Apprentissage par renforcément du jeu Super Mario Bros avec un Duel DQN

Raoufdine Said **Noauffal Abdullatief** **Mohamed Elazzouzi** **Ilyes Korichi**

Department d'Informatique
Université Lyon 1, Claude Bernard
Villeurbanne 69100, FR

{raoufdine.said, noauffal.abdullatief, mohamed.el-azzouzi, ilyes.korichi}@etu.univ-lyon1.fr

***Résumé*- Cet article présente notre projet de deep learning consistant à utiliser l'apprentissage par renforcement pour jouer au jeu Super Mario Bros en utilisant une technique de Duel DQN. Nous avons basé notre travail sur le code de notre TP d'apprentissage par renforcement, ainsi que sur des codes sources disponibles sur des repositories Github sur l'apprentissage par renforcement pour le jeu Super Mario Bros. Grâce à ces ressources, nous avons réussi à mettre en place une implémentation fonctionnelle de ce modèle. Après 171 heures d'entraînement et plus de 29000 épisodes, notre modèle a réussi à atteindre le niveau 3 du jeu.**

Key word – Super Mario Bros, Duel DQN, apprentissage par renforcement, PyTorch, deep learning

I. INTRODUCTION

L'intelligence artificielle (IA) est un domaine de la recherche qui vise à créer des systèmes informatiques qui peuvent effectuer des tâches qui normalement nécessitent une intelligence humaine, comme la reconnaissance de la parole, la compréhension de la langue, la résolution de problèmes et la prise de décisions. Il existe plusieurs sous-domaines dans l'IA, notamment la reconnaissance de la parole, la vision par ordinateur, la robotique et l'apprentissage automatique.

L'apprentissage automatique (ou apprentissage statistique) est un sous-domaine de l'IA qui se concentre sur la création de systèmes informatiques qui peuvent apprendre à partir de données, sans être explicitement programmés. Il y a deux types d'apprentissage automatique : l'apprentissage supervisé et l'apprentissage non supervisé.

L'apprentissage profond (ou deep learning) est un sous-domaine de l'apprentissage automatique qui utilise des réseaux de neurones profonds (deep neural networks) pour effectuer des tâches d'apprentissage automatique. Les réseaux de neurones sont des modèles mathématiques inspirés du fonctionnement du cerveau humain, qui sont utilisés pour traiter des données de manière automatique. Les réseaux de neurones profonds sont des réseaux de neurones avec des couches cachées supplémentaires, ce qui leur permet de traiter des données plus complexes. Les réseaux de neurones profonds sont utilisés dans des tâches telles que la reconnaissance de la parole, la vision par ordinateur et l'apprentissage par renforcement.

Pour notre projet de deep learning, nous avons décidé d'implémenter un Duel DQN qui joue à Super Mario Bros. Le but était de pouvoir créer un système capable de naviguer efficacement dans les niveaux de jeux en prenant des décisions adaptées en fonction de l'environnement.

Un Duel DQN (Deep Q-Network) est un type de modèle d'apprentissage automatique utilisé pour résoudre des tâches d'apprentissage par renforcement. Il est basé sur un réseau de neurones appelé Q-network, qui est utilisé pour estimer la valeur future d'une action dans un état donné. Le Duel DQN se distingue du DQN standard par l'utilisation de deux réseaux de neurones distincts pour estimer la valeur de chaque action et la valeur de l'environnement, plutôt qu'un seul réseau de neurones pour estimer les deux *[1]*.

Un Duel DQN par rapport à un DQN standard permet une meilleure corrélation des données d'entraînement, une meilleure stabilité, une meilleure convergence et des meilleures performances. Enfin un Duel DQN est plus facile à adapter pour des tâches spécifiques.

Ce rapport décrit notre implémentation et les résultats que nous avons obtenus en utilisant un Duel DQN pour apprendre à jouer au jeu Super Mario Bros. Nous décrirons en détail les étapes de notre projet, de la configuration de l'environnement à l'entraînement et à l'évaluation de notre modèle. Nous présenterons également les résultats obtenus et les comparaisons avec les résultats obtenus avec un DQN standard. Enfin, nous discuterons des limites de notre approche et des pistes de recherche pour les améliorations futures.

II. MÉTHODOLOGIE

A. Environnement de jeu du projet

Pour notre projet, nous avons utilisé un environnement de simulation OpenAI Gym pour les jeux Super Mario Bros. & Super Mario Bros. 2 sur la Nintendo Entertainment System (NES) en utilisant l'émulateur nes-py. Cet environnement

nous a permis de configurer facilement les paramètres de jeu et de collecter les données de l'état de jeu pour l'entraînement de notre modèle. Il nous a également permis de tester facilement notre modèle en utilisant différents niveaux de jeux. Cet environnement était une solution idéale pour notre projet car il était facile à utiliser et offrait une grande flexibilité pour configurer les paramètres de jeu.

B. *Méthodologie d'entraînement et de test du modèle*

L'environnement OpenAI Gym utilisé pour notre projet offre une variété de méthodes facilitant la gestion de l'environnement de jeu tels que la possibilité de réinitialiser le jeu, de récupérer l'état courant, la récompense obtenue (reward), le statut de fin d'épisode et des informations détaillées sur les actions effectuées telles que le nombre de pièces récupérées, le score, le niveau et bien d'autres informations pertinentes pour notre étude.

Afin de maximiser les performances de notre modèle, nous avons créé un environnement personnalisé en utilisant un "wrapper" sur l'environnement de base fourni par OpenAI Gym. Ce wrapper nous a permis d'utiliser des actions plus complexes pour notre modèle, de limiter le nombre de frames consécutives où l'IA ne prend pas d'action à 30, de sauter des frames inutiles, de normaliser les frames pour une meilleure analyse de l'état de jeu, et de ne conserver que les 4 dernières frames les plus pertinentes (le modèle ne voit que les 4 dernières frames les plus pertinentes).

Notre méthode d'entraînement consiste à faire jouer notre modèle sur des milliers d'épisodes successifs. Pendant ces épisodes, le modèle apprend de ses erreurs en utilisant un Duel DQN. Pour éviter de se retrouver dans un minimum local, l'agent joue de manière aléatoire avec une probabilité *epsilon* qui débute à 1 et qui diminue progressivement jusqu'à atteindre 0,001. Cela permet de découvrir de nouveaux états et ainsi d'optimiser les performances de notre modèle.

Le test du modèle consiste à faire jouer notre modèle à un épisode de jeu. Pendant cet épisode, notre modèle prédit toutes les actions à effectuer en utilisant les informations de l'état de jeu. Nous avons également affiché en temps réel les frames du jeu après chaque action effectuée pour pouvoir visualiser les performances de notre modèle sous forme d'animation. Cela nous a permis de voir comment notre modèle interagissait avec l'environnement de jeu, et de déterminer si les performances étaient satisfaisantes.

C. *Hyperparamètres*

Dans ce projet de Duel DQN, nous avons fixé le nombre maximal d'épisodes pour l'apprentissage à 1000000. Le paramètre gamma, qui est utilisé pour pondérer l'importance des récompenses futurs, a été fixé à 0.99. La vitesse d'apprentissage (learning rate) a été fixée à 0.0001 et la taille du lot (batch size) utilisée pour l'entraînement a été fixée à 256. Nous avons utilisé une replay memory de taille 50000 pour stocker les expériences passées de l'agent. Les poids des réseaux de neurones ont été sauvegardés tous les 50 épisodes sur le disque dur. Comme fonction de loss nous avons utilisé la Smooth L1-loss qui peut-être interprétée comme une combinaison de la L1 loss et de la L2 loss, elle se comporte comme une L1 loss lorsque la valeur absolue de l'argument est élevée et comme une L2 loss lorsque la valeur absolue de l'argument est proche de zéro, elle combine les avantages de la L1 loss (gradients constants pour de grandes valeur de x) et de la L2 loss (moins d'oscillations lors des mises à jour lorsque x est petit).

D. *Agent et modèle*

Dans ce projet, nous avons développé une classe Agent qui utilise une replay memory et deux copies d'un même modèle (*q* et *q_target*), la loss est calculée sur les résultats de ses deux réseaux de neurones.

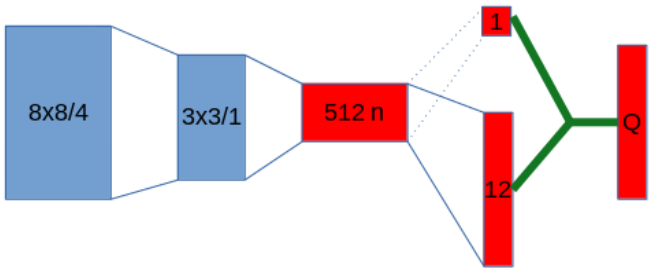

Figure 1: Duel DQN du projet

Le modèle (Figure 1) est constitué d'une première couche de convolution dont la taille des filtres est de 8x8 avec un pas de 4 et d'une deuxième couche dont la taille des filtres est de 3x3 avec un pas de 1. Ensuite viens une couche fully connected de 512 neurones, puis une première couche de sortie avec autant de neurones que de nombre d'actions possibles (12 neurones) et une deuxième sortie d'un neurone. Les valeurs de sorties de ses deux dernières couches sont additionnées pour produire la prédiction finale Q (tableau de 12 valeurs).

Pour l'entraînement, la fonction train de l'agent est appelée. Elle commence par remplir la replay memory en jouant aléatoirement. Une fois que celle-ci est suffisamment remplie (2000 entrées), l'agent utilise ses réseaux de neurones pour prédire la prochaine action. Après chaque prédiction, l'agent appelle sa fonction d'optimisation qui calcule la loss entre la prédiction de *q_target* et celle de *q* sur un lot. Ensuite, les nouveaux poids des neurones sont calculés en utilisant la rétropropagation du gradient.

L'agent possède également des fonctions pour charger les poids des réseaux de neurones enregistrés sur le disque dur et pour jouer un épisode en utilisant ses réseaux de neurones entraînés. Ces méthodes permettent de continuer l'apprentissage à partir des poids sauvegardés et de tester les performances de l'agent dans des situations réelles.

III. Résultats et discussion

A. *Résultats*

En observant les résultats présentés dans la figure 2, nous constatons que l'agent atteint le deuxième niveau du jeu après environ 1500 épisodes et atteint le troisième niveau pour la première fois après environ 13000 épisodes. À partir de l'épisode 19000, l'agent atteint régulièrement le niveau 3, ce qui représente une performance remarquable par rapport à un DQN standard.

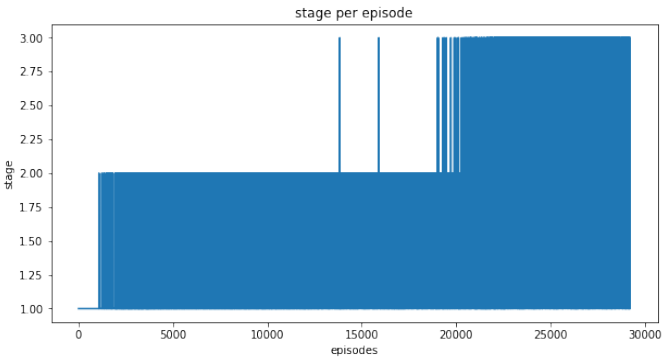

Figure 2: Score par épisode de l'agent

La figure 3 montre une progression rapide du reward au début de l'apprentissage, passant d'environ 1000 pour les premiers épisodes à plus de 4000 autour de l'épisode 2500. Par la suite, le reward a stagné jusqu'à environ l'épisode 19000, où il a augmenté puis stagner à nouveau.

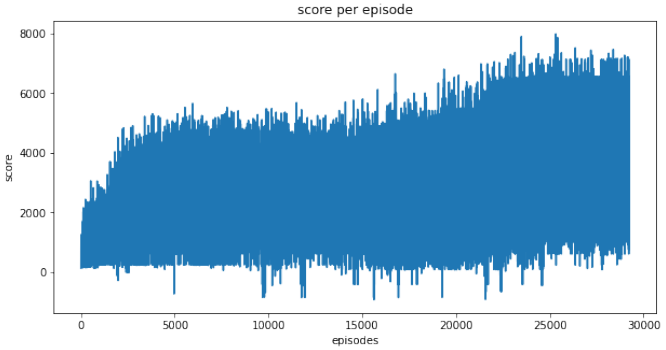

Figure 3: Reward par épisode de l'agent

B. *Comparaison à un DQN standard*

En comparant les résultats présentés dans la Figure 4, nous pouvons constater que le Duel DQN obtient un reward deux à quatre fois plus élevé qu'un DQN standard lors des 10000 premiers épisodes. C'est pour cette raison que nous avons choisi d'utiliser un Duel DQN pour notre projet, car il offre non seulement une simplicité d'implémentation, mais également des performances remarquables.

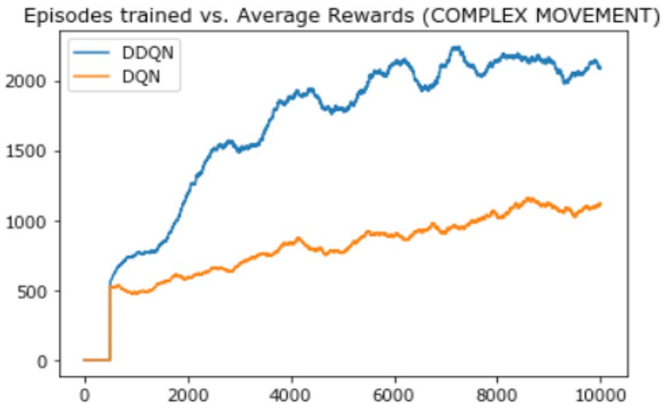

Figure 4: Comparaison duel DQN et DQN standard

C. *Limites du Duel DQN*

Les principals limites de notre Duel DQN sont la taille de la replay memory, le temps d'entraînement et la stabilité d'apprentissage.

Notre replay memory occupe environ 8 Go de mémoire vive, le temps d'entraînement est relativement long, après 171 heures nous avons juste atteint l'épisode 29000 et l'agent n'arrive qu'au niveau 3 et enfin nous constatons que l'apprentissage est assez instable, lorsque notre réseau de neurones fait face à une nouvelle expérience, il arrive souvent qu'il oublie les progrès qu'il a réalisés jusque-là, mais il réapprend rapidement et en généralisant mieux.

D. *Pistes d'amélioration*

Il existe d'autres techniques d'apprentissage par renforcement plus avancées comme les algorithmes A3C, PPO, DDPG pour améliorer les performances de l'agent mais nous avons préféré utilisé un Duel DQN une amélioration du DQN standard car nous avons vu ce dernier en classe.

IV. Conclusion

En conclusion nous avons réussi à faire une implémentation efficace et un code lisible.

La principale difficulté était dans la personnalisation et l'adaptation des codes sources qu'on avait accés. Nous avons surmonter cette difficulté grâce à notre travail d'équipe et notre persévérance.

References

[1] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot , Nando de Freitas. 2016. "Dueling Network Architectures for Deep Reinforcement Learning"

[2] Jiseong Han, 2020, https://github.com/jiseongHAN/Super-Mario-RL