IFT 3150
Projet d'informatique

RAPPORT FINAL

Étendre le concept de généralisation en apprentissage par renforcement
Extending Generalization in Reinforcement Learning

Ronnie LIU
20154429

Remis à
Glen Berseth
Michalis Famelis

30 août 2022

ABSTRACT

L'apprentissage par renforcement (RL) est un type d'apprentissage qui consiste à l'agent d'apprendre, de leurs propres expériences, à résoudre différentes tâches dans un environnement donné. L'agent doit trouver la meilleure série d'actions (politique) afin qu'il puisse maximiser le nombre de récompenses total en moyenne. Afin de qualifier si une action est bonne ou pas, on se sert de la fonction de qualité qui est la moyenne des futures récompenses obtenues par l'agent à une paire d'états et d'actions donnée. Grâce à cela, l'agent pourra accomplir ses tâches de façon optimale. Cependant, dans la vraie vie, l'environnement d'entraînement de l'agent n'est pas nécessairement le même que celui du test, donc cela peut occasionner un problème de surajustement. En d'autres mots, dans un algorithme de RL standard, l'agent n'arrive pas à généraliser ce qu'il a appris dans un nouvel environnement. On conseille de redéfinir le concept de généralisation en proposant la méthode suivante : *Zero-Shot Meta Learning avec IQL*. On a comparé cette méthode avec un algorithme classique DQN. Nous avons appliqué ces méthodes sur trois jeux Atari qui ont des caractéristiques similaires (tirer sur diverses cibles). On constate que la méthode proposée nous donne un score de 357 points au maximum, tandis que l'algorithme DQN ne peut pas dépasser un score de 311 points. On conclut qu'utiliser plus d'un jeu Atari dans la phase d'entraînement et se servir de l'algorithme IQL améliore l'agent à s'adapter dans d'autres jeux Atari qu'il n'a jamais essayé, donc il généralise mieux.

INTRODUCTION

L'apprentissage par renforcement (RL) est une méthode qui se concentre sur l'interaction de l'agent et de son environnement en fonction de différentes actions. Le défi ici est d'optimiser une politique (*policy* en anglais) contenant la meilleure série d'actions à entreprendre compte tenu d'un état afin de maximiser les chances d'obtenir une récompense. En fait, on remarque que ce type d'apprentissage peut occasionner plusieurs problèmes en ce qui concerne de la performance de l'agent. Étant donné qu'une récompense peut être observée après une série d'actions, doit-on rejeter toute la série d'actions au complet lorsque l'agent n'a pas obtenu de récompenses? Une solution possible est d'utiliser plus de données et d'exemples pour l'agent, mais cela crée un autre problème, car nous n'avons pas toujours l'espace pour stocker toutes ces données. En outre, tout comme les autres modèles d'apprentissage, l'apprentissage par renforcement peut mener à un surajustement des données (*overfitting* en anglais). En d'autres termes, l'agent *mémorise* principalement différentes actions dans un environnement spécifique, ce qui réduit considérablement la performance de l'agent dans un nouvel environnement. Nous voulons créer des agents qui apprennent et généralisent à des tâches similaires. Ces agents sont plus utiles dans le monde réel puisqu'ils apprennent non seulement sur une tâche spécifique, mais ont également la capacité de réutiliser leur expérience pour différents futurs objectifs.

Plusieurs articles scientifiques emploient les algorithmes de RL sur des jeux Atari, ce qui est la raison pour laquelle on aimerait étendre le concept de généralisation sur ces environnements. Ces jeux Atari peuvent être obtenus dans la librairie *GYM* sur Python. Nous allons utiliser les jeux Atari suivants : *Carnival, Space Invaders* et *Assault*. Ce sont tous des jeux qui nécessitent de tirer sur des cibles différentes, donc le nombre d'actions différentes restera relativement le même. Le but ici est de voir une augmentation des performances de l'agent face à un jeu qu'il n'a jamais joué après avoir entraîné d'autres jeux Atari. Autrement dit, on veut que l'agent puisse être capable de mieux généraliser face aux nouveaux problèmes.

Lien sur site Web : https://xinyur1.github.io/
Liens pour le code[1] :

- Lien Github pour la version modifiée de rlkit : https://github.com/XinyuR1/rlkit
- Lien Github pour la version modifiée de doodad : https://github.com/XinyuR1/doodad

[1] Les fichiers qui sont essentiels dans ce projet sont *dqn-Atari.py* et *iqn-Atari.py* dans la librairie rlkit. (Voir Annexe A.)

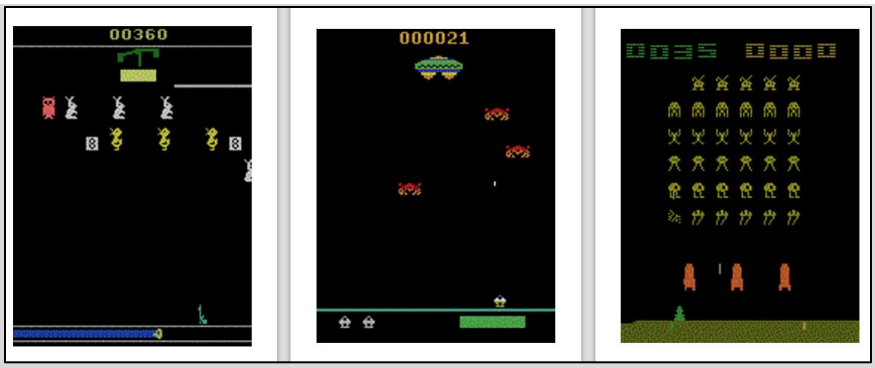

Figure 1: Captures d'écran des jeux Atari utilisés dans ce projet
[De gauche à droite] : *Carnival, Assault, Space Invaders*

OBJECTIFS

Tous les concepts mentionnés ci-dessus sont appris récemment, donc ce rapport focalisera davantage sur notre progrès par rapport à nos connaissances sur l'apprentissage par renforcement au lieu des résultats obtenus. Les objectifs de ce projet sont les suivants :

1. Maîtriser les fondements de RL;
2. Proposer une méthode d'entraînement afin de mieux généraliser pour des tâches similaires;
3. Appliquer et illustrer que cette méthode augmente la performance de l'agent à bien généraliser.

REVUE DE LITTÉRATURE

Ce projet a été inspiré de l'article *Playing Atari with Deep Reinforcement Learning*[2]. Ils effectuent des expériences sur sept jeux Atari 2600 avec la même architecture réseau, qui est le *CNN Policy*. L'objectif dans cet article scientifique est de comparer les performances de différents algorithmes d'apprentissage par renforcement sur sept jeux Atari différents. En fait, nous réalisons que l'algorithme DQN fonctionne le mieux et reçoit des récompenses moyennes totales les plus élevées. Dans ce projet, nous allons utiliser le même *CNN Policy* et le même algorithme de RL que ceux de l'article. Cependant, chaque agent ne s'entraîne que sur un seul jeu Atari en utilisant la même architecture CNN. L'agent de ce projet va entraîner sur plusieurs jeux Atari, puis apprendra par lui-même à jouer à un tout nouveau jeu Atari basé sur sa politique entraînée (*trained policy*). Cet article utilise le concept de généralisation pour un seul environnement. Autrement dit, l'agent entraîne sur un environnement (un jeu Atari) et utilise ce dernier pour évaluer sa performance. Ce type d'environnement s'appelle les environnements singleton[3]. Ce cas est très spécifique puisque les environnements d'entraînement et de test ne sont pas toujours identiques. Par conséquent, l'objectif de ce projet est d'étendre la définition de généralisation en laissant l'agent s'entraîner sur un ensemble d'environnements puis de le laisser jouer sur un nouveau jeu Atari (principe de *Zero-Shot Meta Learning*).

Nous nous référerons aussi à l'article *Offline Reinforcement Learning with Implicit Q-Learning*[4]. L'objectif de cet article est de démontrer la haute performance de IQL (en termes de généralisation), une méthode qui constitue d'un entraînement dit *Offline* suivi par un entraînement standard (*Online*). On propose d'utiliser l'algorithme IQL afin que les agents puissent mieux généraliser sur diverses tâches. La seule différence ici est que nous l'appliquerons sur différents jeux Atari au lieu de différents humanoïdes (guépards, marcheur, etc.).

[2] Playing Atari with Deep Reinforcement Learning: https://arxiv.org/pdf/1312.5602.pdf
[3] Survey of Generalization in Deep RL: https://arxiv.org/pdf/2111.09794.pdf (page 2)
[4] Offline RL with IQL: https://arxiv.org/pdf/2110.06169.pdf

CONTEXTE THÉORIQUE

Processus de décision markovien (MDP)

Définition : Un processus de décision markovien est un 4-tuplet (S, A, R, P) tel que :
S : ensemble d'états
A : ensemble d'actions
R : fonction de récompenses stochastiques – récompenses reçues de l'état *s* à *s'* à l'aide de l'action *a*

$$R(s', s', a) = P\{r_{k+1} | s_{k+1} = s', s_k = s, a_k = a\} \; pour \; k \in \mathbb{N}$$

P : fonction de transition markovien stochastique – probabilité d'atteindre l'état *s'* de *s* à l'aide de l'action *a*

$$P(s', s', a) = P\{s_{k+1} = s' | , s_k = s, a_k = a\} \; pour \; k \in \mathbb{N}$$

On peut dire que MDP peut être comparé à un langage de programmation avec un ensemble de règles spécifiques et une syntaxe unique à respecter. Nous utilisons ce « langage » pour créer une « librairie » appelée Apprentissage par renforcement (RL) afin de résoudre plusieurs problèmes réels.

Apprentissage par renforcement (RL)

L'apprentissage par renforcement (RL) est une branche de l'apprentissage automatique qui se concentre sur l'agent qui entraîne et apprenne sur différentes règles dans un environnement donné en utilisant sa propre expérience. Ce type d'apprentissage contient trois composants : l'agent, l'environnement et les récompenses.

Essentiellement, étant donné un état, l'agent décide d'entreprendre une action spécifique dans l'environnement. En fonction de la performance de l'agent, il recevra une récompense (ou pas). Ensuite, l'agent passera à un nouvel état et le cycle poursuivra jusqu'à ce que l'entraînement soit terminé. Ce processus d'entraînement est comme jouer à un jeu de tir Atari. La figure 2 illustre un exemple avec un joueur ou un robot qui joue une partie du jeu Atari (*Carnival*).

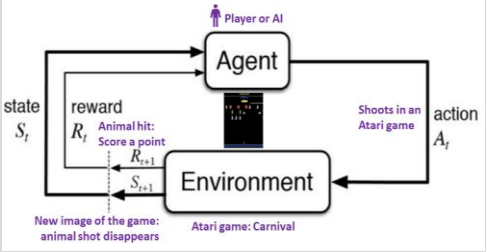

Figure 2 : Schéma de l'apprentissage par renforcement avec un exemple du jeu *Carnival*.

L'agent peut être soit le joueur soit le robot. Il tire (action) et frappe un animal dans le jeu Atari Carnival (environnement). L'agent reçoit 1 point comme récompense et on se trouve dans un nouvel état. Ce cycle continue jusqu'à tant qu'on atteint la fin du jeu.

Politique

Si nous utilisons toujours l'exemple de jouer à un jeu Atari, nous devons trouver le moyen le plus efficace de gagner le jeu en créant une politique optimale pour l'agent. Une **politique** π (a | s, θ) est une distribution sur des actions étant donné un état et un ensemble de paramètres dans un réseau de neurones de la politique (*Policy Deep Neural Network*). On peut dire de façon simple que cette dernière est la probabilité d'effectuer une action *a* sachant l'état présent *s* à l'aide des paramètres θ. L'objectif est d'optimiser cette politique de sorte que le total des récompenses reçues dans le MDP soit maximisé. Dans un jeu Atari, on aimerait trouver une distribution d'actions optimale afin que l'agent puisse obtenir le plus de points possibles.

Équation d'une politique (1)
Soient une action a, un état s et θ, un vecteur des paramètres pour entraîner la politique :

$$\pi(a|s, \theta) = \mathbb{P}\{A = a \mid S = s, \theta\}$$

Fonction de valeur

Afin d'optimiser notre politique dans un jeu Atari, il faut trouver une mesure qui permet de discerner quelles actions sont bonnes ou mauvaises. Cette mesure est la fonction de valeur (*Value Function* en anglais). Cette fonction peut être définie comme l'espérance des futures récompenses obtenues par l'agent à un état initial *s*. Autrement dit, on veut savoir combien de points l'agent a reçu en un épisode en moyenne. Par conséquent, une politique optimale est celle qui mène l'agent à avoir le plus de récompenses (grande valeur pour la fonction de valeur).

Équation de la fonction de valeur suivant une politique π (2)
Soient $p(s_0)$ une distribution des états initiaux, γ le taux d'actualisation (discount rate), R la fonction de récompenses :

$$V_\pi(s) = \mathop{\mathrm{E}}_{s\sim p(s_0)}\left\{\sum_{t=0}^{\infty}\gamma^t\, R(s_t, a_t, s_{t+1}) \mid s_0 = s\right\} pour\ 0 < \gamma < 1$$

Équations de valeur et de politique optimisées en fonction de V (3) (4)

$$V^*(s) = \max_{\pi \in \Pi} V_\pi(s)$$
$$\pi^* = \operatorname*{argmax}_{\pi \in \Pi} V_\pi(s)$$

Fonction de Q (Qualité)

La fonction Q (Q pour Qualité) est l'espérance des futures récompenses obtenues par l'agent à un état initial *s* avec une action choisie *a*. Elle est optimale si cette dernière mène l'agent à obtenir le plus de points possibles dans un jeu par exemple. Remarquez que l'équation de la fonction Q est pratiquement la même que celle de la fonction de valeur. La seule différence est que la fonction Q est conditionnée sous une action.

Équation d'une fonction de Q pour une paire d'états et d'actions (5)

$$Q_\pi(s, a) = \mathop{\mathrm{E}}_{s\sim p(s_0)}\left\{\sum_{t=0}^{\infty}\gamma^t\, R(s_t, a_t, s_{t+1}) \mid s_0 = s, a_0 = a\right\} pour\ 0 < \gamma < 1$$

Équation d'une fonction de Q optimale (6)

$$Q^*(s, a) = \max_{\pi \in \Pi} Q_\pi(s, a)$$

Une fonction de valeur optimale est l'espérance des récompenses totales en commençant par l'état *s*, donc cette dernière peut être exprimée comme le maximum de la fonction Q optimale pour toutes les actions possibles. Finalement, quant à la politique optimale, on choisit donc l'action *a* qui maximise la valeur Q optimale à partir de l'état *s*.

Équations de valeur et de politique optimisées en fonction de Q (7) (8)

$$V^*(s) = \max_{a \in A} Q^*(s, a)$$
$$\pi^* = \operatorname*{argmax}_{a \in A} Q^*(s, a)$$

Pour optimiser la politique, nous avons décidé d'utiliser la fonction de Q au lieu de la fonction de valeur puisque c'est plus facile à implémenter en pratique. Elle fait partie de la méthode *Model-Free* puisque l'agent fait des essais et erreurs durant l'entraînement en prenant diverses actions pour optimiser la fonction Q. C'est plus évident que d'implémenter un algorithme *Model-Based* qui estime directement la fonction de valeur ou la politique même. *Model-Based* prend des décisions plus logiques en se servant des connaissances acquises (actions prises dans le passé) par l'agent.

Q-Learning et Deep Q-Learning

Ce projet se concentrera sur l'utilisation de l'algorithme *Q-Learning*. Cet algorithme utilise la fonction de qualité afin de maximiser les récompenses totales. Il n'apprend pas toujours des actions qui sont à l'intérieur de la politique actuelle, ce qui signifie que le *Q-Learning* encourage plus d'exploration dans l'environnement. La variable epsilon (ε) est la probabilité qu'un agent entreprenne une action aléatoire au lieu de se référer à une action avec la valeur Q la plus élevée. Pour chaque paire état-action, la valeur Q sera mise à jour avec l'équation suivante :

Équation de Bellman pour Q-Learning (9)
À noter que α est le taux d'apprentissage, r est la récompense à présent and γ est le taux d'actualisation.

$$Q(s_t, a_t)_{new} = Q(s_t, a_t)_{old} + \alpha\left[\mathrm{r} + \gamma \max_{a_{t+1}} \mathrm{Q}\,(\mathrm{s}_{t+1}, \mathrm{a}_{t+1}) - Q(s_t, a_t)_{old}\right]$$

Supposons que dans le jeu Atari *Carnival*, vous choisissiez de viser lapin et que vous le frappiez. Cette paire état-action actuelle a initialement une valeur Q ($Q(s_t, a_t)_{old}$). Vous recevez une récompense puisque vous frappez un animal. Maintenant que le lapin frappé disparaît de l'écran (état s_{t+1}), vous devrez effectuer la prochaine action qui maximise la valeur Q. Supposons que parmi toutes les actions possibles, vous réaliserez que la meilleure action est de se déplacer vers la droite ($Q\ (s_{t+1}, a_{t+1})$). L'ancienne valeur Q pour la paire état-action t sera mise à jour avec la valeur Q maximisée pour l'action suivante, la récompense actuelle, le « *discount rate* » et le taux d'apprentissage (*learning rate*). Notez que les valeurs Q seront mises à jour pour chaque étape de chaque épisode et que l'action prise par l'agent est soit une action aléatoire (avec un taux d'epsilon ε), soit une action qui a la valeur Q la plus élevée compte tenu d'un état (à l'aide de l'équation de Bellman), qui est le concept d'algorithme *Epsilon-Greedy Q-Learning*.

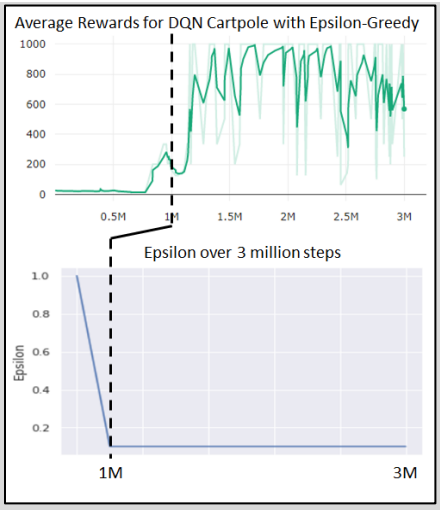

Figure 3 Illustration sur l'effet aux récompenses obtenues par rapport à la valeur d'epsilon dans un problème de poteau (Cartpole) avec Q-Learning.

Illustrons cet algorithme avec l'exemple du poteau (qui consiste à maintenir le poteau droit et stable le plus longtemps possible) (Figure 3). On observe que les récompenses ne sont pas très élevées au tout début, ce qui n'est pas surprenant. On veut que l'agent puisse explorer l'environnement (avec une grande valeur d'epsilon). Après 1 million d'étapes, l'agent commence à optimiser la politique de ce jeu puisqu'on lui donne uniquement 10% de chance de prendre une action au hasard. On voit que la courbe verte augmente et atteint un score de 1000 plusieurs fois (maximum de points). On s'attend à avoir un phénomène similaire lorsqu'on emploie cet algorithme classique aux jeux Atari.

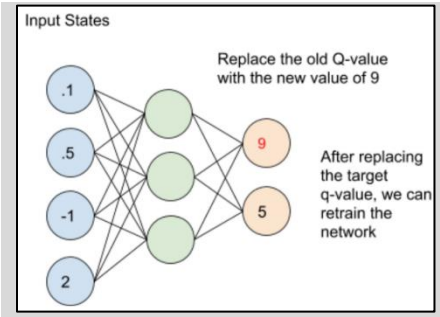

Figure 4 : Réseau de neurones pour Deep Q-Learning (valeur de Q mise à jour en rouge). *Les neurones orange représentent les sorties (les actions différentes d'un jeu Atari).*

Au lieu d'utiliser une paire état-action avec une valeur Q, DQN (*Deep Q-Networks*) utilise des réseaux de neurones pour mettre à jour les valeurs Q. De plus, DQN utilise deux réseaux de neurones : l'un est le réseau principal où se déroule l'entraînement et l'autre est le réseau cible, pour stabiliser le processus d'apprentissage. Il utilise également un *Replay Buffer* afin d'accélérer la mise en œuvre du DQN. La figure 4 illustre un exemple d'états d'entrée donnés mis à jour d'une valeur Q dans un réseau de neurones de la politique.

Réseaux de neurones convolutifs (CNN)

Étant donné que DQN nécessite des réseaux de neurones, nous utiliserons des réseaux de neurones convolutifs. CNN est un réseau de neurones composé de nombreuses couches convolutives cachées, placées les unes à côté des autres. Ces couches sont produites en modifiant l'image d'entrée à une certaine résolution à l'aide des noyaux (*kernels*). Elles sont suivies de couches d'activation qui créent une fonction non linéaire et de couches de regroupement (*pooling layers*) qui réduisent la taille de l'image. Étant donné que CNN est principalement utilisé en vision par ordinateur, nous utiliserons ce réseau de neurones comme réseau de politique pour l'algorithme DQN dans l'apprentissage par renforcement puisque les jeux Atari sont une succession d'images. Ici, l'entrée de notre modèle CNN sera l'image du jeu Atari et la sortie de cette dernière sera la valeur de qualité pour chaque action permise dans le jeu Atari.

Généralisation en apprentissage par renforcement

Lorsqu'on parle de la généralisation en apprentissage automatique, on regarde toujours la performance de l'agent dans l'environnement de l'entraînement (là où la politique est apprise) et dans l'environnement de test (là où la politique est appliquée). Habituellement, on se sert de la fonction de perte (d'erreur) afin d'optimiser notre politique, mais il est plus adéquat d'utiliser les récompenses totales moyennes comme mesure en apprentissage par renforcement. La fonction de perte est une mesure qui juge si l'agent lui-même trouve qu'il a bien performé ou non dans un jeu, mais cela ne veut pas dire que c'est le cas. On peut avoir une valeur de perte très basse tout en ayant peu de récompenses. Bref, il est mieux d'utiliser les récompenses pour voir si l'agent a bien fait sa tâche durant l'apprentissage.

On juge qu'un agent généralise bien lorsqu'il reçoit de hautes récompenses dans le jeu d'entraînement et dans le jeu de test. La généralisation est importante puisqu'il encourage l'utilisation efficace des données ainsi que la capacité d'adaptation de l'agent. Supposons qu'un robot apprenne à marcher en une ligne droite. Si ce dernier est habitué de marcher sur un plancher fait en bois (entraînement), il perdra sa mobilité lorsqu'il se déplace sur un tapis (test) par exemple. Au lieu de construire un nouvel modèle, il serait mieux de modifier quelques éléments du robot afin qu'il puisse s'adapter dans différents environnements. On évitera le gaspillage des expériences cumulées par le robot et il aura la capacité de marcher sur différents terrains.

Dans l'apprentissage par renforcement standard, nous nous concentrons sur les environnements où les environnements d'entraînement et de test sont les mêmes. La seule différence est que l'ensemble de test utilise une politique déterministe, alors que nous ajoutons du bruit aléatoire lors de l'entraînement de la politique. Cependant, dans des situations réelles, les environnements d'entraînement et de test ne sont pas identiques, ce qui signifie que l'agent doit apprendre à bien généraliser sur des tâches similaires dont il n'a pas acquis de connaissances préalables. De plus, le RL classique ne s'entraîne que sur un seul environnement, ce qui crée le problème de surajustement. Par exemple, si on ne s'entraîne que sur le jeu Atari *Carnival*, on est certain que les performances de l'agent pour ce jeu seront bonnes, mais ce n'est pas garanti lorsqu'il essaie de jouer un nouveau jeu Atari, comme *Assault* par exemple.

MÉTHODOLOGIE

Plan général

La méthode proposée pour ce projet s'appelle *Zero-Shot Meta Learning* avec IQL. Nous allons étendre le concept de généralisation en laissant l'agent s'entraîner sur plusieurs environnements Atari. En d'autres termes, après un entraînement sur 2 environnements (*Carnival* et *Space Invaders*), nous voulons voir les performances de l'agent sur un nouveau jeu Atari sans entraînement supplémentaire d'aucune sorte (*Assault*). Cette méthode s'appelle *Zero-Shot Meta Learning* puisqu'on entraîne sur plusieurs environnements (*Meta Learning*) et celui du test n'est pas appris du tout dans la phase d'entraînement (*Zero Shot Learning*). Avec cette méthode, on se sert d'un algorithme classique qui est DQN et un autre qui est proposé par ce projet : IQL.

On doit comparer les récompenses obtenues en fonction de différents algorithmes et voir si l'agent généralise mieux avec IQL au lieu de DQN, un algorithme classique RL. Par la suite, on observe comment la performance de l'agent se diffère lorsqu'on ajoute plus d'environnements lors de la phase d'entraînement. Pour ces algorithmes, on va se servir de CNN comme le réseau de neurones de la politique.

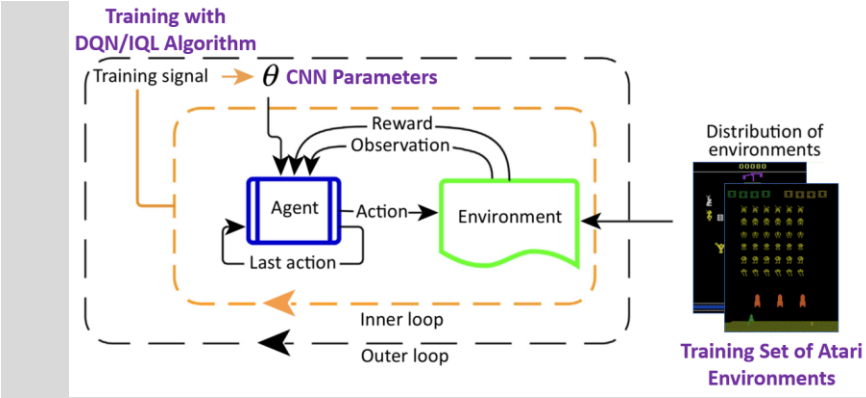

Figure 5 : Visualisation sommaire de l'entraînement d'un agent pour ce projet

Prétraitement des images Atari

Pour tous les jeux Atari, les images sont de dimension 210 x 160 pixels avec 3 canaux (R, G, B). En CNN, les réseaux de neurones acceptent des images qui sont en forme carrée, donc on a réduit les images en dimension 84 x 84 avec 1 canal (blanc/noir). L'article *Playing Atari with Deep Reinforcement Learning* utilise 4 canaux en créant quatre images successives du jeu Atari, mais on a décidé d'utiliser un seul canal pour simplifier la tâche.

Architecture du modèle CNN

L'entrée d'un modèle est une image de taille 84 x 84 x 1 qui est prétraitée. Le réseau CNN est directement tiré de l'article *Playing Atari with Deep Reinforcement Learning*. En fait, ils ont commencé par une première couche convolutive avec 16 filtres (*kernels*) de taille 8 x 8 avec un pas de 4 (*stride*) suivie par une fonction d'activation ReLU ($f(x) = max(0, x)$ pour tout réel x). La deuxième couche a 32 filtres de taille 4 x 4 avec un pas de 2 suivie de la même fonction d'activation. Finalement, la dernière couche est complètement connectée avec 256 neurones comme sortie. La couche de sortie contient k neurones où k représente le nombre d'actions différentes dans le jeu d'Atari. À noter que le *batch size* pour ces expériences est 256.

Algorithme proposé: IQL (Offline RL avec Q-Learning Implicite)

Dans Q-Learning ou DQN, l'agent interagit toujours avec l'environnement afin de collecter des échantillons. Cependant, dans *Offline RL*, l'agent doit optimiser sa politique avec une quantité fixe d'ensemble de données avec des interactions fixes. Par conséquent, l'agent ne peut pas accéder aux données supplémentaires de l'environnement. Nous pouvons comparer cet ensemble de données fixe à l'ensemble de données d'entraînement en apprentissage supervisé.

La principale différence entre le Q-Learning et le Q-Learning implicite est l'équation de Bellman. Dans Q-Learning, nous mettons à jour les valeurs Q en fonction de la prochaine action effectuée qui a maximisé la valeur Q. Le Q-Learning implicite utilise également la même idée, mais les actions doivent se trouver dans le support de la distribution des données. Cette idée et l'utilisation de la régression expectile garantissent que nous n'obtenons pas de valeurs Q à partir d'actions hors-distribution. Sinon, le calcul des fonctions de qualité seront erronées et occasionne un problème de surajustement. Cette conséquence est notre motivation d'utiliser cet algorithme dans notre projet. La figure 6 (voir prochaine page) montre plus clairement la différence entre ces deux algorithmes.

Q-LEARNING (OFF-POLICY)

$$Q(s_t, a_t)_{new} = Q(s_t, a_t)_{old} + \alpha \left[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old} \right]$$

IMPLICIT Q-LEARNING (OFFLINE RL)

$$Q(s_t, a_t)_{new} = Q(s_t, a_t)_{old} + \alpha \left| r + \gamma \max_{a_{t+1} \in A,\ s.t.\ \pi(a_{t+1}|s_{t+1})>0} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old} \right|$$

Figure 6 : Équations de Bellman pour Q-Learning et Q-Learning implicite. La différence est la condition ci-dessous le maximum de la valeur de qualité à l'étape (t+1)

Dans l'algorithme (figure 7), on commence par initialiser une fonction de perte de la valeur (*value function loss*) sur la distribution des actions du jeu. C'est dans cette étape où on restreint à l'agent de prendre des actions qui ne se trouvent pas dans le support. Nous mettons ensuite à jour les valeurs Q avec la perte MSE (perte quadratique) à l'aide de la fonction de perte de la valeur. Enfin, puisque nous n'avons optimisé que les valeurs Q, nous devons encore nous assurer que la politique est également optimisée. Par conséquent, l'article suggère d'utiliser une méthode d'extraction de politique appelée régression pondérée des avantages (*Advantage Weighted Regression*).

Algorithm 1 Implicit Q-learning

Initialize parameters ψ, θ, $\hat{\theta}$, ϕ.
TD learning (IQL):
for each gradient step **do**
$\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
$\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
$\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta$
end for
Policy extraction (AWR):
for each gradient step **do**
$\phi \leftarrow \phi - \lambda_\pi \nabla_\phi L_\pi(\phi)$
end for

Figure 7 : Algorithme de Q-Learning Implicite prise de l'article *Offline Reinforcement Learning with Implicit Q-Learning.*
1. Variable Violette : Fonction de perte de la valeur
2. Variable Verte : Fonction de perte des valeurs de qualité
3. Variable Bleue : Régression pondérée des avantages pour extraire la politique

L'article *Offline RL with IQL* utilise cet algorithme sur des humanoïdes (apprendre à se déplacer dans plusieurs directions) et encore une fois, les environnements d'entraînement et de test sont les mêmes. Dans ce projet, on applique cet algorithme aux jeux Atari avec deux environnements d'entraînement et un environnement de test.

PROGRÈS

Avant de comprendre IQL, plusieurs concepts mentionnés dans le contexte théorique et la méthodologie sont complètement nouveaux pour nous, donc commencer à être familier avec l'apprentissage par renforcement tout en ajoutant d'autres concepts comme le CNN, la généralisation est un grand défi. Nous avons commencé par comparer les similitudes et différences entre l'apprentissage supervisé et par renforcement et tranquillement, on les applique dans des exemples comme celui du poteau (*Cartpole*) et ceux des jeux Atari. En outre, quant à l'implémentation, utiliser différents logiciels et applications comme *Docker*, *Comet ML* est nouveau pour nous, mais essentiel dans ce projet. La méthode IQL nous a permis de découvrir un nouveau sujet qui est le méta-apprentissage. Nous avons remarqué que plusieurs articles utilisent un seul environnement pour entraîner et tester la performance d'un agent, donc cette méthode nous a permis de comprendre un concept de généralisation beaucoup plus vaste.

EXPÉRIENCES

Pour les deux algorithmes, on emploie la méthode *Epsilon-Greedy* avec epsilon qui varie de 1 à 0.1 pendant 1 million d'étapes et demeure 0.1 jusqu'à la fin de l'entraînement. Pour DQN, on va entraîner l'agent pendant 5 000 epochs (5 millions étapes) et pour IQL, 1 000 epochs (1 million étapes) avec *Offline RL* et 4 000 epochs ordinaires (4 millions étapes). La taille du *Replay Buffer* est de 100 000. Pour plus d'informations sur l'exécution du code en lien avec les expériences : consulter la section *Annexe A : Soumission du code*. Le tableau suivant illustre les quatre expériences qu'on va tester dans ce projet en fonction de l'algorithme utilisé et des environnements choisis :

Tableau I – Liste des expériences en fonction de l'algorithme et des environnements utilisés

Numéro	Algorithme utilisé	Environnements d'entraînement	Environnement de test
1	*DQN*	*Assault*	*Assault*
2	*DQN*	*Carnival, Space-Invaders*	*Assault*
3	*IQL*	*Assault*	*Assault*
4	*IQL*	*Carnival, Space-Invaders*	*Assault*
Noter que l'expérience 4 est la méthode proposée du projet.			

On propose de comparer les algorithmes de façon suivante :

- Utiliser les expériences 1 et 2 comme expériences de base.
- Comparer les expériences 1 et 3 – DQN vs IQL avec un seul environnement d'entraînement
- Comparer les expériences 2 et 4 – DQN vs IQL avec un ensemble d'environnements d'entraînement
- Comparer les expériences 3 et 4 – Différents nombres d'environnement d'entraînement avec l'algo IQL

Ces comparaisons nous permettent d'illustrer qu'utiliser IQL avec plusieurs environnements d'entraînement (notre méthode proposée) est meilleur que d'utiliser l'algorithme classique DQN. On s'attend qu'en général, peu importe le nombre d'environnements d'entraînement, IQL va avoir une performance beaucoup mieux que DQN (plus de récompenses) étant donné que IQL est un algorithme qui améliore la performance de l'agent en lui donnant une certaine restriction lorsqu'il décide quelle action il doit entreprendre. Finalement, on suggère que l'expérience 4 est la meilleure parmi les quatre puisqu'elle tient compte de l'utilisation de plusieurs environnements durant l'entraînement tout en évitant le problème de surajustement avec IQL.

RÉSULTATS

Graphique I. Récompenses totales moyennes durant la phase d'entraînement (haut) et de test (bas) en fonction de nombre d'étapes (pas)

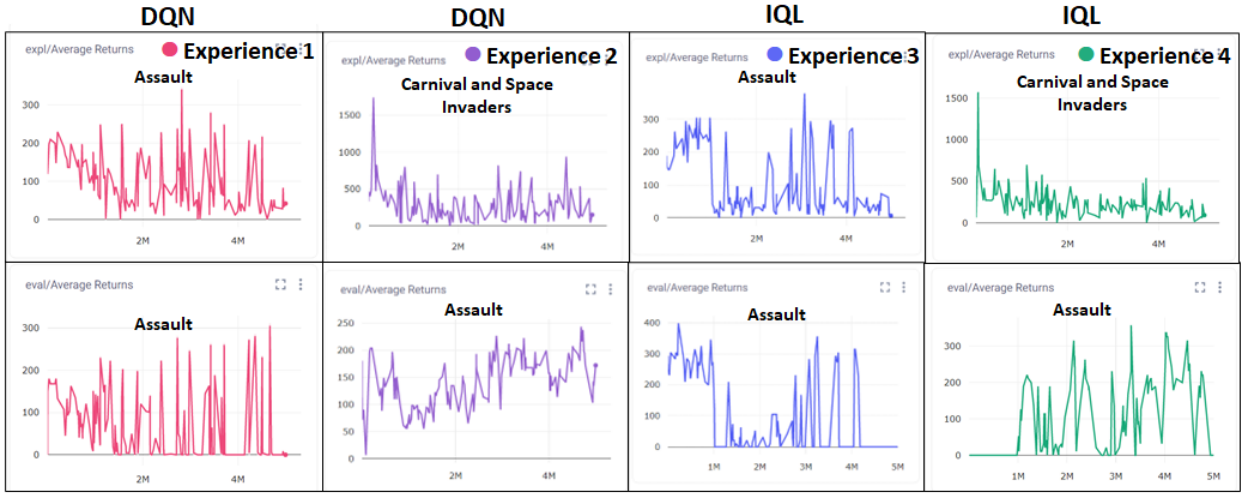

Tableau II. Récompenses totales moyennes maximales durant la phase de test

Numéro d'expérience	Récompense maximale	Nombre d'étapes pour atteindre la valeur maximale
1	311 points	Environ 4.7 millions
2	244 points	Environ 4.7 millions
3	357 points	Environ 3.2 millions
4	357 points	Environ 3.3 millions

DISCUSSION

DQN vs IQL sur un environnement d'entraînement

Tout d'abord, nous avons entraîné l'agent sur le jeu *Assault* et on le laisse jouer selon la politique que ce dernier a entraîné. Dans l'expérience 1, on s'est servi de l'algorithme classique DQN et dans la phase de test, on voit que la courbe possède des oscillations de grande amplitude. Cependant, les récompenses obtenues augmentent de 200 à 300 environ au cours de 5 millions d'étapes. Dans la phase de test, étant donné que l'agent est déjà familier du jeu *Assault*, il n'est pas surprenant que sa performance soit similaire à celle durant sa phase d'entraînement, avec un score maximal de 311 points pour 4.7 millions étapes. Nous avons consulté un rapport[5] où les auteurs de ce dernier emploie la même méthode que l'expérience 1 sur Assault; le score est de 201.4 points lorsque la politique n'est pas entraînée, donc nous sommes au moins certains que notre expérience de base est mieux que de prendre des actions au hasard.

Dans l'expérience 3, on s'attend que la courbe de test augmente de façon significative. Cependant, durant *Offline RL* (1 million premières étapes), on reçoit un score maximal de 400 environ, mais la courbe descend entre 1 et 3 millions étapes et réaugmentent jusqu'à un score maximal de 357 points avec 3.2 millions étapes. Une possibilité d'avoir ce phénomène est illustrée dans la figure 8. En fait, lors de la phase *Offline RL*, l'agent a une quantité fixe d'ensemble de données d'*Assault*, donc il performe relativement bien étant donné qu'il est déjà familier de ce jeu dans la phase d'entraînement. Cependant, dès qu'on est dans la phase *Online RL*, l'agent n'a plus de données fixes (flèche noire de la figure qui pointe vers le bas), donc il faut qu'il se réajuste dans cet environnement en prenant d'autres décisions (flèche qui pointe vers le haut).

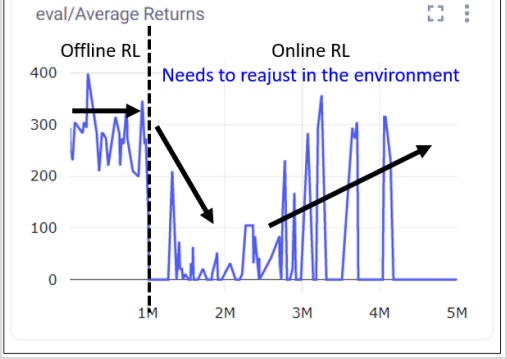

Figure 8 : Explication possible de la courbe d'évaluation de l'expérience 3

Peu importe la cause de ce phénomène, on observe que le score maximal en moyenne de l'expérience 3 (357) est plus élevé que celui de l'expérience 1 (311) avec 3.2 millions d'étapes. On sauve 1.5 millions d'étapes et 5 heures d'entraînement puisque l'expérience 1 a pris 13 heures, tandis que l'expérience 3 n'a pris que 8 heures pour apprendre la politique.

DQN vs IQL sur un ensemble d'environnements d'entraînement

La courbe de l'expérience 2 demeure constante dans la phase d'entraînement si on ignore la donnée aberrante qui se trouve au tout début des itérations. Cependant, la courbe de test augmente jusqu'à un score maximal de 244 points avec 4.7 millions d'étapes. Cela veut dire que l'agent commence à être familier au nouveau jeu Assault après avoir joué *Space Invaders* et *Carnival* étant donné que ces jeux sont de même style (*shooter games*).

Cependant, on remarque que l'expérience 4 obtient un score plus élevé (357), avec une différence de 113 points lorsqu'on compare avec le score de l'expérience 2. Ceci n'est pas surpris puisque l'algorithme DQN ne tient pas compte du concept de généralisation sur différents environnements comme IQL. Il focalise uniquement sur les essais et erreurs de différentes actions, mais n'améliore pas le choix de ces dernières. Encore une fois, on sauve aussi 5 heures du temps d'entraînement et 1.4 millions d'étapes.

[5] Deep Learning on Atari Games using Deep Q-Learning Algorithm: https://www.jetir.org/papers/JETIRBW06017.pdf

Même si l'expérience 4 obtient un meilleur score maximal, dans la courbe de test, nous n'avons aucune récompense durant *Offline RL*, mais le score augmente jusqu'à 357 avec 3.3 millions étapes environ. Ce phénomène n'est pas attendu puisqu'on s'attend à avoir une courbe qui augmente. Une proposition par rapport à la cause derrière ce phénomène est illustrée dans la figure 10. Puisque l'agent n'est pas du tout familier au jeu *Assault* après avoir entraîné dans *Space Invaders* et *Carnival*, on n'obtient aucune récompense durant la phase *Offline RL* étant donné que les interactions et les données sont fixes. Cela représente un problème de surajustement puisque l'agent n'a pas la capacité de s'adapter dans un nouveau jeu durant les 1 million premières étapes. Cependant, dès qu'on est dans la phase *Online RL*, l'agent a accès directement à l'environnement au lieu des données fixes du jeu Assault et peut apprendre graduellement. On constate que la courbe augmente (avec oscillations), ce qui signifie que l'agent s'adapte tranquillement à un nouveau jeu, donc généralise de mieux en mieux.

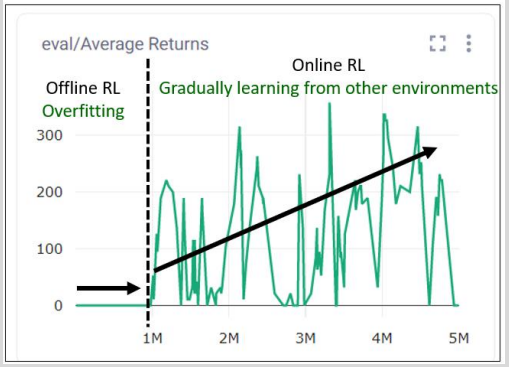

Figure 9 : Explication possible de la courbe d'évaluation de l'expérience 4

Derniers mots à la méthode proposée

On constate que IQL obtient le plus haut score maximal lorsqu'on compare à DQN. En outre, si on compare les expériences 3 et 4, il serait plus préférable d'utiliser l'expérience 4. En fait, dans l'expérience 4, l'agent est capable d'avoir un score de 357 en utilisant des environnements qui sont différents d'*Assault*, donc cela montre qu'il a la capacité de généraliser aux jeux similaires. Certes l'expérience 3 a aussi un score de 357, mais l'agent utilise Assault comment environnement d'entraînement et de test, ce qui n'indique pas si l'agent peut généraliser. On est seulement certain qu'il est capable de jouer seulement un jeu correctement au lieu de s'adapter aux autres jeux similaires à *Assault*.

Les figures 8 et 9 sont toutes des propositions derrière les phénomènes étranges des expériences 3 et 4, elles ne sont pas tout à fait certaines. Une autre possibilité plus probable est l'implémentation erronée de l'algorithme IQL; on ne remarque pas de tendances étranges dans l'algorithme DQN. Elle est probable à cause de la courbe de l'expérience 4 (figure 9). Étant donné qu'IQL est un algorithme qui tente de régler le surajustement de l'agent, il n'est pas logique de voir une récompense nulle au tout début du jeu lors de la phase de test, surtout quand l'algorithme DQN n'a pas de problème de surapprentissage. IQL est implémenté selon les environnements des humanoïdes, donc les actions prises par l'agent sont exprimées à partir des vecteurs réels, non pas des actions discrètes. Le changement de code (à partir des actions à valeur continue aux actions discrètes) peut causer ce problème dans le comportement de l'agent. En outre, on a mentionné que la taille du *Buffer Replay* est uniquement 100 000. Plusieurs articles scientifiques utilisent une taille d'un million, mais nous n'avons pas assez d'espaces pour stocker toute cette information dans le laboratoire, donc on pense aussi que la taille de cette dernière influence grandement la performance de l'agent aussi.

CONCLUSION

En conclusion, la méthode proposée dans ce projet est d'utiliser plusieurs environnements Atari dans la phase d'entraînement et d'utiliser l'algorithme IQL afin d'améliorer un agent à bien généraliser d'autres jeux Atari similaires. Lorsqu'on utilise le même environnement pour l'entraînement et le test (*Assault*), IQL nous mène un score maximal moyenne de 357 points comparés à DQN avec 311 points. On obtient aussi 357 points avec l'algorithme IQL lorsqu'on utilise *Space Invaders* et *Carnival* comme environnements d'entraînement et *Assault* comme environnement de test, tandis qu'on a uniquement 244 points avec DQN. Finalement, on remarque IQL avec plusieurs environnements d'entraînement est la meilleure méthode afin que l'agent puisse s'adapter dans différents jeux.

Quant aux futures contributions à ce projet, on remarque que nous n'avons pas mis l'accent sur divers choix d'hyperparamètres. Donc, afin d'augmenter encore plus la performance de l'agent, on propose d'utiliser les mêmes expériences, mais en prenant plusieurs hyperparamètres et sélectionner la meilleure combinaison de ces derniers. Outre le fait qu'il faut varier les hyperparamètres, on peut comparer aussi avec un algorithme de *Meta-Learning*, une méthode qui suggère d'apprendre un peu les données du test afin de mieux d'approfondir davantage le concept de généralisation en apprentissage par renforcement.

RÉFÉRENCES

Ressources bibliographiques

Alzantot, M. (2017). *Deep Reinforcement Learning Demystified (Episode 2) – Policy Iteration, Value Iteration and Q-Learning*. Medium. https://medium.com/@m.alzantot/deep-reinforcement-learning-demysitifed-episode-2-policy-iteration-value-iteration-and-q-978f9e89ddaa

Berseth, G. (2022). *Robot Learning: Generalization for Robotics* [fichier PDF].

Brunton, S. et Kutz, J. (2021). *Data Driven Science & Engineering*. University of Washington. https://faculty.washington.edu/sbrunton/databookRL.pdf

Chilamkurthy, K. (2020). *Off-Policy vs On-Policy vs Offline Reinforcement Learning Demystified*. Medium. https://kowshikchilamkurthy.medium.com/off-policy-vs-on-policy-vs-offline-reinforcement-learning-demystified-f7f87e275b48

Dayan, P. and Niv, Y. (2008). *Reinforcement Learning: The Good, The Bad, and The Ugly*. Science Direct. https://www.princeton.edu/~yael/Publications/DayanNiv2008.pdf

Kadel, M. et Jain, R. (2019). *Deep Learning on Atari Games Using Deep Q-Learning Algorithm*. Jetir. https://www.jetir.org/papers/JETIRBW06017.pdf

Kirk, R. et al. (2022). *A Survey of Generalisation in Deep Reinforcement Learning*. Cornell University. https://arxiv.org/pdf/2111.09794.pdf

Kostrikov, I. et al. (2021). *Offline Reinforcement Learning with Implicit Q-Learning*. Cornell University. https://arxiv.org/pdf/2110.06169.pdf

Lee, M. (2005). *3.7 Value Functions*. Incomplete Ideas. http://www.incompleteideas.net/book/ebook/node34.html

Lee, M. (2005). *3.8 Optimal Value Functions*. Incomplete Ideas. http://www.incompleteideas.net/book/ebook/node35.html

Sharma, S. (2020). *The Ultimate Beginner's Guide to Reinforcement Learning*. Towards Data Science. https://towardsdatascience.com/the-ultimate-beginners-guide-to-reinforcement-learning-588c071af1ec

Sreenath, S. (2020). *Getting to Grips with Reinforcement Learning via Markov Decision Process*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/11/reinforcement-learning-markov-decision-process/

Violante, A. (2019). *Simple Reinforcement Learning: Q-Learning*. Towards Data Science. https://towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56

Volodymyr, M. et al. (2013). *Playing Atari with Deep Reinforcement Learning*. Cornell University. https://arxiv.org/pdf/1312.5602.pdf

Wang, M. (2020). *Deep Q-Learning Tutorial: minDQN*. Towards Data Science. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc#:~:text=Critically%2C%20Deep%20Q%2DLearning%20replaces,process%20uses%202%20neural%20networks

Yamashita, R. et al. (2018). Convolutional Neural Networks: An Overview and Application in Radiology. Springer. https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

Références du code

Mitr, G. (s.d.). *Deep Q-Network Implementation*. Google Colab. https://colab.research.google.com/github/GiannisMitr/DQN-Atari-Breakout/blob/master/dqn_atari_breakout.ipynb#scrollTo=_IA-czvUwbOn

Montreal Robotics. (s.d.). *doodad*. Github. https://github.com/montrealrobotics/doodad

Neo-X. (s.d.). *doodad*. Github. https://github.com/Neo-X/doodad

Neo-X. (s.d.). *SMiRL_Code*. Github. https://github.com/Neo-X/SMiRL_Code

Rail-Berkeley. (s.d.). *rlkit*. Github. https://github.com/rail-berkeley/rlkit

Figures

Figure 1 – Image d'*Assault* : *Assault* [image]. (s.d.). Gym Library. https://www.gymlibrary.ml/environments/atari/assault/

Figure 1 – Image de *Carnival* : *Carnival* [image]. (s.d.). Gym Library. https://www.gymlibrary.ml/environments/atari/carnival/

Figure 1 – Image de *Space Invaders* : *Space Invaders* [image]. (s.d.). Gym Library. https://www.gymlibrary.ml/environments/atari/space_invaders/

Figure 2: Bhatt, S. (2018). [image modifiée]. KD Nuggets. https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

Figure 3 – Graphique d'epsilon en fonction du nombre de pas : *Epsilon over 10 Million Frames* [graphique modifié]. (2021). Medium. https://medium.com/nerd-for-tech/reinforcement-learning-deep-q-learning-with-atari-games-63f5242440b1

Figure 3 – Graphique de récompenses pour le problème de *Cartpole* : *Eval/average rewards* [graphique]. (2022). Comet ML. https://www.comet.com/xinyur1/dqn-breakout-v0/49c55d5893294354a27398278022e9b2?experiment-tab=chart&showOutliers=true&smoothing=0.591&transformY=smoothing&xAxis=step

Figure 4: *A Neural Network Mapping an Input State to its Corresponding (Action, Q-Value) Pair* [image modifiée]. (2020). Towards Data Science. https://towardsdatascience.com/deep-q-learning-tutorial-mindqn-2a4c855abffc

Figure 5: *Illustration of Meta-RL* [image modifiée]. (2019). Lil'Log. https://lilianweng.github.io/posts/2019-06-23-meta-rl/

Figure 7: *Algorithm 1: Implicit Q-Learning* [algorithme]. (2021). Cornell University. https://arxiv.org/pdf/2110.06169.pdf

ANNEXE

A. Soumission du code

Les expériences sont réalisées à partir de la librairie *rlkit* qui contient les algorithmes d'apprentissage par renforcement. Les expériences 1 et 2 peuvent être exécutées à l'aide du fichier *dqn-Atari.py*; pour les expériences 3 et 4, le fichier *iqn-Atari.py*.

Le code pour ce projet se trouve dans les librairies *rlkit* et *doodad*. Ces librairies sont déjà créées sur Github et on modifie plusieurs fichiers et on en rajoute d'autres afin de réaliser nos expériences. En fait, ces dernières ont été exécutées dans une machine virtuelle Ubuntu. Donc, nous utilisons Linux comme système d'exploitation. Nous avons utilisé ensuite le GPU de l'ordinateur du laboratoire afin que l'agent puisse s'entraîner à divers jeux Atari dans un contenant Docker. Tout cela est possible grâce à doodad, une bibliothèque qui nous permet de lancer différents fichiers Python sur d'autres machines. Pour ce projet, on se sert des ordinateurs « blue » et « green » du laboratoire de DIRO. Noter que tous les changements sont documentés dans les fichiers README.md de chaque répertoire Github.

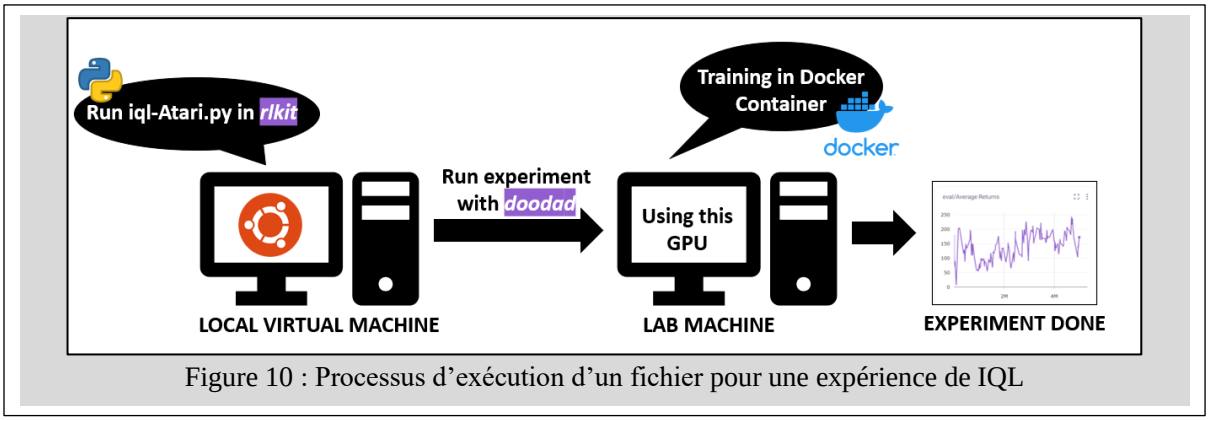

Figure 10 : Processus d'exécution d'un fichier pour une expérience de IQL

Lien Github pour la version modifiée de rlkit : https://github.com/XinyuR1/rlkit
Lien Github pour la version modifiée de doodad : https://github.com/XinyuR1/doodad

Pour plus d'informations sur les expériences que nous avons faites, consulter les liens suivants pour voir des statistiques autres que les récompenses obtenues par l'agent :

- Expérience 1 : https://www.comet.com/xinyur1/ift-3150/d7a34fbb018d440a868cefa3302ac0f7?experiment-tab=chart&showOutliers=true&smoothing=0&transformY=smoothing&xAxis=step

- Expérience 2 : https://www.comet.com/xinyur1/ift-3150/ea8e40ddc4cf4c368a8c0b9105a90b1b?experiment-tab=chart&showOutliers=true&smoothing=0&transformY=smoothing&xAxis=step

- Expérience 3 : https://www.comet.com/xinyur1/ift-3150/91cc5a4423314e9f84f92c324420f9fd?experiment-tab=chart&showOutliers=true&smoothing=0&transformY=smoothing&xAxis=step

- Expérience 4 : https://www.comet.com/xinyur1/ift-3150/7eb412c861cb432c92efd9a1d8dd580a?experiment-tab=chart&showOutliers=true&smoothing=0&transformY=smoothing&xAxis=step