Rapport de TP
Apprentissage profond par renforcement

Théo Rabut 11307400
Tristan Syrzisko - 11809087

1 Introduction

Ce rapport a pour but de décrire, en synthétisant, ce qu'on a fait pour ce projet. Notre objectif, dans un premier temps, était d'aller le plus loin possible mais nous avons vite été limité par la durée d'apprentissage. Nous allons reprendre question par question ce que l'on a réalisé.

2 Cartpole

L'agent aléatoire a été fait, il nous sert encore pour remplir la mémoire d'apprentissage pour accélérer la généralisation de l'apprentissage.

Pour évaluer l'agent nous avons choisi de faire 3 courbes :

- Courbe de résultats, où l'on note pour chaque épisode (x) le résultat de l'agent (y).
- Courbe d'erreur (loss function), où l'on note l'erreur commise à chaque pas d'apprentissage (x) de l'agent.
- Courbe de Qvaleurs, où l'on note pour chaque action (x, couleur) choisie par l'agent (hors exploration) sa qvaleur (y).

Ces méthodes nous ont permis d'évaluer le progrès, la stabilité et la performance de l'apprentissage.

L'expérience replay a été fait avec une simple liste python. Nous aurions pu utiliser une dequeue mais nous trouvé plus simple de procéder comme nous l'avons fait. Nous créons les mini-batchs à la volée grâce au paquet random.

Les différents réseaux que l'on a construit ont été fait en utilisant les modules pytorch.nn

Les deux politiques (e-greedy / exploration de Boltzmann) ont été faites. Nous avons choisi d'utiliser e-greedy puisque nous sommes plus familiers avec et que la politique importe peu, tant qu'il y a une exploration. Epsilon va se dégrader de plus en plus, pour finalement atteindre une valeur minimale.

L'erreur (équation de Bellman) est calculée grâce au module mse_loss.

La mise à jour du target network a été faite grâce à un compteur, plutôt qu'à chaque pas d'apprentissage.

Résultats : Notre agent apprend correctement. On aurait aimé avoir une courbe de progression linéaire mais ce n'est pas le cas. Vous trouverez une vidéo de cartpole qui fait le meilleur score (500).

3 Breakout

Le pré-traitement des observations du breakout a été fait, nous nous sommes inspirés d'un wrapper existant mais nous avons principalement suivi les instructions de l'article mentionné dans le sujet.

- Lorsque l'agent fait une action, nous avons fait 4 fois l'action sur l'environnement en gardant toute les observations/récompenses.
- Pour chaque frame et pour chaque pixel, nous avons pris le maximum entre la frame courante et la frame précédente (mise en mémoire de la dernière frame pour l'action suivante)
- Si nous sommes en mode apprentissage, l'agent perd la partie lorsqu'il perd une vie.
- Si la fin de la partie arrive avant les 4 répétitions, nous copions simplement la dernière frame le nombre de fois restantes.
- On prend le maximum des trois couleurs de chaque pixel (réduction de dimensions 3 -> 1)
- On transforme les dimensions de l'image en utilisant OpenCV
- Il est possible de normaliser l'image
- Lorsqu'on reset l'environnement, on produit 4 fois l'opération NO-OP pour renvoyer 4 frames à l'agent.

Puisqu'on est passé par une liste python, le buffer d'expérience n'a pas eu besoin d'être adapté.

Nous avons fait deux versions différentes des réseaux convolutionnels :

- une qui correspond à l'architecture décrite dans le papier publié dans Nature,
- l'autre plus « custom ».

Nous initialisons la mémoire de l'agent (40 000 épisodes) avec des actions aléatoires, pour éviter de sur-apprendre les premières configurations.

Les méthodes d'évaluation que l'on a utilisé sont les mêmes que pour l'agent cartpole.

Le nombre d'épisodes pour l'apprentissage importe peu. En effet, nous avons pu voir dans les différents articles et projets que l'on a pu rencontrer que l'important est principalement le nombre de pas d'apprentissage.

Malheureusement, nous n'avons pas réussi à avoir un apprentissage satisfaisant avec un DQN-CNN simple et nous n'avons pas réussi à identifier la cause du problème. Selon nous, la structure et les hyper-paramètres sont identiques à ceux présentés dans l'article.

4 Conclusion

Nous avons eu quelques difficultés pour incorporer la dimension de Batch dans l'apprentissage. En effet, dans un premier temps, nous envoyions les éléments un par un dans le réseau pour rétro-propager l'erreur sur chaque élément.
Nous aurions aimé aller plus loin afin d'étudier d'autres architectures que le réseau convolutionnel simple (DoubleQDN / Récurrent). L'article intitulé « Prioritized experience replay. » nous paraissait être une bonne façon d'améliorer ce composant indispensable de l'apprentissage.

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.*
- *[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv :1511.05952.*
- *https://github.com/openai/gym/blob/master/gym/wrappers/atari_preprocessing.py*