Université de Montréal

MAT3450 – Modélisation mathématique

AlphaGo Zero

Apprentissage *tabula rasa* du jeu *connect4*

---

par :
Alexandre Adam
20090755

Éric Pfleiderer
20048978

4 juillet 2023

1 Sommaire

En 2017, AlphaGo Zero réussit à développer des capacités surhumaines dans le jeu de Go sans apprentissage supervisé. Le réseau de neurones employé par AlphaGo Zero est entraîné sur des plates-formes spécialisées à l'aide d'apprentissage par renforcement et dépasse les performances de son prédécesseur, AlphaGo, en quelques jours seulement. Une problématique pertinente est alors soulevée : est-il possible pour AlphaGo Zero d'obtenir de telles capacités, lorsqu' entraînées sur des systèmes non spécialisés ? Plus fondamentalement, les applications d'AlphaGo Zero sont-elles généralisables ?

Afin d'explorer ces problématiques, l'algorithme d'apprentissage automatique est appliqué sur un nouvel environnement, soit le jeu *Connect4*, et son réseau neuronal est entraîné sur des plates-formes non spécialisées. Deux versions d'AlphaGo Zero sont étudiées, soit une version implémentant des blocs résiduels et l'autre implémentant des blocs convolutionnels. Le rythme d'apprentissage des architectures est évalué par compétition ; les générations du réseau neuronal doivent affronter les autres générations de la même architecture. De plus, les générations doivent affronter les générations équivalentes provenant de l'architecture complémentaire. La performance est évaluée par l'apparition et l'usage de tactiques connues dans le cadre du jeu *Connect4*. Les résultats montrent un lien entre la performance d'AlphaGo Zero et l'expérience acquise, mais n'indiquent pas de différence importante entre les deux architectures.

2 Introduction

Le développement d'agents intelligents a longtemps été un enjeu central dans le domaine de l'informatique. Étant donné un agent, un ensemble de règles et un but, comment est-il possible pour l'agent de naviguer à travers son environnement pour accomplir son objectif de manière optimale ? Les solutions traditionnelles telles que les techniques d'optimisation et les automates déterministes sont efficaces pour la résolution de problèmes relativement simples, mais ces solutions ne demeurent pas toujours viables lorsque les actions légales mènent à une explosion combinatoire du nombre d'états futurs possibles. Dans de tels cas, où l'intelligence ne peut pas être simulée de manière explicite, l'apprentissage *tabula rasa* et automatique de l'agent devient cruciale ; il doit être en mesure d'apprendre sans intervention extérieure à partir des règles de l'environnement seulement.

En 2015, AlphaGo, un algorithme d'apprentissage automatique, l'emporte sur l'un des champions du monde au jeu de Go. AlphaGo emploie un réseau de neurones entraîné à l'aide d'apprentissage par renforcement et d'apprentissage supervisé. L'algorithme apprend par joutes successives contre lui-même et la prise de décision, effectuée par une recherche arborescente Monte Carlo, est guidée par un échantillon de coups d'experts. AlphaGo Zero, un successeur qui apparaît sur la scène deux ans plus tard, vient éliminer cette dépendance sur un échantillon d'exemples et donne naissance à un véritable apprentissage *tabula rasa* ; le réseau de neurones est entraîné sans intervention extérieure et développe pourtant des performances supérieures à son prédécesseur.

3 Renseignements

[1] Le concept d'apprentissage machine par renforcement réfère à l'algorithme d'entraînement du réseau neuronal et de l'ensemble arborescent Monte Carlo (méthode MCTS). Le problème lié à cette méthode peut être généralisé en considérant un *agent* à la recherche d'une *police d'action* (évaluée par la méthode MCTS) qui maximise les *gains* associés à l'action choisit. Il *apprend* cette police par *expérience*, c'est-à-dire par joutes successives contre lui-même. Cette section s'intéresse donc a lié ces concepts avec les différentes parties du modèle qui y correspondent.

3.1 L'agent

L'agent doit posséder la capacité de *percevoir* son environnement, d'*explorer* son environnement et la capacité de *décision*.

Pour lui donner ces capacités, on lui donne la structure telle que représentée à l'annexe par la figure 12. La matrice de données est un tenseur de taille (2, 6, 7), c'est-à-dire deux images de l'état du jeu, une pour chaque joueur. Cette information est traitée dans le bloc de vision de l'agent, par le réseau de neurones correspond à la fonction

$$f_\theta(s) = (\mathbf{p}, \nu) \tag{1}$$

où s est l'état de la partie et θ sont les paramètres internes au réseau.

Le vecteur $\boldsymbol{p}$ représente un vecteur de probabilité de choisir une action particulière, parmis l'ensemble des $6 \times 7 = 42$ cases du jeu standard de *Connect4*. Parmi ces possibilités, seules les actions légales sont considérées à tout moment et fournies à l'évaluateur MCTS.

Le paramètre scalaire ν est une évaluation de l'état de la partie sur la probabilité de l'agent de gagner. Il représente l'estimation à long terme des gains potentiels.

La fonction f_θ est crucial, car elle permet à l'agent de percevoir son environnement et lui donne la capacité d'évaluer si cette situation est bonne ou mauvaise. L'algorithme d'apprentissage vise ultimement à améliorer les paramètres θ.

3.2 Simulation MCTS

La capacité de l'agent à *explorer* les choix légaux est simulée par le simulateur Monte Carlo. Avant chaque action, lors d'une partie, le simulateur explore 50 parties possibles. Cette exploration est semi-stochastique : la sélection d'une branche fait appel à la fonction 2 dont le paramètre P est modulé par un bruit η_α.

Notons aussi quelques définitions. Chaque noeud possède une branche pour chaque action $a \in \mathcal{A}(s)$ possibles selon l'état. Chaque noeud est équipé d'un ensemble de statistique $\{N, W, Q, P\}$ où N est un compteur de visites, W est la valeur totale de l'action, Q est la valeur moyenne de l'action et finalement P est la probabilité de choisir une certaine branche évaluée par f_θ.

L'exploration consiste en trois étapes : la sélection, l'évaluation et le l'ajustement à rebours. La sélection décide de la branche à explorer plus avant à l'aide de l'évaluation de la fonction f_θ de chacune des branches qui partent du noeud. La sélection est une fonction *argmax* :

$$a_t = \operatorname{argmax}\left[Q(s,a) + c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right] \tag{2}$$

1. Cette section réfère à l'article où le modèle fût originellement publié [5] et aussi au code que nous avons utilisé, produit par David Foster [6].

où c_{puct} contrôle le niveau d'exploration et

$$P(s,a) = (1-\epsilon)\boldsymbol{p_\alpha} + \epsilon\eta_\alpha. \quad (3)$$

Cette statistique prend l'évaluation du réseau de neurones $\mathbf{p}$, et y ajoute un bruit tiré d'une distribution de Dirichlet $\eta_\alpha \approx D(0.3)$, pesé par le paramètre $\epsilon = 0.2$.

Une valeur plus élevée de c_{puct} encourage la simulation à sélectionner des branches qui ont été visitées souvent. Dans notre cas, $c_{puct} = 1$, de sorte que l'algorithme adopte ce biais préférentiel rapidement.

L'évaluation consiste à produire une valeur de P et ν associé à une branche. Lorsque l'arbre atteint une feuille (simulation se termine) les statistiques de chaque noeud qui ont mené à cette feuille sont ajustées :

$$N(s_t,a_t) = N(s_t,a_t) + 1 \quad (4)$$
$$W(s_t,a_t) = W(s_t,a_t) + \nu \quad (5)$$
$$Q(s_t,a_t) = \frac{W(s_t,a_t)}{N(s_t,a_t)} \quad (6)$$

3.3 Vision machine

La vision machine est un concept utilisé ici de façon lousse, qui réfère aux méthodes associées à la reconnaissance d'image dans le champ d'études sur l'intelligence artificielle. Deux architectures sont utilisées est comparées dans l'analyse des résultats. C'est cette portion de l'architecture qui s'occupe de donner à l'agent la capacité de *percevoir* l'état du jeu.

La première architecture est basée sur le travail de Kaiming He *et al.* [7] et consiste en une amélioration sur l'architecture des réseau neuronaux convolutionel, proposés originellement par les travaux de LeCun et Bengio [8] vingt ans plus tôt. On se propose de comparer ces deux architectures dans le contexte de l'apprentissage du jeu *Connect4*.

3.4 Apprentissage

Comme f_θ est l'évaluateur des états s, la performance du modèle dépend grandement de la précision des paramètres θ. On utilise l'algorithme d'optimisation SGD (*Stochastic Gradient Descent*), avec objectif de minimiser la fonction de perte

$$l = (z-\nu)^2 - \boldsymbol{\pi}^T \log \mathbf{p} + c||\theta||^2 \quad (7)$$

où z correspond au gagnant ($z = \pm 1$) et c est un paramètre de régularisation qui empêche le surapprentissage. Minimiser la différence $z - \nu$ permet d'optimiser l'évaluateur de l'état de f_θ. Aussi, on considère que la police évaluée par le simulateur MCTS $\boldsymbol{\pi}$ est une meilleure approximation de la police optimale. Le terme $\boldsymbol{\pi}^T \log \mathbf{p}$ est un produit scalaire qui mesure la similarité de la prédiction de f_θ et $\boldsymbol{\pi}$. Minimiser la fonction l reviens donc à maximiser cette similarité.

Une collection limitée des parties jouées est sauvegardée à tout moment. Cette collection est utilisée comme un ensemble de données d'entraînement, correspondant à 30 000 coups joués et leur ensemble de statistiques associés provenant de la simulation MCTS. L'algorithme SGD tente alors de minimiser l'équation de perte 7 en ajustant les paramètres θ qui génère $\boldsymbol{p}$ et ν.

4 Recherche

4.1 Objectifs

AlphaGo Zero a développé des capacités surhumaines dans le jeu de Go lorsqu' entraîné sur des plates-formes spécialisées à hautes performances ; on cherche à montrer que l'algorithme demeure une solution viable en l'absence de tels superordinateurs. On cherche plus explicitement à montrer que les applications d'AlphaGo Zero

sont généralisables ; l'algorithme est employé sur un nouvel environnement, le jeu Connect4, ainsi que sur des plates-formes considérées sous-optimales. Finalement, on cherche à comparer la performance de deux versions de l'algorithme ; une implémentant des blocs résiduels et une implémentant des blocs convolutionnels.

4.2 Méthodologie

L'étendu des applications d'AlphaGo Zero est évaluée en implémentant deux architectures différentes de l'algorithme, un employant un réseau de neurones résiduel (ResNet) et l'autre un réseau de neurones convolutionnel (CNN). L'algorithme est entraîné par joutes successives contre lui-même. Cet entraînement produit de multiples générations du réseau de neurones, où les paramètres θ de f_θ sont mis à jour à chaque nouvelle génération afin de maximiser la vraisemblance entre $\boldsymbol{p}$ et $\boldsymbol{\pi}$.

Afin de mesurer la performance de chaque génération, des compétitions intra architecture sont organisées. Le succès des générations les plus âgées est un indicateur d'optimisation de la police $\boldsymbol{p}$ et donc ultimement d'apprentissage. Par contre, cette métrique est relative ; il est possible que l'entraînement du réseau neuronal optimise la police de manière à atteindre un minimum local. Cette situation correspondrait à un joueur d'échec amateur qui se trouve à être le meilleur joueur dans son cercle d'amis, sans savoir qu'une communauté de grands maîtres ont des connaissances sur le jeu de loin supérieur.

Pour compenser cet effet, on mesure les connaissances du modèle en lui montrant des états de jeu et en analysant sa réponse à cet état. En s'appuyant sur les travaux de V. Allis [4], on analyse la recommandation $\boldsymbol{p}$ du modèle et son estimation ν de l'état devant 3 situations : le premier coup, la tactique de la double menace et la réponse à cette tactique.

Les compétitions sont performées en posant $\tau = 0$ pour forcer le modèle à jouer selon la recommandation $\boldsymbol{p}$ (voir annexe, la fonction qui détermine l'action). Le comportement semi-stochastique est utile lors de l'entraînement afin d'éviter les minimums locaux en simulant une curiosité d'apprentissage. Par contre, lors de l'évaluation des architectures, un comportement stochastique est indésirable.

4.3 Résultats

Durant chaque compétition, les générations du réseau neuronal sont mises à l'épreuve les uns contre les autres dans une série de 5 joutes. Le réseau neuronal qui gagne une joute se voit attribuer un point, tandis que le perdant se voit pénaliser d'un point. Les joutes nulles impliquent le statu quo. Le total des points pour chaque génération participante est tabulé et comparé graphiquement.

Le temps de calcul nécessaire pour simuler une compétition sous ces conditions est relativement lourd ; le nombre total de parties jouées est de l'ordre de $\Theta(n^2)$, où n est le nombre de générations participantes. Puisque la puissance de calcul à notre disposition est limitée, le nombre de participants est restreint. Parmi les 37 générations d'architecture ResNet et les 40 générations d'architecture CNN produites durant l'entraînement, seulement une sur quatre est choisie pour participer à la compétition.

4.3.1 Compétition intra architecture

La compétition entre les générations de l'architecture ResNet implique 8 participants qui forment l'ensemble (0, 4, 8, 12, 16, 20, 24, 28). Chaque élément de l'ensemble affronte chaque autre élément dans une série de 5 joutes tel que décrite précédemment :

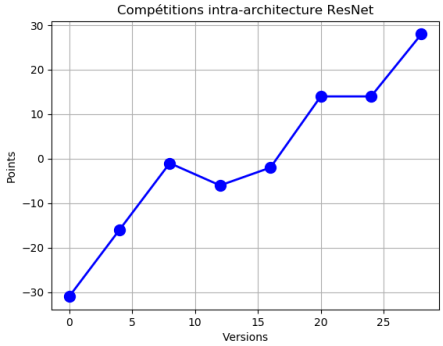


FIGURE 1 – Résultats de la compétition intra architecture ResNet

On remarque par inspection des données une corrélation possible entre le nombre de points accumulés durant la compétition et l'âge du réseau en générations. Lorsqu'une régression linéaire est effectuée, on obtient un coefficient de détermination de $R^2 = 0.916$. Un coefficient de détermination élevé signifie une forte confiance en la qualité de la prédiction de la régression linéaire. Dans ce contexte, cela signifie que les versions plus avancées sont de loin meilleures que les précédentes, et gagnent presque toutes leurs parties. Les versions plus avancées ont soit appris à contrer le comportement des versions précédentes ou appris des techniques gagnantes. La deuxième option sera envisagée plus loin, lors de l'analyse des tactiques.

La compétition entre les générations de l'architecture CNN implique 11 participants qui forment l'ensemble (0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40) :

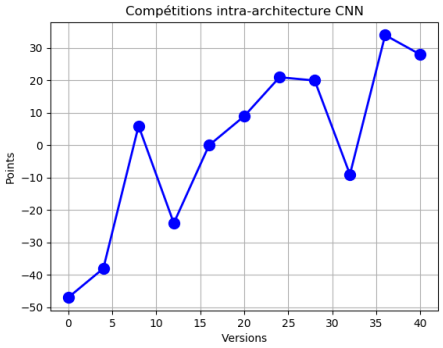


FIGURE 2 – Résultats de la compétition intra architecture CNN

Lorsqu'une régression linéaire est effectuée, on obtient un coefficient de détermination de $R^2 = 0.667$, un résultat plus faible que celui obtenu par l'architecture ResNet. Une première interprétation de ce résultat est qu'il y a amélioration comme dans la précédente architecture. Les améliorations ne sont toutefois pas suffisantes pour démarquer définitivement certaines versions de ses précédentes comme les versions 8 et 32. L'apprentissage avec une architecture ResNet serait donc plus stable. On ne peut cependant parler de force relative entre les deux architectures. C'est le sujet des prochains résultats.

4.3.2 Compétition inter architecture

La compétition inter architecture implique les générations (0, 4, 8, 12, 16, 20, 24, 28, 32, 36) de chaque architecture. Chaque génération affronte la génération de même âge provenant de l'architecture complémentaire.

Lorsqu'une régression linéaire est effectuée, on obtient un coefficient de détermination de $R^2 = 0.002$. Il semble qu'il n'y ait pas de conclusion définitive à porter sur ce résultat. Des fluctuations statistiques auraient pu générer les oscillations observées à la figure 3. Si un

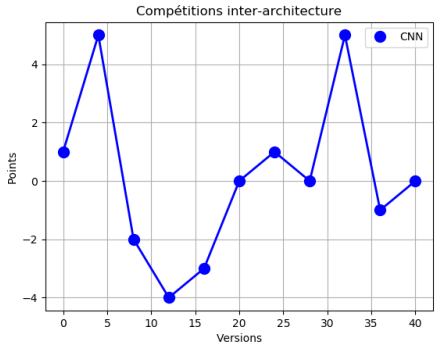


FIGURE 3 – Résultats de la compétition inter architecture

modèle avait été meilleur qu'un autre, la courbe aurait montré une préférence pour la structure CNN (valeur du score surtout positive) ou la structure ResNet (valeur du score surtout négative).

4.3.3 Techniques et tactiques développées

L'environnement *Connect4* correspond à une grille de 6 rangées par 7 colonnes. On identifie les cellules dans la figure 4.

0	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	32	33	34
35	36	37	38	39	40	41

FIGURE 4

Trois états possibles du jeu *Connect4* sont présentés au réseau neuronal convolutionnel, qui évalue de façon déterministe l'action optimale dictée par sa police. Cette action est comparée avec une solution optimale connue respective à chaque cas.

Le premier état en question est la configuration initiale vide où le joueur blanc doit décider de sa première action. On présente cet état à chaque génération du réseau de neurones CNN et on tabule la probabilité de trois actions :

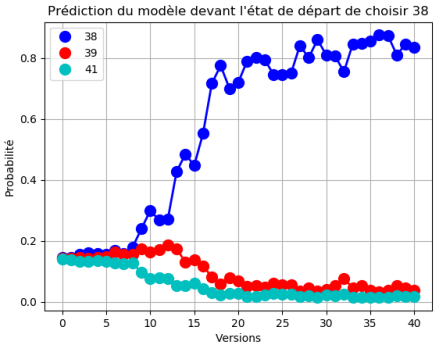


FIGURE 5 – Probabilité de choisir l'action 38, 39 ou 41 au premier tour en fonction de la génération du réseau

Selon Allis [4], le joueur noir peut garantir une partie nulle à condition que le joueur blanc ne joue pas la case du centre dès son premier tour. L'action 38 serait la case optimale à jouer pour les blancs au premier tour.

Plus généralement, le choix de jouer au centre ouvre plus de possibilités que de jouer sur les bords.

En se référant à la figure 4, on remarque que la probabilité de choisir la cellule 38 augmente drastiquement autour de la génération 10 et atteint rapidement un plateau autour de 0.8, tandis que la probabilité des deux autres actions diminue asymptotiquement vers 0.

Le prochain état prend la configuration de la figure 6.

Dans cette configuration du jeu, le joueur blanc doit décider d'une action. Deux des actions disponibles peuvent mener à une victoire rapide, soit la cellule 37 et la cellule 40. Si ces cellules sont jouées, aucune action disponible au joueur noir ne parviendrait à empêcher la victoire du joueur blanc s'il joue optimalement. Cet état est évalué par les différentes générations du réseau de neurone :

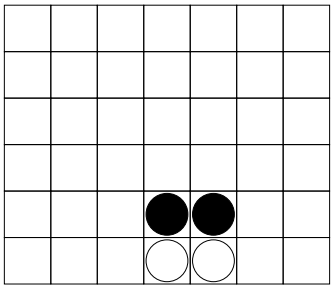


FIGURE 6 – Position qui s'avère après 1. d1 d2, 2. e1 e2... C'est au tours des blancs de jouer et la tactique de la double menace va permettre aux blancx de gagner assurément à leurs prochains tours avec les coups 3. c1 ! ou 3. f1 !

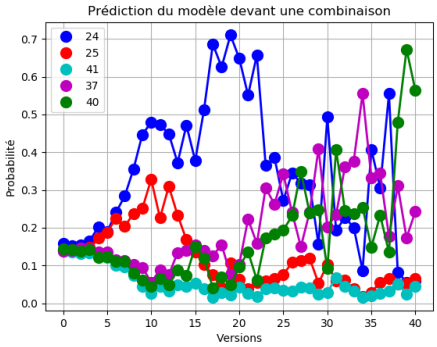


FIGURE 7 – Évaluation des actions 24, 25, 37, 40 et 41 lorsque présenté par une configuration telle que présentée à la figure 6

On remarque en particulier que les coups idéaux 37 et 40 ne sont remarqué par le modèle qu'autour de la version 20, en même temps que la probabilité de jouer la cellule 24 descend. Déjà, à la version 39 et 40, le coup 24 voit sa probabilité descendre au même niveau que le coup sur le bord (41) et le coup 25. Ceci nous indique que le modèle comprend que la situation lui est favorable. Cette interprétation est corroborré par l'évaluation de l'état du jeu par la modèle (ν) à la figure 8.

L'état final évalué prend la configuration de la figure 9.

Dans ce contexte, le joueur noir doit

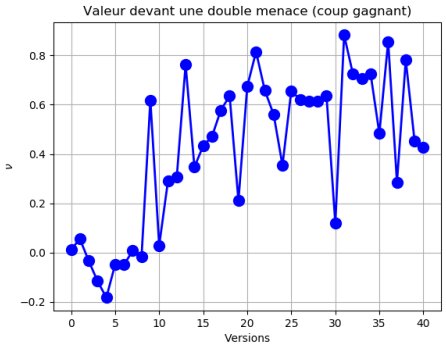


FIGURE 8 – Évaluation de l'état du jeu par le modèle avec vision CNN de la position de la figure 6.

choisir un coup qui lui permettra de bloquer l'apparition de la tactique évaluée précédemment. Le joueur noir doit jouer 37 ou 40 avant que le joueur blanc puisse le faire. Si le joueur noir choisit la cellule 32, la configuration devient identique au cas précédent. La figure 10 montre qu'encore une fois, le coup 24 est d'abord comme optimal. Autour de la version 20, la même version à laquelle le modèle à découvert la tactique de la double menace, la probabilité de jouer les coups qui contrent cette tactique augmente ! Cette analyse du modèle dans les deux positions (joueur blanc et joueur noir) montre que le modèle à identifier la tactique de la double menace en début de partie, qu'il *découvre* à la vingtième version.

L'évaluation de la position à la figure 11 montre que le modèle évalue sa position à la négative, avec un premier pic négatif à la version 20. Toutefois, il n'est pas clair, avec la valeur de ν, que le modèle évalue cette position négativement à cause de la tactique ou parce qu'il évalue l'évolution à long terme de la partie. En effet, la valeur oscille à des valeurs négatives, mais ne semble pas retourner autour de 0.

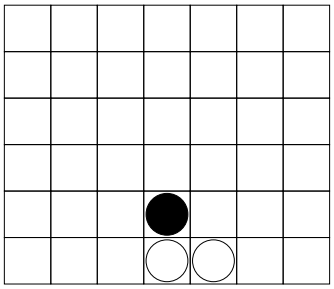


FIGURE 9 – Position qui s'avère après 1. d1 d2, 2. e1... C'est au tours des noirs de jouer et ils doivent contrer la tactique de la double menace des blancs.

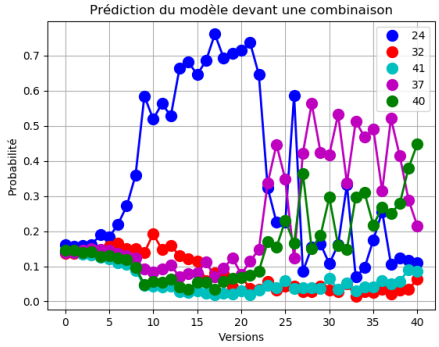


FIGURE 10 – C'est au noir de jouer, selon la position représenté à la figure 9.

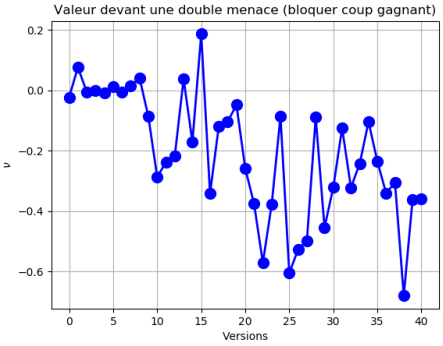


FIGURE 11 – Évaluation de la position représenté à la figure 9.

5 Conclusion

À partir des règles de son environnement et d'un but, AlphaGo Zero acquiert la capacité de faire des choix calculés et intelligents. Appliqué au contexte de *Connect4*, l'algorithme d'apprentissage par renforcement adopte des stratagèmes optimaux, tels que jouer pour le centre ou bloquer une double menace, une indication d'apprentissage par expérience seulement. La performance d'AlphaGo Zero demeure donc fondamentalement liée à la puissance de calcul disponible pour l'entraînement du réseau de neurones. Le développement de capacité surhumaine demeure impratique lorsque restreint à des systèmes non spécialisés, mais le développement de l'intelligence au sens large est réaliste. Les différences entre les architectures évaluées, soient le réseau résiduel et le réseau convolutionnel, semblent être minimales lors de leur application dans le contexte de l'algorithme AlphaGo Zero.

Références

[1] Tristan Cazenave, *Residual Networks for Computer Go*. Residual Networks for Computer Go. IEEE Transactions on Computational Intelligence and AI in Games PP(99) :1-1 · March 2017

[2] Richard S. Sutton et Andrew G. Barto *Reinforcement Learning : An Introduction*. Second edition (2015). A Bradford Book. The MIT Press.

[3] Levente Kocsis et Csaba Szepesv'ari, *Bandit based Monte-Carlo Planning*, Proceeding ECML'06 Proceedings of the 17th European conference on Machine Learning, pp. 282-293. Septembre 2006.

[4] Victor Allis, *A Knowledge-based Approach of Connect-Four*, Vrije Universiteit, Octobre 1988.

[5] David Silver, Julian Schrittwieser, *et al*, *Mastering the game of Go without human knowledge*. Nature volume 550, pp. 354–359. 19 Octobre 2017.

[6] David Foster, *How to build your own AlphaZero AI using Python and Keras*. Applied Data Science - Medium, URL : https ://medium.com/applied-data-science. Git repository : https ://github.com/AppliedDataSciencePartners/DeepReinforcementLearning

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition* (2015).

[8] Yann LeCun, Patrick Haffner, Léon Bottou, Yoshua Bengio. *Object Recognition with Gradient-Based Learning*. Published in *Proceeding Shape, Contour and Grouping in Computer Vision*, p.319. (1999)

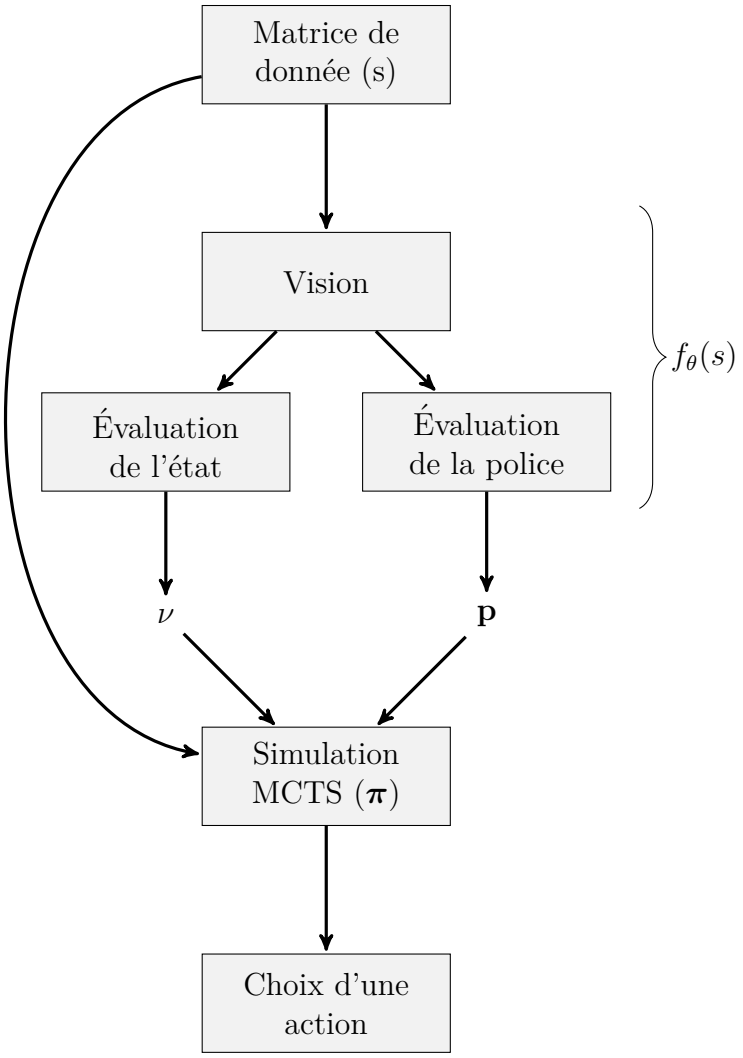
6 Annexe

La fonction qui détermine l'action de l'agent prend comme paramètre la probabilité de choisir une action a donnée l'état du jeu s ($\boldsymbol{\pi} = Pr(a|s)$) et le paramètre τ qui gouverne si l'action doit être déterministe ou stochastique : prend une valeur de 0 ou 1, dépendant du nombre de tours depuis le début de la partie (dans notre cas, 10 tours déclenchent la première conditionnelle). Le comportement stochastique est gouverné par une distribution multinomiale, avec des probabilités d'évènements associées aux probabilités de l'évaluateur MCTS.

```
def chooseAction(self, pi, tau):
 if tau == 0:
  actions = np.argwhere(pi == max(pi))
  action = random.choice(actions)[0]
 else:
  action_idx = np.random.multinomial(1, pi)
  action = np.where(action_idx==1)[0][0]

 return action
```

FIGURE 12


Matrice de donnée (s)
Vision
$f_\theta(s)$
Évaluation de l'état
Évaluation de la police
ν
$\mathbf{p}$
Simulation MCTS ($\boldsymbol{\pi}$)
Choix d'une action