Guimbaud Jean-Baptiste p1408925
Duval Jérémy p1501022

31/12/2019

TP apprentissage profond par renforcement

Intelligence Bio-Inspirée

Master 2 IA

Introduction

L'objectif de ce TP était de nous familiariser avec les techniques d'apprentissages profond par renforcement, en particulier le Deep-Q-Learning. Cette technique récente permet à un agent d'apprendre des politiques efficaces même avec des données d'entrées à nombreuses dimensions.

Nous nous sommes grandement inspirés du papier de recherche pointé par l'énoncé du TP, [Mnih et al., 2015], pour l'implémentation de l'algorithme, ces différentes optimisations et les paramètres des modèles. En effet un tel algorithme requiert d'être entraîné longtemps sur un grand nombre de données et, dans certains cas, il n'est pas garanti de converger. Pour adresser ce problème, le TP nous guide vers différentes méthodes inspirées du papier de recherche, à savoir l'**Expérience Replay** et la **Fixed-Q-Target** qui améliorent grandement les performances du DQL.

Partie 1 : Deep Q-network sur CartPole-v1

Dans cette partie, nous devions implémenter une première version du DQN, sur un environnement relativement simple. L'objectif était de faire apprendre à la machine à faire tenir en équilibre un bâton sur un chariot le plus longtemps possible. La machine ne dispose que de 2 actions, déplacer le chariot à gauche ou à droite, et l'environnement observable n'est constitué que de 4 valeurs correspondantes à la position du chariot et à celle du bâton.

Un réseau de neurones standard composé de fully-connected layers est donc ici suffisant. Le réseau prend en entré les valeurs observables de l'environnement et retourne une sortie par actions (les fameuses Q-Valeurs). La valeur maximale renvoyée nous donne l'action à entreprendre dans l'environnement.

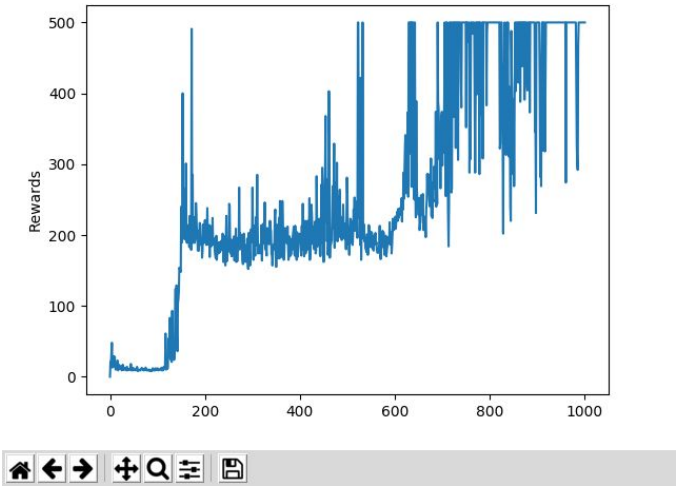
Dans un algorithme par renforcement classique, la machine utiliserait directement la nouvelle observation résultante de l'environnement après l'action de cette dernière. Cependant, pour permettre une meilleure convergence de l'algorithme DQL, nous stockons ces observations dans un buffer de

taille fixé (avec l'action correspondante et l'état précédent). Cela permet ensuite d'apprendre à partir d'un lot tiré aléatoirement dans ce buffer. Ainsi, certaines expériences d'occurrences rares dans l'environnement peuvent être réutilisées plusieurs fois pour apprendre et cela permet, globalement, de mieux tirer profit de l'expérience de la machine.

Afin d'améliorer la convergence de l'algorithme, nous avons ensuite dû utiliser un réseau secondaire pour calculer les Q-Targets. Ce réseau, appelé Target-Network, est mis à jour à chaque itération avec la formule $\theta' = (1-\alpha)\theta' + \alpha\theta$. Avec $\alpha = 0.01$. (nous avons aussi testé avec 0.005 sans obtenir de changements notables sur 1000 épisodes).

Afin de découvrir de nouvelles actions et expériences possiblement ignorées par la machine, nous avons dû implémenter une stratégie d'exploration. La stratégie que nous avons retenue a été *espilon-greedy* pour son ratio efficacité/simplicité. A chaque itération, la machine à une chance "epsilon" de choisir son action aléatoirement (et non en fonction des Q-Valeurs). Cependant cette chance décroît au fur et à mesure que l'algorithme apprend (et gagne en efficacité) en fonction d'un paramètre fixé. On lui fixe cependant toujours une valeur minimale pour être sûr de ne pas rester sur un maximum local. Cela implique donc l'ajout de nombreux paramètres que nous avons dû tester pour obtenir des résultats satisfaisants.

Résultats obtenus :



Partie 2 : Environnement plus difficile : Breakout Atari

Une des difficultés de cet environnement est que l'espace d'entrée de l'algorithme est de dimension beaucoup plus importante, puisqu'il s'agit de l'image du jeu (de casse briques Atari). Cela implique donc un certain nombre de modifications à l'algorithme initial ainsi qu'une étape de pré-traitement de l'environnement. Nous avons commencé par ne traiter l'information que toutes les 4 frames pour accélérer et améliorer l'apprentissage. Nous avons aussi réglé certains soucis de flickering sur Atari en prenant le maximum de chaque pixel entre 2 frames. Puis nous avons réduit

l'image (en 84x84) et l'avons transformée en nuances de gris (pour n'avoir plus qu'un seul canal à traiter par images). Nous avons ensuite créé un buffer constitué des 4 dernières images traitées pour constituer un observable de l'environnement (afin que la machine puisse déterminer des informations telles que la vitesse de la balle, ou son sens de déplacement).

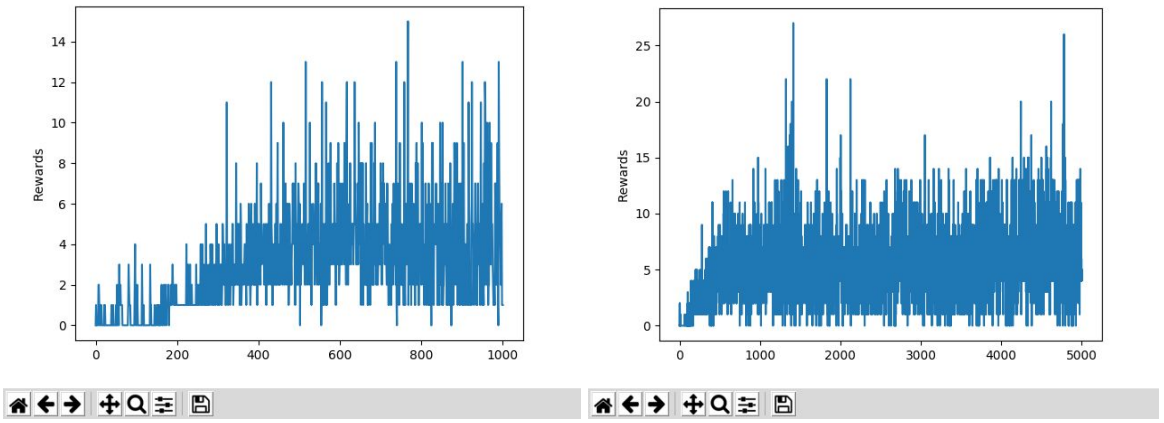
En plus de ces prétraitements, nous avons aussi fait en sorte que la balle soit tirée à chaque début d'épisode, pour accélérer l'apprentissage. Pour cela, nous avons fait en sorte qu'un épisode se termine à la perte d'une vie par la machine (au lieu de perdre ses 5 vies). Sur les 4 actions disponibles dans l'environnement, seules deux sont réellement utiles (l'action 0 ne fait rien et l'action 1 permet juste de tirer la balle au début). Pour améliorer les performances, nous avons cherché à faire en sorte que seule les 2 actions restantes (aller à gauche et à droite) soit à apprendre par la machine. Il peut aussi être intéressant de lui faire tirer la balle par elle même cependant.

Afin de pouvoir traiter efficacement ces images en entrée, nous avons dû mettre en place un réseau profond convolutionnel. La topologie et les paramètres de ce réseau sont aussi inspirés du papier de recherche. Nous avons aussi testé plusieurs optimizers différents afin de voir s'il était possible d'améliorer les performances.

Résultats obtenus:

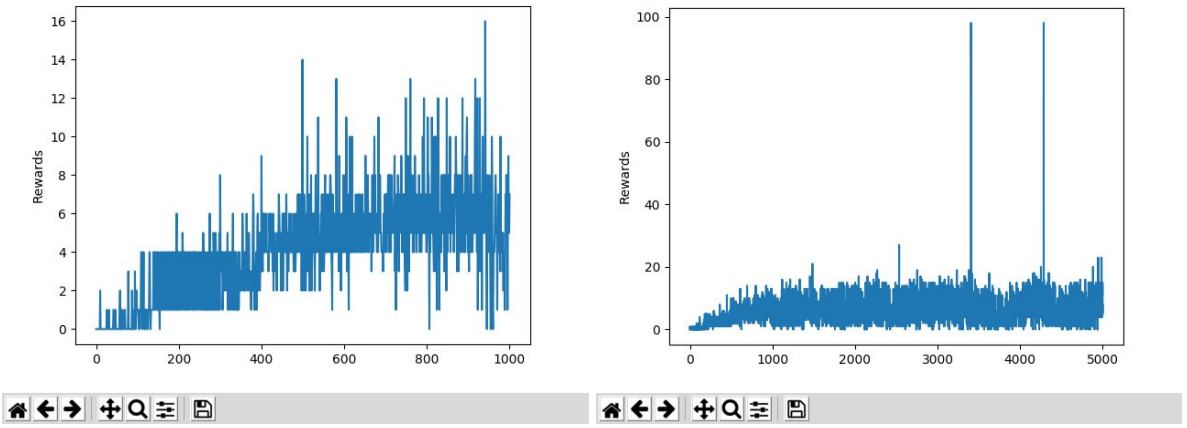
Malheureusement, ne disposant pas d'énormément de RAM sur ma machine (8GB), j'ai dû restreindre la taille du buffer d'expérience replay à 50 000. Le ramener à 100 000 pourrait améliorer les résultats.

Pas d'apprentissage	alpha (coeff de mise à jour du target network).	Optimiseur	Epsilon initial (stratégie d'exploration eps-greedy)	Coeff de decay (eps = eps * decay)	Epsilon min
5e-5	0.01	RMSprop	1	0.998	0.005



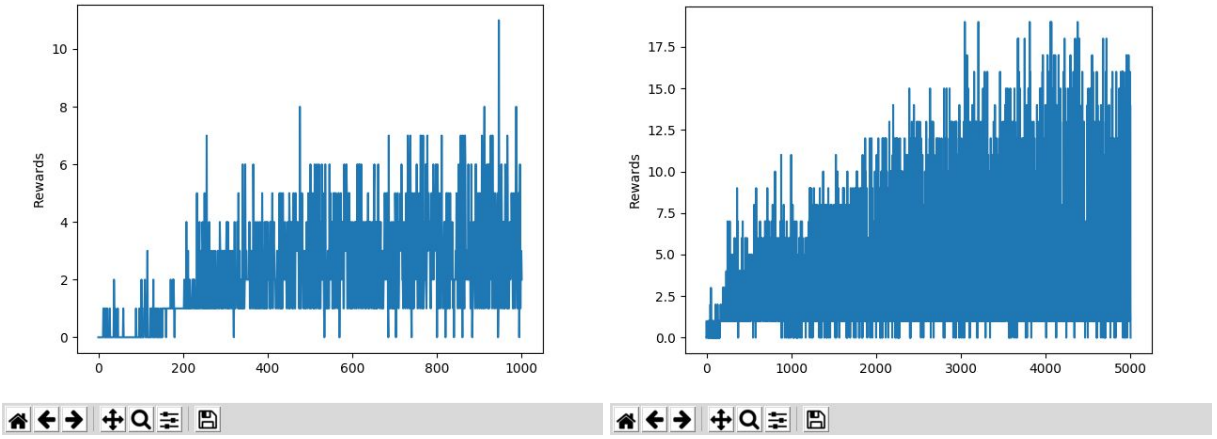
Avec Adam (autre optimiseur) et mêmes paramètres:

Pas d'apprentissage	alpha	Optimiseur	Epsilon initial	decay	Epsilon min
5e-4	0.01	Adam	1	0.998	0.005

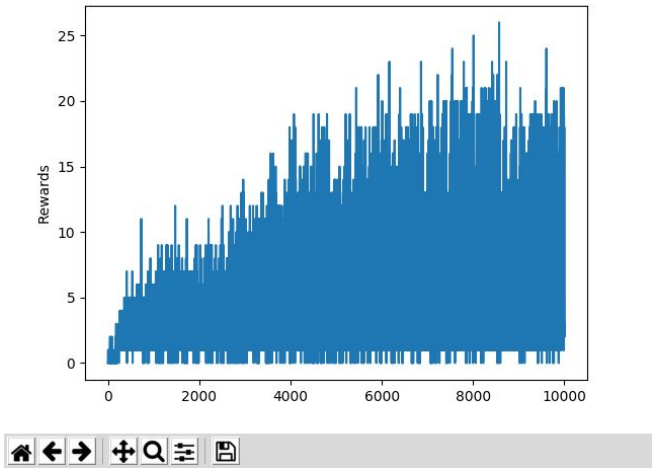


Les résultats sont meilleurs que ceux obtenus avec RMSprop mais ne sont pas encore satisfaisants.

En normalisant les récompenses obtenues de l'environnement (dans {-1, 0 , 1}):

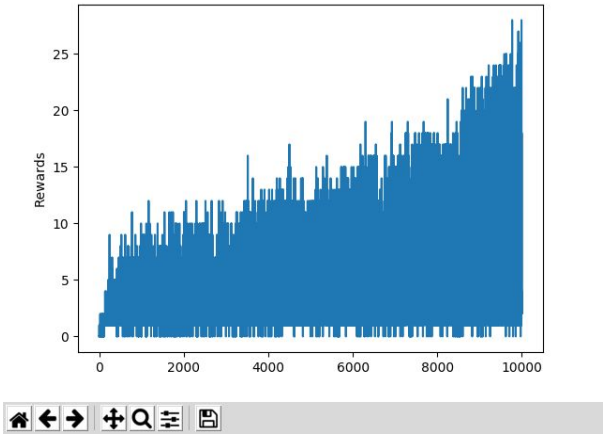


En diminuant le pas d'apprentissage (3e-4):



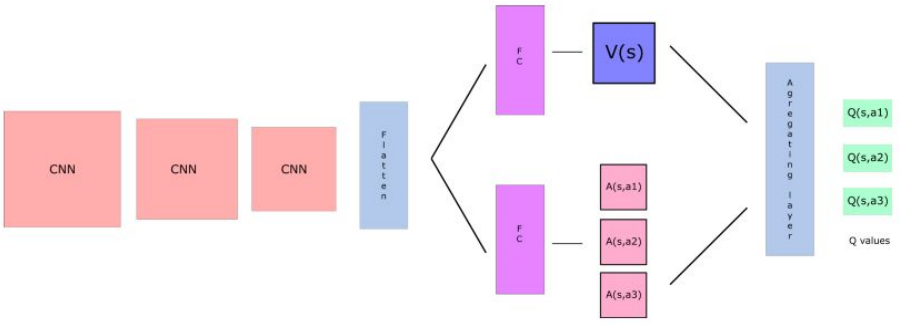
Les résultats semblent stagner.

En diminuant le coefficient de mise à jour du target network (alpha = 0.005):



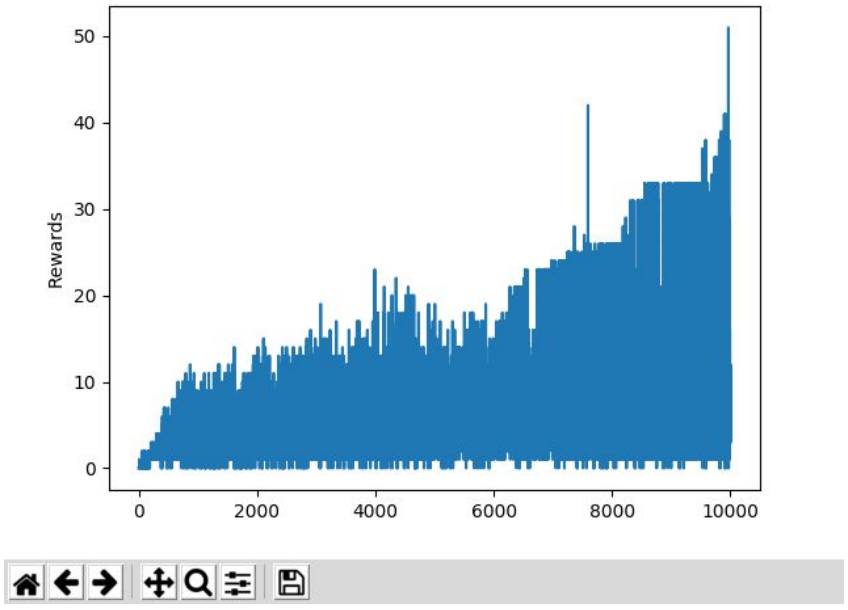
La convergence est plus lente mais progressive.

Afin de pouvoir encore améliorer les résultats, nous avons essayé de changer notre modèle de réseau de neurones pour un dueling Q-Network inspiré du papier de recherche [Wang et al., 2015]. Ce modèle sépare explicitement la fonction qui estime la valeur d'un état V(s) de celle qui estime l'avantage de chaque action A(s,a), avec Q(s,a) = A(s, a) + v(s), en utilisant deux estimateurs différents.



Avec ce nouveau modèles, les résultats s'améliorent encore significativement:

Pas d'apprentissage	alpha	Optimiseur	Epsilon initial	decay	Epsilon min
5e-4	0.005	Adam	1	0.998	0.005



Références :

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540) :529.

[Wang et al., 2015] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv :1511.06581. (voir: https://arxiv.org/abs/1511.06581)