodes suivante : Max-pooling et Average–pooling.

1. **Max-pooling :** au cours du « pooling », la valeur de la case M_{ij} de la matrice résultante est le maximum de la matrice filtrante de pooling appliquée à la matrice initiale.
2. **Average-pooling :** son fonctionnement est similaire à l'algorithme précédant à la différence qu'au lieu de prendre en considération le maximum, la moyenne entre les différents éléments est calculée.

4.4.5 Architecture récurrente

La conception de ce type d'architecture (45) est la plus proche du système nerveux humaine. Elle est utilisée pour apprendre des données séquentiellement organisées et dépendantes les unes les autres à l'image de la traduction automatique, la reconnaissance de voix, génération de musique, classification des sentiments etc. La récurrence provient de la capacité à mémoriser des états passés et garder le contexte ainsi chaque état dépend de ses prédécesseurs. Toutefois, cette mémorisation s'avère très courte ce qui réduit la performance du modèle. C'est ainsi que de nouvelles méthodes révolutionnaires sont créées : Long Short Term Memory (LSTM) et GRU (GattedReccurent Unit).

Il existe cinq types d'architectures RNN (voir annexe B.4) classés selon le type du problème. Ces types sont: one to one, one to many, many to one et enfin deux types différents de many to many.

4.4.5.1 LSTM

Long Short Term Memory(LSTM) (46) est un type révolutionnaire d'architecture RNN. Il se base sur des dépendances, à longues termes comme à courtes termes, entre deux entités

d'entrées. Il est construit d'une cellule mémoire où les informations sont sauvegardées, et de trois portes pondérées dont les poids sont appris durant l'apprentissage ce qui détermine la bonne connexion entre elles. Ces portes sont : porte d'entrée (surveille les conditions pour qu'une nouvelle entrée soit sauvegardée dans la cellule mémoire), porte de sortie (détermine les conditions pour qu'une valeur en mémoire détermine la valeur de sortie) et la porte d'oubli (détermine les contraintes pour qu'une information reste en mémoire).

4.4.5.2 GRU

Les unités récurrentes gated (GRU) (47) est une architecture simplifiée de LSTM où dans ce cas, la porte de sortie et la cellule mémoire sont inexistantes ce qui donne moins de contrôle sur les entrées, mais une exécution rapide et des résultats plus efficaces que l'autre dans certains cas. Avec une simple conception et des résultats prometteurs, GRU est de plus en plus utilisé dans le domaine professionnel.

4.5 Les fonctions d'activations usuelles

Les fonctions d'activations (48) sont des fonctions mathématiques différentiables utilisées au niveau de chaque neurone. Elles permettent la représentation de certaines données complexes tel que les images, les vidéos et audio.

Une fonction d'activation peut être catégorisée selon plusieurs facteurs comme : la monotonie de la dérivé, la linéarité, l'étendue (plage d'activation). Cependant les développeurs optent pour l'utilisation des fonctions non-linéaires car ces dernières offrent une grande puissance au réseau de neurones et une capacité à généraliser des problèmes très complexes avec des résultats efficaces.

Le tableau ci-dessous présente les fonctions d'activations les plus utilisées dans l'apprentissage automatique en réseaux de neurones:

Fonction	Equation	Description
RELU	$F(X) = \begin{cases} 0 \; si \; x < 0 \\ x \; si \; x \geq 0 \end{cases}$	• **Avantage**: - Souvent utilisée notamment dans les couches cachées. - Non linéaire - Efficace. • **Inconvénients** - Sensible au problème de disparition du gradient
Leaky RELU	$F(X) = \begin{cases} 0.01 \; si \; x < 0 \\ x \; si \; x \geq 0 \end{cases}$	- Optimisation de Relu afin de prévenir la disparition du gradient.
ELU	$F(X) = \begin{cases} a(e^x - 1) \; si \; x < 0 \\ x \; si \; x \geq 0 \end{cases}$	- Tire les avantages de RELU et LRELU
PRELU	$F(X) = \begin{cases} ax \; si \; x < 0 \\ x \; si \; x \geq 0 \end{cases}$	• **Avantage:** - Similaire à RELU. - permet d'apprendre sur des valeurs négatives.
Tanh	$\mathrm{F}(X) = \frac{2}{1+e^{-2x}} - 1$	• **Avantage :** - Normalisation des valeurs entre -1 et 1. - Utilisée fréquemment par les couches de sorties - Bénéfique pour la gestiondes données ayant des données neutres, de grandes valeurs positives et négatives. • **Inconvénient :** - Couteuse lors de l'exécution. - Sensibilité à la disparition du gradient.
Sigmoïde	$\mathrm{F}(X) = \frac{1}{1+e^{-x}}$	• **Avantage :** - **Souvent utilisée par les couches de sorties.** - Normalise les valeurs des données entre 0 et 1 - Attribut des probabilités au données mais contrairement à Softmax la somme des probabilités peut êtredifférente de 1 - Utilisée pour les problèmes de classification binaire. • **Inconvénient :** - Les même que « Tanh »
Softmax	$\mathrm{F}(Z)_j = \frac{e^{zj}}{\sum_{k=1}^{k} e^{zk}}$ / j∈ {1 , … , K}	• **Avantage** : - Fréquemment utilisé dans les couches de sorties. - Normalisation des données entre 0 et 1. - Attribut des probabilités pour les éléments sachant que la somme des probabilités égale à

		1. - Une généralisation de «Sigmoïde » - Utilisée pour les problèmes de classifications
Gaussienne	$F(x) = e^{-x^2}$	- Fonction isotropique[10] - Utilisée par les couches cachées.
SWISH	$F(X)=\frac{x}{1+e^{-x}}$	- Développée par Google pour des problèmes de classificationet est considérée comme une optimisation de RELU.

Tableau II-1 Fonctions d'activations

4.6 Entrainement du modèle

4.6.1 Les fonctions de coûtusuelles

La fonction de couts (fonction objective) (49) est une fonction primordiale dans l'entrainement d'un modèle. Elle donne une vue globale sur sa performance en estimant l'incohérence de la sortie attendue avec la sortie prédite qui est corrigée par la suite. Ainsi, la robustesse du modèle est improportionnelle à la valeur calculée par la fonction de couts.

Le choix de la fonction de couts est lié principalement à la fonction d'activation utilisée dans la couche de sortie. Nous distinguons trois types principaux selon le type d'apprentissage (voir tableaux annexe B.5)*:*

- **Fonctions de coût de classification :** Utilisées quand la sortie prédite est sous forme d'une probabilité et la sortie attendu est binaire.
- **Fonctions de coût d'intégration :** Utilisées pour comparer la similarité entre la sortie prédite et celle attendue.
- **Fonctions de coût de régression :** Utilisées pour des problèmes d'apprentissage supervisé de type régression.

4.6.2 Algorithme d'optimisation

Les problèmes d'apprentissage automatique sont des problèmes d'optimisations où des algorithmes avec diverses stratégies et outils sont appliqués aux fonctions d'activations (qui représentent leurs guides de terrain). Ces derniers permettent d'optimiser les erreurs (maximiser ou minimiser) et aussi contribuent à façonner le modèle durant l'entrainement en mettant à jours ses paramètres jusqu'à trouver les valeurs optimales.

[10] Utilisée pour la classification des formes bilinéaire.

Les algorithmes d'optimisations sont classés en deux catégories selon leur taux d'apprentissage : Algorithmes à taux d'apprentissage constant et algorithmes à taux d'apprentissage adaptatif (50).

4.6.2.1 Algorithme à taux d'apprentissage constant

Dans ce type d'algorithme tous les hyper-paramètres propres à l'optimisateur sont initialisés et appliqués sur l'ensemble des données durant l'entrainement. L'un des algorithmes les plus utilisés est le gradient descendant stochastique (GDS).

1. **Le gradient descendant stochastique :** C'est une méthode itérative qui, principalement utilise la méthode du gradient descendant classique (voir section 4.3). Mais contrairement à ce dernier où les échantillons de données sont sélectionnés par lot (Vanilla gradient descent) ou par mini batch (Mini-batch gradient descent) ou même par ordre d'apparition, GDS quant à lui, utilise une sélection aléatoire ce qui facilite la recherche. Par contre, la progression de cette méthode s'avère lente et se fait par des pas constants et généralement ne se dirige pas dans le sens de l'optimisation. En outre, il est souvent sensible et confronté à des problèmes tels que la disparition du gradient ou même le minimum local.
2. **Le gradient descendant stochastique + momentum :** Il s'agit d'une optimisation du GDS classique, où un hyper-paramètre appelé « momentum » (élan) est rajouté et qu'il calcule pour chaque position courante et qui fait converger la recherche vers la bonne direction et permet d'atténuer les oscillations « zigzag » lors de la descente du gradient. Aussi,il utilise le principe de la loi physique d'accélération équivalent du vecteur hessien(voir annexe B.3), et de la vitesse du mouvement (vecteur jacobien inverse) quand il passe du haut d'une courbe et l'utilisepour passer les bornes minimales locales ce qui rend la convergence beaucoup plus rapide que la version classique et résout tous ses problèmes. Cette méthode est très utilisée par les informaticiens et donne de bons résultats. Par contre le choix des valeurs des hyper-paramètres s'avère une tache très difficile et leurs valeurs ne sont pas soumises à des lois prédéfinies.

4.6.2.2 Algorithme à taux d'apprentissage adaptatif

Cette approche permet d'apporterune facilité d'utilisation pour les programmeurs et d'avantage de performances pour les programmes. De ce fait, les hyper-paramètres sont initialisés au début mais se modifient au fur et à mesure de la recherche en utilisant une approche heuristique ce qui permet d'avoir des résultats largement plus prometteurs que l'approche précédente sauf dans des problèmes comme la traduction automatique ou

reconnaissance d'image où ils n'arrivent pas à converger vers la solutions optimale le plus rapidement possible. Les meilleurs optimisateurs qui utilisent ce principe (voir l'annexe B.7) sont à l'exemple de : Adam, Adamax, Nadam, RMSP, AdaGrad.

5 Réseaux de neurones : Problématiques et solutions

Au cours du développement des modèles de réseaux de neurones, les programmeurs sont souvent confrontés à plusieurs problèmes liés à leurs conceptions où à d'autres paramétrages, ce qui nécessite beaucoup de patience et d'essais.

Dans ce qui suit nous présentons les problèmes les plus communs ainsi que des solutions à des problèmes courants qui causent un dysfonctionnement du réseau neuronal artificiel.

5.1 Problématiques

5.1.1 Sur-apprentissage

Le sur-apprentissage se définit comme étant la capacité du modèle -notamment non paramétrique et non linéaire- à apprendre tous les détails des données d'entrainements même les plus inutiles comme le bruit[11]. De ce fait, il réalise de mauvaises prédictions lors des étapes de validation et de test.

5.1.2 Sous-apprentissage

Le sous-apprentissage survient rarement quand le modèle conçue ne convient pas au type et au nombre de données d'entrainement. Par conséquent, le modèle ne parvient pas à apprendre et sera donc incapable de généraliser le problème.

5.1.3 Le problème du « Vanishing gradient »

Il s'agit d'un problème qui se produit dans des modèles cycliques, souvent lié au choix des fonctions d'activations qui influencent avec leur dérivée. En outre, il représente l'une des causes majeures d'un entrainement très long. Précisément, le gradient calculé lors de la mise à jour des paramètres du modèle s'avère très petit ce qui donne une marge d'avancement (dans l'entrainement) réduite et dans d'autres cas interrompt l'entrainement (si le gradient est nul).

5.1.4 Problème du minimum local

La fonction d'apprentissage du réseau de neurones en fonction des couts et des paramètres donne une fonction non convexe qui contient au moins une borne inferieure minimale (globale) minorées par plusieurs bornes minimales intermédiaires (locales).

[11] Données ou parties de données éronnées.

Toutefois, ces dernières peuvent parfois arrêter l'avancement de l'entrainement en croyant avoir trouvé la meilleure optimisation possible alors qu'elle n'est pas encore exploitée. En pratique, cela et due au mauvais choix des hyper-paramètres notamment le nombre de couches cachées. Par conséquence, cela affecte les performances du modèle et ne permet pas d'avoir des prédictions optimales (voir figure ci-dessous).

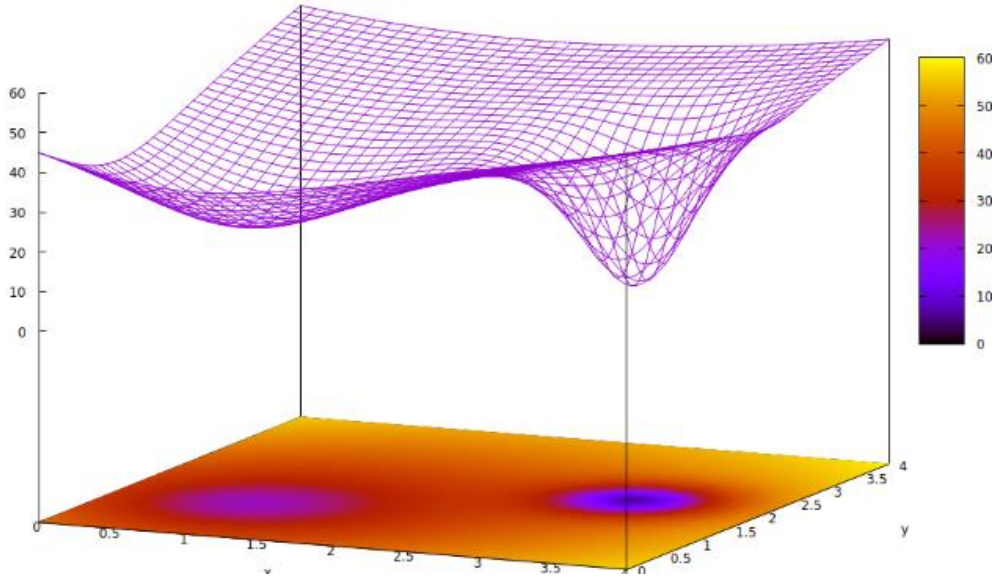


Figure II:18 Problème du minimum local

On constate d'après la figure précédente que l'erreur obtenue lors de l'arrêt de l'apprentissage dans la borne minimale locale est approximativement égale à 0.22. D'autre part, nous aurions pu avoir une valeur optimale de l'erreur avoisinant 0.05, une grande différence qui doit être prise en considération (51)**.**

5.1.5 Bien poser le problème

La compréhension des problèmes en apprentissage automatique joue un rôle crucial dans la conception d'un modèle adéquatet donne une vue globale sur les valeurs des hyper-paramètres à initialiser. De ce fait, un problème mal posé peut conduire à plusieurs solutions où dans d'autres cas sa solution est inexistante. Pour parer à ce problème, les concepteurs se basent principalement sur trois règles :

- **Objectif d'apprentissage** : localiser le type et la tâche à apprendre.
- **Critères de performance** : savoir distinguer entre les erreurs à éviter et ce qu'il faut optimiser.
- **Les données** : possibilité d'obtention des données d'apprentissage.

5.1.6 Choix du modèle et contraintes de temps

La rapidité de la prédiction estune notion très importante notamment lors de l'intégration des modèles d'apprentissage automatique dans des systèmes à temps réels comme la détection d'intrusion où l'utilisation des modèles paresseux[12] comme la méthode du « KPPV » est déconseillée. Ainsi, le choix d'un bon modèle doit s'adapter au phénomène étudié.

Dans les projets informatiques classiques, la gestion des projets notamment avec une approche temporelle est un outil indispensable pour l'estimation de sadurée. Or, dans le domaine de l'apprentissage automatique, il n'existe aucune convention pour l'estimation temporelle du projet car plusieurs facteurs influencent au cours du développementet il faut souvent prendre du temps afin de trouver l'équilibre parfait entre tous les paramètres qui rentrent en jeu. En plus, même les meilleurs ingénieurs et expert du domaine ne peuvent pas prédire le comportement d'un modèle appliqué à d'autres données.

5.1.7 Le problème de la boite noire

La compréhension et l'explication du fonctionnement des modèles de réseaux de neurones artificiels restent des défis à relever par les superviseurs. De nos jours, on ne peut pas expliquer formellement et rigoureusement un dysfonctionnement du modèle ni interpréter des prédictions erronées. Chose qui a même bloquée la commercialisation de nombreux logiciels dotés de cette technologie vu que l'ensemble des utilisateurs cherchent à comprendre la logique derrière leurs « intelligence» et les solutions proposées par les concepteurs en cas d'éventuels échecs ou bugs.

5.2 Solutions

5.2.1 Normalisation des données

Le réseau de neurones artificiel peut être vu comme étant une fonction qui prend en paramètre un ensemble de données et produit un résultat. Ainsi, comme toute fonction a son domaine de définition, le réseau de neurones a besoin que les données propagées soient normalisées dans un même intervalle afin de simplifier l'apprentissage et pour ne pas mettre à l'écart certaines données. De ce fait, cela aide à une convergence rapide vers la solution optimale. Les méthodes les plus utilisées sont :

- **Z-Score :** Elle gère les valeurs aberrantes mais ne met pas les données sur la même échelle. La nouvelle valeur normalisée (v') se calcule comme suit :

[12] Modèle qui s'entraine avant chaque prédiction.

$$V' = \frac{V - \mu}{\sigma}$$

Où $\boldsymbol{V}$ c'est une valeur, la moyenne du groupe de données,$\boldsymbol{\sigma}$: L'écart type

Équation II-4 Z-Score

- **Min-Max :** Permet de mettre toutes les données sur la même échelle mais ne gère pas les valeurs aberrantes. La nouvelle valeur normalisée (v') se calcule comme suit :

$$V' = \frac{V - min}{max - min}$$

Où $\boldsymbol{V}$ c'est une valeur, $\boldsymbol{min}$,$\boldsymbol{max}$, le minimum (respectivement : le maximum) du groupe de données.

Équation II-5 Min-Max

5.2.2 Ajout de bruit

L'ajout de bruit (52) est une méthode utilisée afin de parer au sur-apprentissage du modèle. Elle rentre dans le cadre de sa régularisation particulièrement utilisée lors de l'entrainement du réseau neuronal avec un nombre de données insuffisants ce qui permet dans la plupart des cas d'accroître ses performances. Plus formellement, il s'agit d'un ensemble de perturbations aléatoires appliquées aux signaux électriques d'entrés qui s'ajoutent à celles des données. Ces perturbations étant généralement indésirable s'avèrent bénéfiques dans d'autres cas comme la tolérance aux pannes et l'augmentation des données. L'un des bruits les plus utilisé est le bruit gaussien.

5.2.3 Arrêt prématuré

Il s'agit d'une méthode très utilisée par les développeurs. Elle intervient lors de la validation des données où elle permet d'arrêter l'entrainement dès que l'erreur de validation s'accroît (voir figure ci-dessous).Cela permet d'une part d'éviter le problème de sur-apprentissageet d'autre part de réduire le temps d'apprentissage.

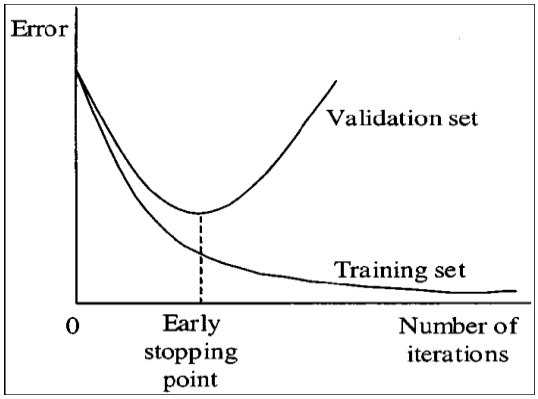


Figure II:19 Méthode d'arrêt prématuré

5.2.4 Régularisation

La régularisation permet de simplifier un modèle complexe ayant de grandes valeurs de pondérations ce qui provoque souvent le sur-apprentissage. Ainsi, avec la régularisation une pénalité est ajoutée au sein de la fonction de coût afin de forcer la diminution des poids. Il existe plusieurs méthodes de régularisations selon la pénalité ajoutée, nous distinguons :

- **Weight decay :** $Cout = erreur + \lambda \sum_i {p_i}^2$
- **L1 :** $Cout = erreur + \frac{\lambda}{2m} \sum \|p\|$
- **L2 :** $Cout = erreur + \frac{\lambda}{2m} \sum \|p\|^2$

5.2.5 Les couches de décrochage

Une autre méthode pour éviter le sur-apprentissage est d'utiliser les couches de décrochage où ces dernières s'appliquent au sein des réseaux de neurones complexes et permettent d'ignorer temporairement et aléatoirement des connexions neuronales du modèle lors de son entrainement (voir figure ci-dessous). Ainsi, une simplification de la structure du modèle sera réalisée et des dépendances qui peuvent être créées entre les neurones et qui empêchent la généralisation du problème seront éliminées.

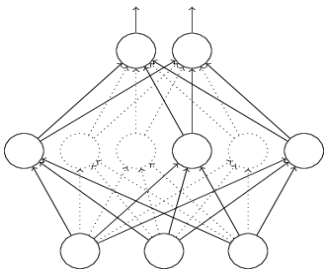


Figure II:20 Effet de décrochage

5.2.6 Elagage

L'élagage est une méthode similaire aux couches de décrochage à la différence où dans ce cas des connexions et des neurones qui n'ont aucun effet sur l'erreur calculée sont éliminés définitivement. Ainsi, une réduction de la complexité du réseau est réalisée (voir figure ci-dessous).

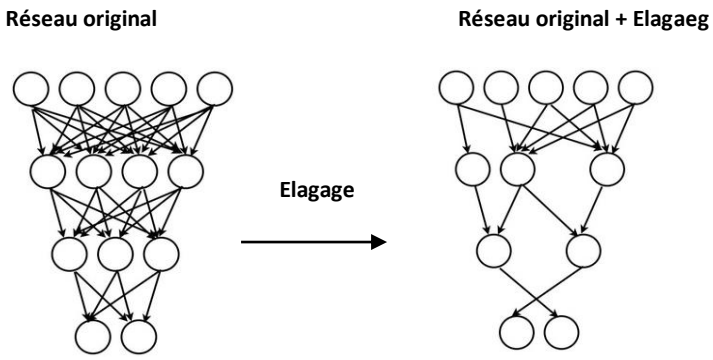


Figure II:21 Elagage

5.2.7 Architecturerésiduelle

L'architecture résiduelle ou « ResNet » est une des solutions proposées pour éviter le problème du « vanishing gradient ».Il s'agit d'un réseau neuronal simple avec des connexions sortantes qui ignorent certaines couches et créent des raccourcis vers d'autres couches ultérieures.

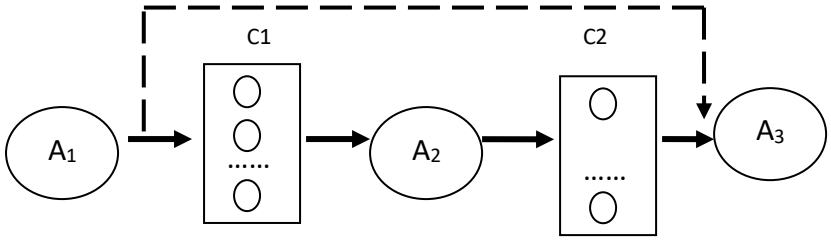


Figure II:22 Architecture résiduelle

On constate d'après la figure précédant qu'avec l'application d'une architecture résiduelle la fonction d'activation« A_3» devient comme suit :

$$A_{3=}F(A_2W_2 + A_1)$$

5.2.8 Autres solution

Il existe des méthodes qui permettent principalement de trouver un compromis biais-variance afin d'éviter le sur /sous apprentissage. Dans ce qui suit nous présentons les méthodes les plus connues et les plus utilisées par les ingénieurs en AA.

5.2.8.1 Méthodes pour atténuer le sur-apprentissage

Plusieurs solutions sont présentées afin de réduire les risques dus au sur-apprentissage à l'image de l'arrêt prématuré, la régularisation, les couches de décrochage et l'élagage.

En outre, les chercheurs en IA recommandent d'utiliser des entrainements parallèles utilisant une initialisation aléatoire des hyper-paramètres. Cette dernière méthode fonctionne

aussi bien que le *Boosting* et le *Bagging* qui sont définie comme étant des heuristiques qui permettent de réduire les valeurs des bias et des variances (53)

5.2.8.2 Méthodespour atténuer le sous-apprentissage

D'autre part, il existe aussi plusieurs solutions pour la réduction du sous-apprentissage, nous citons en l'occurrence : réduction de la régularisation, ajout de données (utilisation de plus d'exemple d'entrainement), utilisation des modèles avec une grande variance et l'ajout de neurones ou de couches cachées.

6 Conclusion

Dans ce chapitre, nous avons présenté les grandes lignes de l'apprentissage automatique. Nous avons exploré son application sur les réseaux de neurones où nous avons décortiqué son fonctionnement ainsi que les algorithmes utilisés. Enfin, nous avons exposé d'une part quelques défis rencontrés par les développeurs et d'autres parts des solutions et recommandations pour une bonne pratique.

Dans le prochain chapitre nous présentons l'application de l'apprentissage automatique dans notre projet à savoir nos modèles de réseaux de neurones artificiels. En plus, nous analysons leurs architectures ainsi que toutes les configurations ajoutées pour leur bon fonctionnement. Aussi, nous avons un passage explicatifdu modèle RNA du projet Leela-chess. Enfin, nous expliquons les métriques implémentées notamment la métrique personnalisée « Metric_Evaluation » qui nous permet d'avoir une vue globale sur les performances de nos modèles.

Chapitre 3: Titre du chapitre 3

1 Introduction

La bonne évaluation d'une position échiquéenne passe par la construction d'un algorithme performant intégrant différents paramètres (exemple : valeurs des pièces, pénalité attribuée à la structure des pions, score attribué à l'espace disponible pour chaque joueur sur l'échiquier, etc.). Le choix de ces paramètres est souvent problématique et est fait après différents tests empiriques nécessitant des temps de calculs très longs. L'utilisation de l'apprentissage automatique vient résoudre ce problème. En effet, il suffit maintenant de disposer d'un modèle général dont les paramètres seront automatiquement calculés par différents algorithmes. Le problème qui se pose alors, réside dans le choix de l'architecture de ce modèle, du nombre des paramètres et des hyper-paramètres utilisés, de l'algorithme d'entrainement à utiliser, etc.

Nous répondons à cette problématique dans ce chapitre, en proposant quatre nouveaux modèles d'apprentissage automatique permettant à terme d'évaluer les positions dans les jeux d'échecs. Mais avant cela, nous présentons d'abords un modèle dans l'état de l'art, à savoir Leela-Chess Zero, pour souligner par la suite les différences avec nos modèles. Ensuite, nous présentons les besoins qui nous ont motivés pour l'élaboration de nos modèles avant de les présenter. Enfin, nous concluons avec notre présage quant à la pertinence de ces modèles.

2 Apprentissage automatique et jeux d'échecs : Cas Leela chess Zero

Au cours du développement de notre application, plusieurs pistes nous ont dirigés et influencés dans nos choix et nos idées. Parmi elles, un des meilleurs projets et exemples typiques de l'application de l'apprentissage automatique dans les jeux d'échecs et qui fait ses preuves ces dernières années: Leela chess Zero (LC0).

LC0 (54) (55) utilise le principe d'AlphaZero, un autre modèle développé par Google, qui utilise un apprentissage par renforcement (voir chapitre 2 section 2.4.4) et ne se basant sur aucunes données au début de son entrainement. Ce qui fait de lui un modèle autodidacte, créatif et sans limites.

L'architecture du modèle intégré dans le moteur de LC0 (56) est caractérisée par ce qui suit:

- La position échiquéenne est représentée sous forme de cinq bitboards : pour les pièces blanches, pour les pièces noirs, pièces glissantes horizontalement, pièces glissantes verticalement et la structure des pions (incluant des informations sur la prise en passant).

- Utilisation de 20 blocs résiduels contenant 256 filtres.
- Utilisation de Squeeze Excite (SE) qui représente une extension des blocs résiduels utilisée par Deepmind. Elle permet d'ajouter une couche de « pooling » globale (squeeze) et puis applique un retour en arrière en transmettant ces informations globales pour toutes les parties de l'échiquier (excite).
- Son réseau neuronal artificiel est constitué d'une seule sortie (P) qui représente la probabilité à ce qu'un coup donné sera le meilleur.
- La normalisation du lot : est utilisée afin d'avoir une variance près de l'unité, et une moyenne qui tend vers zéro.
- La renormalisation du lot: permet de généré des sorties qui ont une relation avec chaque position du mini-batchest non pas en relation avec l'ensemble du lot.
- Utilisation de l'architecture ConvNet qui représente la plus grande partie du réseau neuronal de LC0.
- Utilisation des fenêtres convolutionelles de taille 3x3xN (N : nombre de filtres).
- Utilisation d'une architecture neuronale supplémentaire totalement connectées dans le but d'extraire des informations de la position entière contrairement à l'architecture précédente.
- La fonction MSE est utiliséecomme fonction de coût du modèle.
- Utilisation d'une normalisation dans la couche de sortie à l'aide de la fonction TanH (Tangente Hyperbolique).
- Utilisation d'une méthode Stochastique pondérée qui est une technique qui aide à trouver des minimas globaux et optimise le temps d'entrainement et réduit les probabilités desur apprentissage.
- Offre un entrainement spécifique pour l'estimation de la nullité des positions.
- Utilisation de la méthode de probabilité nommée « Divergence de Kullback-Leibler» (DKL) (voir annexe B.5) qui est une mesure de la distance entre les probabilités. Si la divergence est faible, cela signifie que le réseau neuronal comprend les positions et performe bien. D'autre part, un plus grand nombre de diffusions n'est pas bon pour la précision du réseau et s'explique du fait qu'il est confus quant à la situation actuelle et nécessite davantage de travail pour déterminer la bonne valeur.

En outre, nous présentons ci-dessous quelques-uns des hyper-paramètres de son modèle d'apprentissage :

- Batch size : 2048.
- Nombre d'époques : 140000.

- Taille du buffer : 524288
- Utilisation de l'optimiseur GGT[1]
- Taux d'apprentissages initiaux : 0.02, 0.002, 0.0005 avec une d'option ReducecLROnplateau.

Ce moteur doté d'une technologie de pointe à savoir son réseau neuronal, nous a beaucoup inspiré dans la réalisation de notre application. Premièrement, tout comme LC0, nous avons utilisé une évaluation basée sur des prédictions de son réseau neuronal contrairement aux évaluations classiques linéaires. En outre, l'utilisation d'une architecture convolutive dans la reconnaissance d'une position de jeux d'échecs nous a poussées à appliquer le même principe vu les résultats prometteurs de cette architecture. Aussi, nous nous sommes inspirés des valeurs de ses taux d'apprentissage et d'autres hyper-paramètres ainsi que son architecture résiduelle qui booste les performances du réseau neuronal.

D'autre part, nous présentons ci-dessous quelques-unes des différences entre les modèles proposés (voir la section ci-dessous) et celui utilisé par LC0 :

- La complexité du réseau neuronal : le modèle de LC0 se présente comme étant plus complexe que le nôtre. Cela se présente par la différence du nombre de couches cachées des architectures convolutives ou denses ainsi que le nombre de neurones utilisés dans chaque couche. Cela s'explique par la différence de la qualité des ressources matérielles fournies pour la compilation et l'entrainement des deux modèles. De ce fait, nous étions contraints à réduire ces paramètres vu les performances matérielles utilisées (voir chapitre suivant section 4.1)
- Utilisation de la méthode DKL et l'extension SE.
- Donnée d'entrainement : Aucune données n'est fournie au modèle de LC0 au début de son entrainement vu le type de son apprentissage. Contrairement à notre modèle qui, initialement, utilise un grand nombre de positions échiquéennes.
- La représentation d'une position d'échecs : Notre modèle représente une position d'échecs en utilisant 13 bitboards contrairement à LC0 qui utilise la même représentation qu'AlphaZero avec cinq bitboards.

[1]Optimiseur adaptatif qui utilise le préconditionneur de matrice complète d'AdaGrad avec historique des gradients atténués exponentiellement.

3 Analyse des choix d'apprentissage

Avant d'avoir réalisé les spécifications fonctionnelles de notre application, différents besoins ont orientés nos choix lors de la conception de nos modèles (présentés dans la section suivante). Ces choix ont été parfois inspirés des caractéristiques d'apprentissage de LC0.

Ainsi, nos modèles ont été conçus en se basant sur un apprentissage supervisé contrairement à LC0 qui utilise un apprentissage par renforcement. Par la suite, nous étions confronté à choisir entre deux type d'apprentissage dans le cas de la supervision en l'occurrence entre la régression et la classification. Nous nous sommes inspirés dans ce cas du fonctionnement de LC0 à savoir la régression de son modèle neuronal qui se présente par sa sortie unique sous forme d'une probabilité. Plutôt que d'avoir une probabilité, nous générons une valeur incluse dans l'intervalle [-1,1]. Cela représente une normalisation afin d'avoir une vue sur l'évaluation de la position. Ainsi, si la sortie (x) du réseau neuronal tel que $-1 \leq x \leq -0.5$ cela signifie que l'évaluation de la position est largement défavorable pour les blancs. D'autre part, si $0.5 \geq x \geq 1$ cela signifie une évaluation largement favorable pour les blanc. Autrement, la position est plus ou moins équivalente.

L'application de l'apprentissage supervisé sur nos modèles et notamment le type régression exige une analyse spéciale des données avant celle de l'architecture du modèle. La position échiquéenne initialement fournie à notre modèle est représentée sur un ensemble de 844 bits. En premier lieu, nous avons pensé à séparer l'ensemble de ces entrées en deux types principaux et cela en fonction des architectures neuronales artificielles auxquelles elles sont destinées. Ainsi, notre contribution est d'entrainer chaque type de données à lui seul en formant deux sous-systèmes (sous-modèles) avant d'être concaténés et entrainés une fois de plus afin de relier les informations apprises par chaque sous-système et de partager leur connaissances. Ces deux types sont :

1. **Données sur la position des pièces :** cette partie représente un ensemble de 768 entrées (bits) qui sont réservées pour décrire l'emplacement des 6 différentes pièces des deux couleurs sur les 64 cases de l'échiquier ordonnées comme suite : Pions, cavalier, fou, tour, dame, roi (voir figure ci-dessous).

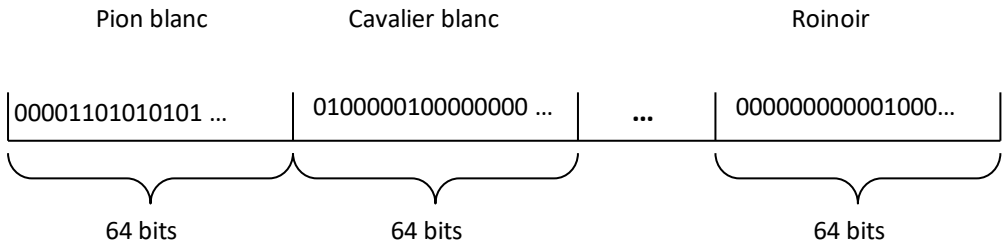


Figure III:1 La position des pièces

- **Les cases vides :** En plus des entrées précédentes, un bitboard supplémentaire (64 bits) sont ajoutées afin d'identifier les cases vides de l'échiquier. Ainsi, en rajoutant ce cas spécial nous aurons 13 bitboards au total (un bitboard représente 64 bits équivalent au 64 cases de l'échiquier).

Ce type de données est défini pour être utilisé par une architecture convolutionelle comme celle utilisée par LC0. Elle est appliquée pour une meilleure compréhension et apprentissage des informations positionnelles. Aussi, ce type d'architecture permet le traitement de la position échiquéenne sous forme d'une image. Théoriquement et en utilisant l'architecture ConvNet (souvent utilisée pour le traitement d'image, la reconnaissance d'objets etc.), nous pouvons considérer une case de l'échiquier comme étant l'équivalent d'un pixel d'une image. Ceci permet d'une part, de faire des relations entre les cases de l'échiquier avec le principe de convolution (ce qui est primordiale pour la compréhension de la position et ainsi avoir une bonne évaluation). D'autre part, cela nous permet d'utiliser moins de paramètres à entrainer ce qui réduit la complexité d'apprentissage et facilite leur mise à jour et nous fait gagner beaucoup de temps d'exécution comparé aux autres architectures.

2. **Données à informations supplémentaires :** Le reste des bits sont organisés de la manière suivante : 4 bits pour le petit et grand roque pour chaque joueur et 8 bits pour représenter la prise en passant. Pour ce dernier cas, nous avons utilisé que 8 bit pour représenter la prise en passant car nous nous intéressons qu'à la prise en passant des blancs vu que l'évaluation des positions est réalisée que pour ces derniers en se basant sur le principe de la symétrie (voir annexe C.1). De ce fait, contrairement au premier type, nous avons pensé à propager les données directement dans un réseau neuronal complétement connecté. Nous l'avons utilisé d'abord pour son statut du réseau neuronal standard et le plus utilisé et aussi dans le but d'apprendre et de relier chaque information supplémentaire avec toutes les autres informations.

En plus des deux architectures neuronales précédemment citées, nous avons testé une troisième architecture sur quelques-uns de nos modèles. Cette architecture est le ResNet (architecture résiduelle). Cette dernière est formée en reliant la n ième couche d'une architecture dense ou convolutionelle vers la (n + x) ième couche (voir figure ci-dessous). Ainsi, en offrant ce raccourci nous résolvons d'abord le problème de la disparition du gradient ainsi que la dégradation de la performance en enregistrant les informations passées entre les couches cachées en cas de perte au cours de l'entrainement.

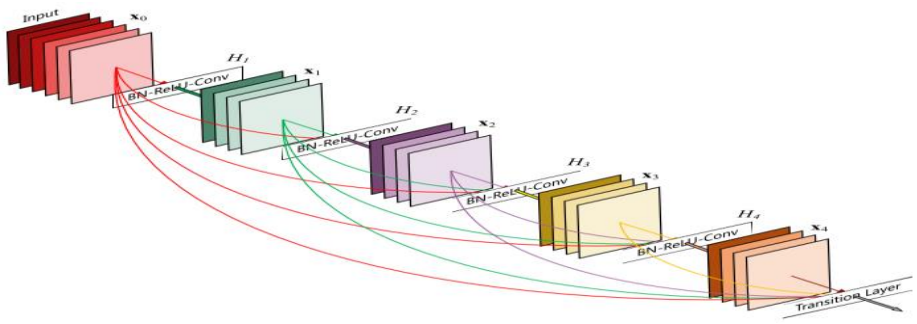


Figure III:2 Architecture ResNet

À noter que nous sommes restreint à utiliser un nombre moyen de couches cachées (maximum trois couches successives), qu'elles soient convolutives, complètement connectées ou résiduelles pour des contraintes matérielles de l'environnement d'entrainement de nos modèles (voir chapitre 4 environnement de développement). Contrairement à LC0, qui contient non seulement un réseau neuronal largement plus complexe que le nôtre notamment l'architecture ResNet (voir la section précédente), celui ci utilise un entrainement parallèle sur des machines clients des utilisateurs ce qui partage le coût de l'entrainement entre les différents clients et facilite l'apprentissage.

4 Modèles proposés

Nos propositions ont été faites en étroite relation avec l'implémentation. Ainsi, l'architecture globale de tous nos modèles répond au digramme décrit ci-dessous. Après avoir entrainé et testé plus d'une centaine de modèles, nous avons sélectionné et proposé quatre modèles parmi ceux qui semblent être les meilleurs de par leurs statistiques d'entrainement (voir chapitre 4 : Environnement) et leur performances lors des tests.

Une première propriété en commun est la gestion des données pour tous les modèles (voir chapitre 4 section 4.2.2). Une autre caractéristique commune consiste à fixer la taille de la fenêtre de convolution à trois. Cela peut être expliqué par le fait que nous voulons apprendre aux modèles à relier l'information requises dans chaque case de l'échiquier avec les 8 cases qui entourent cette dernière. Aussi, nous avons fixé le paramètre padding à « same » dans le but de préserver la taille de données entrantes par cette couche. Une autre similarité se retrouve dans la taille de la fenêtre du pooling qui est fixée à 2 (nombre le plus utilisé dans l'état de l'art) (57). Nous avons laissé la valeur stride par défaut à 2 dans notre cas. En outre, toutes les couches convolutionelles ou denses utilisent la fonction d'activation « relu ». À noter qu'initialement, tous les poids et bias des modèles sont générés aléatoirement.

Concernant l'algorithme d'entrainement (optimiseur), nous avons opté pour Adam en raison de son efficacité dans les problèmes de régressions. En outre, nous avons choisi d'utiliser la même fonction de coût qui est MSE et la même métrique à savoir MAE. Finalement, tous les modèles partagent le même nombre d'époques (1200) et aussi l'utilisation d'un *callback* composé de trois fonctions : Arrêt précoce (40 époques), RedueLROnplateau (réduction de 0.1 chaque 2 époques) et tensorboard.

La différence majeure entre tous les modèles réside dans le nombre des différentes couches utilisées ainsi que le nombre de neurones de chaque couche. Il existe aussi une différence dans les valeurs d'autres hyper-paramètres qu'on précisera ci-dessous pour chaque modèle comme le nombre de batch size, le taux d'apprentissage, etc. Concernant le nombre de neurones des couches cachées choisi où nous avons utilisé une formule appliquée parfois dans le domaine professionnel et souvent dans les petits projets d'apprentissage automatique comme le nôtre. L'équation nous a été proposée par le Dr Andrew NG (chercheur en AA et professeur à l'université de Stanford) et elle est définit comme suit :

$$Nnc=|\ \alpha\sqrt{Nne * Nns}\ |\ (58)\ (59)$$

Nnc, Nne, Nns respectivement le nombre de neurones cachés, d'entré, de sortie de la couche courante. αreprésente un nombre réel arbitraire tel que $1{,}5 \leq \alpha \leq 10$

Équation III-1 Nombre de neurones cachés

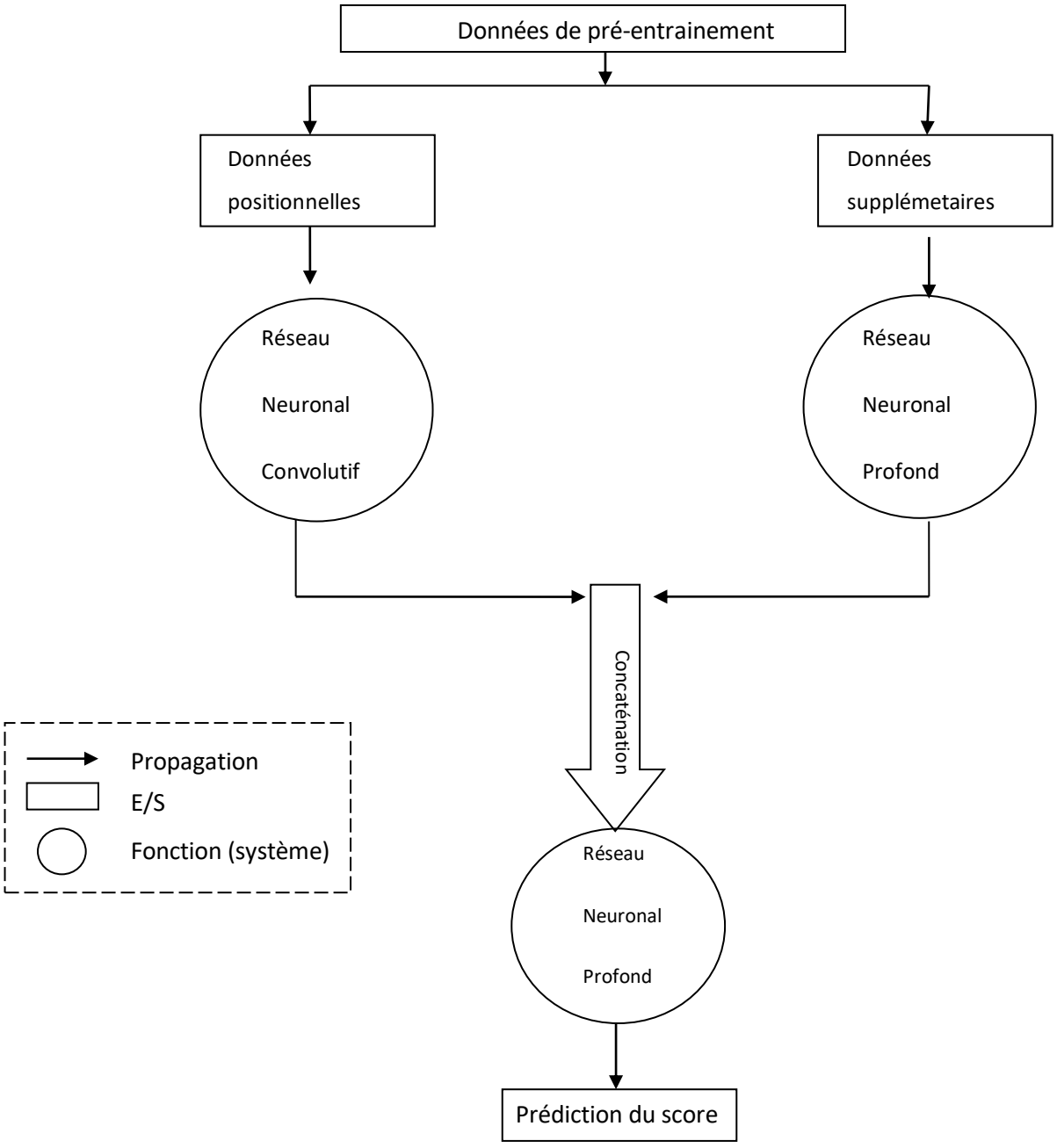


Figure III:3 Diagramme de flux de données générale de nos modèles (DataFlow)

4.1 Modèle RNA1

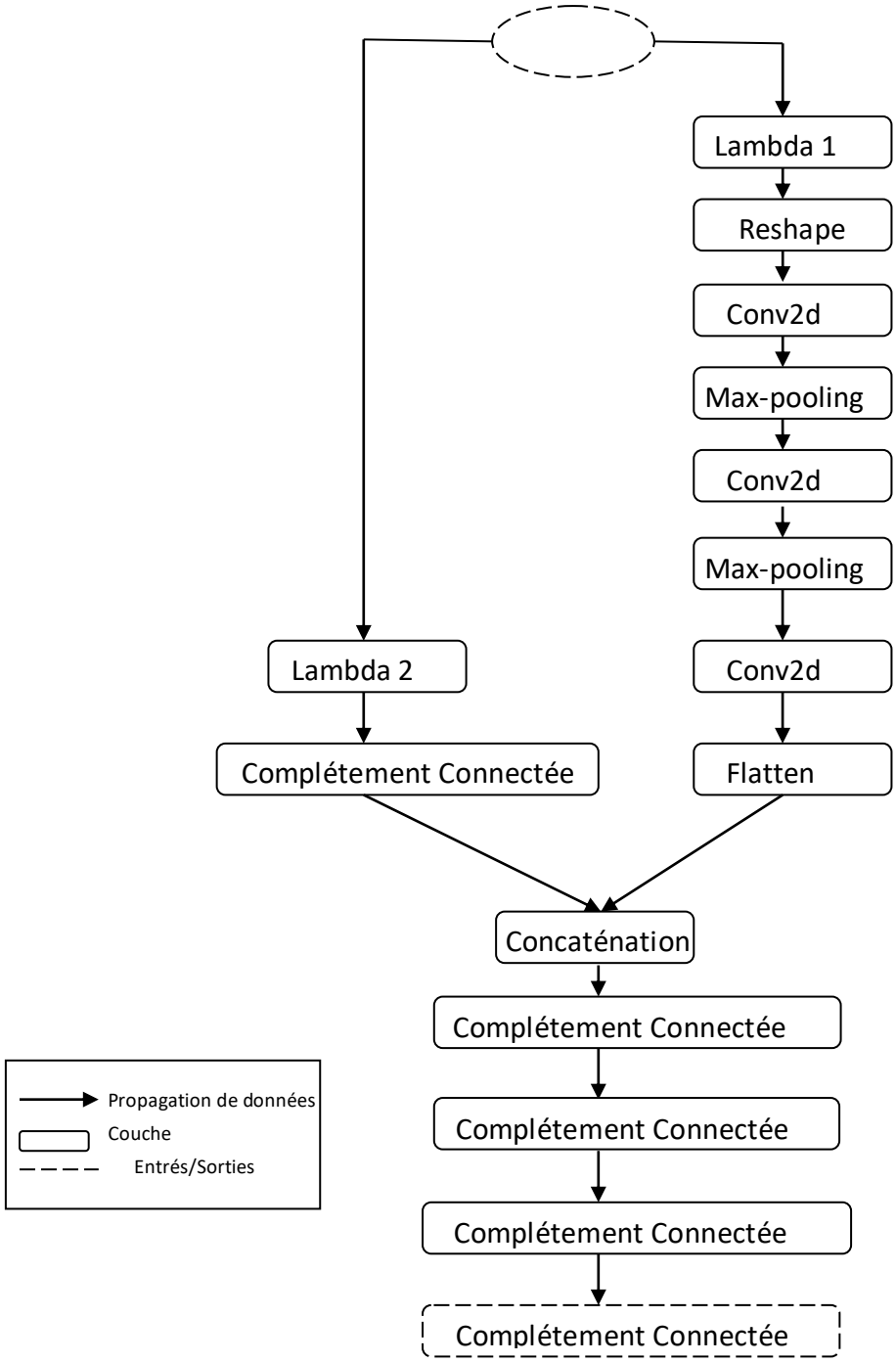


Figure III:4 Architecture du modèle RNA1

Le graphe de ce modèle illustré dans la figure ci-dessus, décrit les composants essentiels de son architecture. Ainsi, le premier sous-système (pour l'entrainement des informations positionnelles) est composé d'une suite de deux couches convolutionelles suivie chacune d'une couche de Maxpooling auxquels en rajoute finalement une dernière couche convolutionelle. Á noter que le nombre de filtres pour toutes les couches convolutionelles est de 200 (aléatoirement choisis). Aussi, nous avons utilisé deux couches de Maxpoolig2D afin

de changer la taille initiale de (8, 8,13) à (2, 2, 200) à la sortie de la seconde couche du maxpoolig.

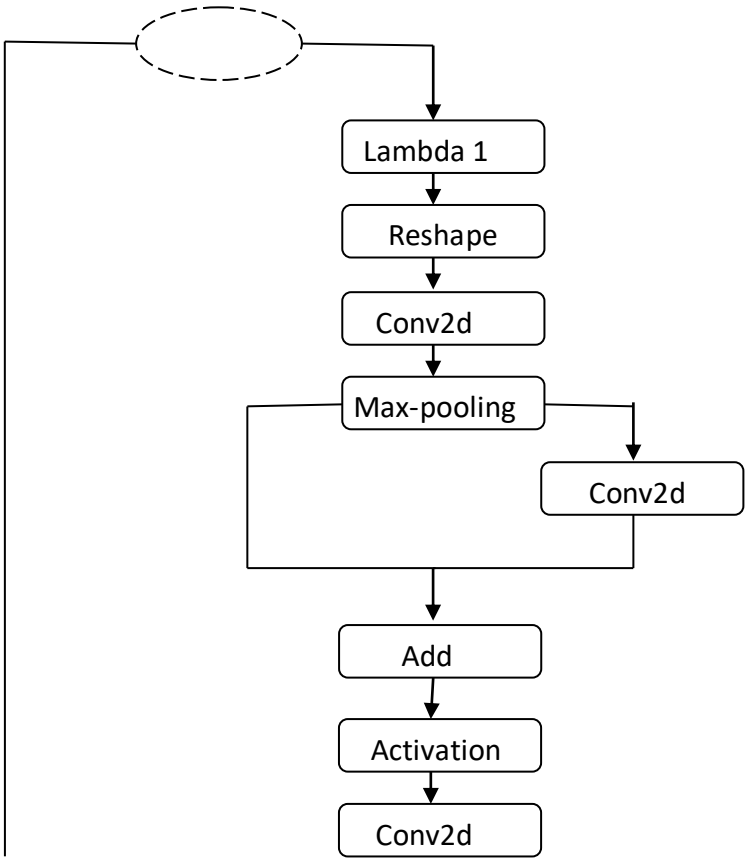
Pour l'autre sous-système (apprentissage des informations supplémentaires) nous avons utilisé une seule couche complétement connectéeavec 50 neurones. Ce nombre a été calculé par l'équation citée précédemment avec les valeurs de variables suivantes : Nne=12, Nns=50, α=2.

Dans le troisième sous-système (réseau complétement connecté après la concaténation), nous avons utilisé trois couches denses de 100 neurones. 100 neurones selon l'Equation 1 avec notamment Nne=850, Nns=1,α=3.5

Enfin, nous avons utilisé un taux d'apprentissage de 0.001 (valeur par défaut pour Adam) et un batch size de 200 (valeur choisie de telle sorte à avoir un temps d'entrainement raisonnable).

Ce modèle se caractérise aussi par son nombre de paramètres d'apprentissage réduit par rapport aux autres modèles, une notion recommandée pour des projets amateurs.

4.2 Modèle RNA2



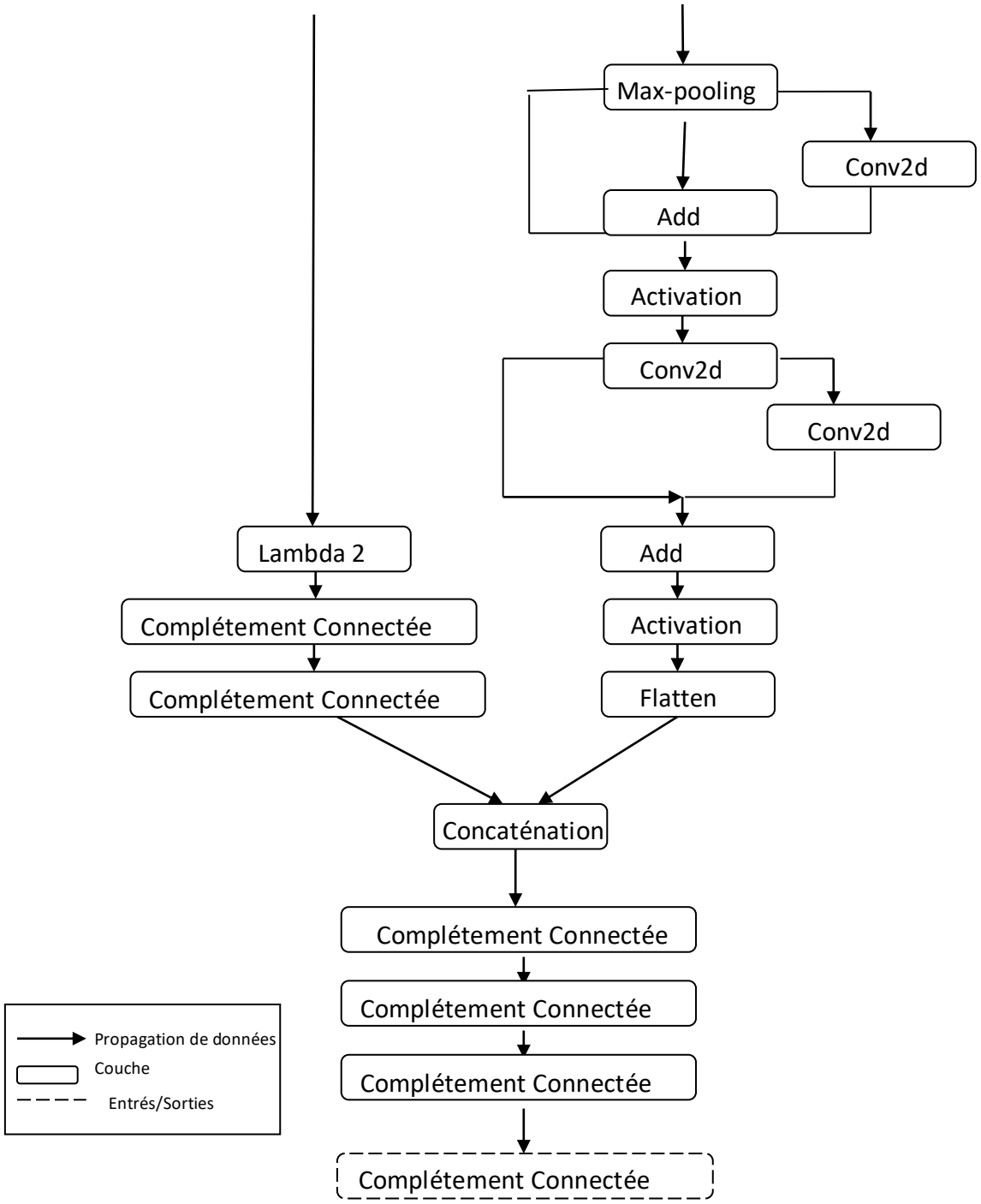


Figure III:5 Architecture du modèle RNA2

L'analyse de la figure ci-dessus pour le deuxième modèle montre que le premier sous-système est composé d'une couche convolutionelle de 150 filtres (aléatoirement choisis dans le but d'accroitre successivement le nombre de paramètres d'un modèle à un autre) suivi d'une couche maxpooling qui est reliée à une architecture résiduelle qui contient une couche convolutionelle de 150 filtres qui se poursuit en reliant les deux bouts et se terminant sur une couche d'activation de type « relu ». A partir de ce modèle, nous avons introduit la notion de ResNet dans le but d'avoir de meilleures performances et pour des besoins de sécurité contre le problème du vanishing gradient. Ainsi, les informations peuvent circuler sans entraves dans

l'ensemble du réseau. La même suite de couches décrite précédemment (S_1) est rajoutée une seconde fois avant de se poursuivre finalement avec une troisième mais sans couche de pooling.

Pour l'autre sous-système (apprentissage des informations supplémentaires), nous avons utilisé deux couches cachées complétement connectées (la norme entre 1 et 3) avec 25 neurones chacune calculé par l'équation citée précédemment avec les valeurs de variables suivantes : Nne=12, Nns=25, α=1,5. .

Pour le dernier sous système, nous avons gardé l'architecture de RNA1 avec les mêmes hyper-paramètres. Nous avons fait ce choix afin de juger la progression du modèle RNA1 en changeant que la structure du premier sous modèle et ses paramètres.

Enfin, nous avons utilisé untaux d'apprentissage de 10^{-4} avec un batch size de 120. Nous constatons une réduction du taux d'apprentissage afin d'éviter le sur-apprentissage. Aussi, une réduction de la taille du lot vu l'augmentation du nombre de paramètres afin d'avoir un temps d'entrainement raisonnable.

4.3 Modèle RNA3

La figure ci-dessous décrit l'architecture de troisième modèle RNA3. Nous constatons une similarité avec le modèle RNA1 à l'exception de l'inexistence d'une troisième architecture ResNet. Nous avons réalisé cela dans le but d'optimiser le second modèle et le nombre de neurones pour le second sous-système où nous avons utilisé 50 neurones pour chaque couche (règle utilisé dans RNA1). Enfin, nous avons utilisé un taux d'apprentissage de 10^{-5}, encore réduit par rapport à RNA2 dans l'idée d'avoir quelques oscillations au cours de l'évaluation de l'apprentissage et ainsi évité le problème de la stabilité de la fonction d'erreur qui est apparue dans RNA2. Notons aussi l'utilisation d'une taille de lot de 300. Une valeur choisie afin d'accroitre le nombre d'exemple présentés dans chaque époque, ce qui permet à notre algorithme d'apprendre d'avantage de relations.

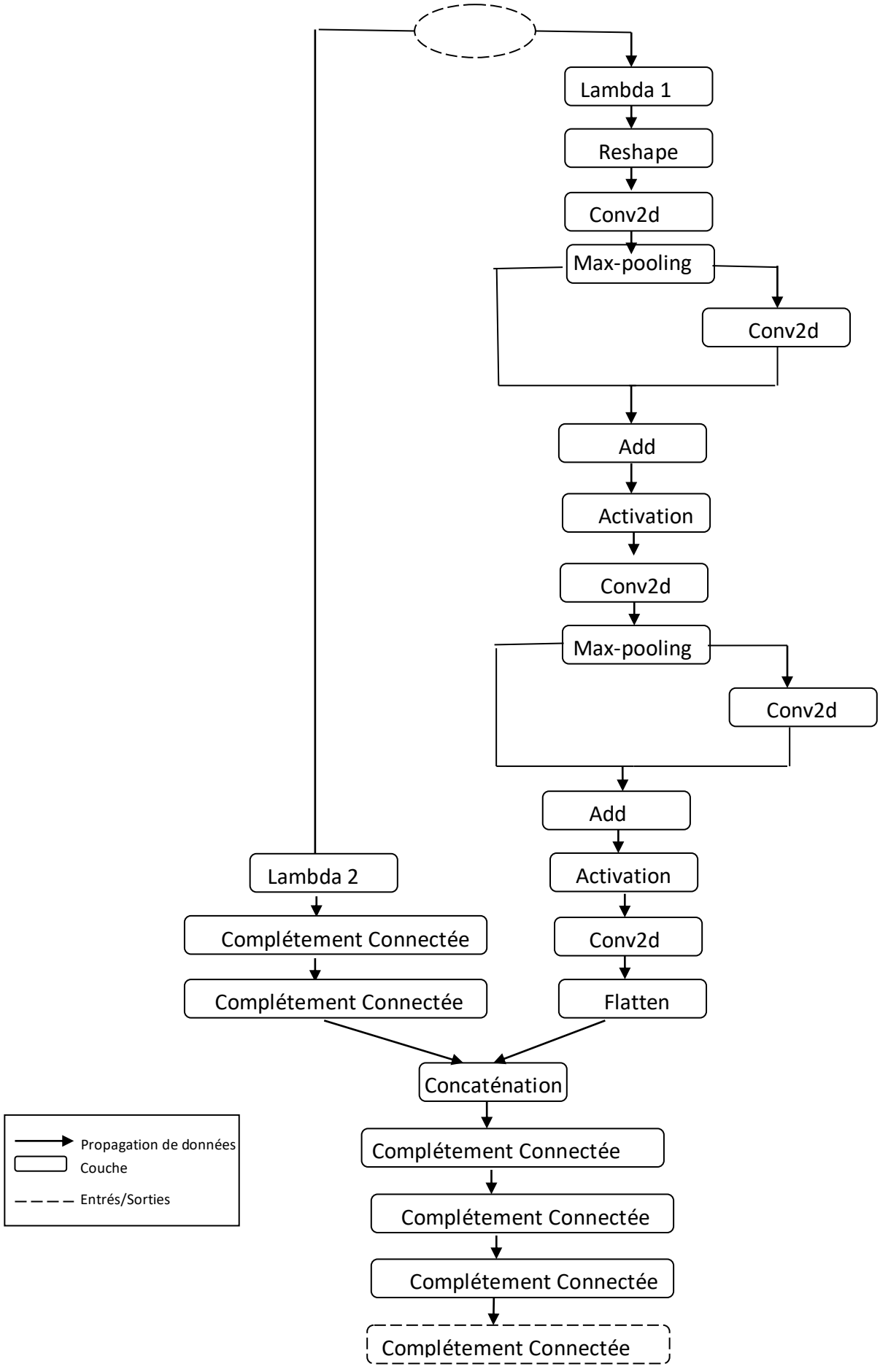


Figure III:6 Architecture du modèle RNA3

4.4 Modèle RNA4

L'architecture du modèle RNA4 est similaire à celle de RNA3 à l'exception du nombre de neurones du premier sous-système qui est de 300 (deux fois celle de RNA3). En effet, c'est une valeur que nous avons choisi aléatoirement juste pour s'appuyer sur le principe disant qu'en augmentant le nombre de filtres dans l'architecture ConNet cela permet à notre algorithme d'assimiler plus de détails sur une position et ainsi nous pourrons extraire d'avantage d'informations concernant chaque case de l'échiquier. Nous avons aussi fixé 150 neurones cachés pour le second sous-système en appliquant l'équation 1 avec les valeurs suivantes : Nne=12, Nns=150, α=3,55.

Pour le reste des couches du dernier sous-modèle, nous avons opté pour ces valeurs respectivement pour les trois couches : 200, 150,100. Un choix qui reste dans la logique de l'équation 1 mais dont nous avons varié les valeurs avec un ordre descendant.

Enfin, il faut mentionner que nous avons utilisé un taux d'apprentissage de 10^{-4} car le modèle précédent risquait d'avoir un problème de sous apprentissage. Enfin, nous avons réduit la taille du lot à 180 dans le but de réduire le temps d'entrainement vu le nombre élevé de paramètres et d'hyper-paramètres utilisés dans ce modèle.

5 Métriques d'évaluation

L'évaluation de l'algorithme d'apprentissage est une étape essentielle dans tous projets d'apprentissage automatique. Nous avons utilisé la fonction « Mean Square Error » (MSE) comme fonction de coût car c'est la plus utilisée dans les problèmes de régression (60). Elle permet de calculer l'erreur quadratique entre les données prédites et celles attendues. En plus, elle permet de calculer le gradient plus facilement que les autres métriques (61) (62).

De plus, nous avons utilisé des métriques afin de suivre l'état et la progression du modèle dans la phase de validation. Par défaut, la fonction de coût utilisée est aussi intégrée comme métrique d'évaluation. Nous avons utilisé la métrique RMSE (voir figure ci-dessous) qui est représentée comme étant la racine carré de MSE. Elle est utilisée généralement pour montrer et détecter les grandes déviations et accorde un poids relativement élevé aux erreurs importantes. Cela signifie que le RMSE devrait être plus utile lorsque des erreurs importantes sont particulièrement indésirables.

Aussi, nous avons appliqué la métrique MAE qui calcule la moyenne des différences absolues des erreurs et cela dans le but de pénaliser les grandes erreurs. Elle est caractérisée

par sa non ambigüité ce qui fait d'elle la métrique la plus utilisée dans tous les projets d'apprentissage automatique.

En plus des métriques citées précédemment, nous avons implémenté une autre métrique (Metric_evaluation) qui permet essentiellement d'estimer le pourcentage des parties dont les signes des évaluations prédites et celle attendues sont différents. De ce fait nous pourrons avoir une idée générale sur la performance du modèle, car plus ce pourcentage est réduit, plus le modèle est bon (voir plus de détails sur l'implémentation dans le chapitre 4).

Enfin, « Metric_evaluation » ne peut pas être utilisée comme fonction de coût de notre modèle à cause de sa non-dérivabilité. Une notion primordiale pour toutes fonctions de coût vu qu'elle sera utilisée lors de la mise à jour des poids où nous aurons besoin de calculer la dérivée de la fonction d'erreur (voir formule chapitre 2 : Entrainement cyclique).

6 Conclusion

Dans ce chapitre, nous avons présenté les caractéristiques architecturales de LC0 qui est l'un des meilleurs moteur de jeu d'échecs de nos temps et qui utilise l'évaluation des positions à l'aide de l'apprentissage automatique. Ensuite, nous avons décrit l'architecture détaillée de chaque modèle que nous avons sélectionné. Enfin, nous avons analysé les métriques d'évaluation utilisées par nos modèles et plus précisément notre métrique personnalisée : Metric_evaluation.

Dans le prochain et dernier chapitre, nous offrons une analyse et une étude des outils utilisés pour la conception et l'implémentation de notre application suivant les normes du génie logiciel. Enfin, nous proposons une expérimentation de nos modèles au cour d'un tournois qui sera organisé à l'issue duquel nous prévoyons la distinction du modèle RNA1 parmi les modèles conçus vu son architecture simple et ses statistiques prometteuses lors de son entrainement. En outre, nous prévoyons aussi une confrontation en finale avec le moteur utilisant une évaluation statique.

Chapitre 4: Conception, implémentation et expérimentations

1 Introduction

Notre projet consiste à réaliser un moteur de jeu d'échecs complet avec notamment en plus d'une évaluation statique explicitement programmée, nous avons proposé un deuxième type à savoir une évaluation à l'aide de l'apprentissage automatique. Cette dernière se caractérise sous forme de plusieurs modèles de réseaux de neurones artificiels conçues, implémentés puis expérimentés.

Toutefois, notre application consiste en deux parties principales en l'occurrence un moteur de recherche échiquéen doté de plusieurs méthodes de recherche, ainsi qu'un modèle de réseau de neurones artificiel utilisé par la première partie (recherche) pour l'évaluation des positions échiquéennes.

Dans ce chapitre, nous présentons les démarches suivies du début du projet jusqu'à son déploiement. Pour ce faire, nous introduisons par les analyses des besoins réalisées et que nous avons classifiés en besoins fonctionnels et non fonctionnels. Ensuite, nous présentons le diagramme des classes et les relations entre elles. Par la suite, nous fournissons les outils et bibliothèques utilisées tout au long de la phase de développement qui est suivie par une description de son déploiement. Ce qui nous emmène à formaliser la méthode du développement utilisée dans notre projet. Enfin, nous présentons une évaluation des résultats visualisés lors de l'expérimentation de notre application.

2 Analyse et conception

Notre application a été conçue afin d'être utilisé par différents acteurs[1]. Il existe deux acteurs principaux auxquels notre application est destinée :

- **Utilisateurs directs:** il s'agit des interfaces graphiques pour utilisateur (GUI) de jeux d'échecs qui utilisent notre application comme moteur de recherche et d'évaluation[2] en utilisant une communication standard , en l'occurrence :UCI (59) (voir annexe D.3).
- **Utilisateurs indirects:** il s'agit de toute personne qui appartient à la communauté échiquéenne qui l'utilise dans ses analyses de parties.

[1] Une personne, un logiciel ou un matériel qui peut interagir avec le système.
[2] A l'image de « Arena », « Scid vs PC » ou « Lucas chess ».

2.1 Besoins fonctionnels et non fonctionnels

Notre projet avec ses deux parties (moteur d'échecs et modèle neuronal) apporte des solutions à deux types de besoins: fonctionnels et non fonctionnels.

2.1.1 Besoins fonctionnels

Les besoins fonctionnels représentent l'ensemble des fonctionnalités de notre application qui s'exécutent lors de la réception d'une action donnée par l'utilisateur.

2.1.1.1 Cas du moteur d'échecs

En premier lieu, nous avons besoin d'intégrer une interface en ligne de commande afin d'utiliser le protocole UCI avec la plupart de ses commandes (voir annexe D.3). L'utilisation d'une telle fonctionnalité offre à notre application la possibilité d'interagir et de communiquer avec les GUIs de jeux d'échecs qui intègrentce protocole largement répandu.

De plus, notre moteur de recherche nécessite la possibilité de choix entre une évaluation statique et une évaluation basée sur les réseaux de neurones artificiels. Cela s'explique par le fait que nous voulons séparer les moteurs de recherche entre eux afin de les préparer à la phase de l'expérimentation.

En outre, nous désirons doter la fonction de recherche de plusieurs heuristiques et de méthodes à l'instar de l'élagage alpha béta, la recherche à base de fenêtre d'aspiration, la recherche à base de variations principales etc. Ces fonctions sont implémentées afin d'optimiser la qualité de la recherche ainsi que sa rapidité en ignorant plusieurs nœuds de l'arbre de recherche.

De plus, nous exigeons l'importation et l'exploitation d'un modèle RNA une seule fois pendant la recherche. Ce besoin est mis en œuvre pour un chargement rapide et facile du moteur d'échecs lors de son intégration dans un GUI ou lors de son exécution en ligne de commande.

Enfin, nous avons besoin de doter le moteur d'une interface simulant des parties entre deux moteurs passés en paramètre et retourner le résultat final de la partie. Ce besoin est ajouté afin d'avoir plus de choix dans l'expérimentation des moteurs d'échecs par les utilisateurs qui ne veulent pas avoir recours à des GUIs de jeux d'échecs. En effet, l'expérimentation en utilisant cette méthode est beaucoup plus facile pour les débutants car elle contient moins de configurations et de bugs.

2.1.1.2 Cas du modèle neuronal

Pour cette seconde partie de notre projet, nous voulons en premier lieu extraire des informations spécifiques à partir de plusieurs parties jouées, évaluées et enregistrées sous format PGN. De ce faite, ceci nous permet de récolter l'ensemble d'entrainement nécessaire pour établir notre apprentissage supervisé et qui sera par la suite propagé dans un réseau neuronal.

En outre, nous désirons avoir une bonne gestion des positions enregistrées dans la BDDregroupant ainsi la lecture, la normalisation et la sauvegarde des données normalisés (de pré-entrainement) sur le disque dur. Ce besoin concerne d'une part les développeurs, car c'est une fonctionnalité qui est considérée comme étant une bonne pratique et qui offre plus d'organisation au projet. D'autre part, cela concerne aussi l'utilisateur, car un chargement de données de pré-entrainement à partir d'un fichier numpy est plus rapide que celui de la BDD auxquels nous appliquant une normalisation à chaque chargement de données.

De plus, nous souhaitons fournir une fonction de représentation de parties sous forme de bitboards unique de son genre. Cette représentation est primordiale pour tout modèle échiquéen. La raison est qu'elle permet de livrer la plupart voir toutes les informations concernant une position échiquéenne donnée. Cependant, la qualité de l'entrainement de nos modèles est relative à la bonne représentation de nos données.

Aussi, nous espérons que cette partie réponde au besoin primordial auquel elle est destinéeà savoir la construction d'un modèle de RNA et son entrainement avant de l'avoir enregistré. Ceci nous permet finalement d'avoir un modèle bien entrainé prêt àêtre importer par le moteur d'échecs.

Finalement, nous avonsbesoin d'intégrer des moyens pour la visualisation des entrainements du modèle neuronal à temps réel ainsi que ses résultats obtenus puisque cela nous permet de suivre l'évolution de l'apprentissage ainsi que les performances du modèle.

2.1.2 Besoins non fonctionnels

Les besoins non fonctionnels représentent les capacités et les performances de l'application en se basant sur les facteurs suivants : la sécurité, l'utilisabilité, le déploiement et le matériel.

2.1.2.1 Cas du moteur d'échecs

Nous voulons offrir à l'application une gestion d'erreur contre les commandes UCI erronées par mesure de sécurité. Cela évite tout dysfonctionnements ou bugs du moteur

d'échecs en analysons toutes les commandes reçues sous le slogan informatique « *Never trust user inputs* [3]». Du côté des performances, nous espérons que notre moteur d'échecs accorde un temps de réponse et de traitement réduit aux communications UCI. Ce dernier besoin s'avère très important car plusieurs moteurs d'échecs génèrent des erreurs lors de leurs déploiements en les intégrant dans des GUIs (comme Arena) qui imposent un temps de réponse bien précis.

De point de vue commercial, nous avons besoin d'un moteur d'échecs compatible avec tous les GUIs de jeux d'échecs. Cela passe par l'identification du protocole en commun utilisé par la plupart des GUIs (plus il est compatible avec les GUIs plus ses ventes augmentent). Aussi, nous voulons offrir une maintenabilité et une testabilité au programme écrit ce qui nous permet d'entretenir facilement l'application et d'adopter des mises à jour ou des corrections d'erreurs après son déploiement.

2.1.2.2 Cas du modèle neuronal

Nous souhaitons avoir la possibilité d'utiliser une base de données afin d'enregistrer les positions récoltées pour l'apprentissage. De plus, nous voulons optimiser la rapidité du chargement des données pour des entrainements ultérieurs en les enregistrant dans des fichiers Numpy ce qui offre une portabilité pour les données. Cela permet principalement de réduire le temps d'exécution. Enfin, tout comme le moteur d'échecs, nous espérons que cette partie nous offre la possibilité de la maintenabilité et de la testabilité vu qu'ils sont des facteurs fondamentaux pour un bon suivi d'une application.

2.2 Diagramme de classes

Le diagramme de classes (voir figure ci-dessous) est un outil incontournable offert par l'UML[4] et qui permet de décrire l'état statique d'un système (notre application avec ses deux parties) en tenant compte de toutes les relations entre les différentes classeset en faisant abstractions du comportement du système.

Ainsi, comme l'illustre le diagramme de classe, le thread principal « lecture » reçoit des commandes UCI où en fonction de ces dernières elle communique avec la classe « engine ». Cette dernière permet de gérer les commandes reçues par « lecture » et générer des réponses qui répondent aux normes du protocole UCI. La méthode « go », déclenche un autre thread (thread secondaire) « Reflexionchess » qui permet quant à lui de lancer la fonction de recherche. Ce dernier, étant le cœur du moteur d'échecs, communique avec plusieurs autres

[3] Ne jamais croire en la validité des entrées de l'utilisateur.
[4] UML : Unifed Modeling Language

classes comme la « Transposition table » qui gère la méthode heuristique de la table de transposition. Aussi, ce dernier thread communique avec la classe qui gère l'ordonnancement des coups et enfin avec une classe abstraite qui gère les deux types d'évaluation à savoir l'évaluation statique et celle qui utilisele modèle RNA généré à partir de la classe Model ANN. Le modèle créé par la classe model ANN et sauvegarder sous l'extension « .h5 » est importé et utilisé par la classe « Reflexionchess ». La classe « Repchess » permet de générer la représentation d'une position d'échecs donnée comme spécifiée dans le chapitre précédant.

La classe « Arena » est un thread qui simule la fonctionnalité principale du GUI Arena Chess. Elle permet la gestion des tours entre les joueurs ainsi que l'interface en sortie. Une autre classe est celle de « Log » qui permet (après avoir reçu de la part de la classe précédente la Fen et l'indexe du moteur ayant le trait) d'invoquer la classe « Startgame » qui, quant à elle permet d'envoyer quelques commandes UCI automatiquement (contrairement à la classe « Lecture » qui se fait par des entrées en ligne de commande) pour la classe « Engine » précédemment décrite.

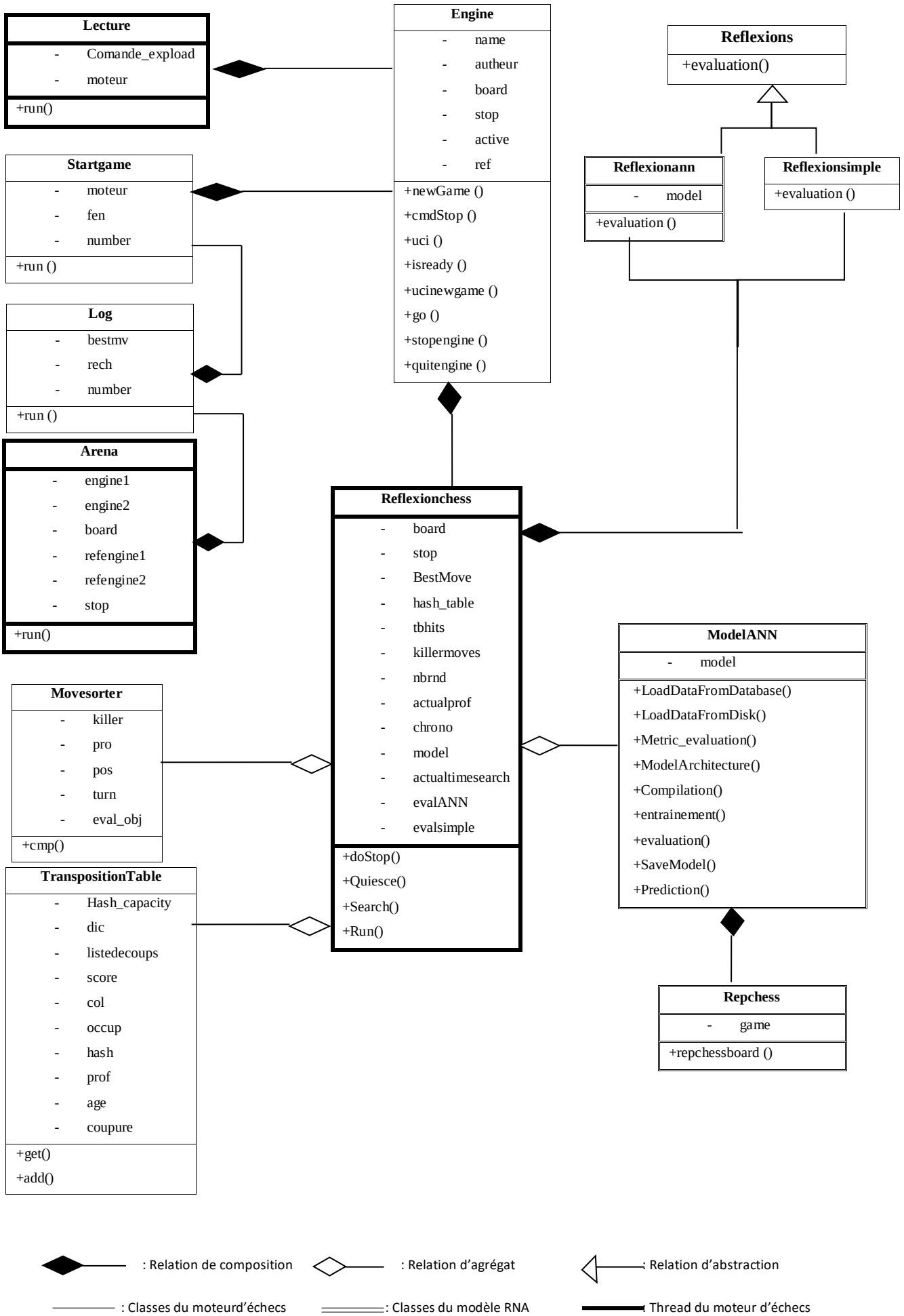


Figure IV:1 Le diagramme de classes de notre application

3 Implémentation

3.1 Outils et environnements de développement

3.1.1 Visual Studio Code

Virtual Studio Code (VSC) (60) est un éditeur de code multiplateforme[5] développé par Microsoft en 2015.Son code source est gratuit et disponible sous la licence MIT[6] et figure parmi les outils de développement les plus utilisés comptant une très grande communauté. En outre, VSC inclut plusieurs langages de programmation, le contrôle GIT et GitHub ainsi que des extensions et des configurations très pratiques pour les développeurs comme la possibilité de débogage, l'invite de commandes intégré ou même la gestion des fichiers et des sources.

3.1.2 Navigateur de base de données pour SQLite

Le navigateur de base de données pour SQLite *(DB Browser for SQLite)* est un logiciel libre, open source, gratuit, autonome, sans serveur, écrit en C. Il fournit une interface graphique afin d'utiliser un moteur de base de données à l'aide des commandes SQL en arrière-plan sans connaissances préalables. Il assure le principe d'ACID [7] ainsi que la création, la conception et l'édition des fichiers de base de données compatibles avec SQLite.

Il est fréquemment utilisé dans des petits projets qui manipulent une petite quantité de données. La base de données créée est stockée dans un fichier en disque dur portant l'extension « .db ». Il est utilisé dans de nombreuses applications en tant que stockage de données interne.

3.1.3 TensorBoard

TensorBoard (61) est une suite d'outils de visualisation graphique à temps réel sous forme d'une application web qui permet de lire les données exécutés sur tensorflow. Il est utilisé comme un débogueur afin d'extraire des informations importantes des données et de suivre les résultats de l'exécution pour aboutir à une performance optimale.

3.1.4 Arena

Arena est une interface graphique créée par Martin Blumeen 2003, et qui supporte des moteurs de jeux d'échecs fonctionnant avec l'un des deux protocoles : UCI ou WinBoard. Elle assure un mode d'analyse simple ainsi que les tournois. En outre, Arena offre des

[5] Possibilité de fonctionnement sous Linux, Windows ou Mac
[6] Massachusetts Institute of Technology
[7] ACID : Atomicité, Cohérance, Isolation, Durabilité.

bibliothèques d'ouvertures et de finales et la possibilité d'importation et d'exportation de parties.

3.2 Bibliothèques logicielles

3.2.1 Python-chess

Python-chess (62) est une librairie de jeux d'échecs écrite en Python par Niklas Fiekas en 2014. Elle inclut plusieurs fonctions de bases comme la génération et la validation des coups, la manipulation des pièces et de l'échiquier et implémente les deux protocoles UCI et XBoard. En outre, elle permet de manipuler les fichiers PGN de même que les tables d'ouvertures et de finales.

3.2.2 Tensorflow

Tensorflow (63) est l'une des meilleures bibliothèques de bas niveau en Python destinées à l'apprentissage automatique. Elle est développée par Google en 2015 et distribuée sous la licence Apache. Son nom est composé de « tensor » qui désigne la représentation vectorielle des données sous forme de tenseurs[8] qui obéissent aux transformations et aux opérations algébriques, et de « flow » qui désigne le flux des données représenté par ces tenseurs.

Tensorflow est multiplateforme (peut être déployé sur CPU, GPU, TPU[9]), et offre la possibilité du parallélisme de données et de modèles comme il offre aussi plusieurs outils de conception, construction et d'entrainement de modèles. Il peut être utilisé seul où comme backend d'un framework de haut niveau où nous pouvons distinguer des concurrents comme Theano, Caffe, CNTK et Torch.

Cette technologie de pointe est tellement puissante qu'elle est utilisée par les plus grandes entreprises du monde comme Snapchat, Uber et Airbus, mais aussi exploitée dans les travaux de recherches aéronautiques du géant NASA et dans le domaine de la sécurité comme au Pentagone.

3.2.3 Keras

Keras (64) est la bibliothèque de haut niveau la plus utilisée dans le développement des applications en apprentissage automatique. Elle utilise Python comme langage de programmation et est exécutée sur Tensorflow ou Theano par défaut. Elle tire sa popularité de sa facilité d'utilisation, de débogage et d'exploration notamment pour les développeurs débutants en intelligence artificielle. Ainsi, elle permet de développer rapidement tout un

[8] Il s'agit d'une généralisation de matrice à plusieurs dimensions (rangs) allant de 0 à n.
[9] TensorProcessing Unit

réseau de neurones en faisant abstractions de plusieurs détailles qui sont pris en charge par le backend.

De plus, Keras supporte la majorité des types de réseaux de neurones comme récurrents, denses, convolutionels. Aussi, elle est doté de deux types de modèles : séquentiel et fonctionnel (voir annexe D.1 et D.2). Ce dernier type a été utilisé pour définir l'architecture du modèle de réseau de neurones de notre application.

3.2.4 Numpy

Numpy (65) est un module de Python utilisé dans la manipulation des objets vectoriels de même type de données comme les matrices ou les tableaux et assure toutes les opérations arithmétiques, logiques, statistiques et même des fonctions qui visent à manipuler les données enregistrées comme le tri , la recherche du maximum et le calcul de la moyenne. En outre, cette bibliothèque permet de sauvegarder les données dans un fichier portant l'extension «. npy ».

3.2.5 Sqlite3

Sqlite3 (66) représente un module de Python introduit par Gerhard Haring et qui fournit une interface SQL conforme à la spécification DB-API 2.0. Il est constitué de plusieurs fonctions qui permettent de réaliser toutes les opérations SCRUD[10].

3.3 Cycle de développement

Dans le cadre de la bonne conduite de notre projet et du développement de notre application de jeux d'échecs, nous avons utilisé une des normes du génie logiciel à savoir le cycle en V (voir figure ci-dessous).

L'intérêt principal que nous portons au cycle en V vient du fait qu'il est intuitif, simple à mette en œuvreet aussi performant notamment avec son approche proactive[11].

Cette méthodologie se compose en trois phases principales : conception, réalisation et tests.

3.3.1 Conception

Afin d'établir une bonne conception de notre application de jeux d'échecs, nous avons eu recours à 4 étapes principales :

[10] SCRUD : Search(rechercher), Create(créer), Read(lire), Update (mettre à jours), Delete(supprimer).
[11] Effectue une vérification à chaque étape de conception.

1. **Expression du besoin :**nous avons présenté dans cette étape l'idée de base et l'objectif principal de notre projet à savoir un moteur de jeu d'échecs et un modèle de réseau de neurones. Aussi, nous avons fait une estimation de la faisabilité du projet c'est-à-dire s'il est techniquement réalisable en fonction du temps et de la complexité de l'application.
2. **Spécifications fonctionnelles :** nous avons défini les fonctions principales du moteur d'échecs et les taches qu'il pourra exécuter une fois quele projet sera réalisé.
3. **Conception Architecturale :** dans cette étape nous avons défini toutes les entités et les classes de notre application ainsi que les relations entre elles en faisant abstractions des détails (voir figure IV.1).
4. **Conception détaillée :** nous avons précisé les méthodes et les attributs de chaque classe décrite dans la conception architecturale.

3.3.2 Réalisation

Dans cette étape nous avons traduit en codage informatique toute la conception détaillée faite auparavant en tenant compte des fonctionnalités exigées.

3.3.3 Tests

Enfin, nous sommes passés à la phase des tests où nous évaluons les performances de nos modèles RNA. Pour ce faire, nous avons suivi 4 étapes successives:

1. **Tests unitaires :** après le développement des méthodes nous testons leur fonctionnement par module. Par exemple, lors de cette étape nous avons développé le programme « EvalPrediction.py » afin de tester le module d'évaluation des modèles RNA.
2. **Tests d'intégration :** après avoir testé les modules développés, nous avons procédé à des tests de l'ensemble des modules qui communiquent entre eux comme l'intégration de la fonction d'évaluation dans la fonction de recherche.
3. **Tests fonctionnelles :** dans cette étape nous avons vérifié si l'application que nous avons développée est bien conforme aux spécifications fonctionnelles décrits précédemment.
4. **Déploiement et maintenance:** la dernière étape que nous avons suivi consiste en un ensemble de testsdes besoins non fonctionnels de notre application et une occasion aussi pour effectuer la correction des bugs et l'optimisation de la performance du modèle de réseau de neurones ainsi que le temps de recherche du

moteur d'échecs et le nombre de nœuds recherchés. Finalement, nous avons procédé à l'intégration du moteur final dans un GUI (Arena) ou en exécutant un programme identique créé afin qu'il soit testé.

.

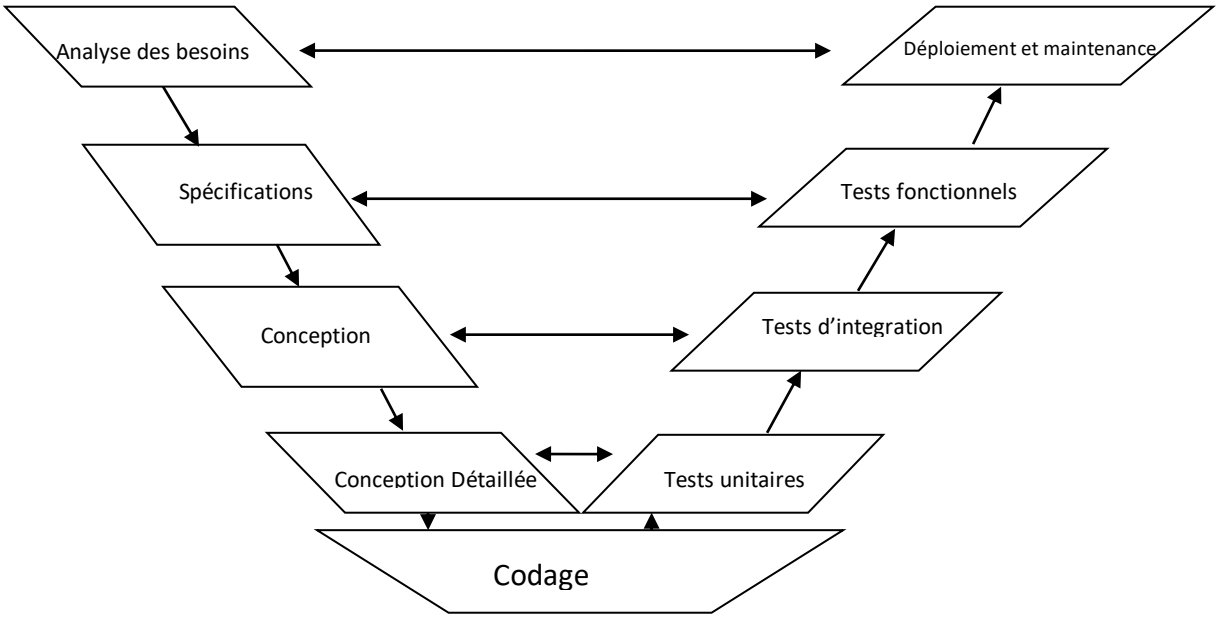


Figure IV:2 Le cycle en V

Fichier	Nombre de lignes	Déscription
Chess1.py	193	- Définit la classe Reflexionchess
Lecture.py	108	- Définit la classe Lecture - lors de l'exécution, ce nom de fichier est suivi d'un nombre allant de 0 à 4 (0 pour utilisation d'une évaluation statique, et le reste et pour l'utilisation d'une évaluation basée sur l'un des quatre modèles créés).
Engine.py	110	- Définitla classe Engine
Chessmodel.py	139	- Définit la classe ModelANN
Movesorter.py	27	- Définit la class Movesorter
Table.py	33	- Définit la classe Transposition table
Reflexion.py	153	- Définit la classe abstraite Reflexions ainsi que les deux sous classes Reflexionann et Reflexionsimple
Repchessboard.py	40	- Définit la classe Repchess
Executer.py	12	- Permet de convertir un fichier Python en un programme exécutable

Insertdata.py	97	- Implémente le code afin d'extraire les informations à partir d'un fichier PGN et les enregistrer dans une BDD
GUI.py	58	- Définit la classe Arena et Log - Permet d'extraire la FEN d'une position et l'indexe du moteur et l'envoyer à la classe Startgame
Inter.py	21	- Lance les commandes UCI automatiquement et les communiquent à la classe Engine et permet de retourner le meilleur coup
EvalPrediction.py	106	- Définit une interface graphique afin d'évaluer des positions données par leur fen en utilisant un modèle de RNA.

Tableau IV-1 Liste des fichiers

3.4 Déploiement

Notre application se décompose en douze fichiers Python dont chacun a un rôle spécifique comme le montre le tableau ci-dessus. Au cours de la phase du déploiement, nous établissons des configurations afin que notre application soit utilisée par les GUI de jeu d'échecs. Ainsi deux méthodes s'offrent à nous :

1. La première méthode consiste à créer un exécutable à partir du fichier principal (Lecture.py) en utilisant principalement le module cx_Freeze qui permet de convertir un fichier Python en un fichier exécutable suivant le code décrit dans la figure ci-dessous. Pour exécuter le code nous utilisons la commande suivante : executer.py build. Enfin lors de la création d'un nouveau moteur d'échecs dans le GUI nous importons l'exécutable créé. Dès lors, un problème est constaté une fois l'exécutable est créé. Ce dernier s'explique simplement par le faite que notre application utilise la bibliothèque Tensorflow. Ainsi l'instruction « import tensorflow as tf » est interprétée par le programme lors de son exécution comme suite : from tensorflow.Python import * . Par conséquent, la partie profiler (tensorflow.Python.profiler) de tensorflow cause cette erreur en voulant faire ce dernier import. Cette erreur peut être résolue en créant un fichier vide « __init__.py » dans le dossier « chemin vers Python \ Lib \ site-packages \ tensorflow \ core \ profiler »

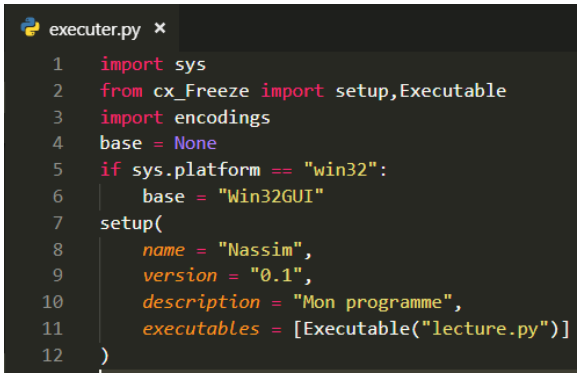


Figure IV:3 Création d'un executable de notre application

2. La seconde méthode consiste à préciser le chemin d'installation de Python suivi par le chemin d'emplacement du fichier Lecture.py (voir figure ci-dessous)

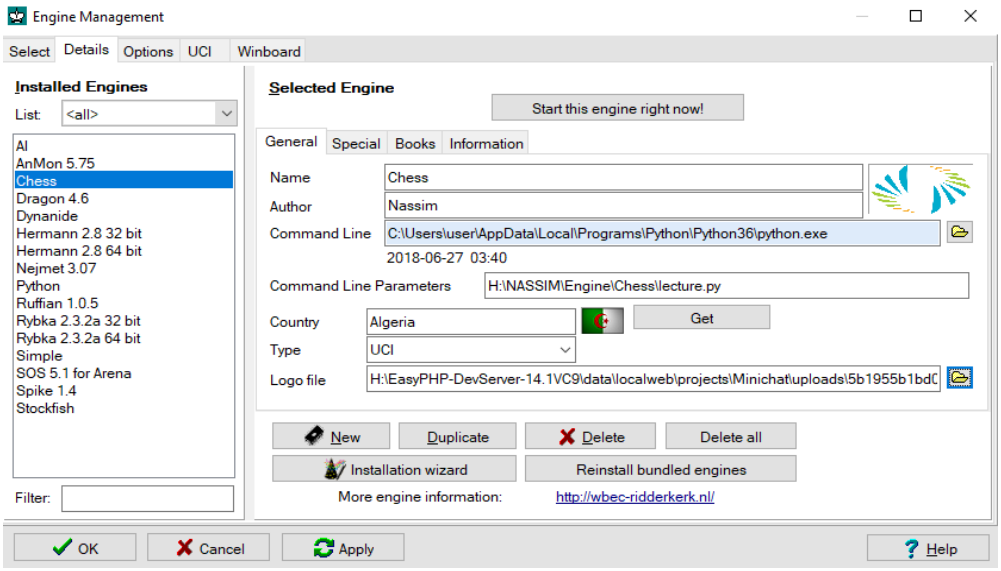


Figure IV:4 Déploiement de notre application sur Arena

Enfin, notre application est utilisable en deux modes :

- Analyse simple (voir figure ci-dessous): c'est le mode principal, il consiste à analyser une position donnée selon les variantes possibles du protocole UCI (voir annexe D.3).

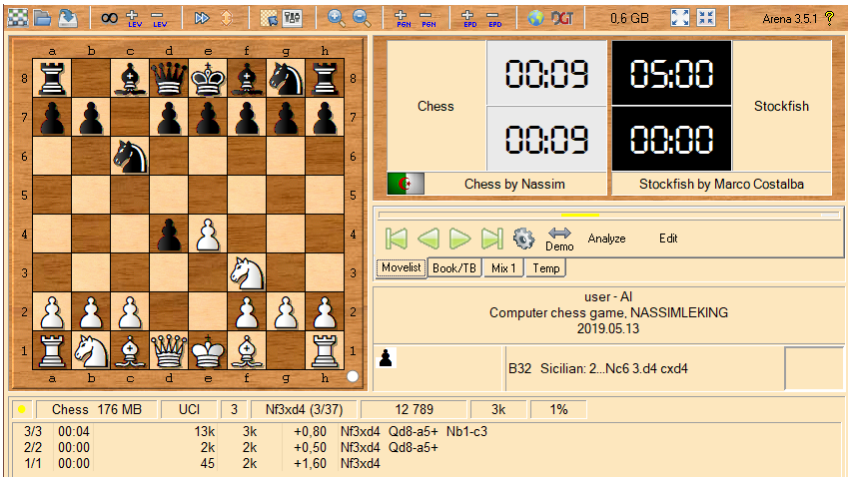


Figure IV:5 Mode Analyse

- Mode tournois (voir figure ci-dessous) : il est possible de faire jouer notre application utilisant une évaluation statique contre celle utilisant une évaluation basée sur un modèle de réseaux de neurones artificielles ou même entre notre application et d'autres moteurs d'échecs intégrés dans le GUI comme Stockfish ou autres. Ce mode utilise

principalement une requête spécifique du protocole UCI où le temps de la pendule est pris en considération.

Figure IV:6 Mode tournois

4 Expérimentations et résultats

4.1 Environnement d'entrainement et de test

Le modèle de réseau de neurones artificielles a besoin de ressources matérielles de qualité afin d'accroitre ses performances. Dans notre cas, nous avons entrainé le modèle sur un ordinateur portable EliteBook v pro ayant un processeur Intel(R) Core™ i5 – 4300U CPU avec une fréquence de 1.90 GHz 2.50 GHz. L'ordinateur utilise le système d'exploitation Windows 10 64 bits avec notamment 4 Go de RAM et un disque dur SSD d'une capacité de 297 GB.

En ce qui concerne l'environnement de test, nous avons utilisé Arena Chess GUI décrit précédemment. Nous avons opté pour ce logiciel pour sa facilité d'utilisation et ses multiples configurations offertes pour les développeurs. En outre, nous avons créé notre propre programme qui simule le fonctionnement des GUI échiquéens. Ce programme fait jouer deux moteurs passés en paramètre en utilisant l'instruction : GUI.py X Y. Où X (moteur ayant les blancs) et Y (moteur ayant les noirs) sont des entiers naturels de 0 à 4 tel que : 0 représente le moteur qui utilise une évaluation statique, autrement ils désignent le numéro d'un des quatre moteurs utilisant les modèles RNA conçus.

4.2 Collection et gestion des données

4.2.1 Collection des données

Dans le cadre d'entrainement de notre modèle de réseaux de neurones artificiels, des données ont été récoltées à partir du site lichess.org. Il s'agit d'un ensemble de positions (des joueurs dont l'élo[12]allant de 2000 à 2800) téléchargées sous format PGN (229400 positions différentes) et qui ont été déjà évaluées par stockfish.

Le fichier « insertdata.py » permet de parcourir les coups joués dans chaque partie et d'extraire la notation FEN de chaque position ainsi que son évaluation. Enfin, il permet d'insérer le résultat dans une base de données avec la structure : ChessCnn (**id**, fen, evaluation, résult)(voir figure ci-dessous)

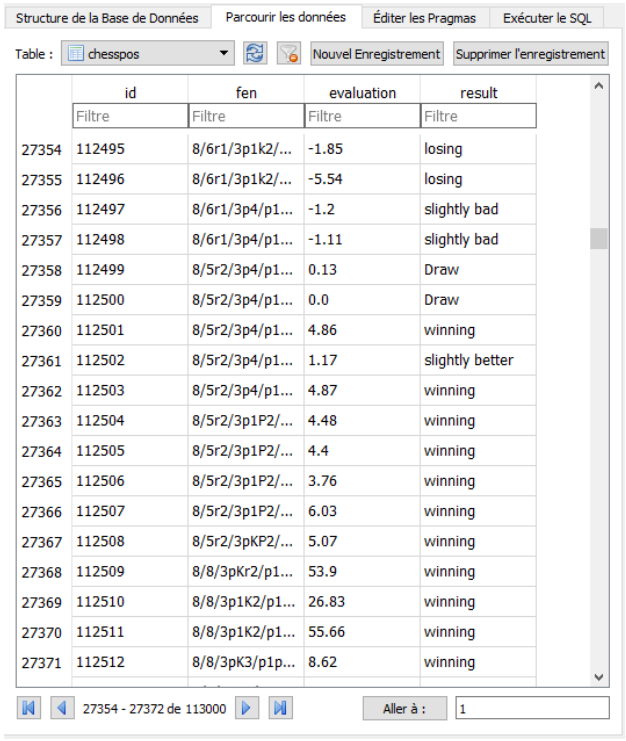


Figure IV:7 Structure de la base de données

4.2.2 Gestion des données

Les données récoltées sont décomposées en trois ensembles principaux :

- **Ensemble d'entrainement :** Il s'agit d'un ensemble de 146 816 positions utilisées dans l'entrainement de notre modèle. Les évaluations des positions sont soumises à deux types de normalisations :
  - **Z-score normalisation :** La moyenne est fixée à 36.5, et l'écart type égale à 1.6.

[12]Classement en un nombre de points associés chaque joueur d'échecs. Il nous permet d'avoir une idée sur le niveau de ce dernier.

- **Tanh :** utilisée dans le but d'avoir des prédictions dans l'intervalle [-1,1].

- **Ensemble de validation :** ils'agit d'un ensemble de 36 704 positions utilisées dans la validation de notre modèle.
- **Ensemble de tests :** ils représentent le reste des données à savoir 45 880 positions utilisées pour évaluer la performance et la qualité du modèle.

Il faut noter que nous avons supprimé toutes les positions d'échecs et mat vu qu'elles sont détectées et gérées par le moteur d'échecs sans nécessairement avoir recours à l'évaluation de la position.

Avant d'alimenter le modèle avec ces données, ces dernières sont représentées sous forme d'un vecteur avec 844 entrées (bits), de telle sorte qu'on puisse décrire les coordonnées des pièces (occupations sur l'échiquier) ainsi que des informations supplémentaires sur la position conformément à ce qui a été décrit dans le chapitre précédent. Enfin, le chargement des données se faite en deux méthodes : à partir de la BDD ou à partir du disque dur.

- **chargement à partir de la BDD :** Il s'agit de la première fonction avant la construction de nos modèles, où nous chargeons les données enregistrées dans notre base de données (DB Browser for SQlite) et nous décomposons et nous définissons au même temps les différents ensemble de données (voir chapitre 2 section 2.3) et dont chacun est représenté par un couple (Fen, Evaluation). Toutes les positions ayant le trait est aux noirs seront normalisées en leur appliquant le principe de la symétrie aux échecs (voir annexe D.1) et ainsi leurs évaluations respectives seront multipliées par -1. Par la suite, en utilisant la méthode fournie par la classe ***repchess ()*** à savoir la méthode ***repchessboard (),*** on représente la position d'échecs donnée par sa FEN sous forme d'un vecteur numpy de 844 entrées décrivant des informations sur la position qui seront précisées par la suite. Ensuite, les évaluations des positions sont normalisées à l'aide des deux fonctions : Z-score (voir chapitre 2 : 5.2.1) suivie de l'application de la fonction Tan dans le but de limité l'intervalle des sorties en l'occurrence [-1,1] (voir annexe B.5). Enfin, les ensembles de données normalisés sont enregistrés sur le disque pour des chargements ultérieurs. (voir la figure ci-dessous).

```
def LoadDataFromDatabase(self):
    compt=0
    connection=sqlite3.connect("ChessCnn.db")
    cursor=connection.cursor()
    cursor.execute("SELECT * FROM chesspos ")
    cursorf = cursor.fetchall()
    for data in cursorf:
        compt=compt+1
        partie=chess.Board(data[1])
        ev = data[2]
        if (partie.turn == chess.BLACK):
            partie = partie.mirror()
            ev = - ev
        rep=repchessboard.repchess(partie).repchessboard()
        if compt<=((len(cursorf) * 4) / 5):
            self.XtrainData.insert(0,rep)
            self.YtrainData.insert(0,ev)
        else:
            self.XtestData.insert(0,rep)
            self.YtestData.insert(0,ev)
    connection.close()
    self.YtrainData = np.array(self.YtrainData)
    self.YtrainData = (self.YtrainData - np.mean(self.YtrainData)) / np.std(self.YtrainData)
    self.YtrainData = np.tanh(self.YtrainData)
    np.save("YtrainData",self.YtrainData)
    self.YtestData = np.array(self.YtestData)
    self.YtestData = (self.YtestData - np.mean(self.YtestData)) / np.std(self.YtestData)
    self.YtestData = np.tanh(self.YtestData)
    np.save("YtestData",self.YtestData)
    self.XtrainData= np.array(self.XtrainData)
    np.save("XtrainData",self.XtrainData)
    self.XtestData=np.array(self.XtestData)
    np.save("XtestData",self.XtestData)
```

Figure IV:8 Chargement depuis la BDD

- **Chargement depuis le disque :** Elle représente la méthode utilisée par tous les modèles une fois que la méthode mentionnée précédemment a été exécutée. Elle permet de charger les données enregistrées dans des fichiers Numpy. (voir la figure ci-dessous)

```
def LoadDataFromDisk(self):
    self.XtrainData=np.load('XtrainData.npy')
    self.YtrainData=np.load('YtrainData.npy')
    self.XtestData=np.load('XtestData.npy')
    self.YtestData=np.load('YtestData.npy')
```

FigureIV:9 Chargement depuis le disque

En plus du chargement de données, une autre gestion est réalisée au niveau des couches d'entrés pour les données chargée en les séparant en deux types de données de pré-entrainement : données positionnelles et données supplémentaires.

- **Données positionnelles :** A ce niveau, une fonction nommée « lambda » est appliquée aux donnée de pré-entrainement. Elle permet dans ce cas d'extraire les 832 premiers entrées du vecteur de données. Cela représente en fait le vecteur bitboard de la position des pièces ainsi que les cases vides (voir chapitre 4 : collection et gestion de données). Enfin, nous redimensionnons le vecteur généré de la forme (832,1) vers une forme

bitboard matricielle (8, 8,13). Par conséquent cette collection de donnée représente les entrés du sous modèles qui traite l'échiquier sous forme d'image en utilisant un réseau de neurones de convolution (RNC). (voir figure ci-dessous)

```
inp = Input(shape=(844,))
side1 = Lambda(lambda x: x[:, 0: 832])(inp)
side1= Reshape((8, 8, 13))(side1)
```

Figure IV:10 Extraction des données positionnelles

- **Données supplémentaires :** D'autre part**,** une autre fonction lambda est appliquée aux entrées de pré-entrainement afin d'extraire les informations restantes (voir la figure ci-dessous). Ces dernières représentent aussi à leur tour les entrés d'un sous-modèle.

```
inp = Input(shape=(844,))
side2 = Lambda(lambda x: x[:, 832: 844])(inp)
```

Figure IV:11 Extractions des données supplémentaires

Á noter que pour faciliter l'entrainement, les positions sont évaluées par rapport aux blancs en tirant profit de la symétrie des positions au jeu d'échecs. De ce fait, avant d'être enregistrée si le trait dans une partie donnée est au noir, la symétrie sera calculée et l'évaluation enregistrée est l'opposée de l'évaluation originale.

4.3 Métriques utilisées

```
def Compilation(self):
    opt=optimizers.Adam(lr=0.001)
    self.model.compile(loss='mse', optimizer=opt,metrics=['mae',root_mean_squared_error,Metric_evaluation])
```

Figure IV:12 Métriques d'évaluations utilisées

Pour les besoins d'évaluation des performances des réseaux de neurones proposés, nous avons utilisés certaines métriques (voir implémentation dans la figure ci-dessus) que nous décrivons dans ce qui suit.

Une fois l'architecture du modèle est définie, nous avons utilisé des métriques qui représentent des fonctions qui permettent de déterminer l'efficacité et la robustesse d'un modèle durant son entrainement. De ce fait, nous avons utilisé trois métriques :

4.3.1 Mean Square Error (MSE)

Elle représente la métrique la plus simple et la plus courante dans l'évaluation des problèmes supervisés de régression. En plus de son rôle dans la compilation de notre modèle en tant que la fonction d'erreur choisie, MSE est aussi utilisée implicitement comme métrique

afin de déterminer l'erreur quadratique moyenne des évaluations de positions prédites. Pour chaque donnée, elle calcule la moyenne des différences carrées entre les prédictions et les valeurs cibles (voir l'équation ci-dessous).

$$\mathrm{MSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 / n$$

$\hat{y}_i$: est la valeur prédite , y_i: représente la valeur attendue, n : est le nombre de prédictions

Équation IV-1 Fonction MSE

Plus cette valeur est élevée, plus le modèle est mauvais. Le résultat fourni est toujours positif, car nous calculons les erreurs de prévision individuelles avant de les additionner, mais serait nul pour un modèle parfait.

4.3.2 Root Mean Square Error (RMSE)

Cette métrique est aussi très utilisée, sa fonctionse présente comme étant la racine carréede MSE. Elle permet d'interpréter plus concrètement la performance du modèle notamment pour des problèmes de régression où la métrique **accuracy** est insensée. Contrairement à la mesure précédant, RMSE utilise directement l'unité cible afin de calculer la différence entre la valeur prédite et celle attendue. De ce fait, si la valeur de l'erreur calculée lors des entrainements est proche à celle des tests, cela signifie que le modèle est bon et c'est un signe d'un bon apprentissage. Cette métrique n'est pas offerte par Keras, Pour cela nous avons développé notre propre implémentation (voir figure ci-dessous).

```
def root_mean_squared_error(self,y_true, y_pred):
    return backend.sqrt(backend.mean(backend.square(y_pred - y_true)))
```

Figure IV:13 Définition de la métrique RMSE

4.3.3 Mean Average Error (MAE)

Dans ce cas, l'erreur est calculée en tant que moyenne des différences absolues entre les évaluations de parties attendues et les prédictions calculées fourni en tant qu'un score linéaire (voir l'équation ci-dessous).

$$\mathrm{MAE} = \sum_{i=1}^{n} |y_i - \hat{y}_i| / n$$

$\hat{y}_i$: est la valeur prédite , y_i: représente la valeur attendue, n : est le nombre de prédictions

Équation IV-2 Fonction MAE

Cette mesure est utilisée dans le but de pénaliser les grandes erreurs qui ne sont pas aussi graves que MSE.

4.3.4 Métrique personnalisée

La dernière métrique « Metric_evalution » (voir figure ci-dessous) est une métrique personnalisée qu'on a définie dans le but d'estimer les erreurs commises par le modèle dans l'évaluation des parties. Le plus important dans une partie de jeu d'échecs est de savoir et d'indiquer si la partie est gagnante ou pas. Ces erreurs se présentent comme étant le pourcentage des parties dont les signes des évaluations prédites sont différents de ceux des évaluations attendues. Ceci nous donne une vision plus précise sur la performance du modèle.

Le calcul de la Metric_evaluation passe par les étapes suivantes :

1. En fonction d'un vecteur qui contient les évaluations attendues (eval_attendues), nous définissons un vecteur (eval_attendues_bin) dont les valeurs sont issues de l'application de la fonction « F » suivante sur celles d'eval_attendues :

$$\mathcal{F}(x) = \begin{cases} 1 & si\ x \geq 0 \\ 0 & si\ x < 0 \end{cases} \quad x\text{: évaluation de la position}$$

2. Nous définissons un autre vecteur (eval_prédites_bin) qui utilise le même principe que l'étape passée mais applique sa fonction sur le vecteur (eval_prédites) qui contient les évaluations prédites par le modèle.
3. Dans cette étape nous réalisons l'opération d'Egalité « = » entre chaque deux cases des deux vecteur eval_attendues_bin et eval_prédites_bin portant le même indice où nous aurons finalement un vecteur (vec) résultat de cette opérations. Á noter que $(a = b) \leftrightarrow (a \wedge b) \vee (\bar{a} \wedge \bar{b})$
4. Dans cette étape nous réalisons l'opération binaire (not) pour chaque entrée du vecteur « vec »
5. Il s'agit de l'étape finale où nous calculons le résultat final en sommant le nombre de 1 dans le vecteur « vec » que nous divisons par le nombre total d'entrés.

```
def Metric_evaluation(self,real,predict):
    shp=backend.shape(real)
    resultat=backend.not_equal(backend.greater_equal(real, 0), backend.greater_equal(predict, 0))
    return backend.sum(backend.cast(resultat, backend.floatx())) / backend.cast(backend.prod(shp), backend.floatx())
```

Figure IV:14 Définition de la métrique « Metric_evaluation »

4.4 Entrainement

La validation du modèle final est l'une des tâches les plus difficiles dans le domaine de l'apprentissage automatique. Il n'existe pas de règles ou de normes formelles afin de former la bonne combinaison des hyper-paramètres qui nous conduisent à un modèle optimal. Par conséquent, les plus grands ingénieurs en IA utilisent quelques approches et heuristiques - que

nous avions appliqué dans notre pour notre cas - afin d'avoir de meilleurs résultats. Ces approches sont sous forme d'un processus dont les démarches sont les suivantes :

- **Augmentation du nombre de données :** Fournir plus de donnée pour le modèle afin d'éviter le sous apprentissage et permettre au modèle d'apprendre sur plus d'exemples et mieux cerner le problème.
- **La modification du taux d'apprentissage :** Il s'agit d'essayer les variantes suivantes :
  - Essayer avec une très grande valeur.
  - Essayer avec une valeur très réduite.
  - Essayer de réduire sa valeur au cours de l'entrainement (ReduceLROnPlateau[13]) (voir la figure ci-dessous).

```
reduc=ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience=2)
```

Figure IV:15 Implémentation de ReduceLROnPlateau

  - Rechercher les valeurs communes utilisées par la communauté de l'apprentissage automatique qui se situe dans l'intervalle [$1*10^{-7}$, 1].
  - Définir la valeur du prochain taux d'apprentissage pour la version suivante selon le résultat du modèle passé suivant la figure ci-dessous :

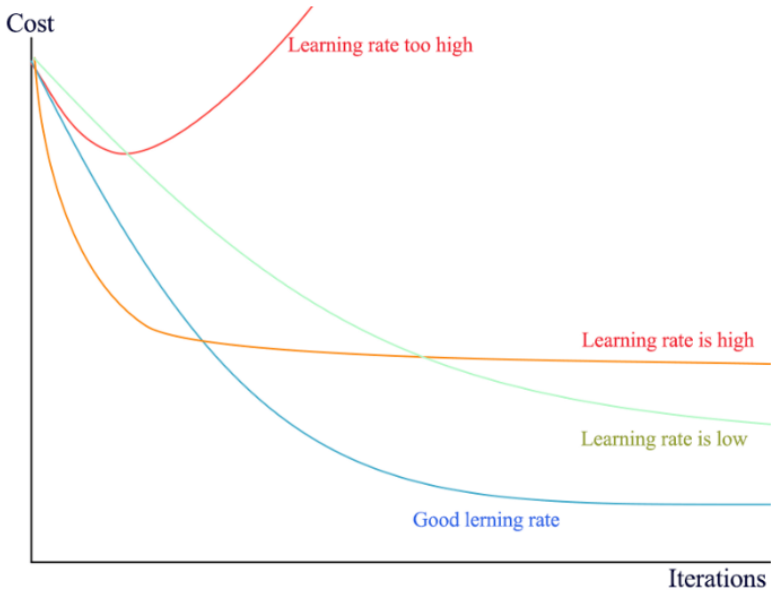


Figure IV:16 : Le choix du taux d'apprentissage

[13] Permet de réduire le taux d'apprentissage avec une valeur bien définie au cours de l'entraiement si le modèle n'arrive plus à minimiser l'erreur.

- **Batch size :** Les valeurs du batch size souvent utilisées sont 8,16, 32 ou 64. En fonction de la complexité de l'architecture utilisée, une valeur réduite peut accélérer l'entrainement mais risque d'avoir des oscillations dans le graphe d'erreur. D'autre part, si la valeur est très grande l'entrainement risque d'être long. Nous avons fixé la valeur de 200 avec tous les modèles entrainés.
- **Dropout :** Nous avonstesté des variantes avec un pourcentage de dropout variant entre 20 à 50%.
- **Arrêt précoce :** Il s'agit d'une méthode de régularisation qu'on a utilisée. Dans notre cas nous avons choisi la valeur 40. (voir figure ci-dessous)

```
earlystop =keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=40, verbose=1)
```

Figure IV:17 Implémentation de l'arrêt précoce.

- **Voir les projets réalisés aux échecs en apprentissage automatique :** Nous avons copié quelques hyper-paramètres et idée de conception des projets déjà réalisés comme Leela Chess Zero (voir chapitre 3).
- **Lancer plusieurs différents modèles en parallèle et valider le plus prometteur :** En plus de ce qui a été présentédans le chapitre précédant (Analyse des choix d'apprentissage), nous avons construit et entrainé plusieurs modèles (plus d'une centaine) à l'issue duquel nous avons opté pour l'utilisation des modèle fonctionnelles pour les raisons suivante :
    - Les modèles fonctionnelles sont plus performants que les modèles séquentiels (cas séquentiel : borne minimal de l'erreur « mse » ne dépasse pas $n*10^{-2}$. Cas fonctionnelle : borne minimal est souvent sous forme $n*10^{-3}$).
    - L'utilisation du modèle fonctionnelle offre plus de flexibilité et gère la complexité des données.
    - Les modèles fonctionnelles donnent les meilleures prédictions dans la phase de tests et ont une meilleure compréhension de la position échiquéenne.
- **Epoque :** Il est conseillé de choisir un nombre très grand dans le cas de l'utilisation d'un arrêt précoce vu qu'on l'atteindra rarement, de ce fait nous lui avons choisi la valeur de 1200.

En utilisant le processus expliqué ci-dessus, nous avons pu valider quatre modèles dont les caractéristiques visualisées avec Tensorboard sont précisée dans le tableau ci-dessous et dont les graphes des métriques sont illustrés dans les sous sections ci-dessous.

	RNA1	RNA2	RNA3	RNA4
mse	**2,5561 * 10^{-3}**	8,2870 * 10^{-3}	6,3193 * 10^{-3}	6,1016 * 10^{-3}
mae	**3,230 *10^{-2}**	5,175 * 10^{-2}	4,725 * 10^{-2}	4,601 * 10^{-2}
val_mse	2,846 * 10^{-2}	1,996 * 10^{-2}	1,950 * 10^{-2}	**1,916 * 10^{-2}**
val_mae	7, 22 * 10^{-2}	7,267 * 10^{-2}	7,146 * 10^{-2}	**7,061 * 10^{-2}**
val_rmse	1,5393 * 10^{-1}	1,3423 * 10^{-1}	1,2966 * 10^{-1}	**1,281* 10^{-1}**
val_Metrique_evaluation	**23,23 %**	24,15 %	23,78%	23,93%
Nombre de paramètres	850.051	1.114.826	1.573.001	3.616.751
Temps d'entrainement	9 h 16 mn 52 s	10 h 9 mn 5s	1 j 12 h 25mn 16s	2j 13h 52mn

Tableau IV-2 Caractéristiques des modèles

4.4.1 RNA1

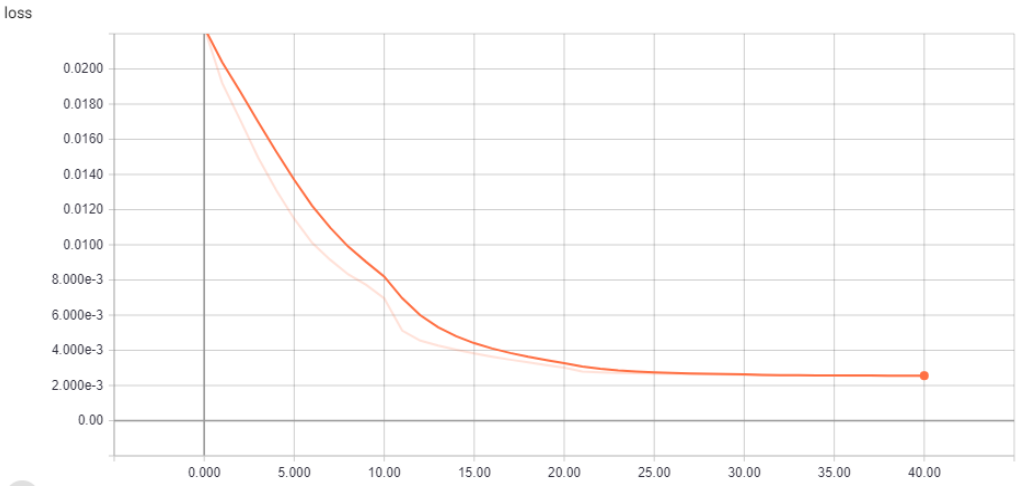


Figure IV:18 MSE de RNA1

La figure ci-dessus décrit l'évolution de l'erreur MSE au cours de l'entrainement. Nous constatons que sa valeur est passée de 0.04 à 2,5561 * 10^{-3}. Il s'agit de la meilleure valeur retrouvée parmi tous les modèles.

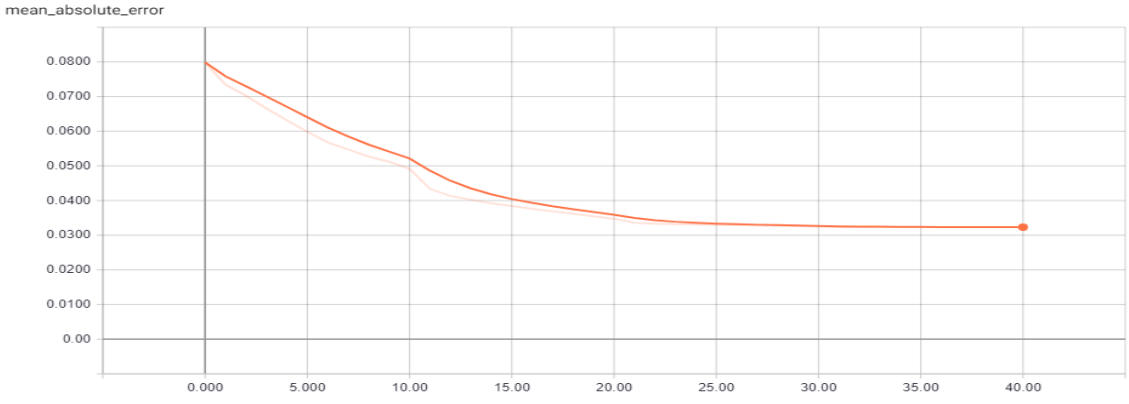


Figure IV:19 MAE de RNA1

La figure passée représente la progression de la métrique d'évaluation MAE qui est passée de 0.08 à 3,230 *10^{-2} qui représente aussi la meilleure valeur parmi celles des autres.

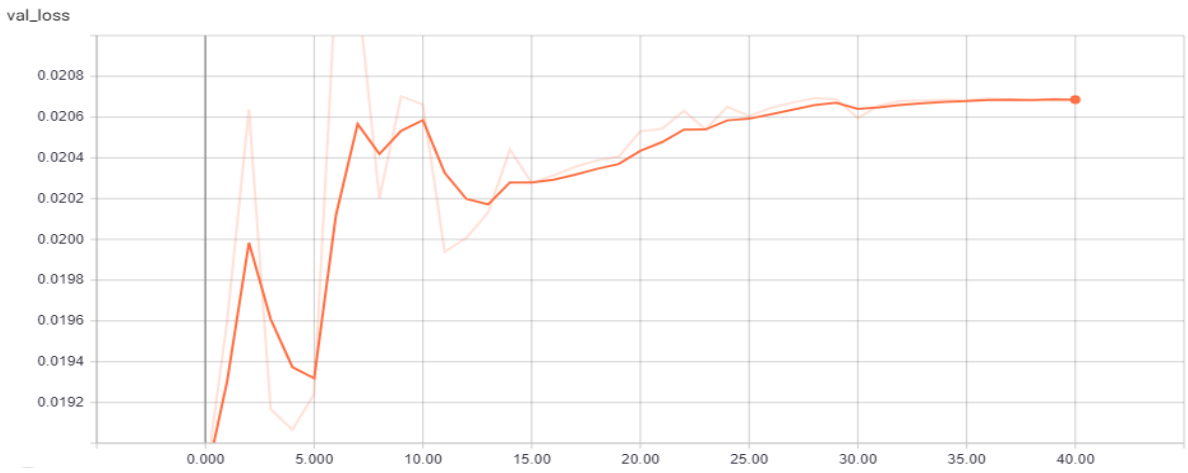


Figure IV:20 Val_MSE de RNA1

La figure ci-dessus décrit l'évolution de l'erreur MSE lors la phase de validation, où nous constatons que sa valeur de progression s'est arrêtée à 2,846 * 10^{-2}.

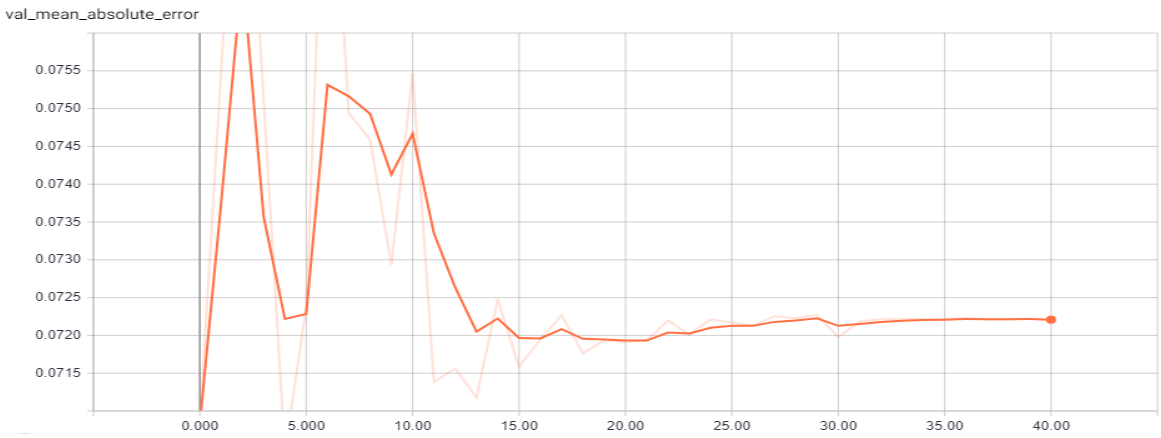


Figure IV:21 Val_MAE de RNA1

Nous notons d'après la figureen haut que l'erreur MAE lors de la phase de validation s'est fixée à 7,22 * 10^{-2}.

4.4.2 RNA2

La figure ci-dessous décrit l'évolution de l'erreur MSE au cours de l'entrainement. Nous constatons que sa valeur est passée 0.033 à 8,2870 * 10^{-3}.

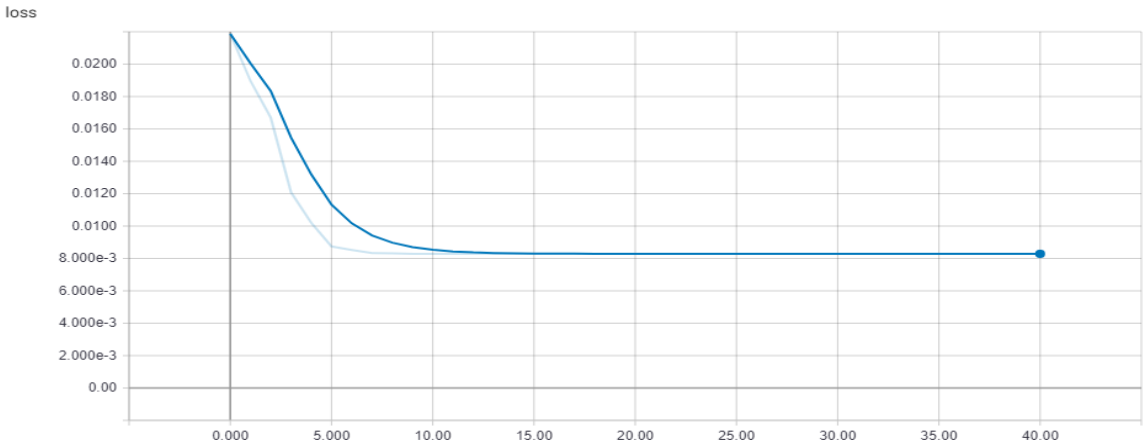


Figure IV:22 MSE de RNA2

La figure suivante représente la progression de la métrique d'évaluation MAE qui est passée de 0.082 à 5,175 * 10^{-2}.

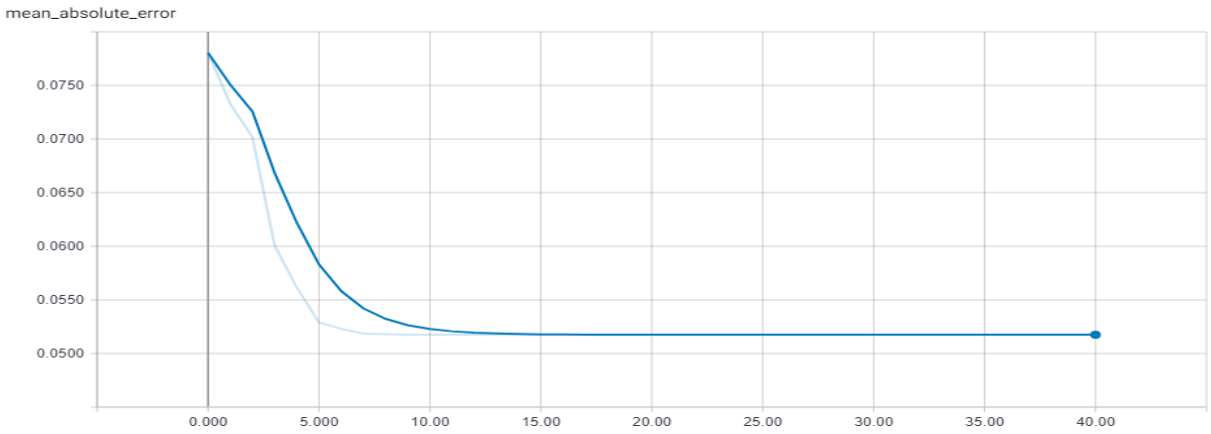


Figure IV:23MAE de RN2

La figure ci-dessous, nous montre l'évolution de la métrique MSE sur l'ensemble de validation en obtenant une valeur finale de 1,996 * 10^{-2}.

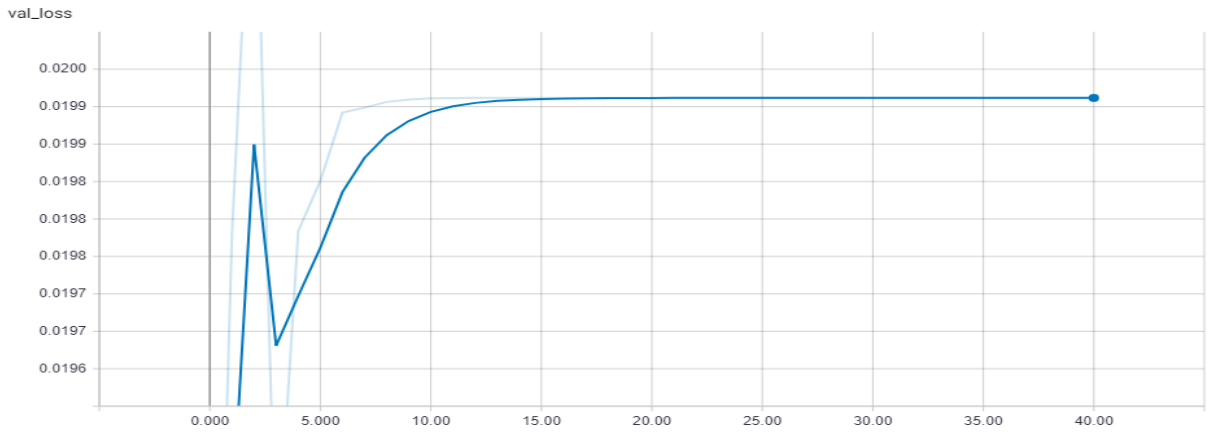


Figure IV:24Val_MSE de RNA2

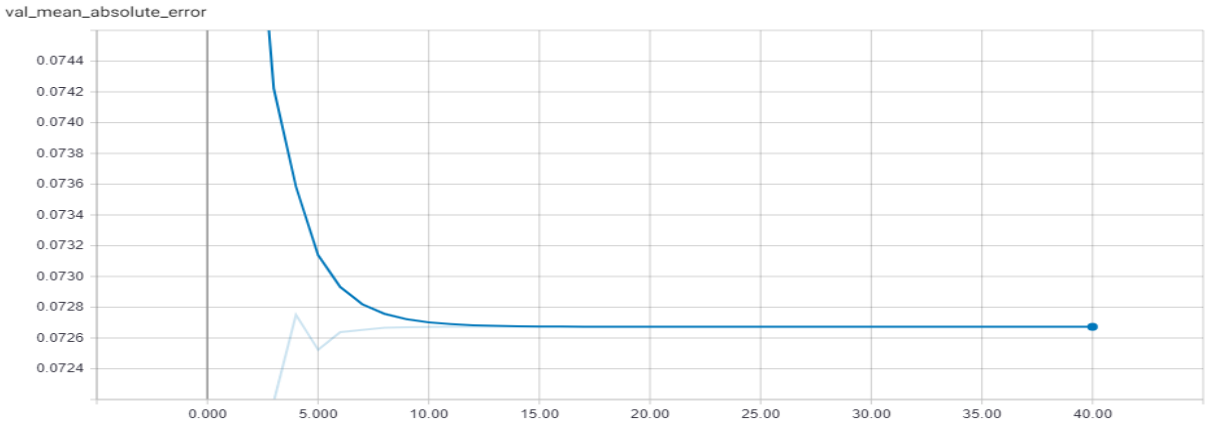


Figure IV:25Val_MAE de RNA2

La figure ci-dessus nous indique l'évolution de la métrique MAE sur l'ensemble de validation on optimisant sa valeur initiale jusqu'à 7,267 * 10^{-2}

4.4.3 RNA3

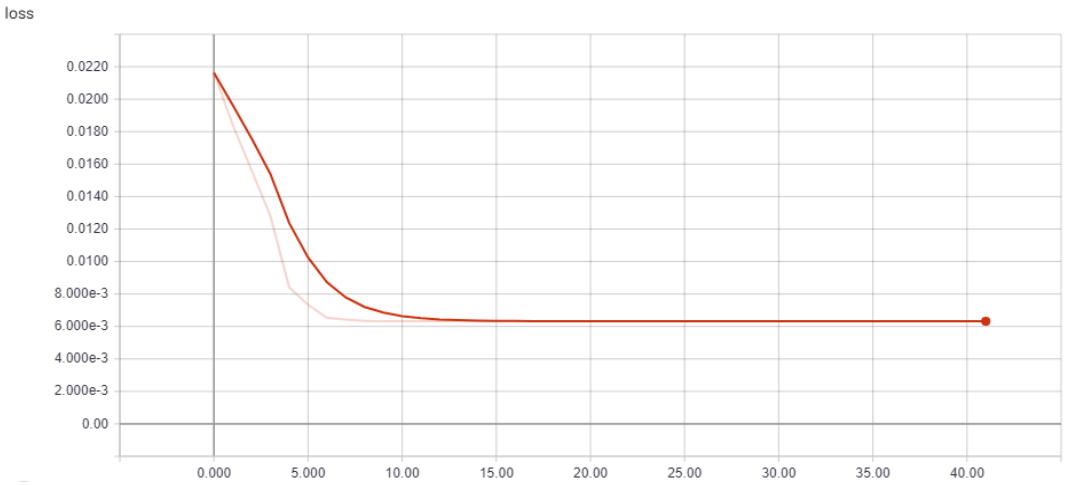


Figure IV:26 MSE de RNA3

La figure ci-dessus décrit l'évolution de l'erreur MSE au cours de l'entrainement. Nous constatons que sa valeur est passée 0.02123 à 6,3193 * 10^{-3}.

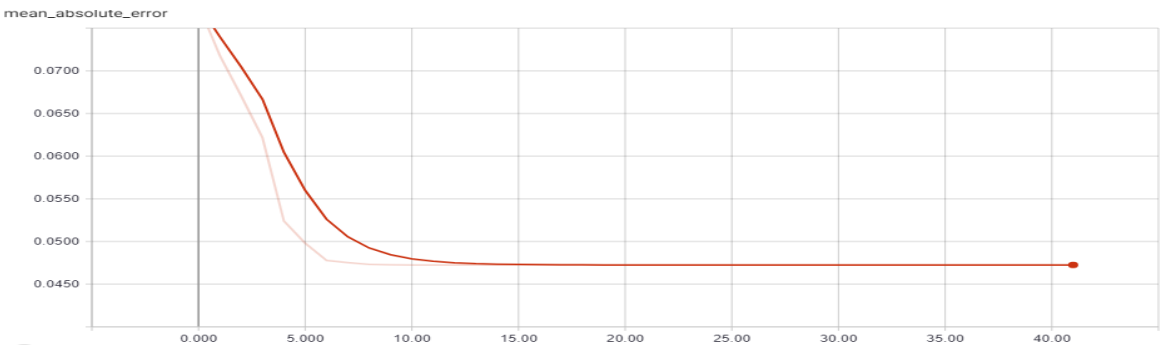


Figure IV:27 MAE de RNA3

La figure ci-dessus nous montre la progression de la métrique d'évaluation MAE qui est passée de 0.097 à 5,175 * 10^{-2}

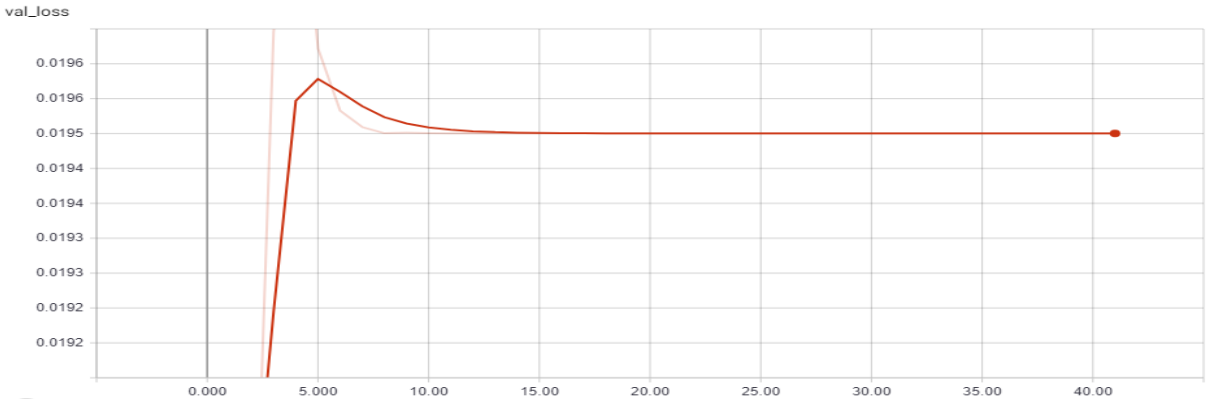


Figure IV:28Val_MSE de RNA3

La figure ci-dessus, nous montre l'évolution de la métrique MSE sur l'ensemble de validation en obtenant une valeur finale de 1,950 * 10^{-2}.

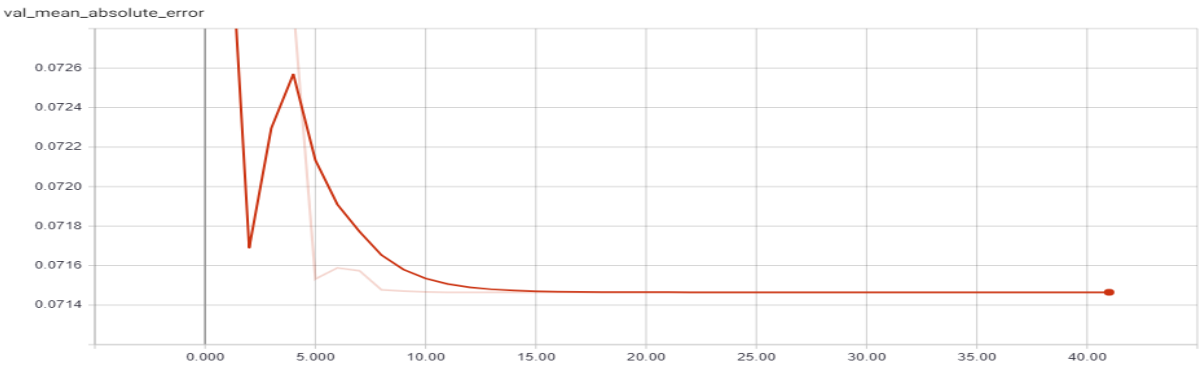


Figure IV:29Val_MAE de RNA 3

La figure ci-dessus nous indique l'évolution de la métrique MAE sur l'ensemble de validation on optimisant sa valeur initiale jusqu'à 7,146 * 10^{-2}

4.4.4 RNA4

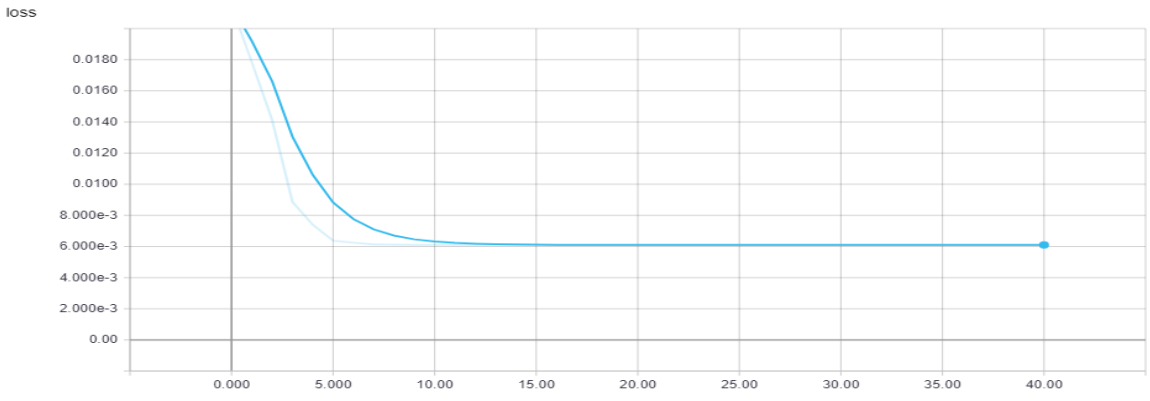


Figure IV:30 MSE de RNA4

La figure ci-dessus représente la progression de l'erreur MSE au cours de l'entrainement. Nous constatons que sa valeur est passée 0.0322 à 6,1016 * 10^{-3}

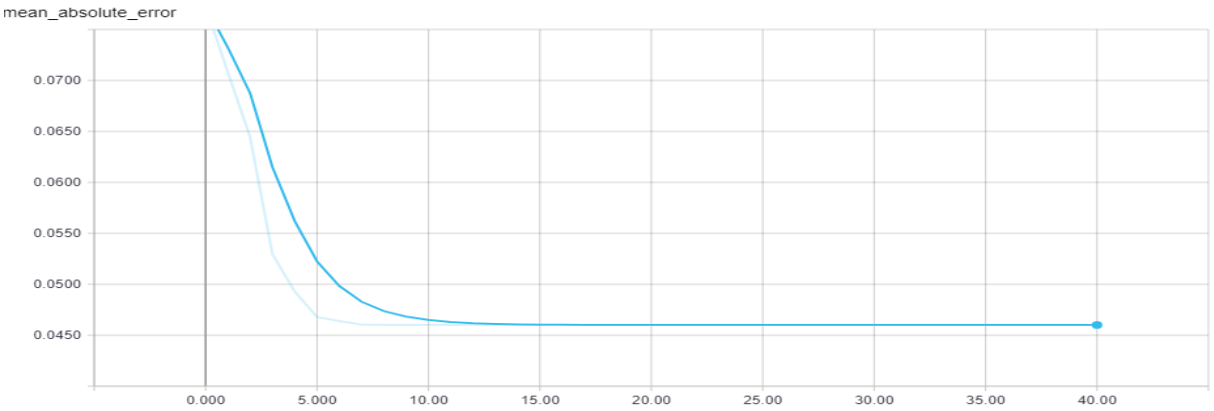


Figure IV:31 MAE de RNA4

La figure ci-dessus nous montre la progression de la métrique d'évaluation MAE qui est passée de 1.09 à 4,601 * 10^{-2}

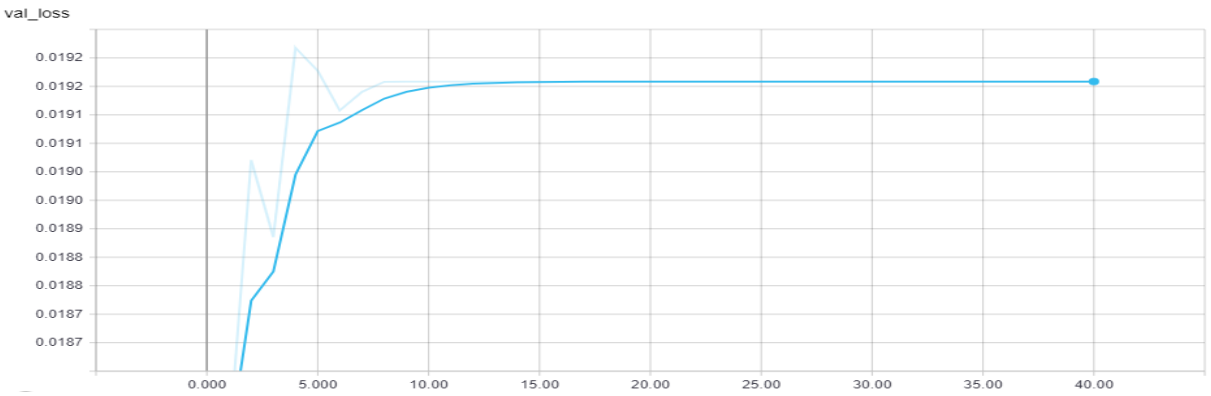


Figure IV:32Val_MSE de RNA4

La figure ci-dessus, nous montre l'évolution de la métrique MSE sur l'ensemble de validation en obtenant une valeur finale de 1,916 * 10^{-2}et qui représente la meilleure valeur trouvée parmi tous les modèles.

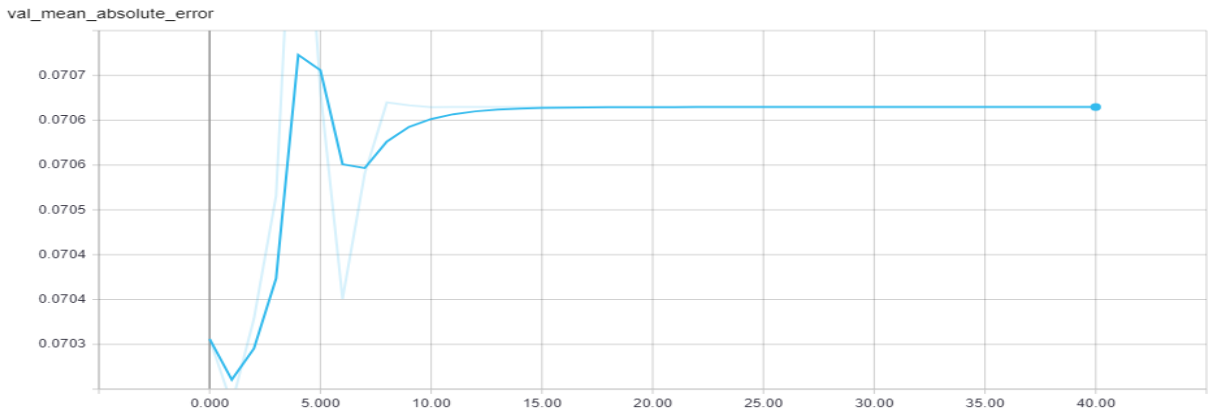


Figure IV:33Val_MAE de RNA4

La figure ci-dessus nous indique l'évolution de la métrique MAE sur l'ensemble de validation on optimisant sa valeur initiale jusqu'à 7,061 * 10^{-2} qui comme la valeur précédente, représente la meilleure valeur trouvée parmi tous l'ensemble des modèles.

4.5 Évaluation et discussions

4.5.1 Évaluation

Dans le cadre de l'évaluation des performances de nos modèles basés sur l'apprentissage automatique, nous avons réalisé un mini championnat (voir les parties dans l'annexe D.4) en utilisant le GUI Arena chess. Notons que les règles du championnat sont les suivants : le vainqueur d'une partie gagnera un point et une partie nulle vaut un demi-point. Après la fin du championnat, le champion aura l'honneur de défier sur deux partiesle moteur basé sur une évaluation statique.

Á noté que la recherche de chaque moteur est conditionnée par une profondeur équivalente à deux.

Les résultats de toutes les parties sont décrits dans le tableau ci-dessous, tandis que la figure IV.34 donne une vue globale sur les résultats du tournoi.

	RNA1		RNA2		RNA3		RNA4	
RNA1			½ - ½	½ - ½	1-0	1-0	½ - ½	½ - ½
RNA2	½ - ½	½ - ½			½ - ½	0-1	½ - ½	0- 1
RNA3	0-1	0-1	1 - 0	½ - ½			1 - 0	½ - ½
RNA4	½ - ½	½ - ½	1- 0	½ - ½	½ - ½	0-1		

Tableau IV-3 Table des résultats

Modèle	**Parties jouées**	**Victoires**	**Nulles**	**Défaites**	**Point**
RNA1	6	2	4	0	4
RNA4	6	1	4	1	3
RNA3	6	2	2	2	3
RNA2	6	0	4	2	2

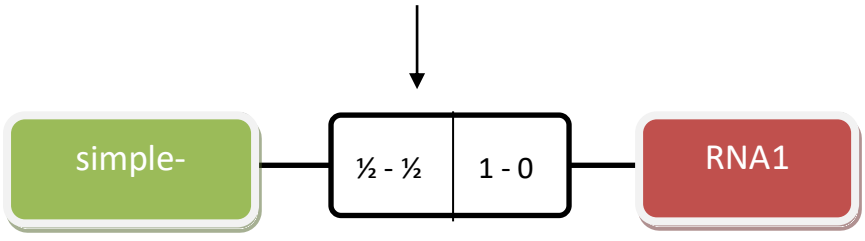


Figure IV:34 Le résultat final du tournoi

4.5.2 Discussions

D'après la figure ci-dessus, nous constatons que le moteur utilisant une évaluation basée sur le modèle RNA1 s'est distingué parmi les autres modèles et a remporté le mini championnat confirmant ainsi sa performance lors des entrainements notamment la valeur réduitede sa métrique personnalisée (23,23%).

Par contre, avec une partie nulle et une autre perdue, notre champion n'a pas réussi à battre lors de la finale le moteur qui utilise une évaluation statique (simple_chess). Cette défaite, est probablement due à plusieurs facteurs. Tout d'abord, l'une des causes principales est la profondeur de la recherche utilisée. Une recherche à la seconde profondeur n'est pas très efficace et produit des évaluations partielles qui ne sont pas très approfondies. En effet, en utilisant une évaluation avec une profondeur supérieure, au minimum quatre, l'évaluation prendra tout son effet et sa valeur et permettra au moteur utilisant le modèle RNA1 de découvrir de meilleurs coups et perspectives et donc une meilleure analyse de la position. Or, avec les moyens matériels utilisés, l'exécution d'une telle profondeur s'avère très lente voir même irréalisable.

En outre, le nombre de nœuds parcourus chaque seconde fait la différence entre les deux finalistes. Ainsi, on a constaté que le moteur utilisant l'évaluation RNA1 parcourt jusqu'à 100 nœuds/s. De l'autre côté, « simple_chess » parcours quant à lui, entre 700 et 1700 nœuds/s. Ce nombre reste quand même à optimiser vu la grande différence avec celui généré par le moteur de Stockfisch avec notamment jusqu'à 20 milles nœuds/s à la seconde profondeur et plus d'un million à la 20ème profondeur. Une statistique qui fait la différence entre la qualité de l'évaluation des positions échiquéennes vu que le moteur ayant une grand valeur de cette statistique évalue plus de positions et ainsi génère une meilleure suite de coups. Dans notre cas, en utilisant la bibliothèque « pyhton_chess » et notamment le langage Python, nous sommes limités en le nombre de coups générés. Conséquemment, afin d'optimiser le nombre de nœud par seconde, nous prévoyons de reprogrammer le moteur d'échecs avec une évaluation RNA1 en utilisant un langage précompilé comme le C ou le C++ avec un bas niveau d'abstraction contrairement au Python qui est interprété. En effet, un langage comme le C est compilé directement dans le code machine qui est exécuté directement par la CPU. Les programmes compilés s'exécutent généralement plus rapidement que les programmes interprétés, ce qui explique pourquoi le C est plus rapide que Python.

Enfin, malgré sa défaite en finale, le moteur utilisant le modèle RNA1 s'est montré plus performant dans certaines positions comme celle montrée par la figure ci-dessous.

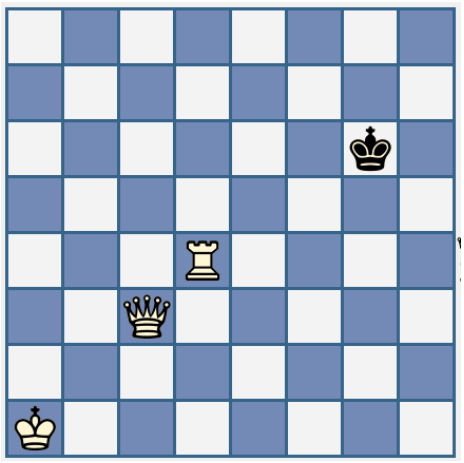


Figure IV:35Position expérimentale 1

Dans cette position, le moteur utilisant le modèle RNA1 (blancs) joue contre « simple_chess » (noirs) et dispose de la meilleure suite de coups afin de réaliser un échecs et mat en seulement cinq coups utilisant la seconde profondeur sachant que stockfish arrive à trouver quant à lui le mat en trois coup à la profondeur 20. La suite est la suivante : 1) d4d1 g6f5 2) c3e3 f5g6 3) e3f4 g6g7 4) d1g1 g7h8 5) f4h6. Par contre, si nous inversons les couleurs de pièces des joueurs à la position initiale, nous avons constaté que simple_chess n'arrive pas à mater RNA1 et la partie se terminée sur un nul avec la suite suivante : 1) d4d8 g6h7 2) d8h8 h7g6 3) c3d4 g6f7 4) a1b2 f7g6 5) d4e5 g6f7 6) e5d4 f7g6 7) d4e5 g6f7 8) e5d4 f7g6 9) d4e5 g6f7 10) e5d4 f7g6 11) d4e5 g6f7 12) e5d4. Un succès qui montre la supériorité de l'évaluation RNA1 sur l'évaluation statique dans quelques positions.

5 Conclusion

Nous avons présenté dans ce chapitre l'analyse de la conception et de l'implémentation de nos modèles de RNA. Par la suite, nous avons décrit l'environnement de développement ainsi que tous les outils qui rentrent dans le cadre de la programmation et du déploiement de l'application. Enfin, nous avons réalisé l'expérimentation de nos modèles ainsi que les résultats obtenues. Ces dernières que nous avions discutés et analysé sont bien confirmé nos prédictions faites dans le chapitre précèdent. En effet, le modèle RNA1 s'est distingué parmi tous les autres modèles en remportant le mini-championnat. Malgré sa défaite lors de la finale, RNA1 a réussi à faire une partie nulle contre le moteur utilisant une évaluation statique et s'est montré supérieur dans plusieurs positions notamment celle analysées dans ce chapitre

Conclusion générale

1 Synthèse

Notre projet a permis de présenter un état de l'art sur la relation des échecs avec la théorie des jeux ainsi que son informatisation en exposant les principaux algorithmes et heuristiques utilisés par la plupart des moteurs échiquéens. Par la suite, nous avons introduit le domaine de l'apprentissage automatique et notamment celui des réseaux de neurones artificiels en raison de sa relation avec l'application mise en œuvre. Concernant cette dernière, nous avons pu développer un moteur de jeux d'échecs doté de la plupart des méthodes de recherches connues.

Aussi, nous avons conçus quatre modèles de réseaux de neurones artificiels pour l'évaluation « intelligente » des positions. Ensuite, nous avons expérimenté ces derniers à l'issue d'un tournoie où nous avons vul'évaluation du modèle RNA1 dépasser celle des autres confirmant ses performance lors de l'apprentissage. Malheureusement, Il n'a pas pu surpasser le moteur « simple_chess » pour plusieurs facteurs détaillés. Néanmoins, RNA1 présente un sens de créativité dans plusieurs positions des parties jouées dépassant parfois l'évaluation générée statiquement.

Cependant, même si notre problématique posée auparavant n'est pas totalement résolue (défaite lors de la finale), plusieurs contributions et points positifs notés sont déjà acquis que nous pourrons optimiser par la suite en utilisant nos perspectives.

2 Perspectives

Nous envisageons plusieurs suites très prometteuses à nos travaux :

- La première perspective serait de reprogrammer le moteur avec un langage compilé plus rapide et plus performant comme le langage C ou C++. En effet, le Python est bien adapté pour l'apprentissage automatique mais est peu adapté pour les calculs intensifs. Ainsi, il serait intéressant d'entrainer les modèles sous Python, et de les utilisés ensuite dans un moteur codé dans un langage compilé.
- Une deuxième perspective serait d'ajouter d'autres heuristiques pour augmenter d'une part la rapidité de la recherche du moteur d'échecs comme la recherche SMP (*Symetric Multi Processing*) qui permet de paralléliser toutes les méthodes utilisées comme alpha béta ou la variation principale, etc. D'autre part, nous prévoyons de rendre l'évaluation de la position plus efficace (l'abandon automatique après un score largement défavorable, la prise en compte du temps de

chaque joueur, changer la fenêtre de convolution de telle sorte à ce qu'elle définit les cases contrôlées par une pièce etc.).

- Une autre perspective consiste à intégrer la table des ouvertures et de finales afin d'optimiser les performances du moteur dans les deux phases de jeux respectives.
- Une quatrième perspective serait d'augmenter le nombre de donnée d'entrainement (pour avoir minimum de 4 millions de positions).
- Une autre éventualité serait de tester d'avantage de combinaisons d'hyper-paramètres de notre modèle afin d'avoir des résultats plus satisfaisants.

  La dernière perspective serait d'imiter le principe d'Alpha Zéro ou de Leela Chess Zero qui serait d'utiliser un apprentissage par renforcement au lieu d'un apprentissage supervisé

Bibliographie

1. **Andrew E. Soltis.** Chess. *Encyclopedia Britannica.* [En ligne] 12 Avril 2019. https://www.britannica.com/topic/chess.

2. **PIerre.Lescanne.** la théorie des jeux. *Wikipédia.* [En ligne] 26 Avril 2019. https://fr.wikipedia.org/wiki/Théorie_des_jeux.

3. **Drew fudenberg ,Jean Tirole.** Game theory. *univie.ac.at.* [En ligne] 1991. https://homepage.univie.ac.at/Mariya.Teteryatnikova/WS2011/FT.pdf.

4. **Bonanno, Giacomo.** Game theory. *faculty.econ.ucdavis.edu.* [En ligne] 2015 - 2018. http://faculty.econ.ucdavis.edu/faculty/bonanno/PDF/GT_book.pdf.

5. **SAFFIDINE, ABDALLAH.** SOLVING GAMES AND ALL THAT. *tel.archives.* [En ligne] 10 Juillet 2014. https://tel.archives-ouvertes.fr/tel-01022750/document.

6. **FIDE.** livre de l'arbitre. *echecs.asso.fr.* [En ligne] 1 juillet 2009. http://www.echecs.asso.fr/LivreArbitre/110.pdf.

7. **M-Tullius-Cicero.** Notation de coups. *Wikipédia.* [En ligne] 11 Fevrier 2019. https://fr.wikipedia.org/wiki/Notation_aux_échecs.

8. **Guillermo Gajate.** FIDE / notation_games. *FIDE.* [En ligne] 2017. http://portablegamenotation.com/FIDE.html.

9. **Vladan VUČKOVIĆ.** an alternative efficient chessboard. *semantic scholar.* [En ligne] Mai 2012.https://pdfs.semanticscholar.org/474d/cbfd78091da7af646b1b6f4c97dc6e45e7ba.pdf.

10. **Jonatan Pettersson.** The 0x88 representation. *mediocrechess.* [En ligne] 14 Décembre 2006. https://mediocrechess.blogspot.com/2006/12/0x88-representation.html.

11. **GerdIsenberg.** FEN. *chessprogramming.* [En ligne] 10 Mai 2018. https://www.chessprogramming.org/Forsyth-Edwards_Notation.

12. **David Barnes.** pgn-extract: a Portable Game Notation (PGN). [En ligne] 1994-2007. http://portablegamenotation.com/Downloads/pgnextract.pdf.

13. **Marc Boulé.** An FPGA Move Generator for the game of chess. *digitool.library.* [En ligne] Aout 2002. http://digitool.library.mcgill.ca/webclient/StreamGate?folder_id=0&dvs=1556807749458~865.

14. **Rhys Rustad-Elliott.** Fast Chess Move. *rhysre.net.* [En ligne] 15 Janvier 2019. https://rhysre.net/2019/01/15/magic-bitboards.html.

15. **GerdIsenberg.** Search. *chessprogramming.* [En ligne] 27 Avril 2018. https://www.chessprogramming.org/Search.

16. **Dirk P. Kroese.** Monte Carlo Methods. *people.smp.uq.edu.au.* [En ligne] 2011. https://people.smp.uq.edu.au/DirkKroese/mccourse.pdf.

17. **Fschwarzentruber .** Minimax. *wikipédia.* [En ligne] 8 Avril 2019. https://fr.wikipedia.org/wiki/Algorithme_minimax.

18. **Jeroen W.T. Carolus.** Alpha-Beta with Sibling Prediction Pruning. *homepages.cwi.nl.* [En ligne] 2006. https://homepages.cwi.nl/~paulk/theses/Carolus.pdf.

19. **GerdIsenberg.** Negascout. [En ligne] 29 Avril 2018. https://www.chessprogramming.org/NegaScout.

20. **Dennis M. Breuker Jos W.H.M. Uiterwijk.** Transposition Tables in Computer Chess. *citeseerx.ist.* [En ligne] 15 Fevrier 2013. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.5189&rep=rep1&type=pdf.

21. **GerdIsenberg.** Endgame_Tablebases. [En ligne] 11 Mai 2018. https://www.chessprogramming.org/Endgame_Tablebases.

22. **Omid David-Tabibi , Nathan S. Netanyahu.** Verified null-move pruning. *elidavid.* [En ligne] Septembre 2012. http://elidavid.com/pubs/vrfd_null.pdf.

23. **Jonatan Pettersson.** guide-aspiration-windows. *mediocrechess.blogspot.com.* [En ligne] 5 Janvier 2007. http://mediocrechess.blogspot.com/2007/01/guide-aspiration-windows-killer-moves.html.

24. **Kieran Greer.** Computer chess move-ordering schemes. *https://core.ac.uk.* [En ligne] 1 Fevrier 2000. https://core.ac.uk/download/pdf/82739320.pdf.

25. **Lyudmil Tsvetkov, Sofia, Bulgaria.** Little Chess Evaluation Compendium. *chessprogramming.* [En ligne] 2012. https://www.chessprogramming.org/images/7/70/LittleChessEvaluationCompendium.pdf.

26. **T.A. Marsland.** Computer chess and search. *semanticscholar.org.* [En ligne] 3 Avril 1991. https://pdfs.semanticscholar.org/f4b5/e9b93c4bf3d6af9a5a3e30b8d0f2ea0d7a00.pdf.

27. **Roll-Morton.** Maching learning. *Wikipédia.* [En ligne] 5 mars 2016. https://fr.wikipedia.org/wiki/Apprentissage_automatique#Applications.

28. **A. M. Turing.** computing machinery and intelligence. *csee.umbc.edu.* [En ligne] 1950. https://www.csee.umbc.edu/courses/471/papers/turing.pdf.

29. **Tomas Borovicka, Marcel Jirina Jr, Pavel Kordik and Marcel Jirina.** Selecting Representative Data Sets. *cdn.intechopen.* [En ligne] 2016. http://cdn.intechopen.com/pdfs/39037/InTech-Selecting_representative_data_sets.pdf.

30. **Mokhtar TAFFAR.** apprentissage automatique. *univ-jijel.* [En ligne] 12 3 2017. http://elearning.univ-jijel.dz/elearning/pluginfile.php/4333/mod_resource/content/1/SupportCours_Mokhtar_Taffar_ApprAuto.pdf.

31. RapidMiner. *univ-lille.* [En ligne] janvier 2011. http://www.fil.univ-lille1.fr/~decomite/ue/MFFDD/tp1/rapidminer.pdf.

32. **Package R.** Data Mining using R. *rattle.togaware.* [En ligne] https://rattle.togaware.com/RattleBrochure.pdf.

33. *eric.univ-lyon2.* [En ligne] http://eric.univ-lyon2.fr/~ricco/tanagra/.

34. **Richard S. Sutton and Andrew G. Barto.** Reinforcement Learning:An Introduction. *incompleteideas.net.* [En ligne] 1 Janvier 2018. http://incompleteideas.net/book/bookdraft2018jan1.pdf.

35. **Kardi Teknomo.** Decision tree. *tanthiamhuat.files.* [En ligne] Octobre 2012. https://tanthiamhuat.files.wordpress.com/2015/10/decision-tree-tutorial-by-kardi-teknomo.pdf.

36. **Karl Pearson.** Statistics is the grammar of science. *cedric.cnam.* [En ligne] 30 Avril 2018. http://cedric.cnam.fr/vertigo/Cours/ml2/coursForetsAleatoires.html.

37. **Rich Zemel, Raquel Urtasun and Sanja Fidler.** Nearest Neighbors. *Toronto.* [En ligne] Janvier 2017. https://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/05_nn.pdf.

38. **Olivier PARENT, Julien EUSTACHE.** Les Réseaux Bayésiens. *perso.liris.cnrs.fr.* [En ligne] 2007. https://perso.liris.cnrs.fr/alain.mille/enseignements/master_ia/rapports_2006/Reseau%20Bayesien%20SYNTHESE%20ECRITE.pdf.

39. **Jean Larsen.** Introduction to artificial neural networks. *imm.dtu.dk.* [En ligne] Novembre 1999. http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/2443/pdf/.

40. **Andrej Krenker, Janez Bešter, Andrej Kos.** Artificial Neural Networks. *intechweb.* [En ligne] 11 Avril 2011. http://cdn.intechweb.org/pdfs/14881.pdf.

41. **Jason Brownlee.** gradient-descent. *machinelearningmastery.* [En ligne] 23 Mars 2016. https://machinelearningmastery.com/gradient-descent-for-machine-learning/.

42. **Sebastian Ruder.** An overview of gradient descent optimization algorithms. *Ruder.* [En ligne] 19 Janvier 2016. http://ruder.io/optimizing-gradient-descent/index.html.

43. **Tuxtla Gutiérrez.** The Best Neural Network Architecture. *researchgate.* [En ligne] Juillet 2014. https://www.researchgate.net/publication/281825506_The_Best_Neural_Network_Architecture.

44. **David Stutz.** Understanding Convolutional Neural Networks. *davidstutz.de.* [En ligne] 30 Aout 2014. https://davidstutz.de/wordpress/wp-content/uploads/2014/07/seminar.pdf.

45. **Roger Grosse et Nitish Srivastava.** Introduction to RNNs. *slazebni.cs.illinois.edu.* [En ligne] 3 Fevrier 2015. http://www.cs.toronto.edu/~rgrosse/csc321/lec9.pdf.

46. **Charles Crouspeyre.** Comment les Réseaux de neurones récurrents et Long Short-Term Memory fonctionnent. *medium.com.* [En ligne] 10 Octobre 2017. https://medium.com/@CharlesCrouspeyre/comment-les-réseaux-de-neurones-à-convolution-fonctionnent-c25041d45921.

47. **Michael Nguyen.** Illustrated Guide to LSTM's and GRU's: A step by step explanation. *towardsdatascience.com.* [En ligne] 24 Septembre 2018. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.

48. **STATISTICA Entreprise.** Réseaux de Neurones. *statsoft.* [En ligne] 2013. http://www.statsoft.fr/concepts-statistiques/reseaux-de-neurones-automatises/reseaux-de-neurones-automatises.htm#fonctions.

49. **Lachlan Miller.** Cost Function. *medium.com.* [En ligne] 10 Janvier 2018. https://medium.com/@lachlanmiller_52885/machine-learning-week-1-cost-function-gradient-descent-and-univariate-linear-regression-8f5fe69815fd.

50. **Piji Li.** Optimization Algorithms for Deep Learning. *lipiji.* [En ligne] 2017. http://lipiji.com/docs/li2017optdl.pdf.

51. **Akarachai Atakulreka and Daricha Sutivong.** Avoiding Local Minima in Feedforward Neural Networks. *springer.* [En ligne] 2007. https://link.springer.com/chapter/10.1007/978-3-540-76928-6_12.

52. **Jason Brownlee.** Train Neural Networks With Noise to Reduce Overfitting. [En ligne] 12 Decembre 2018. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/.

53. **WikiStat.** Agrégation de modèles. *math.univ-toulouse.* [En ligne] https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-agreg.pdf.

54. **LeelaChessZero.** https://github.com. *LeelaChessZero.* [En ligne] [Citation : 10 Juin 2019.] https://github.com/LeelaChessZero/lc0/.

55. **David Silver, Thomas Hubert, et al.** Mastering Chess and Shogi by Self-Play with a. *https://arxiv.org.* [En ligne] 5 Decembre 2017. https://arxiv.org/pdf/1712.01815.pdf.

56. **Veedrac.** Leela Chess—Test40, Test50, and beyond. *https://medium.com.* [En ligne] 13 Février 2019. https://medium.com/@veedrac/leela-chess-test40-test50-and-beyond-c15896becfac.

57. **Andrew NG.** Pooling Layers. *youtube.* [En ligne] 7 Novembre 2017. https://www.youtube.com/watch?v=8oOgPUO-TBY.

58. **Timothy Masters .** Practical Neural Network Recipes in C++. *pdfs.semanticschola.* [En ligne] 1993. https://pdfs.semanticscholar.org/0ca8/7a5fc435deedb1696e4afec4907b23db874a.pdf.

59. **Mark Azzam Jan-Christoph Haag Peter Jeschke.** Application concept of artificial neural networks for turbomachinery design. *pdfs.semanticscholar*. [En ligne] 19 Aout 2009. https://pdfs.semanticscholar.org/4e78/6374980998594d89feb059097d8e4a832be4.pdf.

60. **Siraj Raval.** Loss Functions Explained. *youtube.* [En ligne] 24 Juillet 2018. https://www.youtube.com/watch?v=IVVVjBSk9N0.

61. **Jason Brownlee.** How to Choose Loss Functions When Training Deep Learning Neural Networks. *machinelearningmastery.* [En ligne] 30 Janvier 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

62. **Prince Grover.** 5 Regression Loss Functions All Machine Learners Should Know. *heartbeat.fritz.* [En ligne] 5 Juin 2018. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0.

63. **Stefan-Meyer Kahlen.** UCIProtocol. *wbec-ridderkerk.* [En ligne] Avril 2004. http://wbec-ridderkerk.nl/html/UCIProtocol.html.

64. **Andrea Giarrizzo.** Visual_studio_code. *edutechwiki.* [En ligne] 3 juin 2016. https://edutechwiki.unige.ch/fr/Visual_studio_code.

65. **Aryan mobiny.** how to use tensorboard. *itnext.io.* [En ligne] 9 juin 2018. https://itnext.io/how-to-use-tensorboard-5d82f8654496.

66. **Niklas Fiekas.** python-chess. *python-chess.readthedocs.io.* [En ligne] Octobre 2015. https://python-chess.readthedocs.io/en/latest/core.html.

67. **Karlijn Willems.** TensorFlow Tutorial For Beginners. *datacamp.* [En ligne] 16 janvier 2019. https://www.datacamp.com/community/tutorials/tensorflow-tutorial?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=1t1&utm_creative=332602034361&utm_targetid=au.

68. **Adrian Rosebrock.** Keras Tutorial: How to get started with Keras, Deep Learning, and Python. *pyimagesearch.* [En ligne] 10 Septembre 2018. https://www.pyimagesearch.com/2018/09/10/keras-tutorial-how-to-get-started-with-keras-deep-learning-and-python/.

69. **NumPy community.** NumPy User Guide. *docs.scipy.org.* [En ligne] 29 Mai 2016. https://docs.scipy.org/doc/numpy-1.11.0/numpy-user-1.11.0.pdf.

70. **Smokiev.** Des bases de données en Python avec. *zestedesavoir.* [En ligne] 20 Mars 2019. https://zestedesavoir.com/tutoriels/pdf/1294/des-bases-de-donnees-en-python-avec-sqlite3.pdf.

71. **team chess.** Chess Variants | 5 Amazing Examples. *chess.* [En ligne] 8 Septembre 2006. https://www.chess.com/fr/article/view/cinq-variantes-des-echecs-passionnantes.

72. **Yaron Singer.** Advanced Optimization. *people.seas.harvard.* [En ligne] 24 Fevrier 2016. https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture9.pdf.

Annexes

A Généralités sur les jeux d'échecs et leur informatisation

A.1 Les règles de la variante classiquedu jeu d'échecs

A.1.1 Règles du jeu

- La composition initiale des pièces est identique pour chaque joueur.
- Le joueur ayant les pièces blanches commence la partie.
- L'objectif étant de mettre le roi adverse en position d' « échec et mat [1]»
- Chacune des 6 types de pièces se déplace selon des règles bien spécifiques La prise en passant est un cas spécial de capture de pions
- Le roque est un mouvement particulier -soumit à des conditions-qui offre une sécurité au Roi et une activité pour la Toure
- La promotion permet l'échange du pion -contre une des autres types de pièces- à son arrivé à la dernière rangée.
- On attribue pour chaque pièce un nombre de points symbolique caractérisant leurs forces respectives.
- Le roi est la pièce la plus importante des jeux d'échecs, sa sécurité est primordiale et sa valeur est incomparable aux autres pièces.
- Lorsqu'un joueur gagne une partie il aura 1 point, s'il perd il aura 0 points et enfin une partie nulle lui vaut ½ point.
- Toute pièce touchée est une pièce jouée.
- Une partie de jeux d'échecs est composée en trois phases :
  - **Ouvertures** : représente les premiers coups de la partie le principe se décompose en trois sections principales :
    - Développement de pièces.
    - Contrôle du carré central de l'échiquier
  - Sécurité du Roi (Roque)
  - **Milieux de jeux** : suit immédiatement la phase de l'ouverture et se décompose en deux thèmes principaux :
    - Tactique : se base principalement sur la capacité de calcule elle se présente sous forme de combinaisons, elle inclut plusieurs thèmes tactiques (voir l'annexe A.3) : la fourchette, le clouage, double attaque, attaque à la découverte, attraction, enfilade …
    - Stratégie : en absence de l'un des thèmes tactique la stratégie rentre en jeu afin de créer des positions idéales pour préparer une attaque. Elle inclut plusieurs thèmes: créer une diagonale pour le fou, placer les cavaliers sur les bonnes cases, créer de bonnes colonnes ou rangées pour les toursetc.
  - **finale** : la phase la plus importante et la plus déterminante elle se présente quand il reste peu de pièces sur l'échiquier (moins de 7 pièces)
- Un joueur gagne une partie dans un des cas suivant :
  - Echec et mat.
  - Temps réglementaire de l'adversaire est épuisé.

[1] Position ou le roi maté n'a aucune case de fuite ni une pièce pour le défendre.

- Faux coups [2] de l'adversaire.
- Abandon de l'adversaire.
- Conduite illégale (tentative de tricherie, comportement inapproprié, etc.)

➢ La partie est déclarée nulle dans l'un des cas suivants :

- Offre de nullité acceptée.
- Position de « Pat [3] », ou position théorique ou aucun mat ne peut être fait.
- La position se reproduit trois fois consécutivement.
- 70 coups consécutifs de pièces sans aucun mouvement de pions ni de captures.

A.1.2 Règles de compétition

En plus des règles de bases dans les parties amicales, d'autres facteurs rentrent en jeu lors d'une compétition rajoutant ainsi d'autres contraintes, voici quelques-unes :

➢ La « pendule [4]» est indispensable dans toutes compétitions
➢ Toutes les cadences sont par conventionappliquées selon les normes FIDE.
➢ la « chute du drapeau [5]» implique la fin de la partie.
➢ En cas de situations anormales, l'arbitre intervient met en pause le jeu.
➢ Un « Élo [6] » est attribué à chaque joueur licencié après un nombre de participations.

A.2 Variante des jeux d'échecs

Outre que la variante classique la plus connu de toutes, Il existe de nos jours plusieurs autres variantes des jeux d'échecs avec des règles propres à chacune, mais qui se partagent le même but et règles de déplacements que la variante principale (67)**,** les plus connues sont celles décrites dans le tableau ci-dessous :

Variante	***Petit descriptif***
Le Blitz a quatre	- Une variante a deux équipes - sous-variante de Crazyhouse - Une pièce capturée par un joueur peut être réutilisée par son coéquipier.
Les trois échecs (three checks)	- trois échecs d'un joueur pour son adversaire gagne la partie

[2] Des coups qui contredisent le principe de déplacements de pièces donc les règles de jeux d'échecs.
[3] Position (différente de l'échec et mat) dans laquelle le joueur ayant le trait ne peut en aucun cas déplacer une pièce.
[4] Appareil électronique à deux horloges munies d'un compte à rebours.
[5] La fin du compte à rebours.
[6] Nombre de point qui change en fonction des résultats du joueur.

Le roi de la montagne (King of the Hill)	-	On entre le joueur qui arrive à placer son roi dans le carré central de l'échiquier gagne la partie
Les échecs 960 (Rondom Fisher[7])	-	La composition initiale des pièces dans le début de partie est aléatoire.
Le Crazyhouse	-	Une pièce capturée peut être remise en jeu.
Atomique (atomic)	-	L'ors de la capture les deux pièces (attaquante et attaquée) disparaissentet font disparaitre aussi les pièces se trouvant dans les huit cases autours de la pièce capturée
La course des rois (racing kings)	-	Position initiale différente avec celle de la variante classique
	-	le but étant d'arriver a déplacé le roi vers la 8^{éme} rangée avant l'adversaire

Tableau annexe I-1 Variantes de jeu d'échecs

A.3 Les types de pièces

Pièces	Symboles	Points
Pion	♙	1
Cavalier	♘	3
Fou	♗	3
Tour	♖	5
Dame	♕	9
Roi	♔	

Tableau annexe I-2 Liste des pièces

A.4 Déplacement des pièces

- **Déplacement élémentaire:** se définit comme étant un simple mouvement fondamental d'une pièce d'une case de départ vers une case d'arrivé suivant des règles spécifiques (voir l'annexe A.1 : Règles de déplacements).

[7] Créé par Bobby Fisher ex champion du monde et l'un des pionniers et légendes des échecs.

- **Capture** : il s'agit d'un coup où la pièce attaquante d'un joueur prend la case de la pièce adverse qui sera quant à elle retirée immédiatement de l'échiquier.

• Cas spécial, celui de la prise en passant où un pion se place sur la 5ème rangée (4ème respectivement pour les noirs) et qu'au prochain coup l'adversaire déplace un pion se trouvant sur les colonnes voisine du pion déplacé, dans ce cas ce dernier peut capturer le pion déplacé comme s'il a été déplacé que d'une seule case.

- **Le Roque** : il existe deux types de roque : petit et grand, utilisé pour renforcer la sécurité du roi.
- **Promotion** : un pion blanc qui atteint la huitième rangée ou un pion noir qui atteint la première rangée sera échangée contre une pièce.
- **Echecvséchec et mat** :

• **Echec** : signifie que le roi et temporairement menacé.

• **Echec et mat** : signifie que le roi est en échecs imparable qui se traduit par l'un des cas suivants :

  - Aucune case de fuite pour le roi
  - Aucun défenseur possible pour parer la menace.
  - Impossible de capturer la pièce attaquante.

A.5 Régles tactiques

La tactique c'est un mécanisme qui est utilisé à court terme pour attaquer tout comme pour défendre, elle se repose sur la capacité de calcule combinée avec les règles de déplacement des pièces. Dans ce qui suit les thèmes tactiques les plus utilisés :

A.5.1 Le clouage

Le clouage se définit comme étant l'immobilisation totale (cas où l'otage est le roi) ou partielle (cas où l'otage est une pièce d'une grande valeur que la pièce clouée généralement la dame ou la tour) de la pièce adverse. La figure ci-dessous illustre les deux différents clouages, ainsi les cavaliers noirs en c6 et b6 sont cloués.

Clouage absolu

Clouage relatif

A.5.2 La fourchette

Figure annexe I:1Les types du clouage

Il s'agit d'une attaque double menée par un pion ou un cavalier et qui attaque simultanément deux pièces adverse de plus grande valeur.

Figure annexe I:2 Exemple de fourchette

Dans la figure ci-dessus le fou en c4 et le cavalier en e4 sont pris au piège d'une fourchette du pion d5

A.5.3 L'attraction

L'objectif principal d'une attraction est de forcer le déplacement d'une pièce adverse vers une case spécifique dans le but d'avoir une combinaison avantageuse ou dans le meilleur cas un échec et mat.

Figure annexe I:3 Exemple d'une attraction

Dans figure ci-dessus une possibilité d'attraction s'offre au blancs avec la suite forcée suivante : Fg4 - DxF - Ch6+ suivi de CxD. De ce fait les blancs ont attiré la dame noir en g4 afin d'appliquer une fourchette d'échecs ave le cavalier pour s'emparer enfin de la dame, une belle suite qui montre tout l'intérêt de ce thème.

A.5.4 Attaque à la découverte

C'est une attaque « silencieuse » masquée par une pièce de l'attaquant, ainsi dès que cette dernière se déplace l'attaque deviendra active. Dans le cas où la pièce attaquée est le roi nous parlons alors de l'échec à découverte.

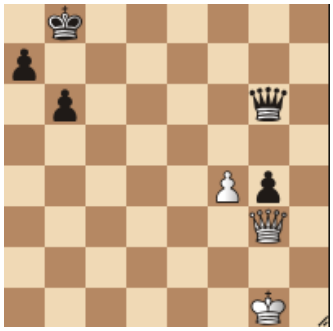


Figure annexe I:4 Exemple d'une attaque à la découverte (échecs à la decouverte)

Dans la figure ci-dessus les blancs peuvent capturer la dame noire par la simple suite suivante en utilisant l'attaque à la découverte : f5+ suivi de fxD ou DxD+.

A.5.5 Le zugzwang

Le zugzwang désigne un coup forcé désavantageux (ou plusieurs coups dans le cas d'un réseau de zugzwang) pour le joueur ayant le trait.

Figure annexe I:5 Exemple de zugzwang

Dans la figure ci-dessus les noirs ont le trait et sont forcés à jouer un coup de roi qui leur fera perdre la partie

A.5.6 Elimination du défenseur

Elle consiste à retirer la défense adverse contre une menace ultérieure et cela en utilisant l'attraction ou sacrifice de pièce ou autres thèmes. Ainsi, dans la figure ci-dessous une bonne suite pour les blanc permet de gagner un fou en utilisant ce thème tactique : TxT - FxT - TxF.

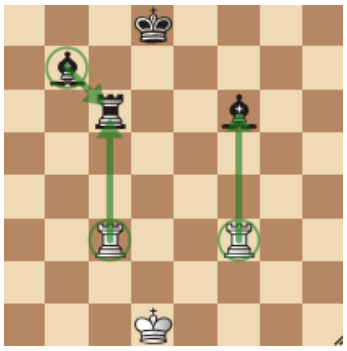


Figure annexe I:6 Exemple de l'élimination du défenseur

A.5.7 La surcharge

Il s'agit du cas où une pièce adverse défense simultanément plusieurs pièces ce qui la rend susceptible à des attaques dangereuses.

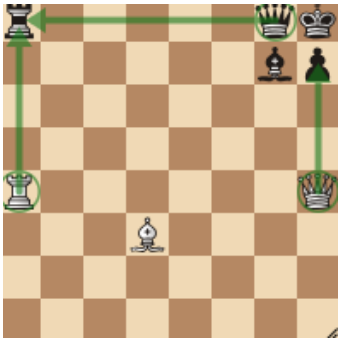


Figure annexe I:7 Exemple de surcharge

Dans la figure ci-dessus la dame noire défend simultanément la tour, le fou et le pion. Par conséquent, les blancs en profité de cette faiblesse afin de concrétiser leur menace directe en h7 ainsi la suite gagnante est : TxT – suivi de Dxh7 #

A.5.8 L'enfilade

Il consiste à attaquer une pièce indirectement à travers l'attaque d'une autre ou un échec.

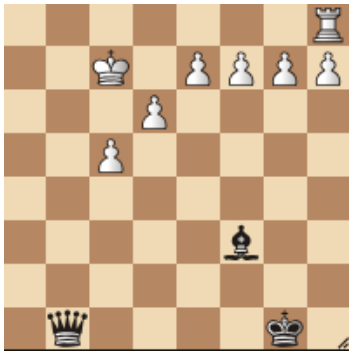


Figure annexe I:8 Exemple d'enfilade

Dans la figure ci-dessus il existe une bonne suite pour les noirs en utilisant ce thème qui leur permettra de s'emparer de la tour blanche : Dg2+ - Re1 – Dg1+ suivi de Dxa1

A.5.9 Echec intermédiaire

Il s'agit du thème à ne pas oublier lors d'un calcul d'une combinaison. Il consiste à faire un échec dans le but de gagner un temps et de mener une attaque imprévisible contre le côté adverse.

Figure annexe I:9 Exemple d'échec intermédiaire

Dans la figure ci-dessus, les noirs viennent de capturer un fou en c4 et menacent un échec et mat en b2. De ce fait, au lieu de reprendre le cavalier par la dame blanche, les blancs appliquent un échec intermédiaire qui s'avère déstabilisant pour les noir. La suite est la suivante : Td8+ - Fxd8 (Dxd8 - Txf7) - Fg7+ - Rg8 (Txg7 - Tf8+ - Tg8- Txg8#)- Fxe5+ ce qui fera entrer les noirs dans un réseau de mat avec la suite suivante : Rf8 - Dg7+ - Re8 - Dxf7#.

A.6 Règles stratégiques

La stratégie est un mécanisme qui est utilisé pour le long terme il sert à améliorer la position et à préparer des positions tactiques gagnantes. Généralement c'est le facteur qui sépare les joueurs du plus haut niveau. Ci-dessous quelque unes des stratégies les plus utilisées :

A.6.1 L'activité des cavaliers

C'est une méthode qui permet d'optimiser le pouvoir du cavalier, utilisée quand une position est statique, dans ce cas le cavalier devient une pièce maitresse et dépasse le pouvoir du fou si le joueur le place dans des cases pertinentes où il ne pourra pas être chassé (voir figure ci-dessous).

Figure annxe I:10 l'activité du cavalier

A.6.2 L'activité des fous

Tout comme le cas précèdent les fous peuvent surpasser le pouvoir de cavalier dans le cas où ils occupent de bonnes diagonales ainsi leur pouvoir s'optimise dans les positions dynamiques (voir la figure ci-dessous).

Figure annexe I:11 l'activité du fou

Dans la figure ci-dessus nous constatons que le fou noir est plus performant que celui de son adversaire car il occupe une bonne diagonale (a7g1) et cloue partiellement le fou en c1.

A.6.3 L'activité des tours

La tour devient une pièce importante quand elle contrôle des couloirs (une colonne ou une rangée) et optimise ses forces quand on les double et notamment sur la 7ème rangée (2ème respectivement pour les noirs).

A.6.4 Structure des pions

La structure de pions est une notion très importante aux échecs, une bonne structure de pions consiste à la garder liée. Ainsi, l'objectif stratégique des joueurs consiste à affaiblir la structure de l'adversaire en créant des pions isolés, pions arriérés, pions doublés. D'autre part, chaque joueur essaye d'améliorer sa structure en créant des pions avancés, pions passés ou une marée de pions.

A.7 Notation des coups

- **Notation Algébrique Figurine (NAF) :** c'est une sous variante de la notation (NAS) qui normalise cette dernière en une seule et unique notation pour tout le monde. Elle consiste à remplacer la première lettre des pièces en une figure de la pièce correspondante (annexe A.3), cequi la rend compréhensible quelle qu'elle soit la langue des joueur.
- **Notation Algébrique Longue (NAL) :** notation identique avec « NAS » qui utilise de plus la rangée et colonne de départ de la pièce déplacée.
- **Notation de Smith :** C'est l'une des notations les plus précises vu qu'elle fait abstraction du type de la pièce et du type du déplacementet en contrepartie elle précise uniquement la case de départ et la case d'arrivé.
- **Notation numérique :** Cette notation est utilisée notamment pour résoudre les ambigüités que peut causer la notation algébrique lors d'une partie par correspondance[8], elle est inspirée de la notation Smith à la différence qu'elle utilise la numérotation des cases au lieu des lettres. (voir figure ci-dessous).

18	28	38	48	58	68	78	88
17	27	37	47	57	67	77	87
16	26	36	46	56	66	76	86
15	25	35	45	55	65	75	85
14	24	34	44	54	64	74	84
13	23	33	43	53	63	73	83
12	22	32	42	52	62	72	82
11	21	31	41	51	61	71	81

Figure Annexe I:12 Codage numérique

A.8 Méthode Delta

Il s'agit d'une méthode utilisée afin de générer les déplacements possibles d'une pièce notamment dans la méthode 0x88. Elle consiste à attribuer à chaque pièce un vecteur delta qui sera utilisé pour la génération des déplacements.

Par exemple, le vecteur delta d'une tour est : Delta= [-1,-16, 1, 16, 0, 0, 0, 0]

[8] Variante de jeux d'échecs à longue distance où les joueurs utilisent les moyens de communication (mails, serveurs...)

Figure annxe I:13 Application de la méthode Delta

En utilisant la figure ci-dessus et le vecteur delta, nous calculons les prochaines positions possibles en utilisant l'équation suivante : position suivante = position actuelle + delta[i].

En appliquant la formule : 52-1=51, 52-16=36, 52+1=53, 52+16=68. Ce qui montre que les prochaines positions possibles sont : 51, 36, 53, 68.

A.9 Calcul de la clé Zobrist

La clé Zobrist est le nombre résultant issu de l'application d'une fonction de hashage appelée « fonction zibrist » à une position d'échecs particulière.

Le calcul de cette clé de zobrist d'une position se fait en suivant les étapes suivantes :

1. Génération aléatoire de douze nombres pour chaque case de l'échiquier qui correspondent au six types de pièces des deux côtés.
2. Pour chaque case, nous sélectionnons le nombre qui correspond à la pièce qui se situe dedans.
3. On convertit tous les nombres sélectionnés en binaire.
4. On applique un XOR entre les nombre.
5. On convertit en binaire le nombre résultant qui correspond à la clé zobrist.

A.10 Algorithme Monte Carlo

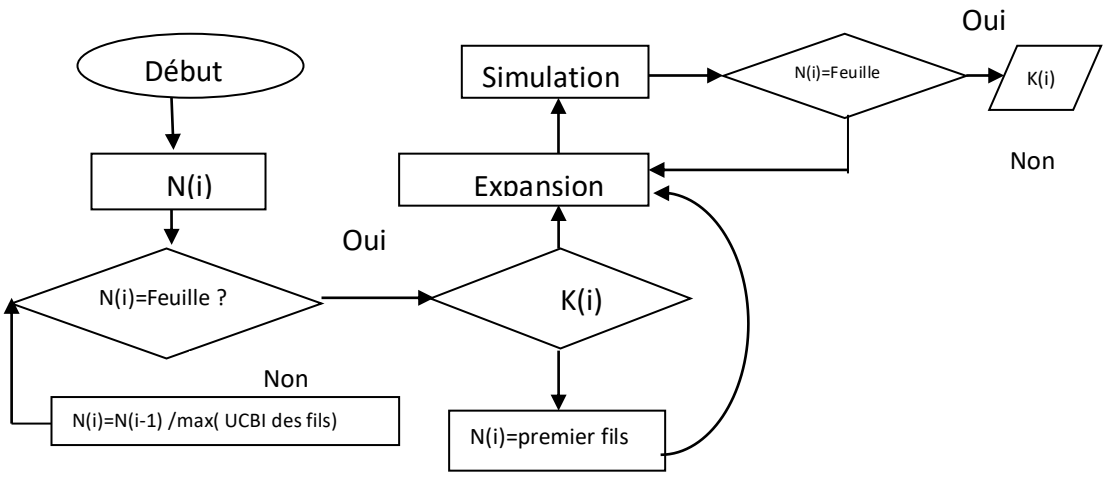


Figure annexe I:14 Organigramme algorithme Monte-Carlo

A.11 Algorithme de Minimax

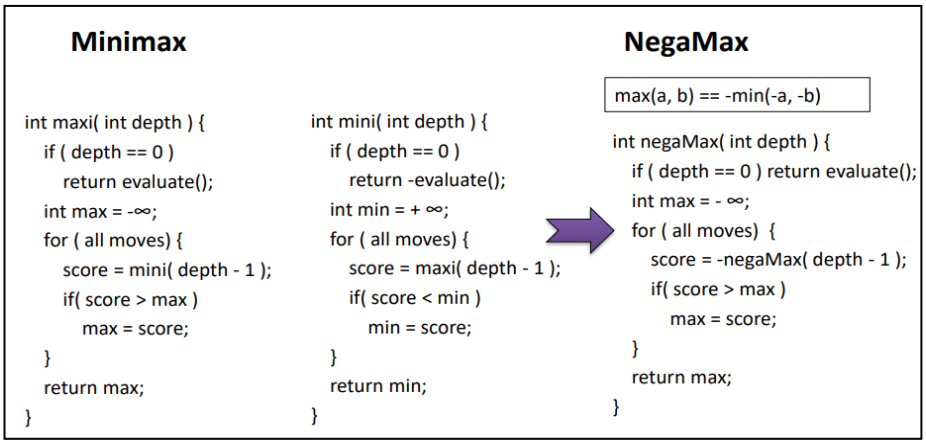


Figure annexe I:15 algorithme Minimax

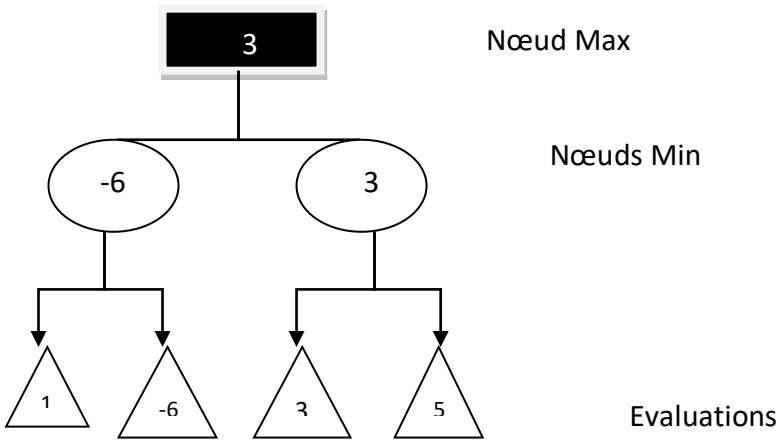


Figure annexe I:16 Exemple de Minimax

A.12 Algorithme Alphabéta

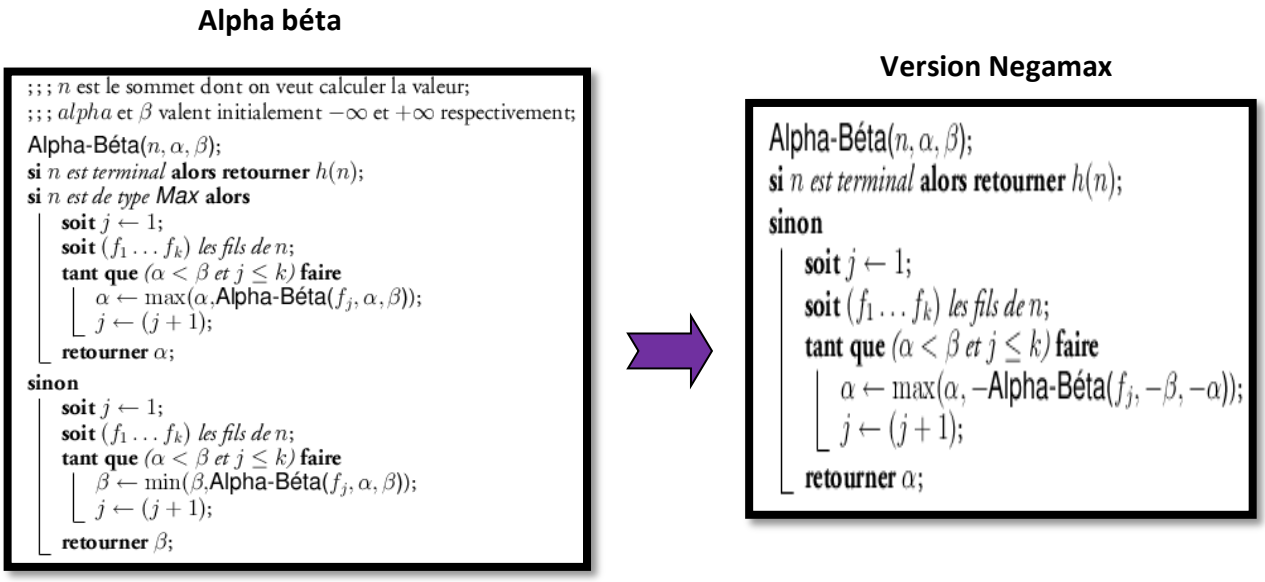


Figure annexe I:17 Algorithme AlphaBeta

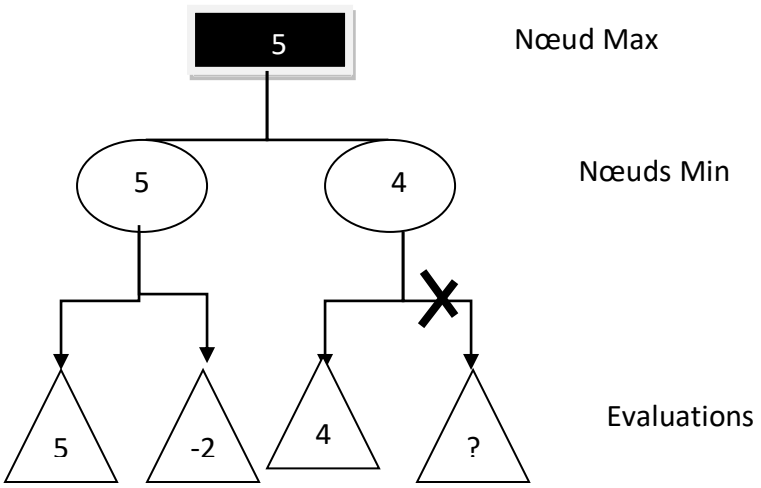


Figure annexe I:18 Exemple d'AlphaBeta

A.13 Algorithme Negascout

```
int NegaScout ( position p; int alpha, beta )
{                       /* compute minimax value of position p */
   int a, b, t, i;
   if ( d == maxdepth )
      return Evaluate(p);                          /* leaf node */
   determine successors p_1,...,p_w of p;
   a = alpha;
   b = beta;
   for ( i = 1; i <= w; i++ ) {
      t = -NegaScout ( p_i, -b, -a );
      if ( (t > a) && (t < beta) && (i > 1) && (d < maxdepth-1) )
         a = -NegaScout ( p_i, -beta, -t );      /* re-search */
      a = max( a, t );
      if ( a >= beta )
         return a;                                   /* cut-off */
      b = a + 1;                          /* set new null window */
   }
   return a;
}
```

Figure annexe I:19 Algorithme Negascout

B Généralités sur les architectures neuronales et leurs algorithmes

B.1 Architecture des reseaux de neurones artificiels :

La figure ci-dessous nous montre les réseaux de neurones les plus utilisés et les plus connus :

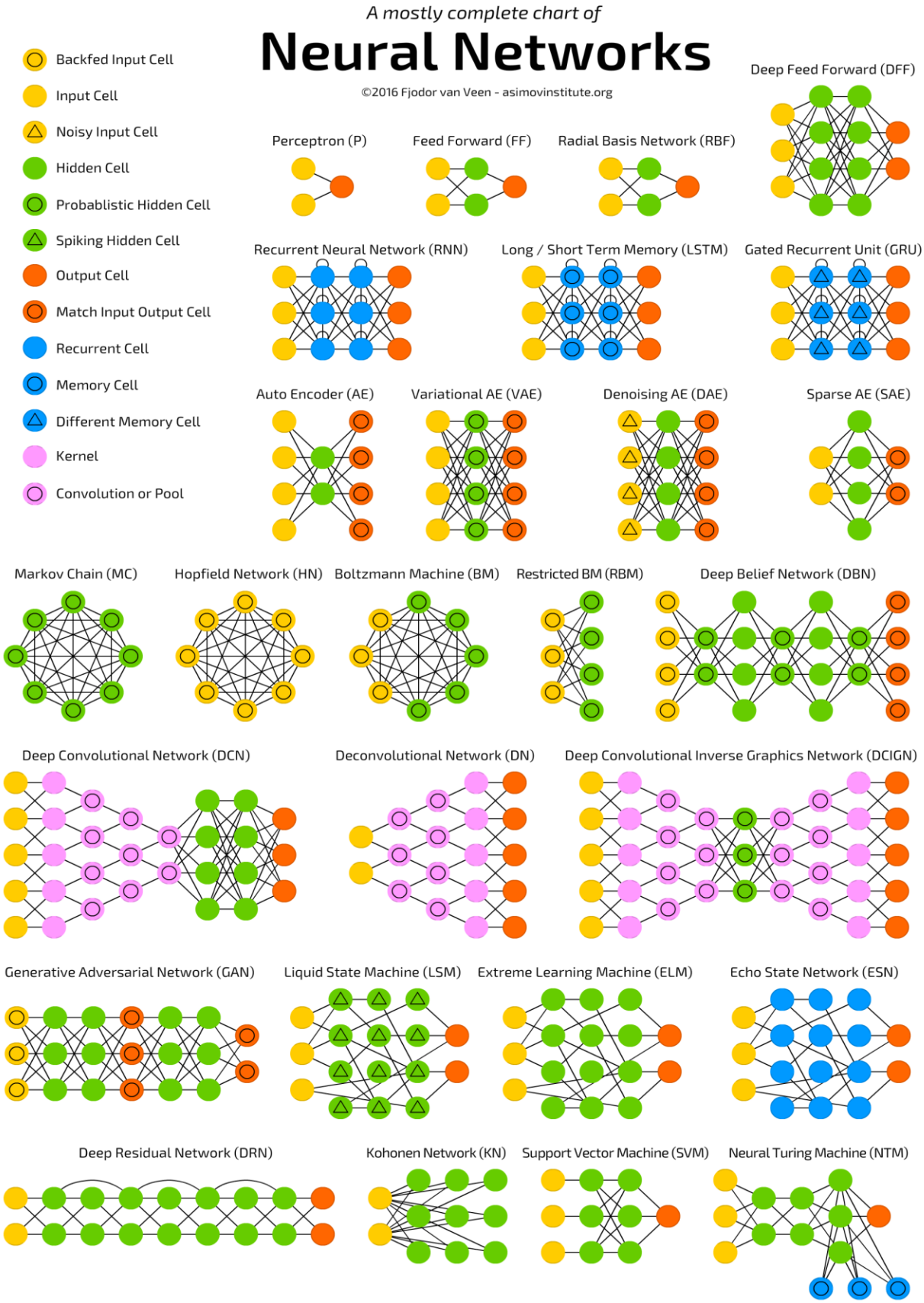


Figure annexe II:20 Les réseaux de neurones

B.2 Calcul du vecteur jacobien

Le vecteur jacobien (∇f) est un vecteur algébrique qui se présente comme étant le résultat de la dérivation d'ordre 1 d'une fonction à plusieurs variables par rapport à chaque variable et qui indique par défaut le sens à suivre afin d'atteindre la borne maximale de la fonction. Formellement il est défini comme suit :

$$\nabla f = \begin{cases} \dfrac{\partial f}{\partial x_1} \\ \dfrac{\partial f}{\partial x_2} \\ \dfrac{\partial f}{\partial x_n} \end{cases}$$

B.3 Calcul du vecteur hessien.

Le vecteur hessien (H) est un vecteur algébrique qui se présente comme étant le résultat d'une dérivation d'ordre 2 d'une fonction multi variables. Il permet d'indiquer la courbure locale du graphe de la fonction.

$$\mathrm{H} = \begin{cases} \dfrac{\partial^2 f}{\partial {x^2}_1} \\ \dfrac{\partial^2 f}{\partial {x^2}_2} \\ \dfrac{\partial^2 f}{\partial {x^2}_n} \end{cases}$$

B.4 Types d'architectures récurrentes

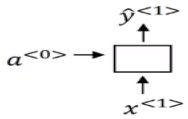
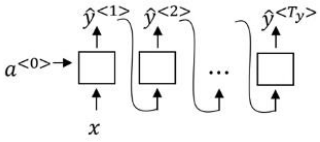
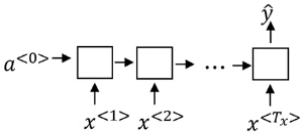
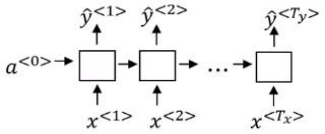
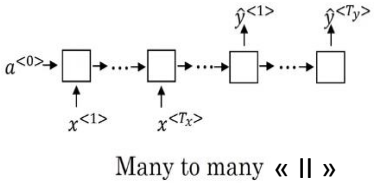
Type	Description	Application
 One to one	Une entrée génèreune sortie. (Classificateur binaire simple)	L'image représente elle un chat ou pas ?
 One to many	Une entrée génère plusieurs sorties.	Génération de musique.
 Many to one	Plusieurs entrées génèrent une sortie	Reconnaissance de fait via une vidéo (plusieurs images consécutives)
 Many to many « I »	Plusieurs entrées génèrent le même nombre de sorties	Analyse des séquences d'ADN
 Many to many « II »	Plusieurs entrées génèrent plusieurs sorties	Traduction automatique

Tableau annexe II-3 type d'architecture récurrente

B.5 Fonctions de cout

Catégorie	Fonction de cout	Equation	Variables
Classification	Binary Cross Entropy (BCE)	$BCE = -\sum_{i=1}^{c'=2} t_i \log S_i$	- t : label 1 ou 0. - Si =p(t_i) +(1- t_i) * log(1- t_i)
	Categorical Cross Entropy (CCE)	$CCE = -\log(\frac{e^{s_p}}{\sum_j^c e^{s_j}})$	- Nombre de classes - Si =score de la classe « i »
	Perte focale	$PF = -\sum_{i=1}^{c=2} (1 - S_i)^{\gamma} \ t_i \log S_i$	Idem que BCE
	Multi class SVM	$SVM = \sum_{j \neq y_i} max(0, S_j - S_{y_i} + 1)$	Idem que CCE
	La Divergence de Kullback-Leibler (DKL)	$KLD(x, y) = y * (\log y - x)$	- x /y : Entré / sortie brute de la fonction de décision du classifieur.
Integration	MarginRanking(MR)	$MR(x, y) = \max(0, -y * (x_1 - x_2) + m)$	- m : la marge - x / y : idem que DKL
	Hinge	$H(x, y) = \begin{cases} x \ , si \ y = 1 \\ \max(0, \Delta - x) \ , si \ y = -1 \end{cases}$	- x /y : idem que DKL - Δ : Distance de Hamming
	Cosine	$C(x, y) = \begin{cases} 1 - \cos(x_1, x_2) \ , si \ y = 1 \\ \max(0, \cos(x_1, x_2) - m) \ , si \ y = -1 \end{cases}$	Idem que MR
Regression	MeanAbsoluteError(MAE)	$\text{MAE} = \sum_{i=1}^{n} \lvert y_i - \hat{y}_i \rvert / n$	- $\hat{y}_i$: valeur prédite. - y_i : valeur attendue. - n :nombre de prédictions
	Mean Absolute Percentage Error(MAPE)	$\text{MAPE} = \sum_{i=1}^{n} \left\lvert \frac{y^{(i)} - \widehat{y^{(i)}}}{y^{(i)}} \right\rvert * 100 / n$	Idem que MAE
	Mean Square Error(MSE)	$\text{MSE} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 / n$	Idem que MAE.
	Mean Squared Logarithmic Error(MSLE)	$\text{MSLE} = \sum_{i=1}^{n} (\log(y^{(i)} + 1) - \log(\widehat{y^{(i)}} + 1))^2 / n$	Idem que MAE
	MeanBiasError(MBE)	$\text{MBE} = \sum_{i=1}^{n} (y_i - \hat{y}_i) / n$	Idem que MAE
	Smooth L1	$S(x, y) = \begin{cases} 0.5 \, a^2 \ , si \ \lvert a \rvert < 1 \\ \lvert a \rvert - 0.5 \ , sinon \end{cases}$	- a : hyper-paramétre généralement égal à 1

Tableau annexe II-4 Fonctions d'erreurs

B.6 Apprentissage par renforcement

Algorithme 2: Apprentissage par Renforcement

Paramètres : espace d'états $\mathcal{S}$, espace d'actions $\mathcal{A}$, fonction (cachée) de transition T, fonction (cachée) de récompense R, facteur de dévaluation γ

Initialisation : L'environnement occupe un état $s_1 \in \mathcal{S}$ et le communique à l'agent

Tours : pour chaque tour de jeu $t = 1, 2, \ldots$
(1) l'agent perçoit l'état s_t et choisit une action $a_t \in \mathcal{A}$
(2) l'environnement retourne à l'agent la récompense r_t choisie aléatoirement selon $R(s_t, a_t)$ et occupe un nouvel état $s_{t+1} \in \mathcal{S}$ choisi aléatoirement selon $T(\cdot \mid s_t, a_t)$

Figure annexe II:21 Algorithme d'apprentissage par renforcement

B.7 Algorithme à taux d'apprentissage adaptatif

Optimisateur	Description
AdaGrad	- Adapte le taux d'apprentissage aux paramètres. - Utilisé pour le traitement de données complexes et denses. - Appliquer pourle SGD afin de l'améliorer. - Sauvegarde les gradients déjà calculés pour avoir une vision globale du chemin empreinte.
NesterovMomentum	- Inspiré du fonctionnement deSGD+M. - Calcule le momentum pour la prochaine position.
RMSProp	- Variante d'AdaGrad. - Un algorithme idéal pour les réseaux de neurones profonds. - Utilise un hyper-paramètre supplémentaire « RHO »
Adam	- Rassemble tous les avantages de RMSProp et SGD+momentum - Donne de bons résultats notamment avec une grande base de données complexe. - Se présente comme étant RMSProp+momentum
Adamax	- Optimisation d'Adam. - Le gradiant courant est le maximum entre le gardiant passé et le suivant. - Utilise des pas plus petits qu'Adam.
Nadam	- Nestrov Momentum appliqué àAdam. - Stable et rapide.
AMSGrad	- Résout des bugs d'Adam - Utilise un taux d'apprentissage plus petit que celui d'Adam. - Pour mettre à jours les hyper-paramètres il utilise le carré du gradient déjà calculé. - chemin emprunté plus stable que celui d'Adam
Adadelta	- Une optimisation d'AdaGrad - Limite le nombre de gradients stockés

Tableau annexe II-5Optimiseur à taux d'apprentissage adaptatif

C Principe de symétrie

La symétrie est une caractéristique particulière des positions des jeux d'échecs. Elle est utilisée notamment dans la programmation information des moteur d'échecs afin d'évaluer toujours la position en fonction des Blanc en faisant la symétrie des pièces en changeant la couleur par rapport à la rangée centrale de l'échiquier. À noter que cette propriété s'applique qu'aux positions ayant le trait aux noirs (voir exemple ci-dessous).

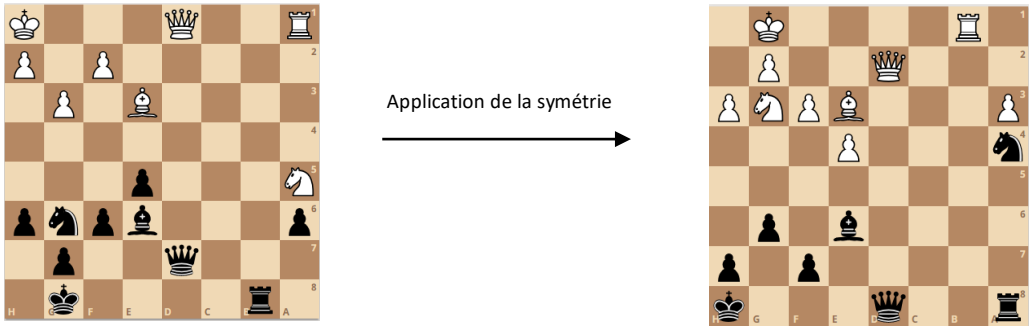


Figure annexe III:22 La symétrie

D Parties jouées et généralités sur les types de modèles.

D.1 Modèle séquentiel

L'API séquentielle de Keras (voir figure ci-dessous) est utilisée pour la plupart des problèmes en apprentissage automatique. Elle permet de créer des modèles comportant des couches successives. Cependant, cette API est limitée car elle ne permet pas de créer des modèles partageant des couches ou comportant plusieurs entrées ou sorties (la réutilisation des couches).

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(2, input_dim=1))
model.add(Dense(1))
```

Figure annexe IV:23 Exemple d'un modèle séquentiel

D.2 Modèle fonctionnel

Cette API offre plus de flexibilité que la précédente. Elle permet de construire des modèles beaucoup plus complexes à l'instar d'architectures résiduelles, avec une possibilité de réutilisation des couches (voir exemple ci-dessous).

```
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
inp = Input(shape=(844,))
side1 = Lambda(lambda x: x[:, 0: 832])(inp)
side1= Reshape((8, 8, 13))(side1)
side1 = Conv2D(200,(3,3), padding='same')(side1)
side1=MaxPooling2D(pool_size=(2, 2))(side1)
side1 =Conv2D(200, (3,3), padding='same')(side1)
side1=MaxPooling2D(pool_size=(2, 2))(side1)
side1 =Flatten()(side1)
side2 = Lambda(lambda x: x[:, 832: 844])(inp)
side2=Dense(200 , activation='relu')(side2)
side2=Dense(200 , activation='relu')(side2)
out = Concatenate()([side1, side2])
out = Dense(200, activation='relu')(out)
out = Dense(1, activation='tanh')(out)
self.model = Model(inputs=inp, outputs=out)
```

Figure annexe IV:24 Exemple d'un modèle fonctionnel

D.3 Protocole UCI

Le protocole de communication UCI (*Universal Chess Interface*) est considéré comme une avancée de l'ancien protocole WinBoard / XBoard. Il est conçu et développé par Rudolf Huber et Stefan Meyer-Kahlen et permet de facilité la communication entre les moteurs de jeux d'échecs. En outre, il s'agit du protocole phare qu'utilisent les moteurs d'échecs pour communiquer avec les GUI (*Graphical User Interface*).

Le protocole UCI contient plusieurs normes de type requêtes/réponses décrites ci-dessous :

- **uci** : demander le nom de l'option et obtenir le nom du moteur et celui du (des) auteur (s). La réponse se termine par « uciok ».
- **isready** : tester si le moteur est prêt à initier la communication.
- **ucinewgame** : crée une nouvelle partie avec la position initiale.
- **position [startpos / fen] moves Y** : permet de définir la position initiale de recherche pour le moteur d'échecs en indiquant optionnellement la suite de coup Y à joués à partir de la position définie précédemment.
- **go** : permet d'interroger le moteur d'échecs afin de recevoir le meilleur coup en utilisant plusieurs variantes :
  - **go infinite** : rechercher le meilleur coup indéfiniment
  - **go depth x** : arrêter la recherche à la profondeur x
  - **go movetime x** : rechercher le meilleur coup jusqu'à x millisecondes
  - **go wtime X btime Y winc Z binc M** : requête qui simule la présence de la pendule. Ainsi les blancs ont X ms avec incrémentation de Z et les noirs ont Y ms avec incrémentation de M après chaque coup.
- **stop**: permet d'arrêter la recherche du moteur s'il est actif.
- **quit**: permet de mettre un terme à la communication UCI.

D.4 Parties jouées

- **RNA1 vs RNA2** :1) c2c3 d7d6 2) d2d4 c7c5 3) e1d2 b7b6 4) d1a4 b8d7 5) a4d7 e8d7 6) d4c5 h7h5 7) c5c6 d7e8 8) b2b4 g7g5 9) d2e3 g5g4 10) f2f3 d8c7 11) c1d2 e8d8 12) g1h3 g8h6 13) f3g4 h6f5 14) g4f5 h8h7 15) b4b5 h7g7 16) e3e4 g7g5 17) d2g5 f8h6 18) g5e7 c7e7 19) e4d3 e7e4 20) d3e4 h6f8 21) c6c7 d8e7 22) c3c4 e7f6 23) b1c3 a8b8 24) c7b8q f8h6 25) b8d6 c8e6 26) d6d8 f6g7 27) d8g5 g7h8 28) g5d8 h8g7 29) d8g5 g7h8 30) g5d8 h8g7 31) d8g5 g7h8 32) g5d8 h8g7 33) d8g5 g7h8 34) g5d8 h8g7.

- **RNA2 vs RNA1** : 1) b2b4 e7e6 2) c2c3 d8h4 3) d1c2 f7f5 4) c2f5 g7g5 5) f5f8 e8f8 6) a2a4 h4f2 7) e1f2 f8f7 8) f2e1 f7f6 9) a4a5 g8e7 10) a5a6 c7c5 11) b4c5 f6g7 12) a6b7 a7a6 13) a1a6 d7d5 14) b7a8b c8d7 15) a8b7 e7c8 16) b7a8 h8d8 17) a8b7 g7f6 18) b7d5 f6g6 19) a6e6 d7e6 20) d5e6 b8a6 21) c5c6 c8b6 22) e6g8 g6f5 23) c3c4 f5e4 24) b1c3 e4f5 25) c3d5 d8a8 26) d5e7 f5g4 27) g8e6 g4h4 28) e7f5 h4h5 29) f5g7 h5h6 30) g7f5 h6h5 31) f5g7 h5h6 32) g7f5 h6h5 33) f5g7 h5h6 34) g7f5 h6h5 35) f5g7 h5h6 36) g7f5 h6h5
- **RNA1 vs RNA3** : 1) c2c3 b8a6 2) f2f3 f7f5 3) g1h3 g7g6 4) b2b4 d7d6 5) c3c4 b7b6 6) e1f2 e8d7 7) f2e3 f8h6 8) h3g5 f5f4 9) e3f4 e7e5 10) f4g3 d8e7 11) g5h7 e7h4 12) g3h4 h6g5 13) h4g3 c7c5 14) h7g5 g8e7 15) b1c3 e7f5 16) g3f2 a6c7 17) b4c5 d7c6 18) f2g1 f5g3 19) c3a4 h8h2 20) g1h2 g3f1 21) h1f1 c7e8 22) d2d3 d6c5 23) a4c5 c8d7 24) c5a6 a8d8 25) f1e1 d8c8 26) c4c5 e8d6 27) a1b1 b6b5 28) g2g3 d6b7 29) b1b3 e5e4 30) e2e3 b5b4 31) e1f1 d7e6 32) a6b8 c6c5 33) d3d4 c5d6 34) d1e1 d6e7 35) b3b4 e6a2 36) b8c6 e7d7 37) c6b8 d7e8 38) b4b7 a2c4 39) e1b4 c4f1 40) b4e7
- **RNA3 vs RNA1 :** 1) c2c4 b8a6 2) a2a4 f7f5 3) g1f3 e8f7 4) f3g5f7f6 5) g5h7h8h7 6) b1a3 d8e8 7) a3b5 e6 8) Cxc7 Fa3 9) Cxe8+ Rg5 10) Cxg7 Th5 11) Txa3 Rf6 12) Cxh5+ Rg5 13) Te3 Rg4 14) h4 Rxh5 15) Th2 Ce7 16) Te5 Cg8 17) Txf5+ exf5 18) e3+ Rh6 19) e4 Rg7 20) exf5 b6 21) Dg4+ Rf7 22) Dxg8+ Rxg8 23) h5 Rg7 24) h6+ Rf7 25) Th4 Cc7 26) Th2 a6 27) Th4 Re7 28) Tf4 Rf7 29) Tf3 d6 30) d4 Rf6 31) Ff4 Ta7 32) Fxd6 Ce8 33) c5 Fb7 34) Tg3 Cxd6 35) cxd6 Fd5 36) h7 Txh7 37) Fxa6 Rxf5 38) f3 Rf4 39) Tg6 Th1+ 40) Ff1 Fc4 41) d7 Re3 42) d8=D Txf1# *
- **RNA1 vs RNA4 :** 1) c2c3 d7d6 2) d2d4 b7b5 3) e1d2 h7h5 4) b2b4 h8h7 5) f2f3 b8a6 6) g2g3 h7h6 7) d1c2 h6f6 8) g1h3 c7c5 9) d4d5 d8a5 10) g3g4 h5g4 11) b4a5 e8d8 12) a2a4 f6g6 13) a1a3 g6f6 14) c3c4 f6g6 15) h3g5 d8c7 16) g5e6 c7b7 17) e6d8 b7c7 18) d2c3 f7f5 19) f3g4 f5f4 20) d8e6 c7b7 21) e6d8 b7c7 22) d8e6 c7b7 23) e6d8 b7c7 24) d8e6 c7b7 25) e6d8 b7c7 26) d8e6 c7b7 27) e6d8 b7c7
- **RNA4 vs RNA1:** 1) d2d4 b8a6 2) d1d2 f7f5 3) e1d1 c7c5 4) d4c5 b7b6 5) d2d7 e8f7 6) c1f4 d8c7 7) d7d5 f7f6 8) f4g5 f6g6 9) g1f3 g6h5 10) a2a4 c8b7 11) d1c1 a8d8 12) g2g4 f5g4 13) d5f7 g7g6 14) f7e7 d8d1 15) c1d1 c7d6 16) e7d6 h7h6 17) d6b6 h8h7 18) b6b7 h7d7 19) b7d7 f8c5 20) f3e5 h5g5 21) d7a7 c5a3 22) b1a3 g8e7 23) d1c1 a6b4 24) a3b5 b4a2 25) a1a2 e7d5 26) c1d2 d5f6 27) d2e3 f6d5 28) e3d2 d5f6 29) d2e3 f6d5 30) e3d2 d5f6 31) d2e3 f6d5 32) e3d2 d5f6 33) d2e3 f6d5 34) e3d2
- **RNA2 vs RNA3:** 1) b2b4 b8a6 2) c2c3 h7h6 3) d1a4 f7f5 4) a4a6 d7d6 5) a6b7 h8h7 6) b7c7 a7a6 7) c7d6 d8a5 8) b4a5 h6h5 9) h2h4 h7h6 10) d6e5 e8d7 11) e5f5 d7c6 12) f5f3

c6b5 13) f3d5 b5a4 14) d5e5 a8a7 15) e5f5 g7g5 16) f5b5 a6b5 17) h4g5 c8h3 18) g1h3 b5b4 19) g5h6 a7a8 20) e1d1 a8c8 21) h3g5 g8h6 22) f2f3 h6g8 23) g5f7 c8c3 24) a5a6 c3c1 25) d1c1 g8f6 26) c1d1 f6h7 27) h1g1 h7f6 28) g1h1 f6h7 29) h1g1 h7f6 30) g1h1 f6h7 31) h1g1 h7f6 32) g1h1 f6h7 33) h1g1 h7f6 34) g1h1

- **RNA3 vs RNA2:** 1) c2c4 d7d5 2) c4d5 c7c6 3) e2e4 g8f6 4) d2d4 d8d7 5) d1g4 e8d8 6) g4g7 f6e8 7) g7f7 d7e6 8) d5c6 h7h5 9) h2h3 e6f6 10) f7e8 d8c7 11) e8c8 c7b6 12) c8b7 b6a5 13) b7b5
- **RNA2 vs RNA4:** 1) b2b4 d7d6 2) b1c3 b8a6 3) a1b1 b7b5 4) c3b5 h7h5 5) b5c7 e8d7 6) c7a6 h8h6 7) f2f3 d8b6 8) a6b8 d7c7 9) b8a6 c7b7 10) a6b8 b6f2 11) e1f2 h6f6 12) f2g3 h5h4 13) g3f2 a7a6 14) d2d3 b7b6 15) c1e3 b6b7 16) d1c1 f6f3 17) e2f3 f7f5 18) b8c6 f5f4 19) c6d8 b7c7 20) e3f4 c7b6 21) d8f7 g7g6 22) f7h8 a8a7 23) h8f7 g8h6 24) f7h8 b6c6 25) h8f7 h6g4 26) f3g4 g6g5 27) f7g5 a7b7 28) g5h7 b7a7 29) h7f6 a7b7 30) f6h7 b7a7 31) h7f6 a7b7 32) f6h7 b7a7 33) h7f6 a7b7 34) f6h7 b7a7 35) h7f6 a7b7 36) f6h7
- **RNA4 vs RNA2:** 1) d2d4 c7c5 2) d4c5 d8b6 3) c5b6 h7h6 4) c1h6 h8h7 5) d1c1 b8a6 6) h6g7 f7f5 7) g7f8 g8h6 8) f8h6 h7f7 9) b1a3 e7e6 10) g1h3 f7h7 11) c2c4 h7f7 12) a3b5 a6c7 13) h3g5 f7h7 14) a2a4 a8b8 15) b5d6 e8d8 16) g5h7 c7e8 17) h6g5 e8f6 18) g5f6
- **RNA3 vs RNA4:** 1) c2c4 d7d6 2) g1h3 b7b5 3) a2a4 h7h5 4) a4b5 h8h7 5) b5b6 c8a6 6) d1a4 c7c6 7) a4c6 d8d7 8) c6d7 b8d7 9) b6a7 d7b6 10) h3g5 f7f5 11) g5h7 f5f4 12) a1a6 a8b8 13) a7b8b b6a4 14) a6a4 e8d7 15) e1d1 d7c6 16) h7f6 g7g6 17) c4c5 g8h6 18) b8d6 h5h4 19) e2e3 f8g7 20) a4a6 c6b7 21) c5c6 b7c8 22) a6a8
- **RNA4 vs RNA3 :** 1) d2d4 h7h6 2) d1d2 d7d6 3) e2e4 b8a6 4) f1a6 b7b5 5) a6b5 d8d7 6) d2a5 h8h7 7) b5d7 e8d8 8) d7c8 f7f6 9) c8a6 a8c8 10) g1f3 e7e6 11) a2a4 d8e7 12) b1a3 e7e8 13) c1f4 c7c5 14) a3b5 h6h5 15) d4c5 g8h6 16) e4e5 c8c5 17) f4h6 g7h6 18) e5d6 h7g7 19) d6d7 e8e7 20) d7d8b e7d7 21) a1d1 d7c6 22) a5b6 a7b6 23) d8f6 g7g4 24) f3e5 c5e5 25) f6e5 g4h4 26) e5d6 h4e4 27) e1d2 e4e2 28) d2d3 e2e3 29) d3c4 f8e7 30) h1e1 c6d7 31) d6c7 d7c6 32) c7b6 e3e1 33) c4d3 e1e2 34) b6c7 e2d2 35) d3e3 d2e2 36) e3f4 e2f2 37) f4e5 f2e2 38) e5f4 e2f2 39) f4e5 f2e2 40) e5f4 e2f2 41) f4e5 f2e2 42) e5f4 e2f2 43) f4e5 f2e2 44) e5f4 e2f2
- **Simple_chess vs RNA1 :** 1) e2e4 b8a6 2) d1h5 d7d6 3) f1b5 d8d7 4) b5d7 e8d8 5) d7c8 f7f5 6) h5f5 g8f6 7) c8b7 f6h5 8) b7a8 c7c5 9) f5h5 d8d7 10) h5f5 d7c7 11) c2c3 h8g8 12) f5h7 g7g6 13) h7g8 e7e5 14) g8f7 c7d8 15) f7f8 d8d7 16) f8f7 d7d8 17) f7f6 d8c8 18) f6d6 a6c7 19) a8c6 c7a8 20) c6a8 g6g5 21) d6e5 a7a5 22) e5c5 c8b8 23) c5a5 g5g4 24) h2h3 g4g3 25) a5d8 b8a7 26) f2g3 a7a6 27) a8c6 a6a7 28) g1e2 a7a6 29) d2d4 a6a7 30)

c6d5 a7a6 31) a2a4 a6a7 32) e2f4 a7a6 33) h1f1 a6a7 34) f1h1 a7a6 35) h1f1 a6a7 36) f1h1 a7a6 37) h1f1 a6a7 38) f1h1 a7a6 39) h1f1 a6a7 40) f1h1 a7a6 41) h1f1

- **RNA1 vs Simple_chess :** 1) c2c3 e7e5 2) g1h3 d8h4 3) d2d4 h4e4 4) d4e5 d7d6 5) d1d6 c7d6 6) c1h6 g7h6 7) e5d6 c8h3 8) d6d7 e8d7 9) f2f3 e4h4 10) g2g3 h4h5 11) g3g4 h5h4 12) e1d1 h3f1 13) b1d2 f1g2 14) d1c2 g2h1 15) b2b4 h4h2 16) a1h1 h2h1 17) g4g5 h6g5 18) d2b1 h1f1 19) b4b5 f1e2 20) c2c1 e2f1 21) c1c2 f1f3 22) a2a4 f3e4 23) c2d1 e4b1 24) d1e2 b1c2 25) e2f3 c2a4 26) f3g3 f8d6 27) g3f3 a4c4 28) f3e3 c4c3 29) e3e4 c3c4 30) e4f3 g5g4 31) f3e3 h7h5 32) e3d2 d6f4 33) d2d1 h8h7 34) d1e1 f4e3 35) e1d1 g8f6 36) d1e1 h5h4 37) e1d1 h7h5 38) d1e1 d7e7 39) e1d1 b8d7 40) d1e1 a8e8 41) e1d1 h5d5 42) d1e1 e7f8 43) b5b6 f8g8 44) b6a7 c4c1 45) e1e2 d5d2