Intelligence Bio-Inspirée
Travaux Pratiques

Compte-rendu des travaux pratiques sur l'apprentissage profond par renforcement

Réalisé par

SUBLET Gary – 11506450
VILLERMET Quentin – 11507338

Polytech Lyon
Université Claude Bernard Lyon 1

2020

Table des matières

1 Réplicabilité

1.1 Environnement virtuel

La totalité des expérimentations suivantes ont été réalisées avec Python 3.7, au sein de *notebooks jupyter* et de scripts *.py*.

Afin de favoriser la facilité de reproduction de ces programmes, nous évoluons dans un environnement virtuel, où les modules python peuvent être plus aisément contrôlés.
Nos dépendances sont listées dans le fichier `requirements.txt` à la racine du projet.

Un guide permettant la mise en place de ce *virtualenv* est présent dans le `Readme.md` du dépôt GitHub.

2 Deep Q-network sur CartPole

2.1 Début

2.1.1 Question 1 – Agent aléatoire CartPole-v1

Le code de l'agent aléatoire est entièrement contenu dans le fichier `cartpole.py`, trouvable dans les sources du projet.

Il est presque entièrement basé sur l'exemple des créateurs de `gym` présent sur le GitHub du module.

2.1.2 Question 2 – Évaluation des récompenses

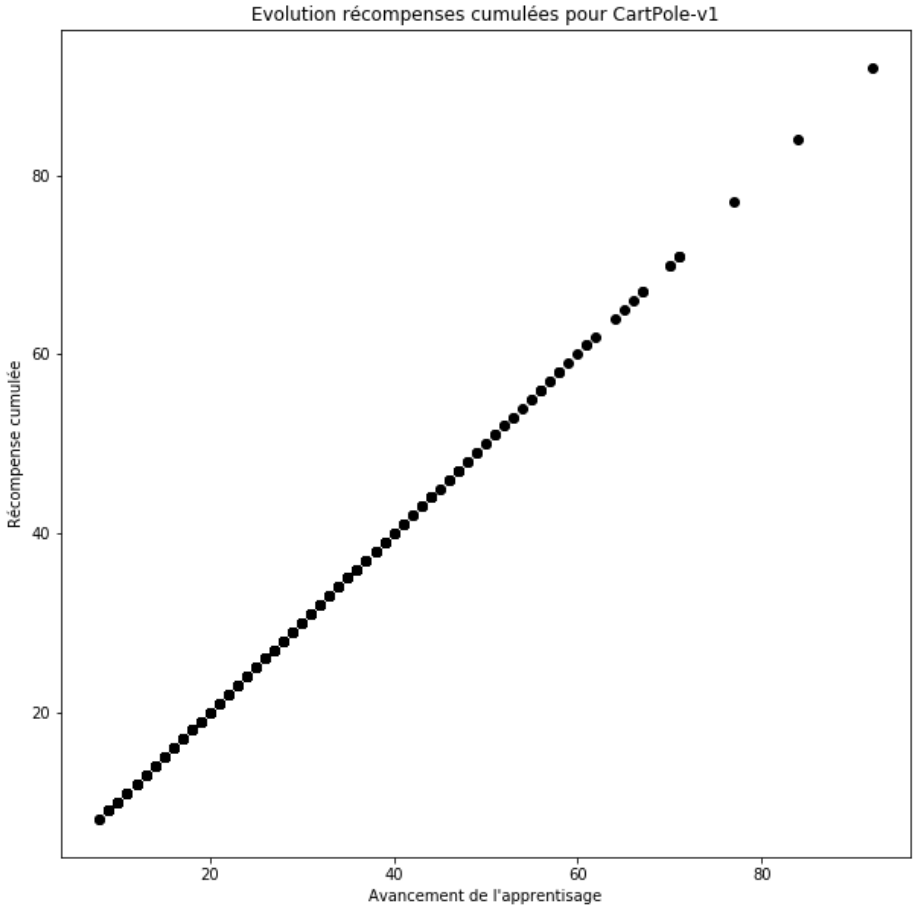
NB : En plus de l'affichage demandé, nous avons pris l'initiative de créer une visualisation de la récompense cumulée au fil des épisodes (dénotant en quelque sorte le « succès » de la session d'apprentissage.

Nous avons opté pour un affichage `matplotlib`, avec *mean-pooling* pour alléger le graphe en cas de grand nombre d'épisodes lorsque nécessaire.

Dans notre premier cas, soit celui d'un agent aléatoire sur l'environnement *CartPole-V1*, le graphe des récompenses cumulées en fonction du nombre d'interactions aboutit forcément à un nuage de points suivant la fonction $f(x) = |x|$.

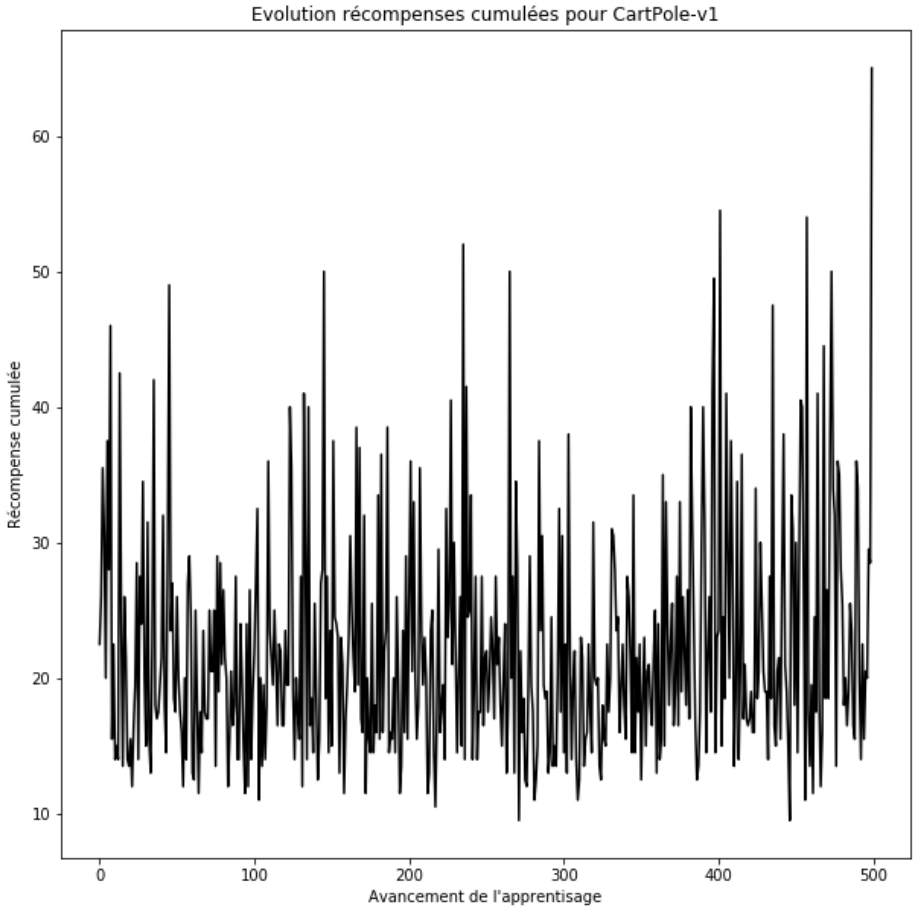
On constate peu d'*outliers* (qu'on considèrera ici comme ayant $x > 60$), en raison du caractère aléatoire des décisions.

19.12.2019



Ce résultat est sans surprise, puisque toute interaction entraîne une récompense de 1 dans cet environnement.

La deuxième visualisation n'est pour le moment pas très utile non plus, puisque notre agent n'apprend pas et agit de façon aléatoire.



Ces deux affichages prendront leur pertinence plus tard dans le développement, lorsque notre agent commencera à apprendre de son environnement.

Il nous indiqueront respectivement la rapidité d'apprentissage des épisodes individuels, et l'efficacité générale de l'apprentissage en fonction du temps passé (*i.e.* des épisodes).

2.2 Experience replay

2.2.1 Question 3 – Buffer de stockage d'expériences

Le code du buffer *experience replay* est dans le fichier `deeprl_agent.py`, plus particulièrement la fonction `experience` de la classe `Agent`.
Cette question marque le début du code des notebooks jupyter.

2.2.2 Question 4 – Sampling de mini-batch

À partir de cette question, nous basculons sur un système de notebooks *Jupyter* pour une implémentation plus rapide des tests.
La récupération de mini-batch de données du buffer se fait donc dans le fichier `experience_replay.ipynb`.

2.3 Deep Q-learning

L'implémentation du Q-learning se fait dans la continuation de la partie précédente, donc dans le fichier `experience_replay.ipynb`. Sauf indication du contraire, c'est au sein de ce notebook que le reste du TP se déroulera.

2.3.1 Question 5 – Réseau de neurones

Pour notre premier exemple, on construit donc un réseau avec 4 neurones d'entrée et 2 neurones de sortie.

En effet, dans l'environnement *CartPole-v1*, l'observation est de type `Box(4,)` et le domaine des actions est un `Discrete(2)` (en l'occurence 0 et 1 pour bouger à gauche et à droite respectivement).

Ces dimensions expliquent la taille de notre réseau.

2.3.2 Question 6 – Approximation des Q-valeurs

Les deux stratégies d'exploitation proposées sont implémentées dans la classe *Agent*, qui est dotée des fonctions `epsilongreedy(self, ob, epsilon)` et `boltzmann(self, ob, tau)` qui renvoient l'action choisie, qui est soit une action aléatoire d'exploration, soit le choix déterminé par la politique d'après les Q-valeurs prédites par le réseau.

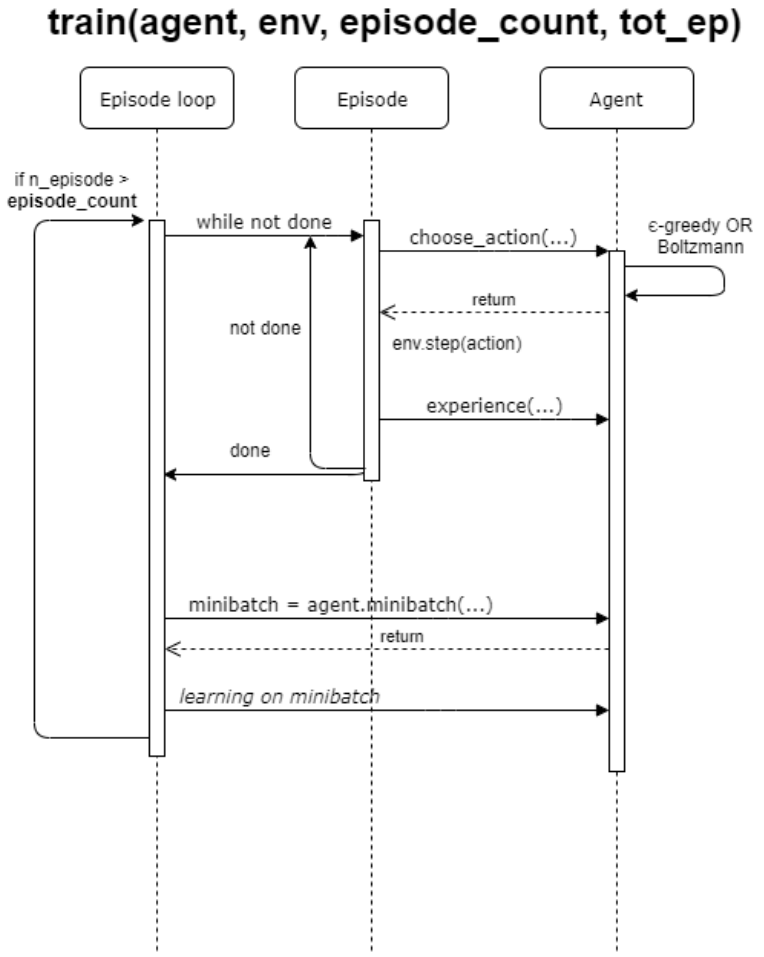
Pour la politique ϵ-greedy, il s'agit de la meilleure action avec une probabilité $(1 - \epsilon)$. Pour Boltzmann, le choix est stochastique mais tend malgré tout à choisir la Q-valeur maximale le plus souvent.

2.3.3 Question 7 – Apprentissage

À ce stade du développement, notre agent utilise désormais toute une panoplie de fonctions ; à chaque fois qu'il doit choisir une action, il fait appel à `choose_action(ob, reward, done)`, qui détermine les Q-values avant de déterminer (selon la méthode d'exploration actuelle) l'action choisie.

Après avoir effectué son action, celle-ci est appliquée à l'environnement par le biais de la fonction `step()`, après quoi on fait appel à l'*experience replay*. Pour ce faire, on utilise la fonction `experience()` de la classe *Agent*.

À la fin de notre épisode, on utilise `minibatch()` pour récupérer des expériences passées de l'agent, puis on apprend sur chacune des expérience de cette mini-batch.



Ci-dessus, un diagramme de séquence simplifié de l'entraînement d'un Agent.

2.3.4 Question 8 – Target Network