DEEP Q LEARNING
APPLIQUE A SUPER MARIO BROS

Kulcsar Jeremy, Fayolle Lucas
Option OSY - Deep Learning
Ecole Centrale Paris - 2020

Abstract – Dans ce projet, nous allons coupler les techniques de Q-learning à celles du Deep Learning afin de développer une intelligence artificielle qui apprendra à jouer à Super Mario Bros. Le réseau sur lequel l'agent apprendra sera convolutif et prendra en entrée plusieurs frames consécutives du jeu afin de déterminer la meilleure action pour l'état suivant. L'agent (Mario) développera ainsi une stratégie de déplacement permettant la complétion autonome des niveaux du jeu.

I – INTRODUCTION

L'utilisation de l'intelligence artificielle dans les jeux vidéo n'est pas une chose nouvelle. Le premier exemple d'IA fut un agent résolvant un jeu de Nim numérisé, développé en 1951. Depuis, les technologies pour rendre les IA plus performantes n'ont cessé d'être développées et/ou améliorées, jusqu'à aboutir à des utilisations avancées du Machine Learning et de ses dérivées.

La branche du Machine Learning que nous allons étudier lors de ce projet est le Reinforcement Learning (Apprentissage par Renforcement). Il s'agit de la branche la plus populaire dans le monde du jeu vidéo autonome : des jeux de plateforme jusqu'aux jeux de plateaux comme les échecs, le Reinforcement Learning est la base de l'apprentissage des IA qui terminent un niveau et/ou jouent contre un joueur.

Nous allons travailler sur le jeu vidéo Super Mario Bros. Nous avons eu l'idée de se baser sur le Q-Learning et de le coupler au Deep Learning afin de garantir le développement d'une IA « intelligente » capable d'interpréter l'environnement et de prendre des décisions sans connaître au préalable les différents états de l'environnement.

II – ENVIRONNEMENT TECHNIQUE

Super Mario Bros est un jeu de plateforme dans lequel Mario interagit avec son environnement. Il peut se déplacer vers la droite, la gauche et sauter. Pour terminer le jeu, il doit se déplacer vers la droite jusqu'à la fin d'un niveau sans mourir (toucher un ennemi ou tomber dans un trou). Mario doit donc avancer en esquivant les différentes menaces comme montré sur la Figure 1.

Il s'agit ainsi d'une représentation agent / environnement, l'agent étant Mario et l'environnement étant le niveau. L'agent peut alors interagir avec l'environnement selon une liste de 7 actions légales :

1 – Rester immobile

2 – Marcher à droite

3 – Courir à droite

4 – Marcher à gauche

5 – Courir à gauche

6 – Sauter à droite

7 – Sauter à gauche

Ainsi, à chaque instant t, Mario se trouve dans un état s_t qui résulte d'une action a_{t-1} prise depuis l'état s_{t-1} (par exemple, « courir à droite » ou « sauter »).

Figure 1 : Image de Super Mario Bros

III – Q-LEARNING

L'agent apprend une stratégie de choix d'action optimale à chaque état s_t en utilisant le Q-learning.

Pour que l'agent détermine quelle est la meilleure action, chacune est associée à une récompense r appelée reward. L'agent doit alors apprendre à maximiser la somme des récompenses qu'il reçoit au cours de son interaction avec l'environnement (dans notre cas, elle s'arrête lors de la mort de Mario ou de la complétion du niveau).

Le principe sera, sachant l'état actuel, de trouver l'action qui donnera la plus haute récompense immédiate tout en maximisant les récompenses obtenues par la suite.

Mathématiquement, cela s'écrit avec une valeur appelée Q-value, définie telle quelle :

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max Q(s_{t+1}, a_{t+1})$$

Où $r(s_t, a_t)$ est la récompense engendrée par l'action a_t depuis l'état s_t.

Et γ est un facteur entre 0 et 1 permettant de nuancer les récompenses obtenues des états suivants, en pratique choisi à 0,7.

Quand Mario arrive dans un état terminal (mort ou niveau terminé) alors :

$$Q(s_T) = r(s_T)$$

L'agent apprend alors à maximiser sa Q-value qui correspond à la somme des récompenses.

Notions $a_{optimale}$ l'action optimale. Sa politique de jeu est alors :

$$a_{optimale} = \text{argmax}\ \{Q(s_t, a_t),\ a_t\ action\ légale\}$$

Le choix de la reward attribuée à l'agent est primordial puisqu'il conditionne son apprentissage Ici, nous souhaitons que l'agent apprenne à terminer un niveau le plus rapidement possible, sans considération pour le score. Ainsi, nous valorisons les déplacements vers la droite qui contribuent à la complétion du niveau et nous pénalisons les déplacements vers la gauche ainsi que les morts.

La reward est ainsi calculée comme suit :

$$r(s_t, a_t) = v_t + c_t + w_t$$

Où v_t est la composante vitesse qui vaut 2 si Mario s'est déplacé vers la droite, 0 s'il est resté immobile et -2 s'il s'est déplacé vers la gauche.

c_t est la composante temps, qui vaut -1 si le timer a avancé d'une seconde et 0 sinon.

Et w_t qui vaut -100 si Mario est mort et 0 sinon.

L'agent est donc encouragé à se déplacer vers la droite sans mourir, dans l'objectif de terminer chaque niveau le plus rapidement possible.

IV – DEEP Q LEARNING

Le Deep Q Learning est l'application du Deep Learning au Q Learning. Cette application permet d'aller au-delà des limites du Q Learning, qui sont rencontrées dans notre projet. La représentation de toutes les permutations possibles du monde de Mario génèrerait un espace d'états-actions trop grand pour les puissances calculatoires de notre machine, et finirait de toutes façons par établir un mapping déterministe de l'univers. Nous avons donc décidé d'utiliser une approche de Deep Q Learning, représentée et comparée à une approche de Q Learning traditionnelle sur la figure 2.

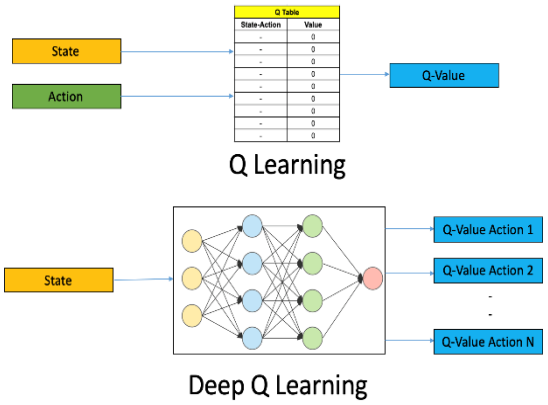

Figure 2 : Représentations des deux méthodes

Commençons par décrire les Deep Q Networks. Un DQN est un réseau qui renvoie un vecteur de valeurs d'actions, basé sur l'état et les paramètres du réseau. Il peut être représenté comme une fonction de $\mathbb{R}^n$ vers $\mathbb{R}^m$, avec n la dimension de l'espace des états en entrée et m la dimension de l'espace des actions.

Le DQN est alors un approximateur de la table des Q-values présentée Figure 2. Celui-ci apprend à estimer les récompenses futures depuis l'état s_t par différence temporelle avec ses propres prédictions dans les états $s_{t+1}, s_{t+2}, \ldots, s_T$.

La différence temporelle est calculée en prenant la MSE entre la Q-value renvoyée et une Q-value prédéfinie, dite Q-value target. Cela définit la loss function utilisée pour apprendre.

On obtient alors l'implémentation algorithmique du Q-Learning :

$$Q_t \leftarrow Q_t + \alpha\left[r_{t+1} + \gamma \max Q_{t+1} - Q_t\right]$$

La partie en bleu représente la Q-value target. Dans le cadre du Deep Q learning, le réseau apprend à prédire ses valeurs futures modulo les récompenses r reçues au cours du temps.

L'un des points primordiaux est donc l'utilisation d'un « Target Network ». Il s'agit d'un réseau identique à celui décrit précédemment, mais avec des poids fixés. Ce réseau sert à déterminer les targets lors de l'apprentissage, afin de calculer la loss function. Les poids du Target Network sont mis à jour à partir du réseau entraîné toutes les C étapes afin de maintenir une target évolutive et améliorée, permettant de se « souvenir » des chemins déjà parcourus par Mario. Le principe d'utilisation du Target Network est représenté sur la Figure 3.

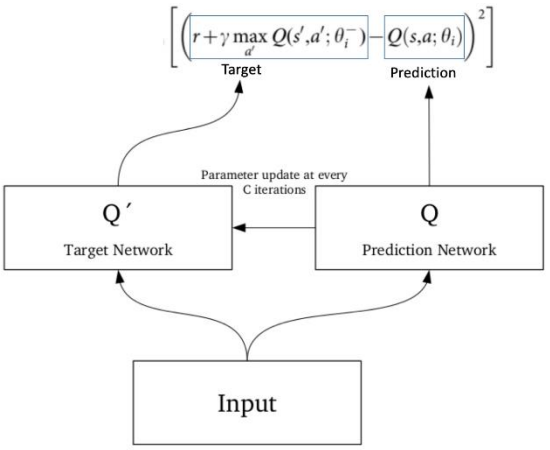

Figure 3 : Principe du Target Network

Pour résumer, les étapes d'un apprentissage par Deep Q Learning sont :

1. Le réseau calcule les Q-values associées à chaque action à partir d'une entrée donnée (ici des images du jeu)

2. L'algorithme sélectionne l'action qui correspond à la Q-value maximale

3. L'agent exécute cette action depuis l'état s_t et atteint un état s_{t+1} : il gagne une reward r. Cette transition est stockée dans un batch d'apprentissage.

4. Le réseau exécute cette étape N fois (ici $N = 1000$) pour remplir le batch d'apprentissage et calculer la loss function :

$$Loss = \frac{1}{N}\sum\left(r + \gamma \max Q_{predict} - Q_{target}\right)^2$$

5. Exécution de la descente du gradient à partir des paramètres actuels pour diminuer la loss function.

6. Après C étapes, les poids du réseau de prédiction sont copiés vers le Target Network.

7. Répéter les actions de 1 à 6.

IV – CHOIX DU MODELE

L'image du jeu dans l'état s_t ne contient pas l'ensemble des informations. En effet, il n'est pas possible, à partir de cette seule image, de déterminer les trajectoires de Mario et des différents ennemis. Ainsi, nous avons décidé d'utiliser les images des 4 derniers états comme entrée de notre réseau de neurones. Par ailleurs, une architecture convolutive a été choisie pour ses performances connues dans le domaine de reconnaissance d'images et de formes. L'architecture du CNN mis en place est la suivante :

1 – Input : 4 images downscalées en 100*100 greyscale

2 – Couche cachée n°1 : 32 filtres de taille 8*8 avec une stride de 4

3 - Couche cachée n°2 : 64 filtres de taille 6*6 avec une stride de 2

4 – Couche cachée n°3 : 128 filtres de taille 4*4 avec une stride de 1

5 – Couche cachée n°4 : 256 filtres de taille 3*3 avec une stride de 1

6 – Couche cachée n°5 : Couche dense de taille 512

7 – Couche de sortie : Couche dense qui renvoie le vecteur des Q-values associées à chaque action (7 au total)

Le taille du réseau a été limitée pour des soucis de performance d'apprentissage, mais aussi pour limiter l'overfitting.

V – APPRENTISSAGE

V.1 Apprentissage

L'apprentissage est réalisé uniquement par exploration de l'agent dans l'environnement, sans aucun set de données.

Pour la suite nous appellerons « step » la réalisation d'une action par l'agent qui permet de passer d'un état s_t à un état s_{t+1}.

L'apprentissage est effectué chaque 1000 steps, sur les données recueillies lors des états parcourus par

l'agent. L'environnement est réinitialisé à chaque mort de Mario.

Afin d'éviter un apprentissage séquentiel, les batchs sont mélangés aléatoirement pour chaque descente du gradient. Nous travaillons par ailleurs avec des mini-batchs de taille 32 pour accélérer l'apprentissage.

L'apprentissage a duré 48h sur une Nvidia GTX 1060. L'agent a fait près de 10 millions de steps pour un total de 15 000 parties.

V.2 Exploration

Pour maximiser l'exploration de l'agent et minimiser le risque d'overfitting, un paramètre d'exploration $e = 0.15$ a été choisi. L'agent choisit normalement l'action qui maximise la Q-value. Ici, l'agent peut effectuer une action aléatoire avec la probabilité e. Cela permet d'augmenter le nombre d'états visités par l'agent et de sortir des minimas locaux lors de l'apprentissage. Au fur et à mesure, ce paramètre a été diminué jusqu'à atteindre la valeur minimale $e_{min} = 0.03$.

V.3 Optimiseur

L'optimiseur Adam a été choisi. Il permet d'utiliser un taux d'apprentissage propre à chaque paramètre et donc de favoriser les neurones les plus activés. La règle de mise à jour des poids selon est Adam est donnée par :

$$\theta \leftarrow \theta - \frac{\alpha}{\sqrt{\hat{v}_t} + \eta} \hat{m}_t$$

Où $\hat{m}_t$ est le gradient moyen et $\hat{v}_t$ la variance (ou le gradient au carré moyen), et $\eta = 10^{-8}$.

VI -RESULTATS ET PERFORMANCE

Dans le cadre du Deep Q Learning, il est difficile d'établir les conventionnels Train, Validation et Test sets puisque le modèle apprend par exploration et sans jeu de donnée. Par ailleurs, entraîner le modèle sur un niveau et le tester sur un autre niveau est peu intéressant étant donné que deux niveaux présentent des difficultés largement différentes.

Ainsi, nous utiliserons comme métrique d'évaluation la Q-value générée par l'agent lors d'une partie. Cette métrique est indépendante pour chaque partie car le paramètre d'exploration place l'agent dans des états inexplorés. Cela permet de créer virtuellement un test indépendant de l'apprentissage.

Le Figure 4 montre l'évolution de la Q-value moyenne par partie au cours de l'apprentissage. La moyenne est prise pour réduire le bruit du au paramètre d'exploration aléatoire. Bien que les résultats ne soient pas à la hauteur des attentes, ils montrent bien que l'agent a été capable de retenir des stratégies efficaces améliorant sa Q-value. Pour donner un point de référence, la complétion du niveau 1 correspond à une Q-value d'environ 2000.

En pratique, l'agent est capable de comprendre les notions basiques comme sauter par-dessus un obstacle, un ennemi ou un trou. La plus grande difficulté rencontrée par l'agent est le manque d'information. En effet, Super Mario Bros est un jeu à information incomplète puisque l'agent ne peut observer que dans un certain périmètre autour de lui. Ainsi, la majorité des morts sont dues à un saut qui amène l'agent dans un trou ou un ennemi qui n'était pas encore visible lors du choix de l'action « sauter » par l'agent.

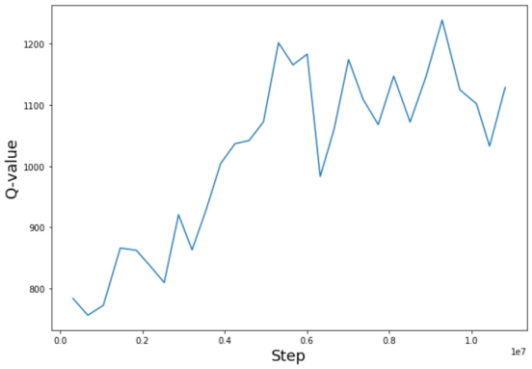

Figure 4 : Q-value obtenue (moyennée toutes les 50 parties) lors de l'apprentissage

Il semblerait qu'à partir de 6 millions de steps, la Q-value moyenne stagne autour de 1150. Il est difficile d'estimer si le modèle a atteint son potentiel maximum ou s'il s'agit simplement d'un plateau de la fonction de coût et que continuer l'apprentissage permettrait d'améliorer les performances de l'agent.

VII – CONCLUSION

Nous avons pu développer un agent indépendant capable d'apprendre à finir le premier niveau du jeu en se basant sur les techniques du Deep Q Learning. On remarque que même si l'agent arrive à atteindre

la fin du niveau 1 et à aller jusqu'à la moitié du niveau 2, la Q-value moyenne montre que la majorité des parties se terminent aux 2/3 du niveau 1.

Ce qui rend ce jeu difficile à apprendre pour l'agent est le fait qu'il s'agisse d'un jeu à horizon long et à information incomplète. Le Deep Q Learning performe en effet au mieux pour les jeux à récompense immédiate et à court horizon.

Voici quelques pistes qui pourraient être explorées pour améliorer le résultat :

- Limiter la reward à un intervalle, par exemple -1 et 1, afin de contrôler la Q-value et éviter de trop grandes différences entre deux états

- Permettre à l'agent de voir l'ensemble du niveau pour permettre une vision globale au réseau et ainsi une meilleure anticipation

- Apprendre plus longtemps, si l'on suppose que la Q-value atteinte ne correspond pas au potentiel maximal du modèle

NOMENCLATURE

s_t : Etat à l'instant t

s_T : Etat terminal (mort ou fin du niveau)

a_t : Action légale depuis l'état s_t

$Q(s_t, a_t)$: Q-value générée par l'action a_t depuis l'état s_t

γ : Facteur de réduction des récompenses futures

$r(s_t, a_t)$: Récompense engendrée par l'action a_t depuis l'état s_t

e : Paramètre d'exploration

θ : Poids du réseau de neurones

α : Taux d'apprentissage

REFERENCES

[1] Sebastian Heinz, "Using Reinforcement Learning to play Super Mario Bros on NES using TensorFlow", 2019

[2] Ferhat Kortaz, "Super Mario Bros. Bot with Reinforcement Learning", 2019

[3] Sean Klein, "Deep Q-Learning to Play Mario", 2016