Rendu du projet

Le code sera envoyé par mail à arthur.aubret@univ-lyon1.fr, avec le lien github du projet. Le github devra contenir un readme.md à la racine expliquant comment lancer le code.

Un rapport récapitulant vos implémentations et expérimentations devra être joint au rapport. Le code doit être **fonctionnel** en **python3** ; tout ce qui n'est pas écrit dans le rapport **ne sera pas noté**.

Date finale de rendu : **5 janvier à 23h59**.

1 Préliminaires

1.1 Réplicabilité

L'objectif est de créer des expérience qui seront **réplicables** facilement par une autre personne, quelque soit son matériel. La **réplicabilité** est centrale à la démarche scientifique ; cela permet à d'autres experts de valider vos expériences et confirmer vos résultats.

Nous vous proposons deux manières de faire :

— Utilisation de Docker ou Singularity.

— Utilisation d'environnements virtuels.

Attention, dans les deux cas, il faudra poser le code en open-source sur github et préciser la procédure de réplicabilité dans un README.

1.1.1 Environnement virtuels

La solution la plus simple est d'utiliser un environnement virtuel. Plusieurs choix s'offrent à vous : virtualenv + pip, pipenv ou anaconda. Un environnement virtuel permet d'isoler les dépendances python du projet des dépendances python installées dans le système ou dans les autres projets. Cela évite de nombreux conflits de version entre vos différents projets.
Si vous souhaitez utiliser pipenv, faites :

```
pip3 install pipenv
mkdir my_project_directory
cd my_project_directory
pipenv shell #Créer et/ou active l'environnement virtuel du projet
pip3 freeze > requirements.txt #Stocke les dépendances pour qu'un autre utilisateur les
```

1.1.2 Docker/Singularity

Dans le cas de Docker et singularity, il faudra fournir avec le code une *recette* (ou dockerfile pour docker) dans laquelle vous définirez les dépendances système et python. Une autre option est de poser votre image dans le hub (répertoire d'images en ligne).

1.2 Gym

1.2.1 Introduction à Gym

En apprentissage par renforcement, il y a deux concepts fondamentaux ; l'agent et l'environnement. L'agent est l'entité apprenante qui observe l'environnement et agit sur celui-ci selon les actions disponibles. Son objectif est de maximiser la récompense cumulée qu'il recoit de l'environnement avec lequel il interagit. Chaque environnement a :

— Un espace d'action.

— Un espace d'état.

— Une fonction de récompense.

Gym (https://github.com/openai/gym) propose une interface unifiée entre un agent et un environnement. Ainsi, il est possible de construire indépendamment un agent de l'environnement et inversement ; il suffit alors de satisfaire l'interface pour qu'un nouvel environnement soit compatible avec presque tous les agents précédemment codés pour d'autres environnements. Lorsque certains pré-traitements sont nécessaires sur les actions, observations ou récompenses, il est possible d'encapsuler l'environnement dans un wrapper, celui-ci se chargera du pré-traitement. De plus gym a un méchanisme de registre qui permet de construire un environnement simplement avec son nom.

Dans notre cas nous allons implémenter un agent que l'on pourra ensuite tester sur plusieurs environnements différents.

Il existe plusieurs fonctions clé pour interagir avec un environnement. Un exemple très simple d'agent agissant de manière aléatoire dans un environnement peut être trouvé sur https://github.com/openai/gym/blob/master/examples/agents/random_agent.py.

Une liste de tous les environnements est disponible sur https://github.com/openai/gym/wiki/Table-of-environments

1.2.2 Installer gym

```
pip3 install gym
```

Pour ce projet, vous aurez besoin d'une librairie de deep learning. Choisissez celle que vous préférez ; par exemple :

```
pip3 install torch #Pour Pytorch
pip3 install tensorflow #Pour tensorflow
```

(Facultatif) Si vous souhaitez utiliser le GPU de votre machine, vous pouvez installer CUDA et cuDNN. Attention cependant à bien choisir une version compatible avec votre librairie de deep learning. Cela peut se révéler laborieux à installer, ce n'est donc pas recommander dans un premier temps.

2 Deep Q-network sur CartPole

2.1 Début

L'algorithme que nous allons implémenter est décrit dans [Mnih et al., 2015]. Plus de précisions et d'aides sont disponibles dans l'article. Nous l'appliquerons à la tâche CartPole-v1, qui consiste à garder un bâton levé le plus longtemps possible.

Question 1. *En vous inspirant du lien donné ci-dessus, faites interagir un premier agent de manière aléatoire avec l'environnement CartPole-v1.*

Question 2. *Pour évaluer notre agent, nous aurons besoin de suivre l'évolution de la récompense obtenue à chaque épisode. Proposez une manière de suivre l'évolution de la somme de récompense par épisodes en fonction du nombre d'interaction de l'agent avec l'environnement. Une manière de faire serait d'afficher des logs et/ou d'afficher la courbe à la fin de l'apprentissage via matplotlib.*

Rappel : Un épisode débute lors du reset et se termine lorsqu'il recoit un signal de fin de la part de l'environnement.

2.2 Experience replay

Nous allons maintenant pouvoir commencer l'implémentation de notre algorithme. Le DQN utilise l'experience replay, i.e., il stocke dans en mémoire toutes les interactions qu'il recoit, où interaction=(état,action,étatsuivant,récompense,finepisode). Le buffer a une taille maximale (100 000 par exemple) ; lorsqu'elle est dépassée, les nouvelles expériences remplacent les plus anciennes. L'agent va apprendre des expériences stockée dans son buffer, il choisira aléatoirement un minibatch d'expériences dans son buffer.

Question 3. *Implémentez le buffer et stockez les données reçues. Faites attention à bien gérer le dépassement de la taille maximale. Le buffer doit être indépendant de l'environnement, et donc de l'espace d'action et d'état.*

Question 4. *Implémentez une fonction de sampling permettant de récupérer un minibatch de données aléatoires dans le buffer.*

2.3 Deep Q-learning

Nous allons utiliser des réseaux de neurones pour approximer l'espérance de récompense cumulée $Q(s, a)$. Ensuite nous appliquerons l'équation de Bellman.

Question 5. *Construisez un réseau de neurone qui, prenant en entrée un état, renvoie la Q-valeur de toutes les actions.*

Question 6. *Plutôt que de choisir des actions aléatoires, nous pouvons utilisez cette approximation de Q pour choisir les actions. A chaque nouvel état que reçoit votre agent, calculez la Q-valeur des actions. Choisissez également une stratégie d'exploration pour découvrir de nouveaux états.*

Nous proposons deux stratégies classiques d'exploration.

La méthode $\epsilon - greedy$ permet de choisir avec une probabilité ϵ, une action aléatoire, et avec une probabilité $(1 - \epsilon)$ la meilleure action. Cette stratégie a l'avantage d'être simple.

L'exploration Boltzmann permet de choisir une action a_k selon la probabilité :

$$P(s, a_k) = \frac{e^{\frac{Q(s,a_k)}{\tau}}}{\sum_i e^{\frac{Q(s,a_i)}{\tau}}}. \tag{1}$$

où τ un hyper-paramètre désignant la stochasticité de l'exploration, plus il est grand, plus la politique sera aléatoire, plus il est faible, plus la politique tendra vers une politique déterministe maximisant la Q-valeur actuelle. Intuitivement, l'exploration Boltzmann pondère la probabilité d'une action selon sa Q-valeur.

Question 7. *Pour l'instant, notre agent n'apprend pas. Après chaque interaction, choisissez aléatoirement dans votre buffer un minibatch de données, calculez les Q-valeurs, appliquez l'équation de Bellman et rétropropagez l'erreur dans le réseau de neurone.*

Rappel :

- $J_\theta = \left[Q_\theta(s, a) - (r(s, a) + \gamma \max_{a'} \hat{Q}_\theta(s', a'))\right]^2$ si l'épisode continue.
- $J_\theta = [Q_\theta(s, a) - r(s, a)]^2$ si l'épisode se termine.

A partir de là, il se peut que votre agent apprenne des comportements intéressants mais qu'ils soient très instables. C'est dû au fait que la fonction d'erreur n'est pas stationnaire, modifié

$Q_\theta(s,a)$ modifie également $\hat{Q}_\theta(s',a)$; une backpropagation du gradient peut alors augmenter l'erreur.

Question 8. *Nous allons utiliser un target network pour stabiliser l'apprentissage. Clonez votre réseau de neurone et utilisez le duplicat par calculer la Q-valeur ciblée. Mettez à jour de manière graduelle le duplicat vers le réseau de neurone original.*

Il y a deux manières possible de mettre à jour le target network :

- Toutes les N étapes apprentissages (10000 par exemple), recopier entièrement le réseau de neurone original dans le duplicat.
- Mettre à jour petit à petit le duplicat à chaque étape d'apprentissage : $\theta' = (1-\alpha)\theta' + \alpha\theta$ où θ' sont les poids du duplicat, θ les poids du réseau original, et α le pas de mise à jour. Souvent $\alpha = 0.005$ ou $\alpha = 0.01$.

Dorénavant, votre DQN est censé fonctionner et votre agent devrait réussir à manipuler le CartPole !

3 Environnement plus difficile : Breakout Atari

Maintenant qu'on est sur que votre algorithme de DQN fonctionne. Nous allons le tester sur un environnement plus compliqué sur lequel il mettra plus de temps à apprendre un comportement intéressant. Par ailleurs, notre agent utilisera cette fois des pixels directement comme états.

Question 1. *Construisez le nouvel environnement BreakoutDeterministic-v4. Cet environnement répète automatiquement 4 fois une action avant de renvoyer l'état, comme dans l'article original. Vous aurez besoin d'un wrapper faisant le pré-processing de l'image : Il faut passer les pixels en gris, rassembler les 4 dernières frames, prendre le pixel maximum des deux dernières frames, etc...*

Vous pouvez notamment vous aider du wrapper disponible ici https://github.com/openai/gym/blob/master/gym/wrappers/atari_preprocessing.py.

Après pré-processing, un état correspondra aux 4 dernières frames, grisées, chacun de taille 84x84. Ainsi, la taille totale d'un état sera 4x84x84, avec 3 dimensions . Vous pouvez éventuellement normaliser les pixels.

Question 2. *Adaptez l'ensemble de votre code, notamment le buffer d'experience, pour qu'il puisse également gérer des états multi-dimensionnels.*

Question 3. *Remplacez votre réseau de neurone entièrement connecté par un réseau de neurone convolutionnel.*

Question 4. *A l'aide de l'article original (plus haut), cherchez les meilleurs hyper-paramètres et essayer d'obtenir la meilleure récompense possible.*

Question 5. *Filmez à l'aide du wrapper de vidéo les comportements de votre agent. Sauvegardez également le réseau de neurone appris par votre agent de manière à pouvoir le relancer et tester facilement.*

4 Bonus possibles : Pour une excellente note

Les trois groupes obtenant la meilleure récompense sur Atari dans la promotion auront immédiatement un bonus de points, sous réserve que la politique apprise soit testable. Les scores et noms des trois meilleurs groupes seront affichés sur la page github.

En bonus, il y aurait plein d'autres choses amusantes/intéressantes à faire, nous vous proposons plusieurs exemples :

Extensions DQN : plusieurs extensions sont possibles : Double-DQN [Van Hasselt et al., 2016], distributionnal DQN [Bellemare et al., 2017], dueling DQN [Wang et al., 2015], prioritized experience replay [Schaul et al., 2015], soft DQN [Haarnoja et al., 2017]. Pour les plus motiver, vous pouvez essayer le rainbow [Hessel et al., 2018].

Réseaux récurrents : implémentez et testez DRQN [Hausknecht and Stone, 2015] sur un environnement partiellement observable comme VizDoom [Kempka et al., 2016]. Il s'agit d'ajouter un réseau de neurone récurrent après le CNN pour gérer la mémoire.

Comparez un autre algorithme avec DQN : par exemple : A2C [Mnih et al., 2016], PPO. [Schulman et al., 2017].

Essayez un autre algorithme Actor-Critic : par exemple DDPG [Lillicrap et al., 2015] ou SAC [Haarnoja et al., 2018]. Vous pouvez essayer l'algorithme sur Pendulum avant de l'essayer sur des environnements plus compliqués.

Essayez d'autres environnements : Vous pouvez tester votre algorithme dans votre propre environnment ou l'essayer sur des environnements plus atypiques trouvables sur internet. Par exemple Mario bros : `https://pypi.org/project/gym-super-mario-bros/`.

Curiosité : Essayez d'explorer votre environnement avec ICM [Pathak et al., 2017].

Références

[Bellemare et al., 2017] Bellemare, M. G., Dabney, W., and Munos, R. (2017). A distributional perspective on reinforcement learning. *arXiv preprint arXiv :1707.06887*.

[Haarnoja et al., 2017] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org.

[Haarnoja et al., 2018] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv :1801.01290*.

[Hausknecht and Stone, 2015] Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *2015 AAAI Fall Symposium Series*.

[Hessel et al., 2018] Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow : Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[Kempka et al., 2016] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom : A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv :1509.02971*.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[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.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv :1511.05952*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv :1707.06347*.

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[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*.