

Ordre...../F.S.S.A/UAMOB/

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA
RECHERCHE SCIENTIFIQUE
UNIVERSITÉ AKLI MOUHAND OULHADJ-BOUIRA



Faculté des Sciences et des Sciences Appliquées
Département d'Informatique

Mémoire

EN VUE DE L'OBTENTION DU DIPLÔME DE **Master** EN :

FILIÈRE : Informatique

OPTION : GSI

PRÉSENTÉ PAR :

Mlle.RABIA Molakhir

Mlle.SAIDANI Siham

Thème

Réseaux de neurones pour les jeux de plateau

Devant le jury composé de :

Mr.BOUDJELABA Hakim	Dr	UAMOB	Encadreur
Mme.MAHFOUD Zohra	Dr	UAMOB	Présidente
Mme.BOUGLIMINA Ouahiba	Dr	UAMOB	Examineur1
Mr.DEMOUCHE Mouloud	Dr	UAMOB	Examineur2
Mme.CHIHATI Sarah	Dr	UAMOB	Invité

Année Universitaire 2019/2020

Remerciements

Avant d'entamer ce mémoire de fin d'étude, nous tenons à exprimer notre sincère gratitude envers tous ceux qui nous ont aidé ou ont participé au bon déroulement de ce mémoire.

Tout d'abord, Nous exprimons nos profondes gratitudes et respectueuse reconnaissance à notre encadreur : **Mr. BOUDJELABA Hakim** pour sa bonne volonté d'accepter de nous encadrer, pour tout le temps qu'il nous a octroyé et pour tous les conseils qu'il nous a prodigué. Aussi que les membres de jury trouvent ici nos remerciements les plus vifs pour avoir accepté d'honorer par leur jugement notre travail. Nos vifs remerciements s'adressent également à nos enseignants et à nos amis, pour leur présence chaleureuse et leur encouragement.

Dédicaces

Je dédie ce modeste travail à

Ma chère mère et à mon cher père à propos de l'encouragement contenu, le réconfort
,la confiance, l'espoir, le soutien que vous me donnez.

Que dieu vous préserve santé et longue vie.

À toute ma famille, mes amis et à tous ceux qui ont contribué de près ou de loin pour
que ce projet soit possible.

Je vous remercie

RABIA Molkir

Dédicaces

Je dédie ce travail à :

Mes chers parents : Nacer, Djamila pour leur soutien, leurs sacrifices.

À mon frère Siphax et ma sœur Silia.

À toute ma famille.

À tous mes amis.

À ma collègue Molahir et sa familles.

SAIDANI Siham

Résumé

Les réseaux de neurones sont des outils très puissants utilisés surtout dans le domaine d'apprentissage automatique, ils peuvent être utilisés dans les jeux de plateau où la machine en partant de la seule connaissance des règles d'un jeu parvient à atteindre rapidement un niveau de jeu très élevé en jouant de nombreuses parties contre elle-même.

L'objectif de notre travail est de proposer un réseau de neurones capable d'apprendre et de jouer à un jeu de stratégie.

Mots clés : Réseaux de Neurones, apprentissage automatique, jeux de plateau.

Abstract

Neural networks are very powerful tools used especially in the field of machine learning, they can be used in board games where the machine, starting from the mere knowledge of the rules of a game, quickly reaches a very high level of play by playing many games against itself.

The objective of our work is to propose a neural network that is capable of learning and playing a strategy game.

Key words : Neural Networks, machine learning, board games.

ملخص

الشبكات العصبية هي أدوات قوية للغاية تستخدم خاصة في مجال التعلم الآلي، ويمكن استخدامها في ألعاب الطاولة حيث يتمكن الجهاز بدءاً من معرفة قواعد اللعبة الوصول بسرعة إلى مستوى عال جداً من اللعب خلال لعب العديد من الألعاب ضد نفسها. الهدف من عملنا هو اقتراح الشبكة العصبية التي هي قادرة على التعلم واللعب لعبة استراتيجية.

الكلمات الرئيسية: الشبكات العصبية، التعلم الآلي، ألعاب الطاولة.

Table des matières

Table des matières	i
Table des figures	iv
Liste des tableaux	vi
Liste des abréviations	vii
Introduction générale	1
1 Réseaux de neurones	3
1.1 Introduction	3
1.2 Définition	3
1.3 Historique sur les réseaux de neurones	4
1.4 Modélisation d'un neurone	5
1.4.1 Le neurone réel (Anatomy) et neurone artificiel (Formel)	5
1.4.2 Les éléments constitutifs d'un neurone artificiel	6
1.5 Fonctionnement d'un réseau de neurones artificiel	10
1.6 Types de perceptron	11
1.6.1 Perceptron simple	11
1.6.2 Perceptron multicouches (MLP)	11
1.7 Les différents types de réseaux de neurones artificiels	12
1.7.1 Réseau de neurones feed-forward	12
1.7.2 Les réseaux de neurones récurrents (RNN)	12
1.7.3 Les réseaux de neurones convolutifs (CNN ou ConvNet)	13

1.8	Apprentissage des réseaux de neurones	13
1.8.1	Apprentissage supervisé (back-propagation)	13
1.8.2	Apprentissage non-supervisé	13
1.8.3	Apprentissage semi-supervisé	14
1.8.4	Apprentissage par renforcement	14
1.9	Règle d'apprentissage	15
1.9.1	Loi de Hebb	16
1.9.2	Loi de Hopfield	16
1.9.3	Loi Delta rule (Règle de Widrow-Hoff)	16
1.10	Les avantages et les inconvénients des réseaux de neurones	17
1.10.1	Avantages	17
1.10.2	Inconvénients	17
1.11	Conclusion	18
2	Les jeux de plateau	19
2.1	Introduction	19
2.2	Travaux connexes	19
2.3	Jeu de Yokai No Mori	19
2.3.1	Version 3 * 4 (version classique)	20
2.3.2	Version 5 * 6	21
2.3.3	Type de réseau de neurones utilisé	21
2.3.4	Description de ce réseau de neurones	21
2.3.5	Résultats obtenus	23
2.4	Le jeu de Go	24
2.4.1	Matériels utilisés	25
2.4.2	Règles du jeu	25
2.4.3	Discussion	25
2.4.4	Type de réseau de neurones utilisé et type d'apprentissage	26
2.4.5	L'architecteur de ce réseau de neurones	26
2.5	Chess et Shogi	27
2.5.1	Travaux sur les échecs informatiques et Shogi	27
2.6	AlphaZero	29
2.6.1	Les connaissances de domaine utilisé par AlphaZero	29

2.6.2	Fonctionnalités d'entrée utilisées par AlphaZero respectivement dans Go, Chess et Shogi	29
2.6.3	L'application de l'algorithme AlphaZero aux échecs, shogi	31
2.7	Le jeu de Morpion	32
2.7.1	Présentation du jeu	32
2.7.2	Matériel du jeu	32
2.7.3	But du jeu	32
2.7.4	Règles du jeu	32
2.8	Conclusion	33
3	Proposition et implémentation	34
3.1	Introduction	34
3.2	Notre problématique	34
3.3	Solution proposée	34
3.3.1	L'architecture de notre réseau de neurones	34
3.3.2	Type du réseau de neurones	35
3.3.3	Type d'apprentissage	35
3.3.4	Étape d'entraînement	36
3.3.5	Implémentation de notre modèle	37
3.4	Implémentation	39
3.4.1	Langage utilisé	39
3.4.2	Plateforme et environnement de développement	39
3.4.3	Bibliothèque Utilisés	40
3.4.4	Architecture de notre application	41
3.4.5	Réalisation	44
3.5	Résultats expérimentaux	45
3.5.1	Discussion	46
3.6	Conclusion	46
	Conclusion générale	47
	Bibliographie	49

Table des figures

1.1	(a) Neurone réel et (b) neurone artificiel [7].	5
1.2	Fonction de Heaviside[8].	7
1.3	Fonction linéaire [9].	8
1.4	Fonction sigmoïde[10].	8
1.5	Fonction Tanh[11].	8
1.6	Fonction softmax [12].	9
1.7	Fonction ReLu[13].	9
1.8	Architecture d'un réseau profond à plusieurs couches[15].	10
1.9	Perceptron simple[17].	11
1.10	Perceptron multi-couches[18].	12
1.11	Apprentissage supervisé [22].	13
1.12	Apprentissage non-supervisé[22].	14
1.13	Architecture de l'approche d'apprentissage actif semi-supervisé[24].	14
1.14	Apprentissage par renforcement[25].	15
1.15	Algorithm d'apprentissage[2].	16
2.1	Jeu de Yokai No Mori[30].	20
2.2	La structure du réseau de neurones[32].	22
2.3	Jeu de Go[34].	24
2.4	(a) Jeu de Chess[36] et (b) Jeu de Shogi[37]	27
2.5	Le jeu de Morpion[41].	32
3.1	L'architecture du réseau de neurones.	35
3.2	Entraînement.	36

3.3	La Dataset.	37
3.4	L'Entrainement de notre réseau de neurone.	38
3.5	Implémentation de notre modèle.	39
3.6	Création du modèle.	42
3.7	Compilation du modèle.	42
3.8	Entrainement du modèle.	43
3.9	Architecture de notre application.	44
3.10	(a) et (b) et (c) Réalisation.	44
3.11	(a) Accuracy et (b) loss du modèle de base selon 20 époques.	45
3.12	(a) Accuracy et (b) loss du modèle de base selon 100 époques.	45

Liste des tableaux

- 1.1 Neurone réel et neurone artificiel 6
- 2.1 Fonctionnalités d'entrée utilisées par AlphaZero respectivement dans Go, Chess et Shogi 30
- 2.2 Représentation d'action utilisée par AlphaZero dans Chess et Shogi respectivement 31

Liste des abréviations

ADN	Acide DésoxyriboNucléique
ADALINE	ADaptive LINar Element
RNA	Réseaux de Neurones Artificiels
RNN	Recurrent Neural Networks
CNN(ConvNet)	Convolutional Neural Networks
Tanh	Hyperbolic Tangent Function
MLP	Multi Layer Perceptron
IA	Intelligences Artificielles
ReLU	Rectified Linear Unit
TD	Tower Defense
MCTS	Monte Carlo Tree Search
TCEC	Top Chess Engine Championship
CSA	Computer Shogi Association
DNN	Deep Neural Networks
SGD	Stochastic Gradient Descent
MSE	Mean squared Error

Introduction générale

Les jeux de plateau sont en fait préhistoriques, ce qui signifie que nous avons des jeux de société avant d'avoir un langage écrit. Le tout premier jeu c'est les dés, une pièce essentielle dans la plupart des jeux de plateau d'aujourd'hui. En effet les jeux sont amusants et peuvent entraîner notre cerveau à réfléchir à la manière de résoudre un problème, en particulier les jeux de plateau.

Dans le domaine de la science et de la technologie le terme d'intelligence artificielle joue un rôle prépondérant et ses récentes avancées lui ont permis de gagner en popularité pour les concepts d'intelligence artificielle et d'apprentissage automatique. Le rôle de l'intelligence artificielle a permis aux machines d'apprendre par expérience afin d'exécuter des tâches plus efficacement, "Faire jouer une intelligence artificielle, c'est avant tout accomplir une performance", Philippe Preux.

L'affrontement entre les hommes et les machines est un thème cher à la science-fiction. De nos jours, elles nous ont d'abord battus aux échecs. En 1997, Deep Blue, développé par IBM, battait le champion Garry Kasparov. Vingt ans plus tard, c'est au Go que s'illustre une intelligence artificielle. AlphaGo, un programme développé par une filiale de Google, a vaincu plusieurs champions de ce jeu de plateau réputé pour sa complexité. Au-delà de ces victoires très médiatisées, les intelligences artificielles jouent depuis plusieurs années.

L'objectif de notre travail consiste à créer une approche qui pourra jouer au jeu de plateau. C'est un problème qui peut être résolu par diverses méthodes d'apprentissage. Les techniques du Machine Learning et du Deep Learning permettent d'atteindre de bons résultats.

Pour réaliser notre travail nous allons suivre le plan suivant :

- **Chapitre 1** Est consacré aux réseaux de neurones, nous présenterons les différents types de réseaux de neurones, leurs fonctionnements, ainsi que les types d'apprentissage utilisés par ces réseaux de neurones et les algorithmes d'apprentissage existants.
- **Chapitre 2** Nous présenterons quelques travaux qui se sont intéressés sur l'utilisation des réseaux de neurones aux jeux de plateau.
- **Chapitre 3** Nous détaillerons notre proposition, les différentes étapes nécessaires pour la création et l'entraînement de notre modèle. Nous présenterons aussi quelques résultats expérimentaux de notre proposition et son implémentation.

Et nous terminerons par une conclusion générale et quelques perspectives.

Réseaux de neurones

1.1 Introduction

L'apprentissage profond (*Deep Learning*) est une forme d'intelligence artificielle, dérivée d'apprentissage automatique (*Machine Learning*), différentes techniques de Machine Learning ont été développées pour créer des algorithmes capables d'apprendre et de s'améliorer de manière autonome. Parmi ces techniques, on compte les réseaux de neurones artificiels.

Dans ce chapitre, nous allons parler des réseaux de neurones, nous présenterons les différents types de réseaux de neurones existants, leurs principes de fonctionnements, ainsi que les types d'apprentissage utilisés par ces réseaux de neurones et les algorithmes d'apprentissage existants.

1.2 Définition

Les réseaux de neurones artificiels sont présentés comme la vague du futur dans le domaine de l'informatique et neuro-informatique. Il s'agit en effet de mécanismes d'auto-apprentissage qui permettent aux ordinateurs de résoudre des problèmes en fonction de la méthode d'intelligence artificielle utilisée [1].

Les réseaux de neurones artificiels sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit [2].

1.3 Historique sur les réseaux de neurones

Une brève histoire des réseaux de neurones :

- 1890 : **W. James**, célèbre psychologue américain introduit le concept de mémoire associative, et propose ce qui deviendra une loi de fonctionnement pour l'apprentissage sur les réseaux de neurones connue plus tard sous le nom de loi de Hebb.
- 1943 : **J. McCullough et Walter. Pitts** laissent leurs noms à une modélisation du neurone biologique (*un neurone au comportement binaire*). Ceux sont les premiers à montrer que des réseaux de neurones formels simples peuvent réaliser des fonctions logiques, arithmétiques et symboliques complexes (*tout au moins au niveau théorique*) [2].
- En 1949, **D. Hebb** présente dans son ouvrage "The organization of behavior" une règle d'apprentissage. De nombreux modèles de réseaux aujourd'hui s'inspirent encore de la règle de Hebb.
- En 1958, **F. Rosenblatt** développe le modèle du Perceptron. C'est un réseau de neurones inspiré du système visuel. Il possède deux couches de neurones : une couche de perception et une couche liée à la prise de décision. C'est le premier système artificiel capable d'apprendre par expérience. Dans la même période, Le modèle de L'ADALINE a été présenté par B. Widrow, chercheur américain à Stanford. Ce modèle sera par la suite le modèle de base des réseaux multi-couches.
- En 1969, **M. Minsky et S. Papert** publient une critique des propriétés du Perceptron. Cela va avoir une grande incidence sur la recherche dans ce domaine. Elle va fortement diminuer jusqu'en 1972, où T. Kohonen présente ses travaux sur les mémoires associatives. Et propose des applications à la reconnaissance de formes [3].
- En 1982, **Hopfield** développe un modèle qui utilise des réseaux totalement connectés basés sur la règle de Hebb pour définir les notions d'attracteurs et de mémoire associative.
- En 1984 c'est la découverte des cartes de Kohonen avec un algorithme non supervisé basé sur l'auto-organisation et suivi une année plus tard par la machine de Boltzman (1985) [4]. Malheureusement, à l'époque, les réseaux de neurones étaient limités par les ressources techniques. Par exemple, les ordinateurs n'étaient pas assez puissants pour traiter les données nécessaires au fonctionnement des réseaux

de neurones. C'est la raison pour laquelle la recherche dans le domaine des Neural Networks est restée en sommeil durant de longues années [5].

- La nouvelle situation : Il aura fallu attendre le début des années 2010, avec l'essor du Big Data et du traitement massivement parallèle, pour que les Data Scientists disposent des données et de la puissance de calcul nécessaires pour exécuter des réseaux de neurones complexes.
- En 2012, lors d'une compétition organisée par ImageNet, un Neural Network est parvenu pour la première fois à surpasser un humain dans la reconnaissance d'image. C'est la raison pour laquelle cette technologie est de nouveau au cours des préoccupations des scientifiques. A présent, les réseaux de neurones artificiels ne cessent de s'améliorer et d'évoluer de jour en jour [5].

1.4 Modélisation d'un neurone

1.4.1 Le neurone réel (Anatomy) et neurone artificiel (Formel)

Les réseaux de neurones artificiels sont des modèles mathématiques inspirés de la biologie. Le neurone artificiel, est né initialement du désir de modéliser le fonctionnement d'un neurone biologique [6] (voir FIGURE 1.1 à la page 5) et (Voir table 1.1 à la page 6 fait référence au[7]).

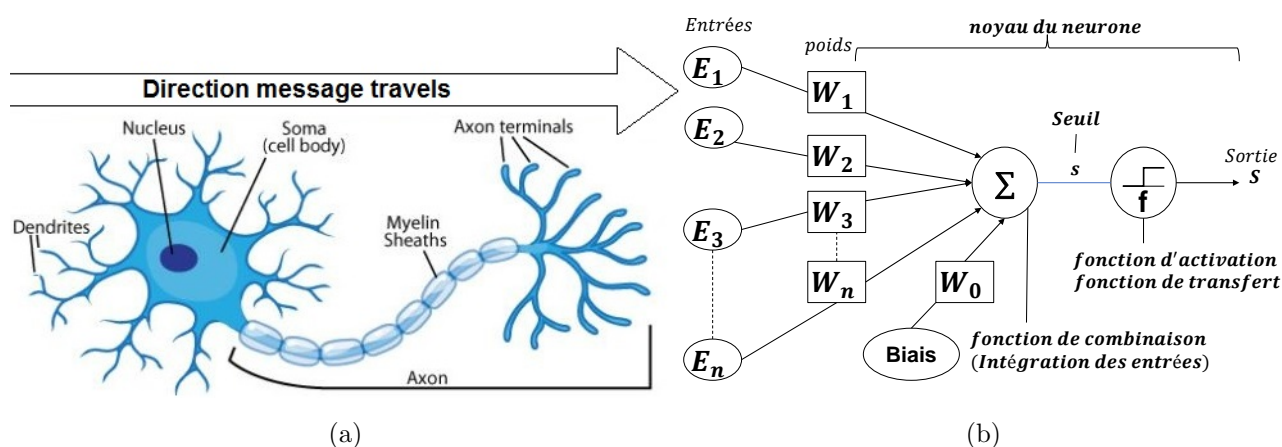


FIGURE 1.1 – (a) Neurone réel et (b) neurone artificiel [7].

Neurone réel (<i>Anatomy</i>)	Neurone artificiel (<i>Formel</i>)
Synapses : Envoyer des impulsions électriques aux neurones voisins.	Poids de la connexion : Facteurs multiplicateurs qui affectent l'influence de chaque entrée sur la sortie du neurone.
Axon : Transfère les signaux d'impulsion électrique du corps cellulaire à la synapse.	Sortie : Directement une des sorties du système ou peut être distribuée vers d'autres neurones.
Soma(<i>cellule</i>) : Le corps cellulaire qui contient la plupart des organites de la cellule	Neurone : Est une unité de calcul qui reçoit des informations, effectue des calculs simples avec celles-ci et les transmet ensuite.
Noyau : Contient l'ADN de la cellule	Noyau de neurone : Intègre toutes les entrées et le biais et calcule la sortie du neurone selon une fonction d'activation.
Dendrites : Reçoivent des impulsions électriques des neurones voisins.	Entrées : Directement les entrées du système ou peuvent provenir de d'autres neurones.

TABLE 1.1 – Neurone réel et neurone artificiel

1.4.2 Les éléments constitutifs d'un neurone artificiel

Les éléments qui constituent un neurone artificiel sont les suivants [7] :

- Les entrées "E" du neurone proviennent soit d'autres éléments "processeurs", soit de l'environnement.
- Les poids "W" indiquent l'influence de chaque entrée.
- Entrée de biais (*bias input*) : une unité fictive dont le poids permet d'ajuster le seuil de déclenchement du neurone.
- La fonction de combinaison "s" combine les entrées "E" et les poids "W" comme suite :

$$s = \sum_{i=1}^n W_i E_i \quad (1.1)$$

W_i : Poids de la connexion à l'entrée i .

E_i : Signal de l'entrée i .

— La Fonction de Transfert " f " détermine l'état du neurone en sortie " S ".

- Calcul de la sortie :

$$S = f(s) \quad (1.2)$$

- Ou encore :

$$S = f\left(\sum_{i=1}^n W_i E_i\right) \quad (1.3)$$

— La fonction de transfert " f " calcule la sortie " S " du neurone en fonction de la combinaison " s " en entrée et peut avoir plusieurs formes :

1. **Fonction en échelon (*fonction de Heaviside*)** est une fonction H discontinue prenant la valeur 0 pour tous les réels strictement négatifs et la valeur 1 partout ailleurs, (voir FIGURE 1.2 à la page 7).

$$\forall x \in R, H(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases} \quad (1.4)$$

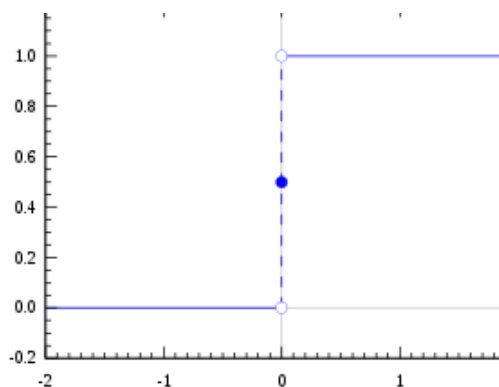


FIGURE 1.2 – Fonction de Heaviside[8].

2. **Fonction linéaire** (voir FIGURE 1.3 à la page 8).

$$y = s. \quad (1.5)$$

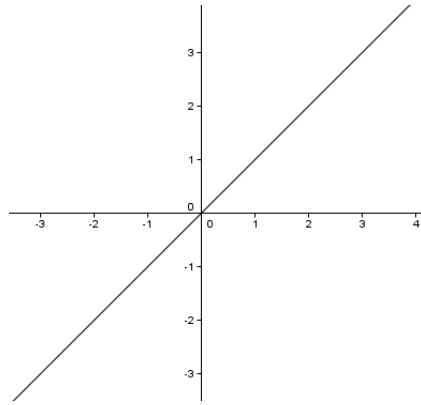


FIGURE 1.3 – Fonction linéaire [9].

3. **Fonction dérivable (*sigmoïde*)** (voir FIGURE 1.4 à la page 8).

$$\forall x \in \mathbb{R}, f(x) = \frac{1}{1 + e^{-x}}. \quad (1.6)$$

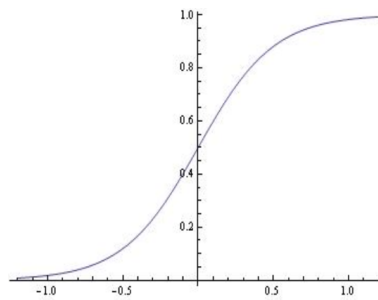


FIGURE 1.4 – Fonction sigmoïde[10].

4. **Fonction Tanh** est une fonction plus lisse centrée sur le zéro dont la plage est comprise entre -1 et 1, donc la sortie de la fonction de Tanh est donnée par l'équation(1.7) [11] (voir FIGURE 1.5 à la page 8)

$$\forall x \in \mathbb{R}, f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (1.7)$$

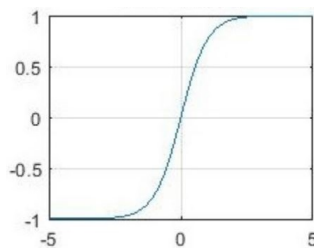


FIGURE 1.5 – Fonction Tanh[11].

5. **Fonction Softmax** utilisée pour calculer la distribution de probabilité à partir d'un vecteur de nombres réels. La fonction Softmax produit une sortie qui est une plage de valeurs entre 0 et 1, la somme des probabilités étant égale à 1. La fonction Softmax est calculée en utilisant la relation, voir l'équation(1.8) [11], (voir FIGURE 1.6 à la page 9).

$$\forall x \in R, f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}. \quad (1.8)$$

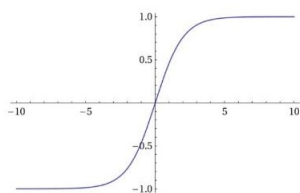


FIGURE 1.6 – Fonction softmax [12].

6. **Fonction Relu** est la fonction d'activation la plus utilisée pour les applications d'apprentissage profond. Elle offre les meilleures performances et la généralisation de l'apprentissage profond par rapport aux fonctions d'activation Sigmoid et Tanh. ReLU représente une fonction presque linéaire et préserve donc les propriétés des modèles linéaires qui les rendaient faciles à optimiser, avec des méthodes de gradient de descente. La fonction d'activation ReLU effectue une opération de seuil pour chaque élément d'entrée où les valeurs inférieures à zéro sont fixées à zéro donc le ReLU est donné par(1.9) [11], (voir FIGURE 1.7 à la page 9).

$$\forall x \in R, f(x) = \max(0, x) = \begin{cases} 0 & \text{si } x_i < 0 \\ x_i & \text{si } x \geq 0 \end{cases} \quad (1.9)$$

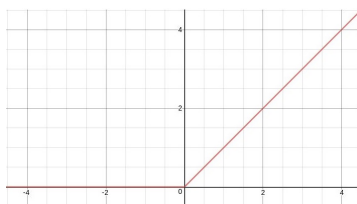


FIGURE 1.7 – Fonction ReLu[13].

1.5 Fonctionnement d'un réseau de neurones artificiel

Un réseau de neurones comprend quelques dizaines, centaines, milliers, voire millions de neurones artificiels appelés unités, disposés en une série de couches, chacune d'entre elles étant reliée aux couches de chaque côté. Certaines d'entre elles, appelées unités d'entrée, sont conçues pour recevoir du monde extérieur diverses formes d'informations que le réseau tentera d'apprendre, de reconnaître ou de traiter d'une autre manière. D'autres unités se trouvent de l'autre côté du réseau et signalent comment il réagit aux informations qu'il a apprises, ce sont les unités de sortie. Entre les unités d'entrée et les unités de sortie se trouvent une ou plusieurs couches d'unités cachées, qui ensemble, forment la majorité du cerveau artificiel. La plupart des réseaux de neurones sont entièrement connectés, ce qui signifie que chaque unité cachée et chaque unité de sortie est connectée à chaque unité des couches de part et d'autre. Les connexions entre une unité et une autre sont représentées par un nombre appelé poids, qui peut être soit positif ou négatif. Plus le poids est élevé, plus une unité a d'influence sur une autre [14] (voir FIGURE 1.8 à la page 10).

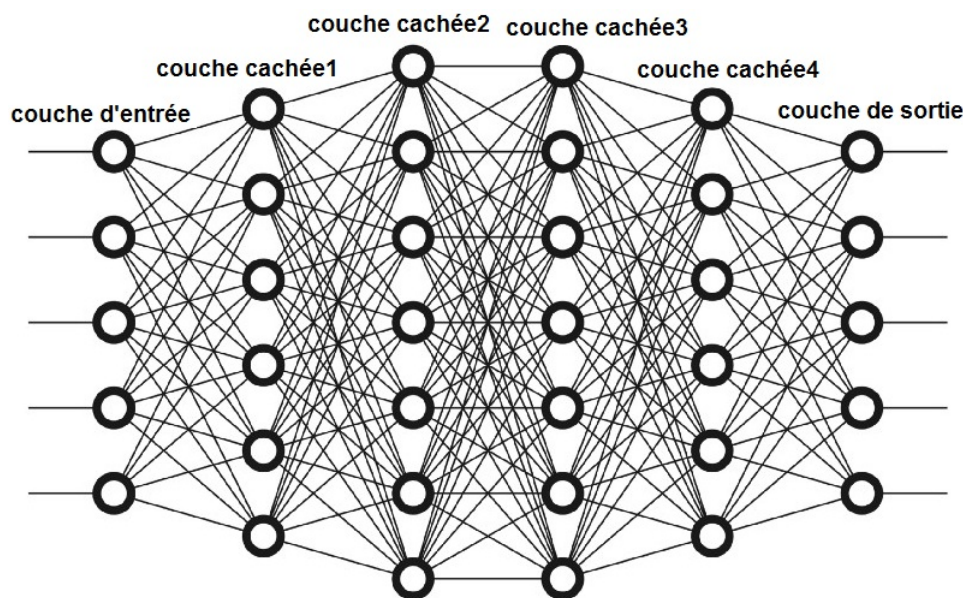


FIGURE 1.8 – Architecture d'un réseau profond à plusieurs couches[15].

1.6 Types de perceptron

Il existe deux types différents de perceptron, perceptron simple et perceptron multicouches.

1.6.1 Perceptron simple

Le perceptron à une seule couche est le premier modèle de neurones proposé créé. Le contenu de la mémoire locale du neurone se compose d'un vecteur de poids. Le calcul d'un perceptron d'une seule couche est effectué sur le calcul de la somme du vecteur d'entrée chacun avec la valeur multipliée par l'élément correspondant du vecteur des poids. La valeur qui est affichée dans la sortie sera l'entrée d'une fonction d'activation [16] (voir FIGURE 1.9 à la page 11).

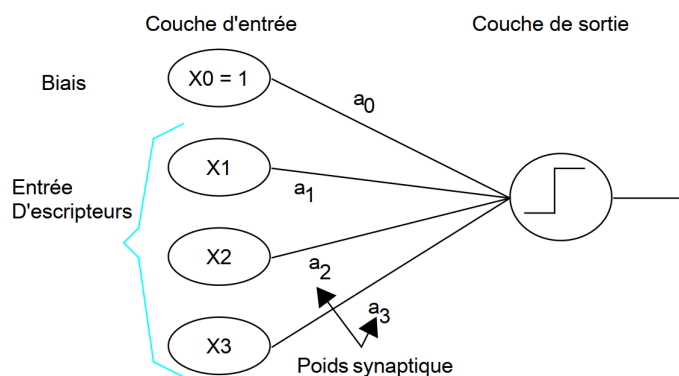


FIGURE 1.9 – Perceptron simple[17].

1.6.2 Perceptron multicouches (MLP)

Est un type de réseau de neurone Feed-Forward qui se caractérise par une couche d'entrée, un certain nombre de couches intermédiaires et une couche de sortie, qui sont entièrement connectées. Un MLP utilise back-propagation pour la formation. Le terme Feed-Forward se réfère à l'architecture en couches dans le réseau, en particulier qu'il n'y a pas de cycles dans le réseau. La structure de la couche garantit qu'il n'existe aucun cycle, car les couches ne sont autorisées à avoir des poids que de la couche directement précédente. Le terme entièrement connecté se réfère au fait que dans MLP, tous les nœuds dans une couche donnée ont un poids à tous les nœuds dans la couche précédente [19] (voir FIGURE 1.10 à la page 12).

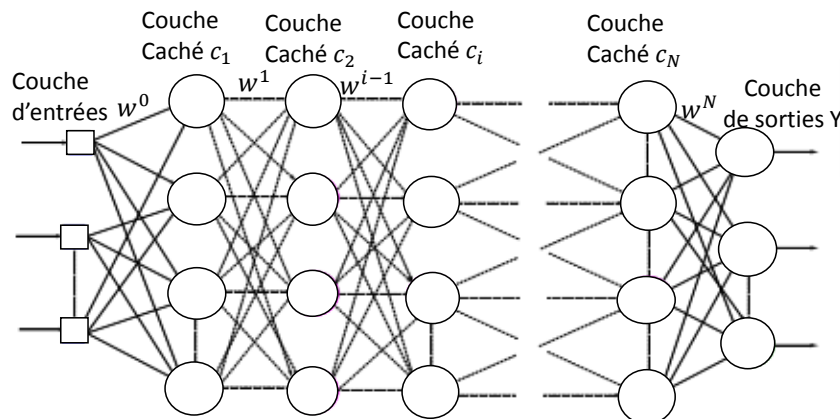


FIGURE 1.10 – Perceptron multi-couches[18].

1.7 Les différents types de réseaux de neurones artificiels

Les réseaux de neurones sont classés en catégories selon le nombre de couches qui distinguent l'entrée des données de la sortie du résultat, selon le nombre de nœuds cachés dans le modèle, ou le nombre d'entrées et de sorties de chaque nœud.

En fonction du type de réseau, la propagation de données entre les divers tiers des neurones peut différer. Il existe plusieurs types de réseaux de neurones [2] :

1.7.1 Réseau de neurones feed-forward

Dans ce type de réseaux, les informations passent directement de l'entrée aux nœuds de traitement puis aux sorties.

1.7.2 Les réseaux de neurones récurrents (RNN)

Sont des réseaux de neurones dans lesquels l'information peut se transmettre dans les deux sens, ils incluent des couches profondes aux premières couches. Ils sont plus proches du mécanisme réel de fonctionnement du système nerveux, qui n'est pas à sens unique [2].

1.7.3 Les réseaux de neurones convolutifs (CNN ou ConvNet)

Est un algorithme d'apprentissage profond qui peut prendre une image d'entrée, attribuer de l'importance (*poids et biais apprentissables*) à divers aspects/objets de l'image et être capable de les différencier les uns des autres. Le prétraitement requis dans un ConvNet est beaucoup moins important que pour les autres algorithmes de classification. Alors que dans les méthodes primitives, les filtres sont conçus à la main, avec une formation suffisante, les ConvNets ont la capacité d'apprendre ces filtres/caractéristiques [20].

1.8 Apprentissage des réseaux de neurones

Il existe quatre types d'apprentissages principaux :

1.8.1 Apprentissage supervisé (back-propagation)

L'algorithme d'apprentissage machine est fourni avec un exemple suffisamment important de données d'entrée et de sortie, généralement préparé en consultation avec l'expert d'un domaine respectif. L'objectif de l'algorithme est d'apprendre des modèles dans les données et de construire un ensemble général de règles pour mettre en correspondance les données d'entrée avec la classe ou l'événement [21] (voir FIGURE 1.11 à la page 13).

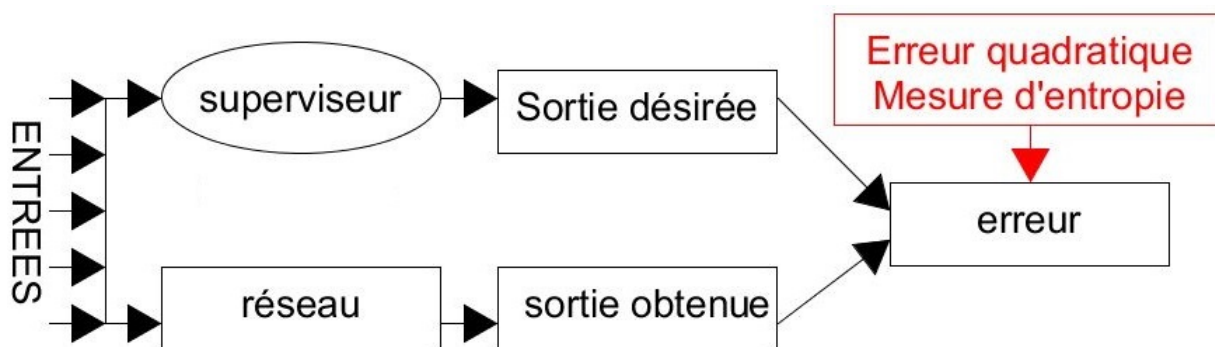


FIGURE 1.11 – Apprentissage supervisé [22].

1.8.2 Apprentissage non-supervisé

Il existe des situations où le résultat souhaité est inconnu pour les données historiques. L'objectif dans de tels cas serait d'étudier les modèles dans l'ensemble des données d'entrée pour mieux les comprendre et identifier des modèles similaires qui peuvent être

regroupés en classes ou événements spécifiques. Ces types d'algorithmes ne nécessitent aucune intervention préalable de la part des experts [21] (voir FIGURE 1.12 à la page 14).

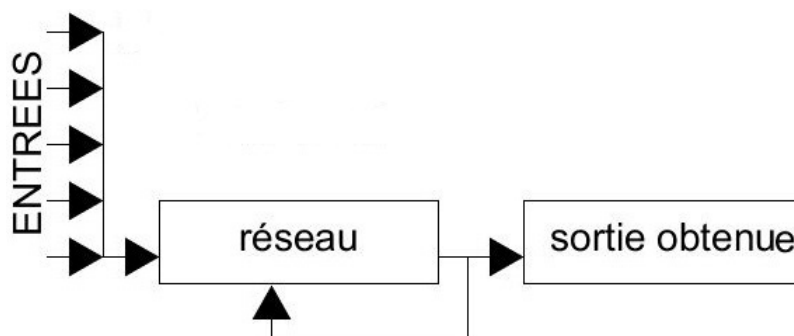


FIGURE 1.12 – Apprentissage non-supervisé[22].

1.8.3 Apprentissage semi-supervisé

L'apprentissage semi-supervisé est un type de technique d'apprentissage automatique. Il se situe entre l'apprentissage supervisé et non supervisé, c'est-à-dire que l'ensemble de données est partiellement étiqueté[23](voir FIGURE 1.13 à la page 14).

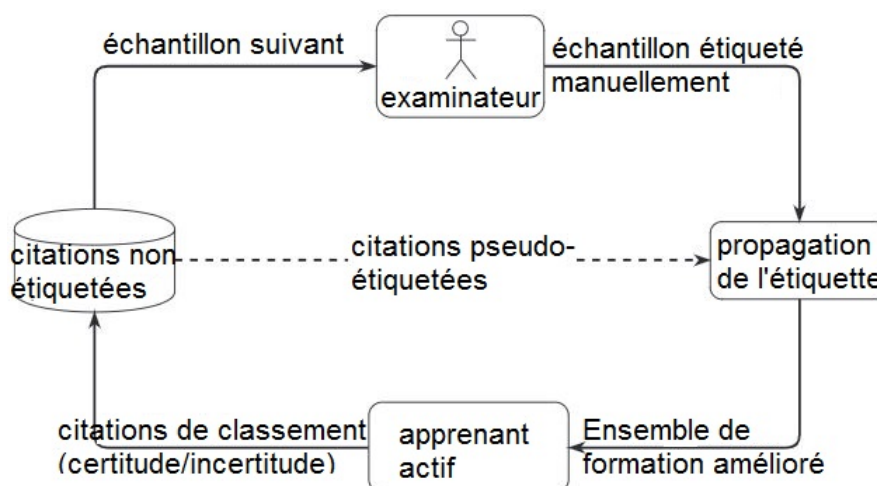


FIGURE 1.13 – Architecture de l'approche d'apprentissage actif semi-supervisé[24].

1.8.4 Apprentissage par renforcement

Le réseau de neurones est renforcé pour les résultats positifs et sanctionné pour les résultats négatifs. C'est ce qui lui permet d'apprendre au fil du temps, de la même manière

qu'un humain apprend progressivement de ses erreurs [5], (voir FIGURE 1.14 à la page 15).

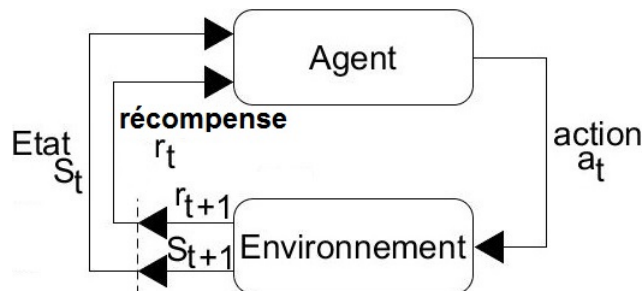


FIGURE 1.14 – Apprentissage par renforcement[25].

1.9 Règle d'apprentissage

La vitesse à laquelle les ANN apprennent dépend de plusieurs facteurs contrôlables (le temps, la complexité du réseau, sa taille, le choix du paradigme, l'architecture, le type de règle d'apprentissage, et la précision souhaitée). Ces facteurs jouent un rôle important dans la détermination du temps nécessaire pour former un réseau. La modification de l'un de ces facteurs peut soit allonger le temps de formation à une longueur déraisonnable, soit même aboutir à une précision inacceptable.

Il est évident qu'un rythme plus lent signifie que l'on passe beaucoup plus de temps à effectuer l'apprentissage pour produire un système correctement formé. Cependant, avec des taux d'apprentissage plus rapides, le réseau peut ne pas être en mesure de faire les discriminations fines possibles avec un système qui apprend plus lentement.

La plupart des fonctions d'apprentissage prévoient un certain taux d'apprentissage. Ce terme est généralement positif et se situe entre zéro et un. Si le taux d'apprentissage est supérieur à un, il est facile pour l'algorithme d'apprentissage de dépasser la correction des poids, et le réseau oscille. De petites valeurs du taux d'apprentissage ne corrigeront pas l'erreur actuelle aussi rapidement, mais si de petites mesures sont prises pour corriger les erreurs, il y a de bonnes chances d'arriver à la meilleure convergence minimale [26](voir FIGURE 1.15 à la page 16).

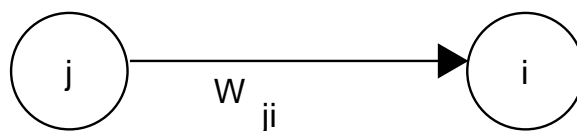


FIGURE 1.15 – Algorithm d'apprentissage[2].

1.9.1 Loi de Hebb

La première, et sans doute la plus connue, règle d'apprentissage a été introduite par Donald Hebb. La description est apparue dans son livre *The Organization of Behavior* en 1949. Sa règle de base est la suivante : Si un neurone reçoit une entrée d'un autre neurone, et si les deux sont très actifs (*ont mathématiquement le même signe*), le poids entre les neurones doit être renforcé [26].

$$\text{Loi de Hebb : } \Delta W_{ij} = R a_i a_j. \quad (1.10)$$

1.9.2 Loi de Hopfield

Elle est similaire à la règle de Hebb, à l'exception du fait qu'elle précise l'ampleur du renforcement ou de l'affaiblissement. Elle stipule que si la sortie et l'entrée souhaitées sont toutes deux actives ou toutes deux inactives, il faut augmenter le poids de connexion [26].

$$\text{Loi de Hopfield : } \Delta W_{ij} = R a_i a_j. \quad (1.11)$$

1.9.3 Loi Delta rule (Règle de Widrow-Hoff)

Cette règle est une autre variante de la règle de Hebb. Basée sur l'idée simple en modifiant continuellement les forces des connexions (*poids synaptiques*) d'entrée pour réduire la différence (*le delta*) entre la valeur de sortie souhaitée et la sortie réelle d'un élément de traitement [26].

$$\text{Widrow - Hoff : } \Delta W_{ij} = R a_i (a_j - W_{ij}). \quad (1.12)$$

1.10 Les avantages et les inconvénients des réseaux de neurones

1.10.1 Avantages

Sont les suivants [27] :

- Le parallélisme de leur structure, leur adaptabilité ainsi que dans leur mémoire distribuée.
- Les réseaux de neurones sont en réalité des "Approximateurs universels". Leur utilisation permet de passer directement des données au prédicteur, sans intervention, sans recodage, sans discrétisation, sans simplification ou interprétation douteuse.
- Un réseau de neurones possède également une grande résistance au bruit ou au manque de fiabilité des données.
- L'idée d'apprentissage est plus simple à comprendre.
- Les réseaux de neurones sont capables d'analyser des relations spatiales et topologiques.
- Les régressions se font sur des fonctions de dépendance simples (*linéaire, logarithmique*) qui ne sont pas toujours très réalistes.
- Un réseau de neurones bien conçu est capable de présenter n'importe quelle dépendance fonctionnelle et d'extraire des données sans modèle préconçu.

1.10.2 Inconvénients

Sont les suivants [27] :

- Un réseau de neurones ne dispense pas de bien connaître son problème, de définir ses classes avec pertinence, de ne pas oublier de variables importantes, ...
- Un réseau de neurones est une "boîte noire" qui ne justifie pas ses décisions.
- Les réseaux de neurones ont une très forte capacité de prédiction statistique, mais ils sont totalement impossibles à inspecter.

1.11 Conclusion

Les réseaux de neurones sont des outils très puissants qui sont très utilisés dans différents domaines de l'intelligence artificielle et qui offrent de bons résultats.

Dans ce chapitre nous avons présenté les réseaux de neurones, leurs différents types, leurs modes de fonctionnement ainsi que les méthodes d'apprentissage existantes. Dans le chapitre suivant, nous allons aborder les différents travaux qui se sont intéressés à l'utilisation des réseaux de neurones pour faire apprendre à une machine à jouer à des jeux de plateau d'une façon intelligente.

Les jeux de plateau

2.1 Introduction

Les progrès de l'intelligence artificielle ne cessent d'augmenter et de battre, les uns après les autres, tous les jeux que les humains ont parfois mis des années à bien maîtriser. Pourtant, le lien entre intelligence artificielle et jeu ne date pas d'hier. En 1979 déjà, le robot "Gammanoid" battait Luigi Villa, un champion de backgammon, à Monte-Carlo[28].

Dans ce chapitre nous allons présenter les différents jeux de plateaux qui utilisent les réseaux de neurones avec les méthodes qu'ils appliquent.

2.2 Travaux connexes

Les réseaux de neurones sont utilisés dans plusieurs domaines tels que le domaine scientifique, le secteur industriel, . . .

Dans notre travail, nous nous intéressons aux jeux de plateaux tels que le Yokai No Mori, jeu d'échecs, Shogi et jeu de Go.

2.3 Jeu de Yokai No Mori

Est un jeu de plateau créé en Octobre 2013. Inspiré du Shogi (*jeu traditionnel japonais*) Yokai no mori est un jeu aux règles simples, mais d'une grande richesse tactique et stratégique [29] (voir FIGURE 2.1 à la page 20).



FIGURE 2.1 – Jeu de Yokai No Mori[30].

2.3.1 Version 3 * 4 (version classique)

- Elle se joue sur un plateau de 3 cases sur 4.
- Chaque joueur n'a que quatre pièces : koropokkuru, tanuki, kitsune et kodama.
- **Le koropokkuru** : en gros l'équivalent du roi, sauf qu'il peut gagner par promotion tant que cela ne le met pas en échec.
- **Le Tanuki** : se déplace d'une case orthogonalement.
- **Le kitsune** : se déplace en diagonal en avant et en arrière.
- **Le kodama** : ne se déplace que vers l'avant, peut être promu au rang de samouraï de kodama [31].

Le but du jeu

- Soit d'attraper le koropokkuru de l'adversaire.
- Soit de déplacer son propre koropokkuru sur le côté opposé du plateau de jeu [31].

Règles du jeu

- Les pièces ne peuvent se déplacer que d'une case à la fois.
- Les mouvements possibles sont indiqués par les points situés sur les pièces.
- Replacer une pièce capturée précédemment n'importe où sur le plateau.
- Promouvoir le kodama : si on arrive à déplacer le kodama sur la dernière rangée du plateau (*la zone de contrôle adverse*) il est promu. Ce Kodama se transforme alors en Kodama samouraï avec de nouvelles options de mouvement [31].

2.3.2 Version 5 * 6

La version expert a un plateau de jeu 5 * 6 plus grand, chaque joueur ayant huit pièces (*3 kodama, 2 oni, 2 kirin et 1 koropokkuru*).

- **Le kirin** : peut se déplacer dans tout les sens sauf dans les diagonales arrières.
- **L’oni** : qui peut être lui aussi promu au super oni afin de modifier son mode de déplacement [31].

Règles du jeu

- Vous ne gagnez qu’en capturant le koropokkuru de votre adversaire, et non en atteignant le côté opposé du plateau de jeu.
- Vous ne pouvez pas avoir deux de vos propres kodama dans la même colonne.
- la zone de promotion correspond aux deux dernières rangées (*cette zone est délimitée par une ligne orange sur le plateau*).
- Vous ne pouvez pas laisser tomber un kodama pour le mat [31].

2.3.3 Type de réseau de neurones utilisé

Dans ce jeux les chercheurs ont utilisé les réseaux de neurones a convolution.

2.3.4 Description de ce réseau de neurones

La FIGURE 2.2 à la page 22 montre la structure du réseau de neurones utilisée [32].

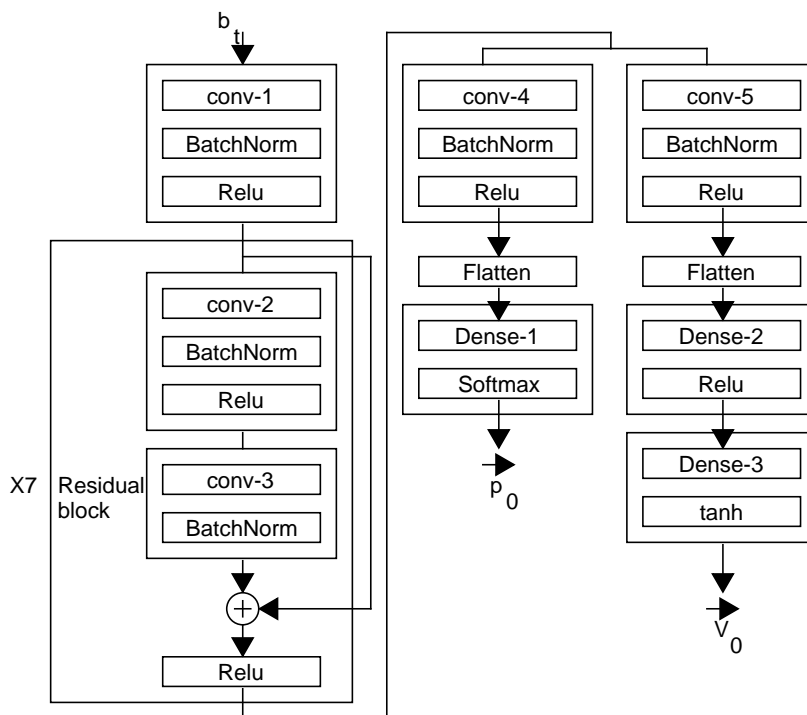


FIGURE 2.2 – La structure du réseau de neurones[32].

— **conv-1**

Couche de convolution (*masques de convolution de taille $5 * 5$, output : 256 matrices*) avec normalisation par batch et une fonction d'activation ReLU.

— **conv-2 et conv-3**

Un bloc résiduel composé de deux couches de convolution (*pour chaque couche : masques de convolution de $3 * 3$, output : 256 matrices*) avec, dans les deux cas, une normalisation par batch et une fonction d'activation ReLU. Ce bloc résiduel étant répété 7 fois on a donc 14 couches.

Le réseau se sépare ensuite en deux parties : une première partie qui aura en sortie le vecteur \vec{p}_0 et une seconde partie qui aura en sortie V_0 .

— **conv-4** (partie " \vec{p}_0 ")

Couche de convolution (*masques de convolution de taille $1 * 1$, output : 2 matrices*) avec normalisation par batch et une fonction d'activation ReLU.

— **conv-5** (partie " V_0 ")

Couche de convolution (*masques de convolution de taille $1 * 1$, output : 4 matrices*) avec normalisation par batch et une fonction d'activation ReLU.

- **Flatten** Ces deux couches permettent de passer d'une matrice à un vecteur (*en "aplatissant" la matrice*).
- **Dense-1** (partie " \vec{p}_0 ") Couche complètement connectée avec une fonction d'activation softmax. En sortie de cette couche on trouve un vecteur de taille 358 pour la version 5 * 6 du jeu et 130 pour la version 3 * 4 (*dans les deux cas cela correspond au nombre de coups possibles*).
- **Dense-2** (partie " V_0 ") Couche complètement connectée avec une fonction d'activation ReLU. En sortie de cette couche on trouve un vecteur de taille 256.
- **Dense-3** (partie " V_0 ") Couche complètement connectée avec une fonction d'activation ReLU. En sortie de cette couche on trouve un scalaire (V_0). Nous avons donc en tout 17 couches pour la partie " \vec{p}_0 " et 18 couches pour la partie " V_0 " [32]

2.3.5 Résultats obtenus

Dans la **version 3 * 4** Après une centaine d'itérations, il est difficile, pour un joueur moyen, de battre le programme. En revanche, si le joueur évite les erreurs grossières, le programme aura aussi énormément de mal à gagner. Ils ont donc souvent des parties qui s'éternisent. À noter qu'au-delà d'une centaine d'itérations, ils constatent une stagnation des progrès, même si cette notion de progrès reste très subjective vues les conditions de l'expérimentation.

Dans la **version 5 * 6** Ici aussi l'expérimentation porte sur une version de programme qui a effectué une centaine d'itérations. Les résultats obtenus sont moins positifs que pour la version 3 * 4. En effet, un joueur moyen gagne facilement ses matchs. Au cours d'une partie, le programme finit souvent par commettre une erreur qui lui sera fatale quelques coups plus tard. Il est assez simple de prendre les pièces de programme, elle a en effet la fâcheuse tendance à ne pas les protéger (comportement qui est plus difficile à mettre en évidence dans la version 3 * 4 vue la taille réduite du plateau de jeu). Afin d'améliorer la situation, il a été tenté d'augmenter le nombre d'itérations, mais comme dans le cas de la version 3 * 4, le sentiment que le programme ne progresse plus s'impose assez rapidement [33].

2.4 Le jeu de Go

Le jeu de Go (voir FIGURE 2.3 à la page 24) a longtemps été considéré comme le plus difficile des jeux classiques pour l'intelligence artificielle en raison de son énorme espace de recherche et de la difficulté d'évaluer les positions et les mouvements du plateau. Ils ont présenté ici une nouvelle approche du Go sur ordinateur qui utilise des "réseaux de valeurs" pour évaluer les positions du plateau et des "réseaux de politiques" pour sélectionner les coups. Ces réseaux de neurones profonds sont formés par une nouvelle combinaison d'apprentissage supervisé à partir de jeux d'experts humains et d'apprentissage de renforcement à partir de jeux d'auto-jeu. Sans aucune recherche d'anticipation, les réseaux de neurones jouent au Go au niveau des programmes de recherche par arbre de Monte Carlo de pointe qui simulent des milliers de jeux aléatoires d'auto-jeu. Ils ont introduit également un nouvel algorithme de recherche qui combine la simulation de Monte Carlo avec les réseaux de valeurs et de politiques.

Grâce à cet algorithme de recherche, le programme AlphaGo a obtenu un taux de réussite de 99,8% face à d'autres programmes de go et a battu le champion européen de go humain par 5 parties à 0.

C'est la première fois qu'un programme informatique bat un joueur professionnel humain dans une partie de go, un exploit que l'on croyait impossible avant au moins dix ans[35].



FIGURE 2.3 – Jeu de Go[34].

2.4.1 Matériels utilisés

Le matériel utilisés dans ce jeu est [38] :

- Le goban est le plateau sur lequel va se dérouler la partie. Une grille de $19 * 19$ ou $13 * 13$ ou bien $9 * 9$.
- Les pierres du jeu sont de couleur noires ou blanches, et se trouvent généralement dans un bol. La coupole peut servir à garder les prisonniers capturés.

2.4.2 Règles du jeu

Sont présenté comme suit [39] :

- Le jeu commence avec un plateau vide, sauf si les joueurs acceptent de placer un handicap.
- Le noir doit toujours faire le premier pas vers le coin supérieur droit.
- Le blanc fait le deuxième mouvement vers le coin inférieur droit.
- Un joueur effectue un mouvement lorsqu'il place une pierre sur une intersection libre du plateau.
- Les joueurs sont autorisés à passer leur tour.
- Deux passes successives mettent fin au jeu.
- Le joueur qui occupe la zone la plus étendue gagne.
- La zone du joueur comprend tous les points qu'il a encerclés.
- Une pierre doit être enlevée lorsque l'ennemi occupe des intersections adjacentes.

2.4.3 Discussion

Dans ce travail, ils ont développé un programme Go, basé sur une combinaison de réseaux de neurones profonds et de recherche par arbre, qui joue au niveau des joueurs humains les plus forts, réalisant ainsi l'un des "grands défis" de l'intelligence artificielle. Ils ont développé, pour la première fois, des fonctions efficaces de sélection des mouvements et d'évaluation des positions pour le Go, basées sur des réseaux de neurones profonds qui sont formés par une combinaison inédite de l'apprentissage supervisé et l'apprentissage par renforcement. Ils ont introduit un nouvel algorithme de recherche qui combine avec succès les évaluations des réseaux de neurones avec les déploiements de Monte Carlo. Le programme AlphaGo intègre ces composants ensemble, à l'échelle, dans un moteur de

recherche arborescent très performant. Lors du match contre Fan Hui, AlphaGo a évalué des milliers de fois moins de positions que Deep Blue lors de son match d'échecs contre Kasparov, il a compensé en sélectionnant ces positions de manière plus intelligente, en utilisant le réseau de politiques, et en les évaluant de manière plus précise, en utilisant le réseau de valeurs - une approche qui est peut-être plus proche de la manière dont les humains jouent. En outre, alors que Deep Blue s'appuyait sur une fonction d'évaluation artisanale, les réseaux de neurones d'AlphaGo sont formés directement à partir du jeu, uniquement par des méthodes d'apprentissage supervisées et de renforcement à usage général.

Le go est exemplaire à bien des égards des difficultés rencontrées par l'intelligence artificielle : une tâche décisionnelle difficile, un espace de recherche intraitable, et une solution optimale si complexe qu'il semble impossible de s'en approcher directement en utilisant une fonction de politique ou de valeur.

En combinant la recherche par arbre avec les réseaux de politiques et de valeurs, AlphaGo a enfin atteint un niveau professionnel dans le domaine du Go [35].

2.4.4 Type de réseau de neurones utilisé et type d'apprentissage

Ils ont utilisé les réseaux de neurones à convolution avec un apprentissage supervisé et apprentissage par renforcement [35].

2.4.5 L'architecteur de ce réseau de neurones

L'entrée du réseau politique est une pile d'images de $19 * 19 * 48$ composée de 48 plans de caractéristiques.

La première couche cachée zéro transforme l'entrée en une image $23 * 23$, puis convolue k filtres de taille $5 * 5$ avec un pas de 1 avec l'image d'entrée et applique une non-linéarité de redressement.

Chacune des couches cachées suivantes (2 à 12 zéros) transforme la couche cachée précédente respective en une image $21 * 21$, puis convolue k filtres de taille de noyau $3 * 3$ avec un pas de 1, à nouveau suivi d'une non-linéarité de redressement.

La dernière couche convolue 1 filtre de taille de noyau $1 * 1$ avec un pas de 1, avec un biais différent pour chaque position, et applique une fonction softmax.

La version de correspondance d'AlphaGo utilisée $k = 192$ filtres montre en outre les résultats de l'entraînement avec $k = 128, 256$ et 384 filtres [35].

L'entrée du réseau de valeurs est également une pile d'images $19 * 19 * 48$, avec un plan de caractéristiques binaires supplémentaire décrivant la couleur actuelle à jouer.

Les couches cachées 2 à 11 sont identiques au réseau de valeurs, la couche cachée 12 est une couche de convolution supplémentaire, la couche cachée 13 convolue 1 filtre de taille de noyau $1 * 1$ avec un pas de 1, et la couche cachée 14 est une couche linéaire entièrement connectée avec 256 unités de redressement.

La couche de sortie est une couche linéaire entièrement connectée avec une seule unité de tanh [35].

2.5 Chess et Shogi

Le jeu d'échecs est le domaine le plus étudié dans l'histoire de l'intelligence artificielle et l'étude des échecs informatiques est aussi ancienne que l'informatique elle-même [40] (voir FIGURE 2.4 à la page 27).



FIGURE 2.4 – (a) Jeu de Chess[36] et (b) Jeu de Shogi[37]

2.5.1 Travaux sur les échecs informatiques et Shogi

Dans cette partie, nous discutons quelques travaux antérieurs sur l'apprentissage par renforcement dans les échecs informatiques [40].

Kaneko and Hoki

A entraîné les poids d'une fonction d'évaluation de shogi comprenant un million de caractéristiques, en apprenant à sélectionner des mouvements humains experts lors de la recherche alpha-bêta. Ils ont également formé une optimisation à grande échelle basée sur une recherche minimax régulée par des journaux de jeu experts, cela a fait partie du moteur Bonanza qui a remporté le championnat du monde de shogi informatique en 2013.

Giraffe

A évalué les positions par un réseau de neurones qui comprenait des cartes de mobilité et des cartes d'attaque et de défense décrivant l'attaquant et le défenseur de chaque carré ayant la plus faible valeur. Il a été formé par auto-jeu en utilisant le TD (*feuille*), atteignant également un niveau de jeu comparable à celui des maîtres internationaux.

DeepChess

A formé un réseau de neurones pour effectuer des évaluations de postes par paires. Il a été formé par un apprentissage supervisé à partir d'une base de données de parties humaines expertes qui a été pré-filtrée pour éviter les coups de capture et les parties nulles. DeepChess a atteint un niveau de jeu de grand maître.

Tous ces programmes ont combiné leurs fonctions d'évaluation apprises avec une recherche alpha-bêta améliorée par diverses extensions.

AlphaZero

AlphaZero évalue les positions en utilisant une approximation de fonction non linéaire basée sur un réseau de neurones profond, plutôt que l'approximation de fonction linéaire utilisée dans les programmes d'échecs typiques, ce qui donne une représentation beaucoup plus puissante, mais peut également introduire de fausses erreurs d'approximation. Les MCTS font la moyenne de ces erreurs d'approximation, qui ont donc tendance à s'annuler lors de l'évaluation d'un grand sous-arbre. En revanche, la recherche alpha-bêta calcule un minimax explicite, qui propage les plus grandes erreurs d'approximation à la racine du sous-arbre. L'utilisation des MCTS peut permettre à AlphaZero de combiner efficacement ses représentations de réseau de neurones avec une recherche puissante et indépendante du domaine.

2.6 AlphaZero

2.6.1 Les connaissances de domaine utilisées par AlphaZero

- Les caractéristiques d'entrée décrivant la position et les caractéristiques de sortie décrivant le mouvement sont structurées comme un ensemble de plans, c'est-à-dire que l'architecture du réseau de neurones est adaptée à la structure en grille de plateau [40].
- AlphaZero est fourni avec une parfaite connaissance des règles du jeu. Celles-ci sont utilisées pendant les MCTS, pour simuler les positions résultant d'une séquence de mouvements, pour déterminer la fin du jeu et pour noter toutes les simulations qui atteignent un état terminal [40].
- La connaissance des règles est également utilisée pour coder les plans d'entrée (roque, répétition, No-progress) et les plans de sortie (comment les pièces se déplacent, les promotions et les chutes de pièces dans le shogi) [40].
- Le nombre typique de mouvements légaux est utilisé pour évaluer le bruit de l'exploration.
- Les parties d'échecs et de shogi dépassant un nombre maximum de pas (déterminé par la durée typique de la partie) ont été interrompues et se sont soldées par un résultat nul, les parties de go ont été interrompues et ont été comptabilisées selon les règles de Tromp-Taylor [40].

2.6.2 Fonctionnalités d'entrée utilisées par AlphaZero respectivement dans Go, Chess et Shogi

Représentation de tableau d'entrées et la représentation de sorties d'action utilisé par le réseau de neurones dans AlphaZero[40] (voir Table 2.1 à la page 30 fait référence au[40]).

GO		Echecs (jeu d'échecs)		Shogi	
Caractéristique	Plans	Caractéristique	Plans	Caractéristique	Plans
P1 Pierres	1	P1 pièce	6	P1 pièce	14
P2 Pierres	1	P2 pièce	6	P2 pièce	14
		Repetitions	2	Repetitions	3
				P1 prisoner count	7
				P1 prisoner count	7
Colour	1	Colour	1	Colour	1
		Total move count	1	Total move count	1
		P1 Roque	2		
		P2 Roque	2		
		No-progress count	1		
Total	17	Total	119	Total	362

TABLE 2.1 – Fonctionnalités d'entrée utilisées par AlphaZero respectivement dans Go, Chess et Shogi

L'entrée du réseau de neurones est une pile d'images $N * N * (MT + L)$ qui représente l'état en utilisant une concaténation de T ensembles de M plans de taille $N * N$. Chaque ensemble de plans représente la position du plateau à un pas de temps $t - T + 1, \dots, t$, et est mis à zéro pour les pas de temps inférieurs à 1. Le plateau est orienté selon la perspective du joueur actuel. Les plans de caractéristiques M sont composés de plans de caractéristiques binaires indiquant la présence des pièces du joueur, avec un plan pour chaque type de pièce, et un second ensemble de plans indiquant la présence des pièces de l'adversaire. Pour le shogi, il existe des plans supplémentaires indiquant le nombre de prisonniers capturés de chaque type. Il y a un plan d'entrée supplémentaire à valeur constante L indiquant la couleur du joueur, le nombre total de coups et l'état des règles spéciales (voir Table 2.2 à la page 31 fait référence au [40]). Un mouvement dans les échecs peut être décrit en deux parties :

- sélection de la pièce pour déplacer.
- et puis la sélection parmi les mouvements pour cette pièce.

Echecs (jeu d'échecs)		Shogi	
Caractéristique	Plans	Caractéristique	Plans
Mouvement de Dame	56	Mouvement de Dame	64
Mouvement Cavalier	8	Mouvement Cavalier	2
Sous promotions	9	Promotions de mouvement de Dame	64
		Promotions de mouvement de Dame	2
		Drop	7
Total	73	Total	139

TABLE 2.2 – Représentation d'action utilisée par AlphaZero dans Chess et Shogi respectivement

Ils représentent la politique $\pi(a|s)$ par une $8*8*73$ pile de plans encodant une distribution de probabilité sur 4,672 mouvements possibles. Chacune des positions $8 * 8$ identifie la case d'où l'on peut "prendre" une pièce.

Les 56 premiers plans encodent possible "Mouvement de Dame" pour n'importe quelle pièce. Les 8 prochains plans encodent les mouvements de "Cavalier" possibles pour cette pièce. Les 9 derniers plans encodent les sous-promotions possibles pour les mouvements de "Pion" ou les captures dans deux diagonales possibles, à "Cavalier", "Fou" or "Tour". D'autres mouvements de "Pion" ou captures du septième rangée sont promus à une "Dame".

2.6.3 L'application de l'algorithme AlphaZero aux échecs, shogi

Aux échecs, AlphaZero a surclassé Stockfish (2016 TCEC world-champion program) après seulement 4 heures (300k étapes). En shogi, AlphaZero a surpassé Elmo(2017 CSA world-champion program) après moins de 2 heures (110k étapes). AlphaZero vaincu de manière convaincante tous les adversaires, perdre zéro jeux à Stockfish et huit jeux à Elmo. AlphaZero recherche juste 80 000 positions par seconde aux échecs et 40 000 en shogi, contre 70 millions pour Stockfish et 35 millions pour Elmo [40].

2.7 Le jeu de Morpion

2.7.1 Présentation du jeu

Le jeu de morpion, aussi appelé "Tic-tac-toe", est un jeu classique de stratégie et de réflexion qui se joue à deux au tour par tour ou contre l'ordinateur. Il existe plusieurs variantes, grille de $3 * 3$ (voir FIGURE 2.5 à la page 32), grille de $5 * 5$ ou grille de $7 * 7$.



FIGURE 2.5 – Le jeu de Morpion[41].

2.7.2 Matériel du jeu

- Une grille de 9 cases $3 * 3$.
- 5 pions rond "O" et 5 pions croix "X".

2.7.3 But du jeu

Réaliser le premier un alignement avec 3 pions en horizontale, en verticale ou en diagonale.

2.7.4 Règles du jeu

- Chaque joueur, à son tour, pose un pion sur une case vide.
- Un pion joué ne peut être ni repris, ni déplacé.
- La partie est gagnée par le joueur qui parvient le premier à placer 3 de ses pions sur une même ligne horizontale, verticale ou diagonale sur les grilles $3 * 3$ (alors qu'il en faut 4 sur les grilles $5 * 5$ et $7 * 7$).

2.8 Conclusion

Dans ce chapitre nous avons présenté les différents jeux de plateaux qui ont utilisés les réseaux de neurones en précisant leurs types, le type d'apprentissage et les résultats obtenus pour chaque jeu.

Dans le chapitre suivant on va détailler notre proposition, nous expliquerons les différentes étapes nécessaires pour créer et entrainer notre modèle et nous présenterons quelques résultats expérimentaux et l'implémentation de notre modèle.

Proposition et implémentation

3.1 Introduction

Dans le chapitre précédent nous avons vu que les réseaux de neurones sont des outils très puissants qui peuvent être utilisés pour faire apprendre à une machine à jouer à des jeux de réflexion avec un niveau acceptable et des fois même à un haut niveau.

Dans ce chapitre nous allons présenter notre proposition et le réseau de neurones que nous avons proposé pour le jeu de morpion, son implémentation et quelques résultats expérimentaux concernant l'évaluation de notre modèle.

3.2 Notre problématique

- Créer une approche en utilisant les réseaux de neurones profonds qui pourra apprendre à jouer au morpion toute seule avec un niveau acceptable.
- Le programme doit apprendre en jouant un grand nombre de parties du jeu.

3.3 Solution proposée

3.3.1 L'architecture de notre réseau de neurones

Notre réseau de neurones est constitué de neuf neurones en entrée, 4 couches cachées et une couche de sortie avec 3 neurones voir la FIGURE 3.1 à la page 35.

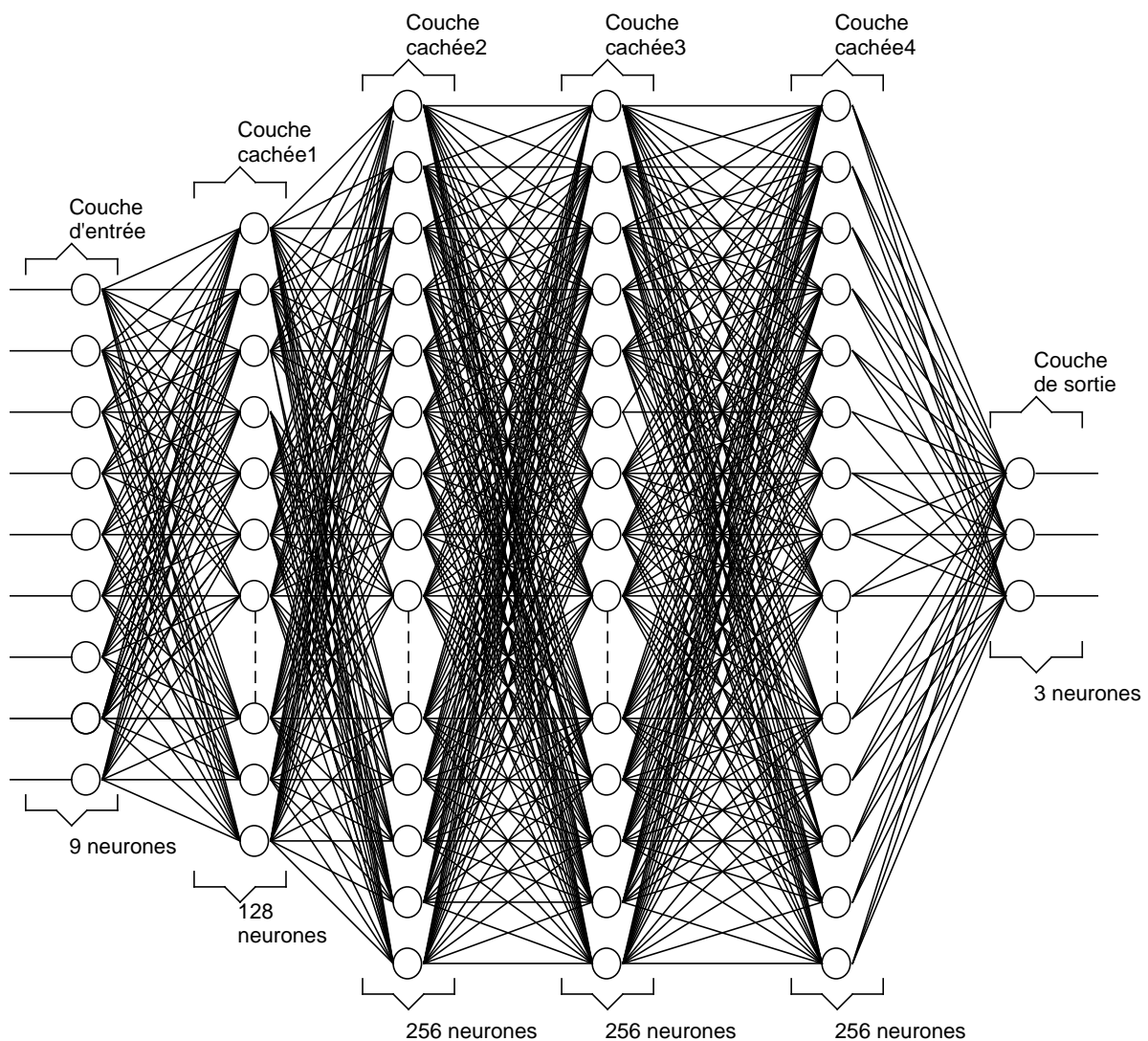


FIGURE 3.1 – L'architecture du réseau de neurones.

3.3.2 Type du réseau de neurones

Notre approche est basée sur les réseaux de neurones profonds.

3.3.3 Type d'apprentissage

Le type d'apprentissage utilisé par notre réseau de neurones est un apprentissage supervisé, pour cela, nous avons utilisé un jeu de données étiqueté que nous avons réalisé.

3.3.4 Étape d'entraînement

Les différentes étapes d'entraînement de notre réseau de neurones est montré sur la FIGURE 3.2 page 36) :

1. **Dataset** : le nombre de parties jouées (10000 partie), chaque partie constitue de 9 état.
2. **Réseau de neurones**
3. **Résultats d'apprentissage**

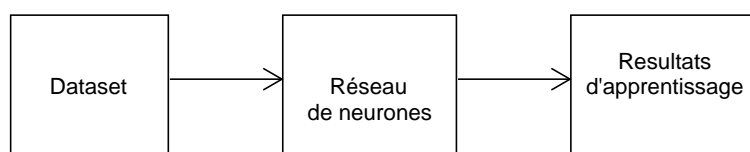


FIGURE 3.2 – Entraînement.

- **Dataset** : Nous avons créé une classe 'Game' pour stocker les informations et traiter le déroulement du jeu de manière aléatoire. Notre dataset comporte 10000 partie de jeu , chaque état d'une partie est stocké sous forme d'un tableau de 9 cellules. Une cellule est marquée par 0 si elle n'a pas été choisie par un joueur, -1 pour le joueur avec un X et 1 pour le joueur avec un O (voir la FIGURE 3.3 page 37).
- L'entraînement de notre réseau de neurones est effectué en jouant plusieurs parties du jeu.
- Au cours de la première partie, le réseau de neurones "Modèle" est initialisé avec des poids aléatoires.
- Après que le programme a joué la première partie, les probabilités produites par le réseau de neurones sont corrigés comme suit :
 - * Si la partie a été gagnée par le programme, attribuez au dernier plateau une probabilité de 1, de même qu'une probabilité de -1 si la partie a été perdue et de 0 si la partie a été nulle.
 - * Pour chacun des états précédents du plateau, attribuez la probabilité qui a été prévu à l'état suivant du plateau.


```
In [25]: df = pd.DataFrame(data=game.HistorieEntrainement)

In [26]: print(df)

   0      1
0  -1  [[0, 0, 0], [1, 0, 0], [0, 0, 0]]
1  -1  [[-1, 0, 0], [1, 0, 0], [0, 0, 0]]
2  -1  [[-1, 0, 0], [1, 0, 0], [0, 0, 1]]
3  -1  [[-1, -1, 0], [1, 0, 0], [0, 0, 1]]
4  -1  [[-1, -1, 0], [1, 1, 0], [0, 0, 1]]
... ..
89995 1  [[1, 1, -1], [0, 1, 0], [0, 0, -1]]
89996 1  [[1, 1, -1], [0, 1, 0], [-1, 0, -1]]
89997 1  [[1, 1, -1], [0, 1, 1], [-1, 0, -1]]
89998 1  [[1, 1, -1], [-1, 1, 1], [-1, 0, -1]]
89999 1  [[1, 1, -1], [-1, 1, 1], [-1, 1, -1]]

[90000 rows x 2 columns]
```

FIGURE 3.3 – La Dataset.

- Mettre à jour les poids du réseau de neurones en l’ajustant aux états du plateau et aux probabilités corrigés.
- Répétez le processus ci-dessus encore et encore, sur un grand nombre de jeux, en actualisant le réseau de neurones après chaque jeu.
- La logique de l’approche est que, pour un joueur, si l’état final du plateau était un état gagnant, alors les états du plateau menant à l’état final seraient également favorables.
- Comme le processus de mise à jour des poids se répète au fil des parties, le réseau de neurones apprend à attribuer une probabilité plus élevée aux états favorables du plateau, et des probabilités plus faibles aux états défavorables du plateau (voir FIGURE 3.4 à la page 38).

3.3.5 Implémentation de notre modèle

L’implémentation de notre modèle est expliqué dans la FIGURE 3.5 à la page 39. Le sélecteur de mouvement sélectionne le coup suivant pour un joueur en fonction de l’état actuel du plateau de jeu.

```

Apprentissage = Apprentissage(9, 3,100, 32)

Apprentissage.train(game.ObtenirHistorieEntrainement())

val_accuracy: 0.7499
Epoch 95/100
72000/72000 [=====] - 19s 267us/step - loss: 0.1059 - accuracy: 0.7620 - val_loss: 0.1103 -
val_accuracy: 0.7502
Epoch 96/100
72000/72000 [=====] - 19s 257us/step - loss: 0.1059 - accuracy: 0.7622 - val_loss: 0.1105 -
val_accuracy: 0.7515
Epoch 97/100
72000/72000 [=====] - 18s 255us/step - loss: 0.1059 - accuracy: 0.7624 - val_loss: 0.1113 -
val_accuracy: 0.7461
Epoch 98/100
72000/72000 [=====] - 19s 257us/step - loss: 0.1059 - accuracy: 0.7624 - val_loss: 0.1110 -
val_accuracy: 0.7484
Epoch 99/100
72000/72000 [=====] - 18s 255us/step - loss: 0.1058 - accuracy: 0.7624 - val_loss: 0.1107 -
val_accuracy: 0.7497
Epoch 100/100
72000/72000 [=====] - 18s 252us/step - loss: 0.1058 - accuracy: 0.7624 - val_loss: 0.1110 -
val_accuracy: 0.7467

```

FIGURE 3.4 – L’Entraînement de notre réseau de neurone.

Générateur de mouvements disponible

Pour un état de plateau donné, ”le générateur de mouvements disponible” fournit l’ensemble de tous les états possibles de plateau suivants pour le programme.

Évaluateur ”modèle de réseau de neurones”

Pour un état donné du plateau, l’évaluateur prédit une probabilité. Cette probabilité sera utilisée pour pouvoir choisir le meilleur coup à jouer par le programme.

Sélecteur de mouvements

Pour un état donné du plateau, ”le sélecteur de mouvements” sélectionne le coup suivant pour le programme en utilisant l’approche suivante et (voir FIGURE 3.5 à la page 39) :

- Utilisez ”le générateur de mouvements disponible” pour obtenir l’ensemble de tous les états légaux possibles du prochain plateau.
- Pour chacun des états légaux possibles du prochain plateau, utilisez ”Modèle” et prévoyez une probabilité.
- L’état suivant possible du plateau avec la probabilité prévu la plus élevée est choisi pour le coup suivant. Ainsi, la tâche d’apprendre à jouer au Morpion, peut être réduite à la tâche d’entraîner ”le modèle” à attribuer des probabilités plus élevées aux états du plateau favorables au programme, et des probabilités plus faibles aux états du plateau qui sont défavorables au programme.

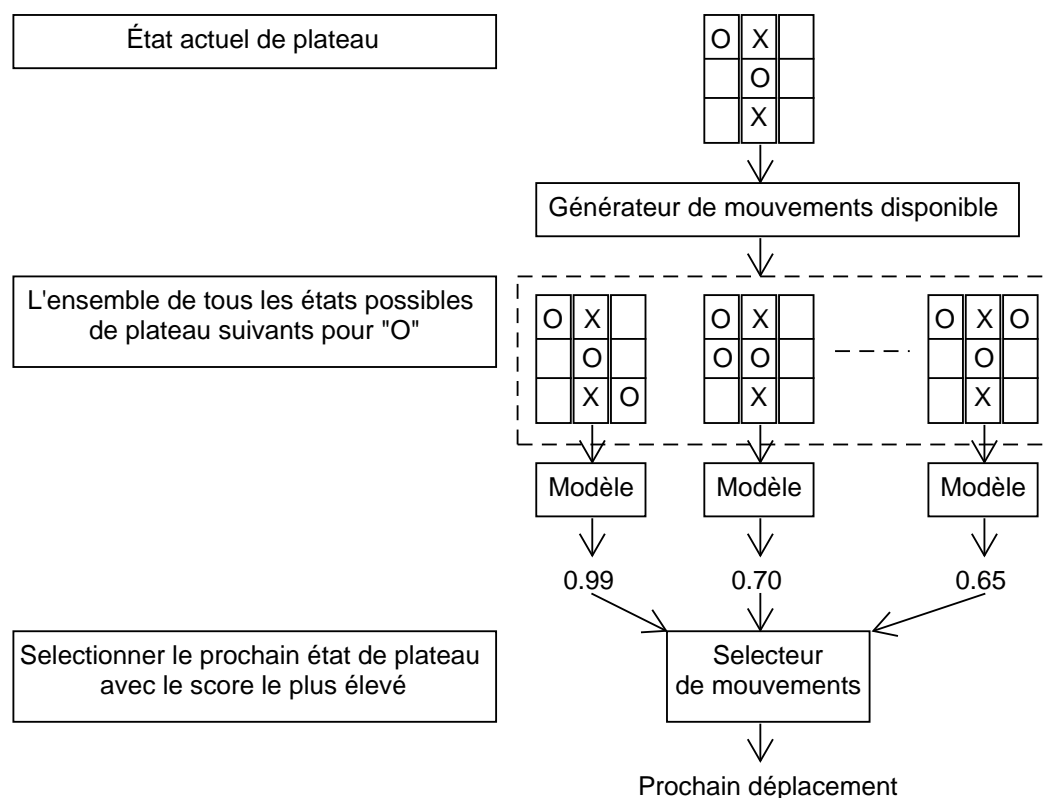


FIGURE 3.5 – Implémentation de notre modèle.

3.4 Implémentation

3.4.1 Langage utilisé

Python


Python est un langage de programmation généraliste et très riche avec de nombreuses fonctionnalités. C'est l'un des principaux objectifs d'un style de programmation connu sous le nom de programmation orientée objet[42]. Il s'agit d'un langage de programmation interprété de haut niveau [43].

3.4.2 Plateforme et environnement de développement

Anaconda

Est une plateforme scientifique de données open source qui rassemble les meilleurs outils pour la science des données. Il s'agit d'une pile de données scientifiques qui comprend plus de 100 paquets populaires basés sur Python, Scala et R. Avec l'aide de son

gestionnaire de paquets, conda, les utilisateurs peuvent travailler avec des centaines de paquets dans différents langages et effectuer facilement le prétraitement, la modélisation, le regroupement, la classification et la validation des données[43].

 **Jupyter Notebook** Est un outil populaire pour écrire du code Python. Basées sur le Web, ce qui signifie que lorsque Jupyter s'ouvre, il le fait dans votre navigateur Web par défaut, qui peut être Google Chrome, Pale Moon, Edge ou Internet Explorer[44].

3.4.3 Bibliothèque Utilisés

Tensorflow

A été créé à l'origine par Google en tant qu'outil interne d'apprentissage automatique. TensorFlow est une bibliothèque de logiciels open source pour l'intelligence artificielle. Elle est de plus en plus utilisée dans la recherche, la production et l'éducation [45].

keras

Est une bibliothèque open source pour les réseaux de neurones. Elle a de très bonnes capacités pour former des fonctions d'activation. Keras peut utiliser différents frameworks Deep Learning en tant que backend. La façon de passer d'un framework à l'autre est de modifier le fichier keras.json, qui se trouve dans le même répertoire que celui où Keras est installé [46].

Numpy

Est une bibliothèque Python open source pour le calcul scientifique. NumPy vous permet de travailler avec des tableaux et des matrices de manière naturelle. La bibliothèque contient une longue liste de fonctions mathématiques utiles, dont certaines pour l'algèbre linéaire, la transformation de Fourier et les routines de génération de numéros. NumPy remplace certaines des fonctionnalités de Matlab et Mathematica, permettant un prototypage rapide et interactif [47].

Panda

Pandas est une bibliothèque Python open source pour l'analyse de données hautement spécialisées [48].

Matplotlib

Est la bibliothèque Python qui est actuellement la plus populaire pour produire des tracés et autres visualisations de données en deux dimensions. Comme l'analyse des données nécessite des outils de visualisation, elle est très répandue dans les milieux scientifiques et techniques [48].

pygame

Est une bibliothèque de commandes qui permet de faciliter l'écriture des jeux. Elle permet de dessiner des formes graphiques, d'afficher des images bitmap, d'animer, d'interagir avec le clavier, la souris et la manette de jeu, de jouer du son, de détecter la collision d'objets [49].

3.4.4 Architecture de notre application

Notre application est divisée en deux parties, partie Apprentissage et partie Game.

La partie Apprentissage

C'est elle qui permet de charger la dataset et la fournir comme entrée à notre réseau de neurones pour pouvoir faire son apprentissage, la FIGURE 3.9 à la page 44 montre les résultats d'apprentissage de notre réseau de neurones.

Sequential spécifie que nous créons le modèle séquentiellement et la sortie de chaque couche que nous ajoutons est entrée dans la couche suivante que nous spécifions.

model.add est utilisé pour ajouter une couche à notre réseau de neurones.

La **Dense** est utilisée pour spécifier la couche entièrement connectée.

Les arguments de Dense sont la dimension de sortie qui est 128 dans le premier cas, la dimension d'entrée qui est 9 et la fonction d'activation à utiliser qui est relu dans ce cas.

La deuxième couche est similaire, nous n'avons pas besoin de spécifier la dimension d'entrée car nous avons défini le modèle comme étant séquentiel, donc Keras considérera automatiquement que la dimension d'entrée est la même que la sortie de la dernière couche,

c'est-à-dire 128, la dimension de sortie qui est 256 et la fonction d'activation utiliser est relu.

Dans la troisième couche, la dimension de sortie est de 256 et la fonction d'activation utiliser est relu.

Dans la quatrième couche, la dimension de sortie est de 256 et la fonction d'activation utiliser est relu.

Dans la cinquième couche (couche de sortie), la dimension de sortie est de 3 (nombre de classes). La couche de sortie prend différentes fonctions d'activation et pour le cas de la classification multiclasse, il s'agit de softmax.

```
self.model = Sequential()

self.model.add(Dense(128, activation='relu', input_shape=(nbrEntree, )))
self.model.add(Dense(256, activation='relu'))
self.model.add(Dense(256, activation='relu'))
self.model.add(Dense(256, activation='relu'))
self.model.add(Dense(nbrSortie, activation='softmax'))
```

FIGURE 3.6 – Création du modèle.

Une fois le modèle est créé, nous l'avons compilé à l'aide de :

- **Loss = 'mse'** : Le travail de la fonction de perte consiste à calculer la différence entre les valeurs prédites et réelles par le modèle qui est entrain de s'entraîner. Cette différence est également appelée perte, moins elle est importante, mieux c'est. Le comportement de la perte aide le modèle à comprendre ce qui doit être fait pour optimiser le modèle afin que la perte puisse être réduite.
- **optimizer='SGD'** : un algorithme d'optimisation. Il optimise le modèle en réduisant la perte calculée par la fonction de perte (mse).
- **metrics=['accuracy']** : Le paramètre de mesure indique que nous voulons juger notre modèle sur la base de la précision.

```
self.model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])
```

FIGURE 3.7 – Compilation du modèle.

Ensuite, nous séparons notre ensemble de données destiné à l'entraînement en deux sous ensembles, 80 % de cet ensemble est réservé pour l'entraînement et les 20 % restants pour l'évaluation de notre modèle.

Ensuite, nous lançons l'entraînement de notre modèle sur les jeux de morpion que nous avons générés précédemment.

Ici, nous devons spécifier les données `X_train`, `y_train`, le nombre d'itérations (epochs), et la taille du lot (`batch_size`). Il renvoie un historique de l'entraînement de modèle. L'historique consiste en la précision (accuracy) du modèle et les pertes (losses) après chaque époque.

l'ensemble de données est très vaste et nous ne pouvons pas faire correspondre des données complètes en une fois, c'est pourquoi nous utilisons `batch_size`. Cela permet de diviser nos données en lots dont la taille est égale à `batch_size`. Maintenant, seul ce nombre d'échantillons sera chargé en mémoire et traité. Une fois que nous avons terminé un lot, il est effacé de la mémoire et le lot suivant sera traité.

- **X_train, y_train** : Nous avons stocké nos données d'entrée et de sortie dans `X_train` et `y_train`.
- **Epochs** : c'est le nombre d'itérations dans l'ensemble de données.
- **batch_size** : précise combien d'échantillons sont inclus dans un lot(batch).

```
border = int(0.8 * len(X))
X_train = X[:border]
X_test = X[border:]
y_train = y[:border]
y_test = y[border:]
self.model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=self.epochs, batch_size=self.batchSize)
```

FIGURE 3.8 – Entraînement du modèle.

Nous avons construit un réseau de neurones qui prend comme entrée un état du plateau de jeu (les données d'entrée sont donc un tableau de neuf éléments, si l'on aplatit l'état du tableau) et qui a comme sortie les probabilités associées à chaque résultat de jeu possible (X gagne, O gagne ou c'est un match nul). Entre les couches d'entrée et de sortie, nous avons 4 couches denses. On a choisi le nombre de couches et leurs dimensions après avoir expérimenté de nombreuses combinaisons différentes. À partir de la sortie, nous choisissons le résultat avec la plus grande probabilité comme résultat de jeu.

La partie Game

C'est la partie qui se charge de lancer le jeu en utilisant le modèle généré lors de l'étape d'apprentissage.

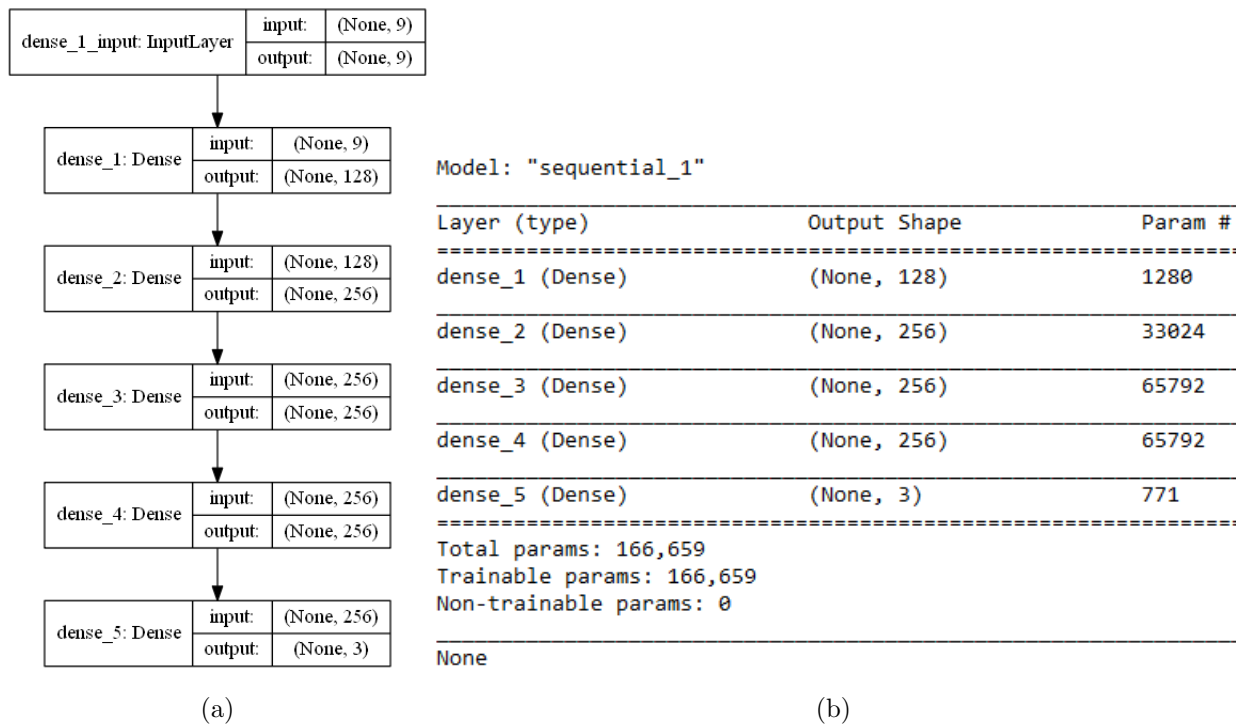


FIGURE 3.9 – Architecture de notre application.

3.4.5 Réalisation

La FIGURE 3.10 à la page 44 montre le résultat de test de notre application en jouant une partie contre elle.

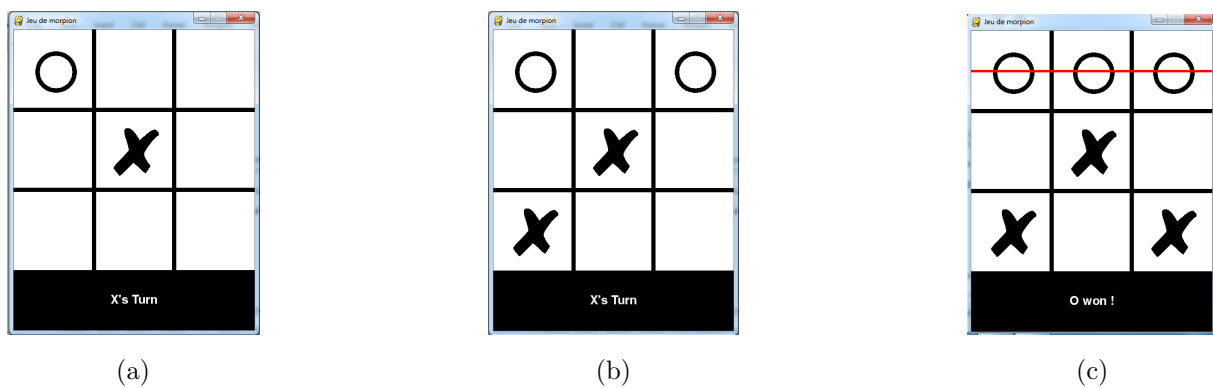


FIGURE 3.10 – (a) et (b) et (c) Réalisation.

3.5 Résultats expérimentaux

Les courbes de précision (Accuracy) et de perte (loss) obtenues lors du processus d'entraînement et de validation sont présentées dans les FIGURES 3.11, 3.12, ?? ci-dessous.

- **Courbe d'apprentissage d'entraînement (Training)** : Courbe d'apprentissage calculée à partir de l'ensemble des données d'entraînement qui donne une idée de la qualité de l'apprentissage du modèle.
- **Courbe d'apprentissage de la validation** : Courbe d'apprentissage calculée à partir d'un ensemble de données de validation qui donne une idée de la qualité de la généralisation du modèle.
- **Courbes d'apprentissage de l'optimisation (Loss)** : Courbes d'apprentissage calculées sur la métrique par laquelle les paramètres du modèle sont optimisés.
- **Courbes d'apprentissage des performances (Accuracy)** : Courbes d'apprentissage calculées sur la métrique par laquelle le modèle sera évalué et sélectionné.

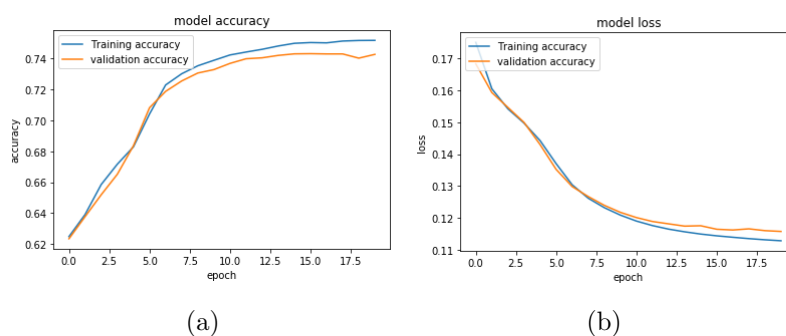


FIGURE 3.11 – (a) Accuracy et (b) loss du modèle de base selon 20 époques.

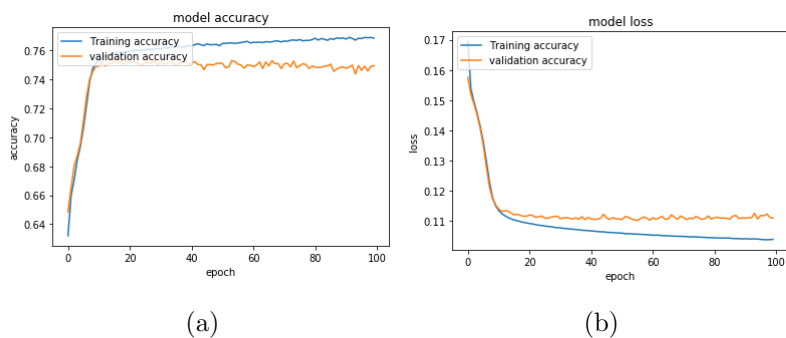


FIGURE 3.12 – (a) Accuracy et (b) loss du modèle de base selon 100 époques.

3.5.1 Discussion

D'après les FIGURES 3.11, 3.12 Les résultats obtenus ont été fait en terme de l'accuracy et de loss :

L'accuracy dans 20 et 100 epoch

- L'accuracy du modèle augmente avec le nombre d'époch jusqu'à ce qu'il devient stable, ceci reflète qu'à chaque epoch le modèle apprend plus d'informations.
- L'accuracy du modèle est presque toujours plus élevé sur l'ensemble des données d'entraînement que sur l'ensemble des données de validation.

Model loss dans 20 epoch

- La courbe de la perte (loss) de l'entraînement diminue jusqu'à un point de stabilité.
- Le courbe de la perte (loss) de validation diminue jusqu'à un point de stabilité et présente un petit écart avec la perte de l'entraînement ce qui signifie un good fit.

Model loss dans 100 epoch

- La courbe de perte d'entraînement continue à diminuer avec l'expérience.
- La courbe de perte de validation diminue jusqu'à un certain point et recommence à augmenter.
- La perte du modèle est presque toujours plus faible sur l'ensemble des données d'entraînement que sur l'ensemble des données de validation.

3.6 Conclusion

Nous avons présenté dans ce dernier chapitre notre réseau de neurones que nous avons proposé pour le jeu de morpion et son fonctionnement et le fonctionnement de notre application et son architecture. Nous avons aussi présenté les différents langages et outils de développement que nous avons utilisé pour implémenter notre solution.

Une validation de notre modèle a été effectué et les résultats obtenus sont satisfaisant et on a testé aussi notre application en jouant quelques parties contre elle et elle a un niveau plutôt acceptable.

Conclusion générale et perspectives

Le réseau de neurones est l'une des avancées d'intelligence artificielle, inspirée par la structure du cerveau humain qui aide les ordinateurs et les machines à ressembler davantage à un humain. Les réseaux de neurones ont une capacité remarquable à extraire des données significatives à partir de données imprécises, ce qui permet de détecter des tendances et d'extraire des modèles difficiles à comprendre, que ce soit par ordinateur ou par l'homme. Un réseau de neurones entraîné peut devenir un "expert" en informations qui ont été données à analyser et peuvent être utilisées pour fournir des projections.

Dans notre travail, nous avons proposé un réseau de neurones qui joue au morpion toute seul. Pour réaliser cette approche, nous avons utilisé le Deep Learning, la méthode d'apprentissage qui a montré ses performances ces dernières années et nous avons choisi la méthode DNN, ce choix est justifié par la simplicité et l'efficacité de la méthode.

Dans cette étude, nous avons parlé des réseaux de neurones en donnant leurs types, leurs fonctionnements, le type d'apprentissage utilisé par ces réseaux de neurones et les algorithmes d'apprentissage. Présenter quelques jeux de plateau qui utilisent les réseaux de neurones. Ensuite, Nous avons détaillé notre solution et discuté les résultats obtenus.

Bien que notre modèle n'est pas un modèle aussi complexe que AlphaGo, mais il a tout de même montré une nette amélioration par rapport au jeu aléatoire même si il ne peut pas jouer aussi bien que deux humains compétents l'auraient fait. De plus, il semble prendre des décisions raisonnables lorsqu'il est joué par un humain. Le modèle se déplace "pour gagner" s'il se perçoit comme étant en avance sur son adversaire, et "pour faire match nul" s'il perçoit son adversaire comme étant en avance.

Comme perspectives nous pouvons ajouter :

- Tester notre modèle sur d'autres jeux de données.
- Proposer d'autres architectures pour améliorer les résultats.
- Proposer un autre modèle qui peut jouer sur différentes grilles (5*5, 7*7).

Bibliographie

- [1] **Dave Anderson and George McNeill.** *Artificial neural networks technology.* *Kaman Sciences Corporation*, 258(6) :1-83, 1992.
- [2] **Claude TOUZET.** *INTRODUCTION AU CONNEXIONNISME COURS,LES RESEAUX DE NEURONES ARTIFICIELS , EXERCICES ET TRAVAUX PRACTIQUES*, Juillet 1992.
- [3] **Gilbert Saporta.** *Une brève histoire de l'apprentissage*, 2018.
- [4] **Mohamed Yessin Ammar.** *Mise en œuvre de réseaux de neurones pour la modélisation de cinétiques réactionnelles en vue de la transposition batch/continu.* PhD thesis, 2007.
- [5] Groupe Publithings Tous droits réservés Mentions légales, *LE BIG DATA*, 2020, disponible à l'adresse : <https://www.lebigdata.fr/reseau-de-neurones-artificiels-definition>. Consulté : 20 mars 2020.
- [6] Centre universitaire Belhadj Bouchaib Ain Temouchent Institut des sciences et de la technologie Département génie électrique POLYCOPIE DE COURS COMMANDE AVANCEE ” *Logique floue et réseaux de Neurone 'application à l'électrotechnique'”* Parcours : commande système électrique M1 Année Universitaire : 2016-2017 Préparé et enseigné par : **Dr : Mendaz Keira**. Consulté : 20 mars 2020.
- [7] **Youcef Djeriri.** *Les Réseaux de Neurones Artificiels*, 20 September 2017.
- [8] sensagent : Encyclopédie en ligne, Thesaurus, dictionnaire de définitions et plus. Tous droits réservés, *LE PARISIEN*, 2000-2016, disponible à l'adresse :

- <http://dictionnaire.sensagent.leparisien.fr/Fonction%20de%20Heaviside/fr-fr/>. Consulté : 20 mars 2020.
- [9] **Mohammed Msaaf and Fouad Belmajdoub.** *L'application des réseaux de neurone de type "feedforward" dans le diagnostic statique.* 2015.
- [10] **Nikolay Kyurkchiev and Svetoslav Markov.** *Sigmoid functions : some approximation and modelling aspects.* LAP LAMBERT Academic Publishing, Saarbrücken, 2015.
- [11] **Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall,** *Activation Functions : Comparison of Trends in Practice and Research for Deep Learning* arXiv :1811.03378v1 [cs.LG] 8 Nov 2018.
- [12] PRIVACY POLICY. HORACE THEME BY JUSTGOOD-THEMES. POWERED BY JEKYLL AND GITHUB PAGES. PRIVACY POLICY, *KRIS BOLTON*, 2020, disponible à l'adresse : <http://krisbolton.com/a-quick-introduction-to-artificial-neural-networks-part-2#>. Consulté : 01 décembre 2020.
- [13] **Abien Fred M. Agarap.** abienfred.agarap@gmail.com, *Deep Learning using Rectified Linear Units (ReLU).* arXiv :1803.08375v2 [cs.NE] 7 Feb 2019.
- [14] Chris Woodfordx. All rights reserved, *EXPLAIN-THATSTUFF!*, 2011, 2020, disponible à l'adresse : <https://www.explainthatstuff.com/introduction-to-neural-networks.html> Consulté : 03/11/2020.
- [15] TechTarget, *LE MAGIT*, 2018 - 2020, disponible à l'adresse : <https://whatis.techtarget.com/fr/definition/reseau-de-neurones>. Consulté : 20 mars 2020.
- [16] All Rights Reserved, *TUTORIALS POINT*, 2020, disponible à l'adresse : https://www.tutorialspoint.com/tensorflow/tensorflow_single_layer_perceptron.htm. Consulté : 20 mars 2020.
- [17] **Ricco Rakotomalala.** *Perceptrons simples et multicouches*, Université Lumière Lyon 2. Consulté : 21 mars 2020.

- [18] **Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil.** *Multilayer perceptron : Architecture optimization and training.* IJIMAI, 4(1) :26, 30, 2016.
- [19] **Benoît Virole.** *Réseaux de neurones et psychométrie.* Editions du Centre de Psychologie Appliquée-ECPA, 2001.
- [20] DATA SCIENCE , Medium, *TOWARDS*, disponible à l'adresse : <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Consulté : 20 mars 2020.
- [21] **Manohar Swamynathan.** *Mastering Machine Learning with Python in Six Steps : A Practical Implementation Guide to Predictive Data Analytics Using Python*, Publisher : Apress, 2017, ISBN : 978-1-4842-2865-4, 978-1-4842-2866-1.
- [22] Search Engine Marketing, *Digital Guide*, 10.03.20, disponible à l'adresse : <https://www.ionos.fr/digitalguide/web-marketing/search-engine-marketing/quest-ce-quun-reseau-neuronal-artificiel/>. Consulté : 20 mars 2020.
- [23] **Georgios Kontonatsios, Austin J Brockmeier, Piotr Przybyla, John McNaught, Tingting Mu, John Y Goulermas, and Sophia Ananiadou.** *A semi-supervised approach using label propagation to support citation screening.* Journal of biomedical informatics, 72 :67-76, 2017.
- [24] **Y CA Padmanabha Reddy, P Viswanath, and B Eswara Reddy.** *Semi-supervised learning : A brief review.* Int. J. Eng. Technol, 7(1.8) :81, 2018.
- [25] **Roche David.** *Deep learning et apprentissage par renforcement pour la conception d'une Intelligence Artificielle pour le jeu Yokai No Mori.*
- [26] **Dave Anderson and George McNeill.** *Artificial neural networks technology.* Kaman Sciences Corporation, 258(6) :1 83, 1992.
- [27] **Benoît Virole.** *RESEAUX DE NEURONES ET PSYCHOMETRIE , Etude prospective des applications possibles des réseaux de neurones formels dans le traitement des données psychométriques ,* Editions du Centre de Psychologie Appliquée , Juin 2001.
- [28] Business Insider Inc., *BUSINESS INSIDER* , 2016, disponible à l'adresse : <https://www.businessinsider.fr/ces-jeux-ou-lintelligence-artificielle-a-battu-lhumain/#le-backgammon>. Consulté : 9 mai 2020.

- [29] Flat Prod. Tous droits réservés. La reproduction totale ou partielle sans permission est interdite, *TRIC TRAC*, 2020, disponible à l'adresse : <https://www.trictrac.net/jeu-de-societe/yokai-no-mori-0>. Consulté : 9 mai 2020.
- [30] Jedisjeux. All rights reserved, *JEDIS JEUX*, 2020, disponible à l'adresse : <https://www.jedisjeux.net/jeu-de-societe/yokai-no-mori>. Consulté : 9 mai 2020.
- [31] geekoutput, *BGG*, disponible à l'adresse : <https://boardgamegeek.com/boardgame/148641/ykai-no-mori>. Consulté : 10/10/2020.
- [32] **David Roche**, *Deep learning et apprentissage par renforcement pour la conception d'une Intelligence Artificielle pour le jeu Yokai No Mori*. Ce travail est publié sous licence Creative Common.
- [33] Fédération Française de Go, *JEUDEGO.ORG*, 1998-2020, disponible à l'adresse : <https://jeudego.org/>. Consulté : 10 mai 2020.
- [34] Sophos Ltd. All rights reserved, *NAKED SECURITY by SOPHOS*, 1997 - 2020, disponible à l'adresse : <https://nakedsecurity.sophos.com/pt/2016/03/10/will-google-ai-conquer-the-worlds-toughest-game-go/>. Consulté : 10 mai 2020.
- [35] **Florian Brunner**. *Mastering the game of go with deep neural networks and tree search* .(silver et al, 2016). 2019.
- [36] Jeroen van der Zijp, *FOX TOOLKIT*, 1997-2007, disponible à l'adresse : <http://www.fox-toolkit.org>. Consulté : 11 octobre 2020.
- [37] Amazon , Inc. ou ses filiales. *Amazon*, 1996-2020, disponible à l'adresse : <https://www.amazon.fr/Chess-Armory-Wooden-Interior-Storage/dp/B01256V578>. Consulté : 01 décembre 2020.
- [38] Collins, *COLLINS*, 2020, disponible à l'adresse : <https://www.collinsdictionary.com/dictionary/english/shogi>. Consulté : 11 octobre 2020.
- [39] SIAMMANDALAY. SUBSCRIBE TO OUR MAILING LIST, *SIAMMANDALAY.*, 2020, disponible à l'adresse :

- <https://www.siammandalay.com/blogs/puzzles/play-go-board-game-instructions-rules-strategies>. Consulté : 11 octobre 2020.
- [40] **David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al.** *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*. arXiv preprint arXiv :1712.01815, 2017.
- [41] *VITRINE MAGIQUE*, disponible à l'adresse : <https://www.vitrinemagique.com/p/jeu-en-bois-morpion/26123/> Consulté : 01/10/2020.
- [42] **Peter Norton, Alex Samuel, David Aitel, Eric Foster-Johnson, Leonard Richardson, Jason Diamond, Aleatha Parker, and Michael Roberts.** *Beginning python*, isbn-10 : 0-7645-9654-3.
- [43] **Yuxing Yan and James Yan.** *Hands-On Data Science with Anaconda : Utilize the right mix of tools to create high-performance data science applications*. Packt Publishing Ltd, 2018. Consulté : 04/10/2020.
- [44] **John Shovic, Alan Simpson.** *Python All-In-One for Dummies*. Series : For Dummies, Publisher : Wiley, Year : 2019, ISBN : 1119557593, 978-1119557593.
- [45] **Sam Abrahams, Erik Erwit, Ariel Scarpinelli, and Danijar Hafner.** *Tensorflow for machine intelligence : a hands-on introduction to learning algorithms*. 2016.
- [46] **Abhishek Nandy and Manisha Biswas.** *Reinforcement Learning : With Open AI, TensorFlow and Keras Using Python*. Apress, 2017.
- [47] **Ivan Idris,** *NumPy 1. 5 Beginner's Guide*. 2011, Packt Publishing Ltd, ISBN : 1849515301,9781849515306.
- [48] **Fabio Nelli.** *Python data analytics : Data analysis and science using pandas. Matplotlib, and the Python Programming Language*, 2015. Publisher : Apress.ISBN : 978-1-4842-0959-2 ,978-1-4842-0958-5.
- [49] **Paul Craven.** *Program Arcade Games : With Python and Pygame*. Apress, 2015.