# Rapport d'article - Reinforcement Learning
# MuZero

Alexandre Martin

28 février 2021

## 1 Contexte

### 1.1 Introduction : autour des échecs

En 1977 sortait le Fidelity Challenger Chess 1, considéré comme le premier jeu d'échecs électronique à être commercialisé (limité cependant aux Etats-Unis, et tiré uniquement à 1000 exemplaires qui se revendent aujourd'hui à plusieurs centaines d'euros). Cette petite révolution pour l'époque permettait de se confronter à un joueur ordinateur d'un niveau autour des 2000ELO (à titre de comparaison, un joueur débutant est aux alentours de 1200ELO et Magnus Carlsen, actuel champion du monde, dépasse les 2850ELO).

Figure 1: Fidelity Challenger Chess (1977)


Les décennies suivantes marquèrent l'explosion de la puissance de calcul, et avec elle le développement d'un nombre conséquent de méthodes permettant de toujours plus améliorer le niveau de ces joueurs d'échecs virtuel, avec comme point marquant la rencontre Kasparov vs DeepBlue (IBM) en 1996 et 1997. Pour la premère fois, un champion du monde fut vaincu par une intelligence artificielle, ouvrant ainsi la voie à une transformation du jeu d'échecs à haut niveau : aujourd'hui les plus grands joueurs préparent leurs futures parties majoritairement avec l'aide de l'ordinateur, afin de découvrir des lignes tactiques qui échapperaient à l'oeil humain.

Figure 2: Kasparov vs DeepBlue (1997)

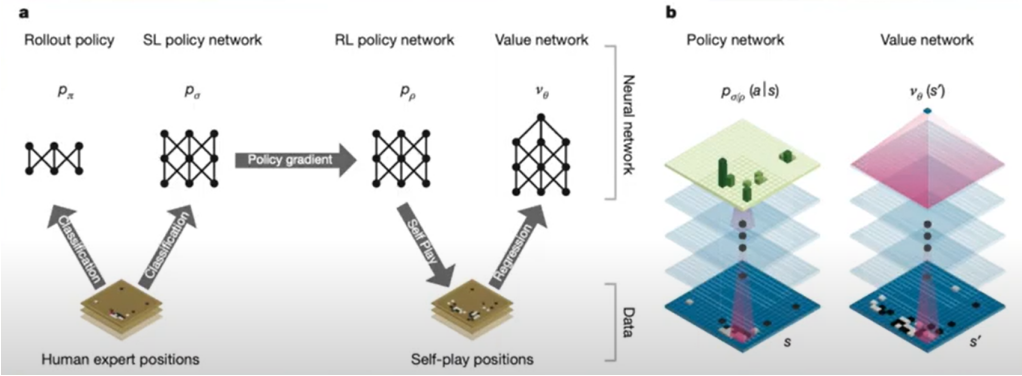
Une question intriguante demeurait cependant : supposons que l'on envoie l'un des jeux électroniques descendants de Fidelity sur une autre planète, chez une espèce qui n'a jamais entendu parler du jeu d'échecs. A partir des seules réponses de la machine aux coups joués et des signaux lumineux qu'elle envoie (clignote bleu pour un coup valide, rouge pour un coup non valide), serait-il possible de recréer les règles du jeu d'échecs dans son entièreté ? Non, pensait-on il y a quelques années, celui-ci reste bien trop complexe, les règles trop détaillées. Pourtant, MuZero, dernier né de DeepMind, semble bel et bien répondre favorablement à cette question.

## 1.2 D'AlphaGo à MuZero

DeepMind, fondée en 2010 et rachetée depuis par Google, est aujourd'hui l'une des figures dominantes du Reinforcement Learning. D'abord spécialisée dans le jeu de Go (AlphaGo fut le premier algorithme à battre un joueur professionnel de Go en 2015), ses dernières réalisations sont applicables à de nombreux jeux (Go, échecs, shogi, Atari), mais aussi à des situations en environnement réel (problème de repliement des protéines).

1 - AlphaGo (2016) utilise la recherche arborescente Monte-Carlo via 3 réseaux de neurones convolutifs déterminant la politique de l'algorithme : - les deux premiers sont entraînés par apprentissage supervisé pour copier les coups des meilleurs joueurs humains - le dernier est entraîné par des parties de l'algorithme contre lui-même La combinaison de ses réseaux de neurones est ainsi utilisée, conjointement avec un réseau séparé calculant la valeur dans la recherche arborescente, pour évaluer les positions que l'algorithme peut rencontrer. En entrée, les réseaux de neurones prennent en compte des caractéristiques directement adaptées au jeu de Go (position des pions, nombres de tours, pions capturés après un coup, ...).

Figure 3: AlphaGo




2 - AlphaGo Zero (2017) supprime l'apprentissage supervisé sur les coups des joueurs humains de haut niveau, et combine politique et valeur en un unique réseau de neurone ; la politique est ensuite mise à jour par MCTS. Le réseau de neurone ne prend ici en entrée que la position de chaque joueur, ainsi que l'identité du joueur pouvant jouer à ce tour.
2bis - AlphaZero (2018) reprend le concept d'AlphaGo Zero en adaptant les entrées pour être étendu à d'autres jeux : échecs et shogi.

Figure 4: AlphaGo Zero et AlphaZero

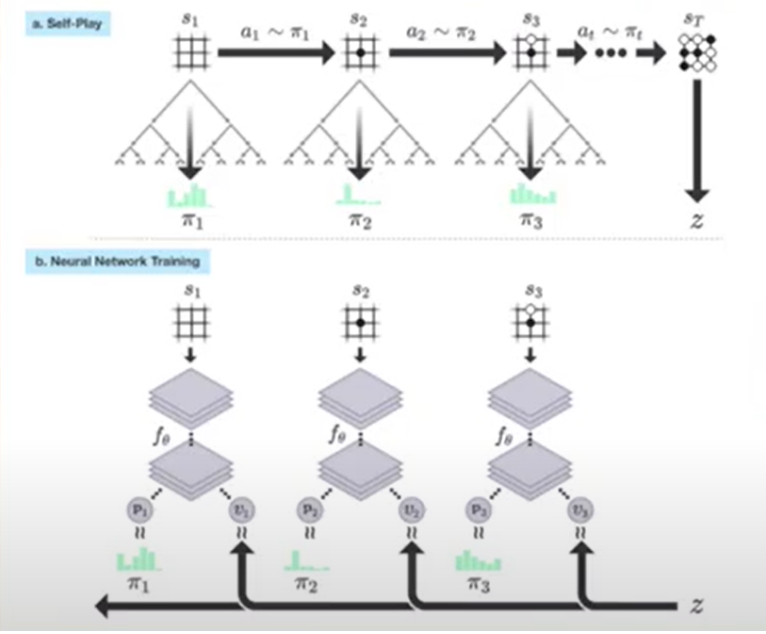
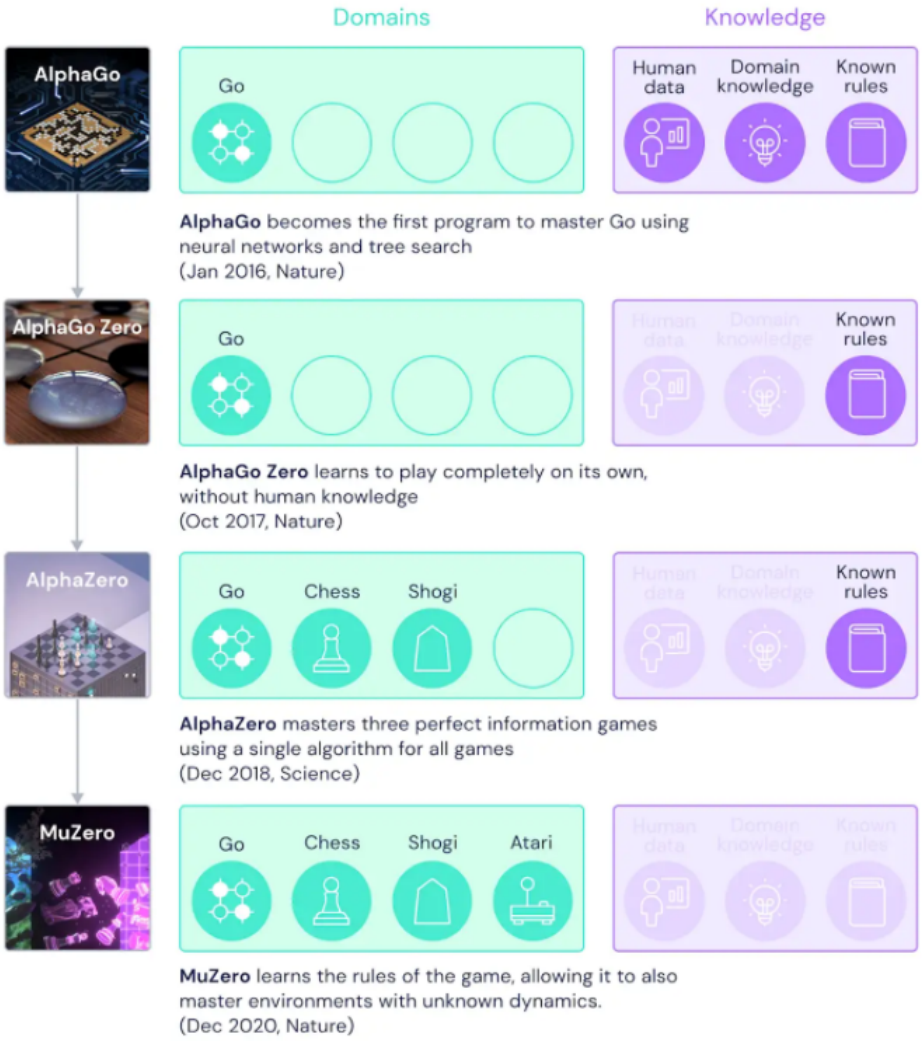


Figure 5: D'AlphaGo à MuZero




3 - MuZero (2020) généralise AlphaZero dans le sens où il apprend par lui-même les règles du jeu auquel il joue : jusqu'ici les autres algorithmes bénéficiaient d'un modèle du jeu auquel ils étaient appliqués, c'est-à-dire qu'ils savaient implicitement quels coups étaient autorisés ou non, ou encore quel était l'impact qu'un coup avait sur sur le plateau de jeu. Si cela peut paraître contre-intuitif pour les jeux comme les échecs où un modèle du jeu est relativement simple, ce concept démontre toute son efficacité dans les cas où des approches sans modèle étaient privilégiées, typiquement pour les jeux Atari. MuZero dispose ainsi d'un pouvoir de généralisation très important, puisqu'il ne nécessite pas

l'ajout d'une série de règles. Pour reprendre l'analogie de la partie précédente, on peut schématiser l'initialisation d'AlphaZero comme étant un enfant connaissant parfaitement les règles des échecs, mais n'ayant jamais joué de sa vie, tandis que l'initialisation de MuZero est un enfant n'ayant jamais entendu parler des échecs de sa vie, mais disposant d'un jeu électronique confirmant si ses coups sont valides ou non. Heuristiquement, MuZero peut ainsi privilégier en interne les règles les plus avantageuses pour lui, et en cela s'affranchir du biais humain de la connaissance de l'ensemble des règles a priori.

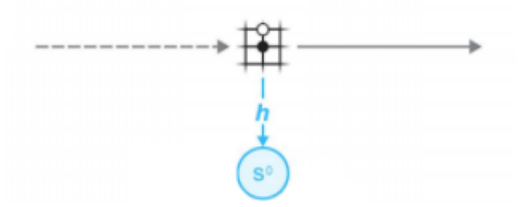
# 2 Présentation détaillée de MuZero

## 2.1 Analyse d'un état

La principale innovation de MuZero est la façon dont il prévoit les actions à venir, et la manière dont celles-ci modifient l'environnement. Jusqu'ici, les algorithmes de DeepMind connaissaient les règles des jeux auquels ils étaient confrontés, mais aussi l'impact précis qu'allait avoir les actions de l'agent : les états futurs étaient effectivement ceux possibles lors d'une partie. MuZero, en revanche, apprend par lui-même les changements de l'environnement relatifs à ses actions, via un arbre d'états latents, qui sont mis à jour au fur et à mesure de l'avancement des parties pour s'adapter à ce qu'il observe effectivement.

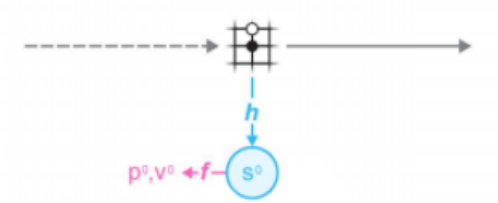
Le premier état caché $s^0$ est traduit de l'état effectif de l'environnement par un réseau de neurones $h$ : $h$ est nommé fonction de représentation.
$s^0 = h(o_1, o_2, ..., o_t)$

Figure 6: Représentation de l'état



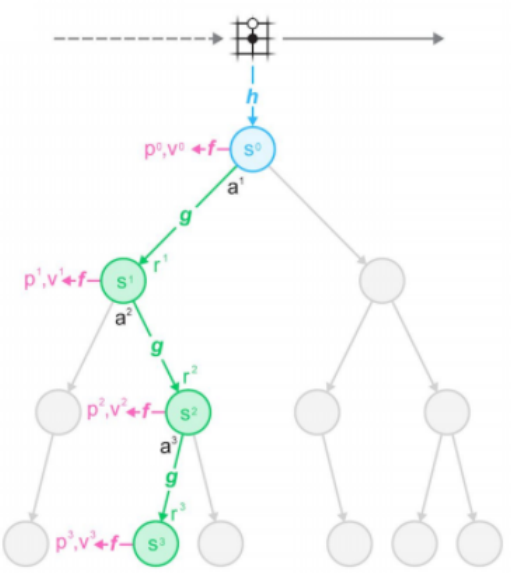

A partir de cet état caché, le modèle prédit la politique $p^0$ et la valeur $s^0$ associés via un autre réseau de neurones $f$, nommé fonction de prédiction.
$p^0, v^0 = f(s^0)$

Figure 7: Prédiction de la politique et de la valeur




Enfin, un dernier réseau de neurones $g$, la dynamique, construit les futurs états cachés $s^1, s^2, ...$ et la récompense $r$ du système à partir de l'état caché précédent et des actions envisagées, tandis que la fonction de prédiction $f$ calcule politique et valeur associée.
$r^k, s^k = g(s^{k-1}, a^k)$
$p^k, v^k = f(s^k)$

Figure 8: Itérations

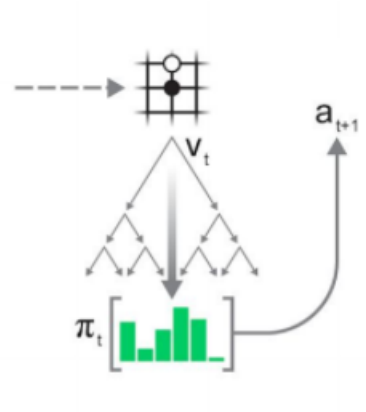



Ainsi, même sans connaître les règles de l'environnement, sans connaître la manière dont celui-ci sera affecté par ses actions, MuZero est tout de même capable de prévoir d'une certaine manière l'évolution de la partie, et par conséquent d'effectuer une recherche arborescente Monte-Carlo dans l'espace des états latents pour en déduire une politique $\pi$ et une valeur $v$ associés à l'état réel actuel.

## 2.2 Suivi d'une trajectoire et entraînement

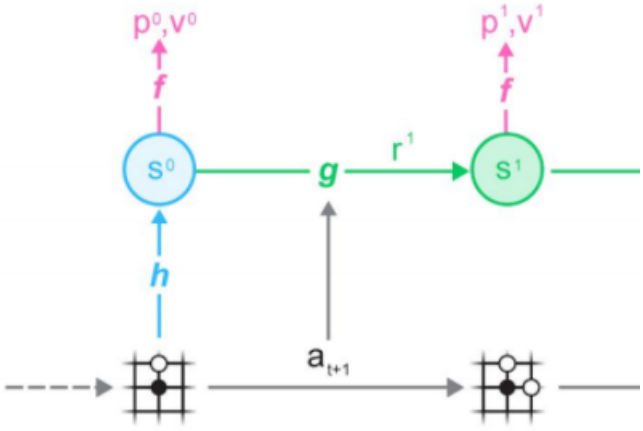
Le modèle détaillé dans la partie précédente permet ainsi de prédire la valeur associée à l'état, ainsi que la distribution de probabilité de la politique suivie. Cette dernière est modifiée par un bruit de Dirichlet d'influence décroissante au cours du temps, afin de privilégier l'exploration au début, puis l'exploitation lorsque l'estimation de la valeur devient meilleure.

Figure 9: Choix de l'action




Cette action choisie modifie alors l'environnement. Cette modification de l'environnement sous l'effet de sa dernière action est alors apprise par l'agent, qui peut ainsi adapter son espace d'états latents pour le rapprocher de ses observations : la dynamique $g$ est alors mise à jour.

Figure 10: Mise à jour du modèle de l'environnement



La mise à jour des poids des 3 réseaux de neurones est réalisée par rétropropagation. Trois quantités sont optimisées conjointement pour chaque état latent k :
- (politique) l'entropie croisée $l^p$ entre le compte des visites dans la recherche Monte-Carlo et les logits des politiques issues de la fonction de prédiction $f$.
- (valeur) l'entropie croisée, ou l'erreur quadratique moyenne, $l^v$, entre la somme réduite des récompenses moyennes et la valeur issue de la fonction de prédiction.
- (récompense) l'entropie croisée $l^r$ entre la récompense observée et celle estimée par la dynamique $g$.
Les fonctions $f$, $g$ et $h$ étant des réseaux de neurones, et donc paramétriques (de vecteur de paramètre $\theta$), on ajoute également un terme de régularisation. La quantité à optimiser est ainsi :
$l_t = \sum_{k=0}^{K}(l_k^p + l_k^v + l_k^r) + c||\theta||^2$

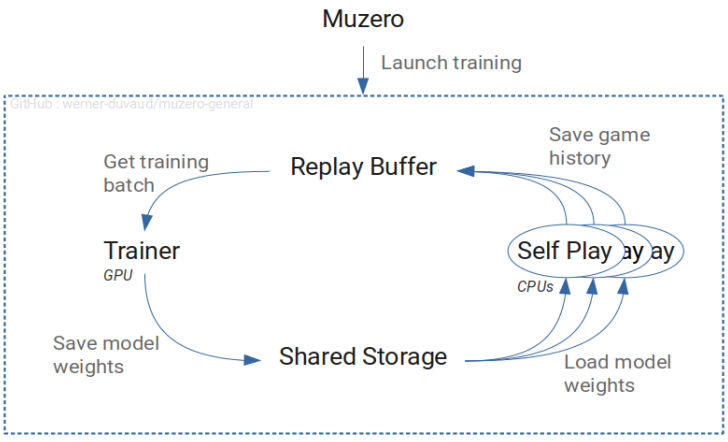
# 3 Etude d'une implémentation

Werner Duvaud - MuZero General (Python)

## 3.1 Structure de l'agorithme

L'algorithme MuZero General est divisé en quatre parties principales :
- dans le Shared Storage sont stockés les poids des réseaux de neurones $f$, $g$ et $h$
- Ces poids sont utilisés dans le Self Play, où l'agent joue des parties contre lui-même . - Ces parties sont enregistrées dans le Replay Buffer - Elles alimentent ainsi le Trainer, qui va modifier les poids des réseaux de neurones pour prendre en compte les nouvelles observations. Ces nouveaux poids sont ainsi stockés dans le Shared Storage, complétant le cycle.

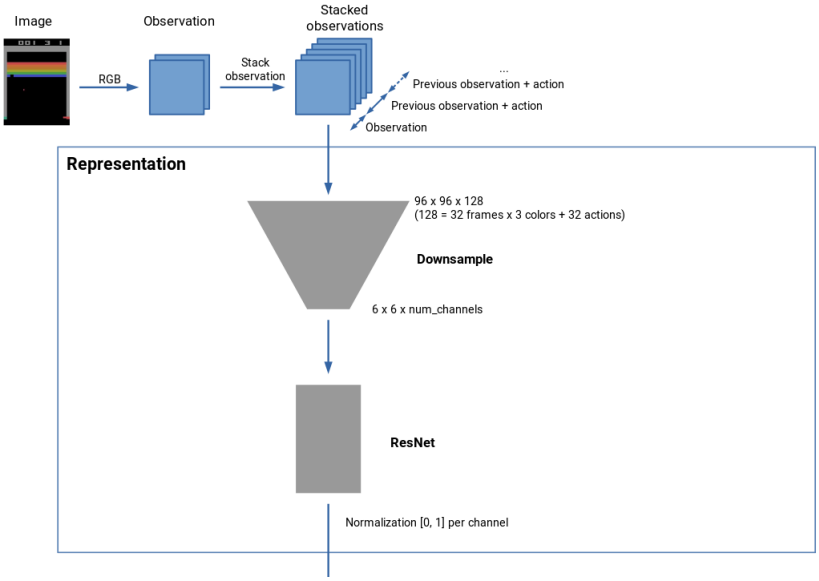
Figure 11: Structure de l'algorithme



## 3.2 Architecture des fonctions : exemple des jeux Atari

Les observations collectées sont ici les images successives du jeu, codées en RVB. Les couples (observation passée, action passée), ainsi que l'observation présente sont stockés au fur et à mesure. La fonction de représentation $h$ telle que $s^0 = h(o_1, o_2, ..., o_t)$ est ainsi composée d'un sous-échantillonage, ainsi que d'un réseau de neurones résiduel, donnant donc en sortie le premier état latent $s^0$.

Figure 12: Fonction de représentation $h$




La fonction de prédiction $f$ prend en entrée cet état latent pour en déduire la valeur $v$ et l'action $\pi$. Elle est composée d'un réseau de neurones résiduel, d'un réseau convolutif, puis d'un dernier réseau de neurones pour chaque quantité à prédire.

Enfin, la dynamique $g$ permet de calculer le prochain état caché $s^1$ ainsi que la récompense $r$ associée, à partir de l'état caché présent et de l'action envisagée. Elle est composée d'un encodage one-hot pour l'action uniquement, puis d'un réseau convolutif et d'un réseau de neurones résiduel pour calculer l'état caché $s^1$. Cet état caché passe ensuite dans un nouveau réseau convolutif et un dernier réseau de neurones calculant la récompense.

Figure 13: Fonction de prédiction $f$

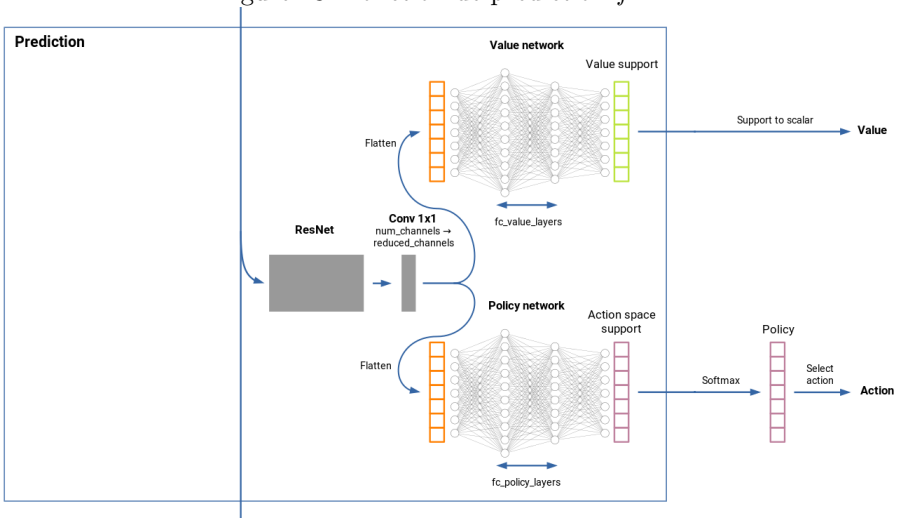
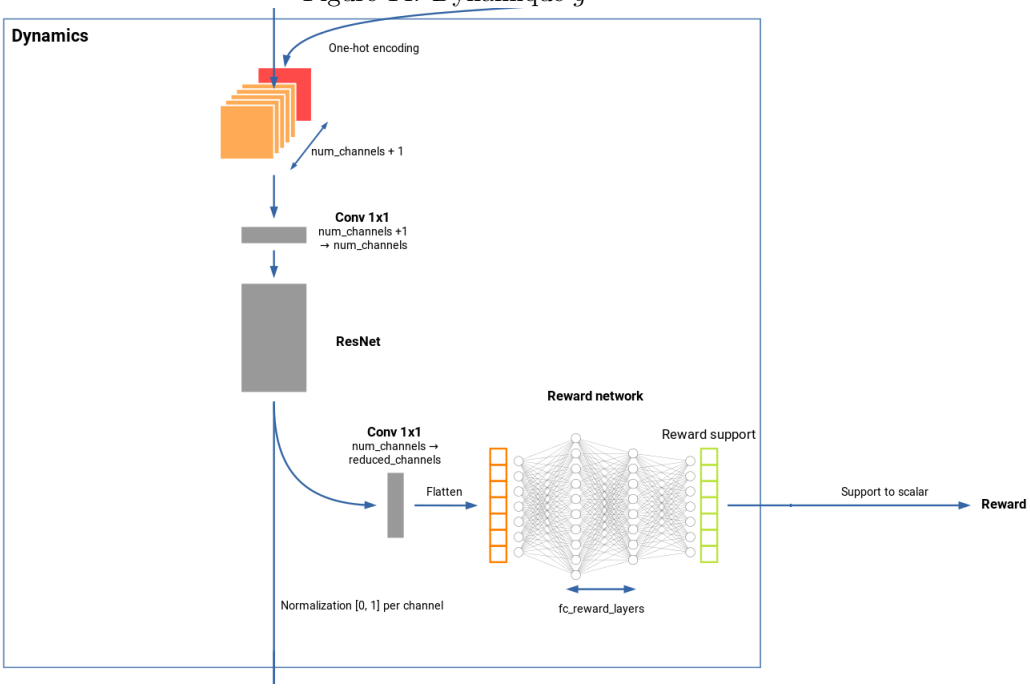



Figure 14: Dynamique $g$



## 3.3 Application au morpion

Voici les résultats obtenus après 10 minutes d'entraînement sur le jeu du morpion avec Google Colab.

Figure 15: Evolution de la récompense

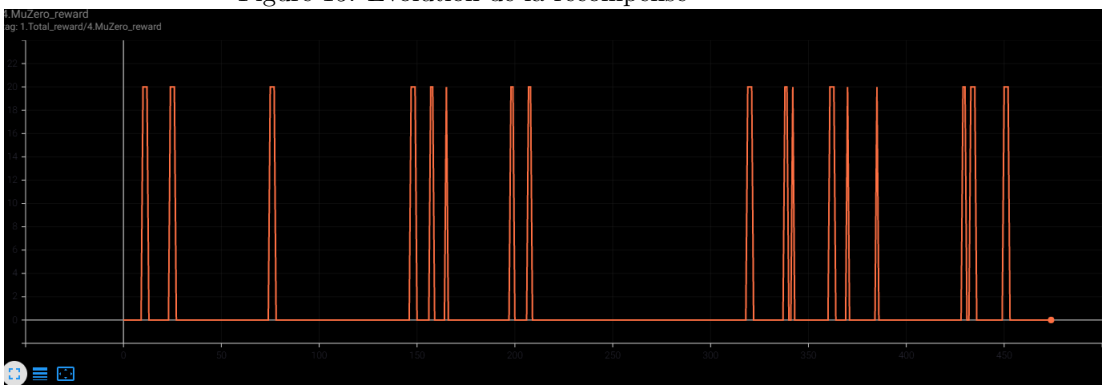
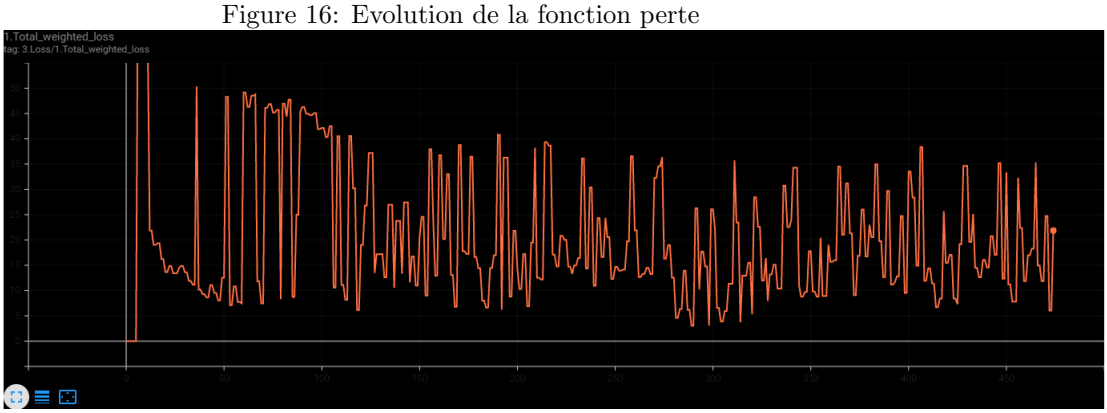



Figure 16: Evolution de la fonction perte



# 4 Applications et évolutions

## 4.1 Reinforcement Learning en situtation réelle

L'innovation de MuZero, en le sens où il ne nécessite pas de modèle de la réalité intégré, ouvre la voie à de nombreuses applications en situation réelle, où les "règles du jeu" sont beaucoup trop complexes pour être définies a priori de manière exhaustive. Aujourd'hui, par exemple, les agents associés aux voitures autonomes ou au contrôle des robots sont entraînés sur simulateur (duckietown, ...) avant d'être intégrées aux nano-ordinateurs physiques. Cependant, ce transfert de la simulation au réel peut être problématique et fait aujourd'hui l'objet de nombreuses recherches (sim2real) : on peut penser aux phénomènes de frottement, grippage des roues, ... MuZero pourrait permettre de régler ce problème en court-circuitant le simulateur, pour faire apprendre le robot directement en situation réelle. Cela risque cependant de ne pas être applicable partout en l'état : difficile d'envoyer 100 voitures autonomes non entraînées directement sur la route en leur demandant d'apprendre par elles-mêmes les règles de la conduite.

Figure 17: Environnement de simulation Duckietown

## 4.2 Digression sur les échecs

- Quelle utilité pour les joueurs d'échecs ?

Si les joueurs d'échecs de haut niveau ont pu énormément profiter de l'avancée des programmes informatiques jusqu'à un certain point, l'avancée exponentielle des performances des algorithmes à un niveau inatteignable pour un être humain pose question de leur utilité : si on ne peut retenir les variantes que sur 20 coups, est-il utile de disposer d'un programme permettant d'en calculer 40 plutôt que 30 ?
L'avancée MuZero va en fait certainement permettre de développer énormément ce qu'on appelle "échecs alternatifs" : échecs à quatre, anti-échecs (toute pièce pouvant être capturée doit l'être, la partie est gagnée si on n'a plus de pièce), échecs atomiques (toute capture déclenche une explosion qui détruit toutes les pièces autour). Notamment, MuZero peut potentiellement atteindre d'excellents résultats pour la variante dite des échecs 960 : la position des pièces y est aléatoire au début de chaque partie.
Le caractère généraliste de MuZero peut ainsi rendre possible la création de théories solides pour ces modes de jeu.

Figure 18: Echecs atomiques : la prise du cavalier fait exploser les pièces environnantes

- Comment battre MuZero aux échecs ?

La question peut paraître futile pour un humain, mais il semble qu'une réponse puisse pourtant se dégager. Dans la base de données des parties libres (pas d'ouverture imposée) jouées entre Stockfish et AlphaZero, ce dernier en a en effet perdu 3, et ces défaites présentent des similarités. Ces trois parties délivrent en effet une finale humainement nulle (au moins par répétition de coups), où AlphaZero va essayer de forcer la gain et au final perdre. Si dans le milieu de partie, AlphaZero domine Stockfishpar une meilleure "intuition", il est à même de faire des erreurs en finale par une légère erreur d'évaluation (si le gain est de 0 pour une nulle, et qu'il calcule un gain légèrement positif sur une variante, il risque de choisir cette dernière, même si elle est non gagnable). Il est toutefois important de noter que ces défaites ont eu lieu dans des situations où

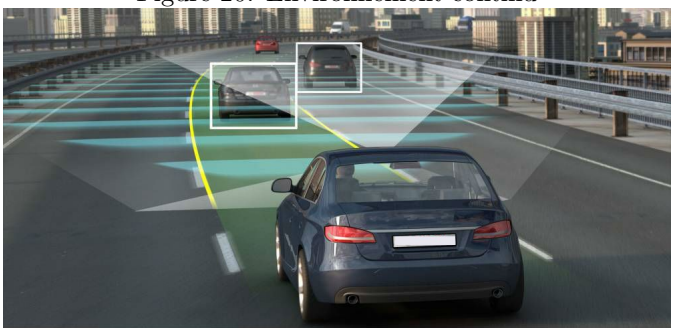
AlphaZero disposait d'un temps de calcul très restreint par rapport à Stockfish. Comme l'algorithme de recherche Monte-Carlo de MuZero est similaire à celui d'AlphaZero, il est raisonnable de penser qu'il peut être victime des mêmes biais.
En bref, pour battre MuZero aux échecs, jouez pour la nulle !

Figure 19: Position où AlphaZero (blanc) essaye de forcer le gain avec Cc3 et perd, alors que Cc1 était une nulle claire.

## 4.3 Aller plus loin ?

Toujours dans un but de généralisation maximale, certains cas non pris en compte par MuZero pourront l'être dans de futures innovations :
- l'environnement est pour le moment supposé déterministe, discret et stationnaire : chacun de ces points pourra faire l'objet d'un axe de recherche.
- l'information est ici disponible de manière complète et parfaite : il reste à introduire un algorithme prenant en compte une information bruitée ou en partie cachée.
- il pourrait être intéressant d'intégrer une valeur de confiance dans le modèle appris par l'agent selon sa concordance avec les retours effectifs de l'environnement.

Figure 20: Environnement continu



# 5 Références

Article :
Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model (Julian Schrittwieser, Ioannis Antonoglou, et al. arXiv 2019)

Blogs :
Deepmind - MuZero
Julian Schrittwieser - MuZero Intuition
Medium - MuZero : The Walkthrough
Chessbase - MuZero figures out chess, rules and all
Mathieu Acher - MuZero : A new revolution for Chess ?

Youtube :
Julian Schrittwieser - MuZero (NeurIPS)
Julian Schrittwieser - MuZero (ICAPS)
Henry AI Labs - MuZero
Henry AI Labs - The Evolution of AlphaGo to MuZero
Blitzstream - Comment AlphaZero perd ses parties d'échecs ?
ainsi que les commentaires sous ces vidéos.

GitHub :
Denny Britz - Reinforcement Learning
Werner Duvaud - MuZero General