Apprentissage par renforcement

Guillaume VANEL, Valérian ACIER

Janvier 2020

1 Introduction

Pour réaliser ce TP nous avons utilisé l'environnement Google Colab nous permettant de travailler en collaboration en ligne et de réaliser les différents calculs directement sur un serveur mis à disposition gratuitement par Google.

Le code du projet est divisé en différentes parties ayant chacune un rôle.

- La fonction **train** a pour but d'entraîner le réseau de neurones à l'aide d'un batch d'expériences, elle va donc appliquer la rétro propagation de l'erreur calculée grâce à l'équation de Bellman.
- Les fonctions **exploration_greedy** et **exploration_boltzmann** sont les fonctions permettant à l'agent de choisir entre l'exploitation (choisir l'action qui maximise notre estimation de Q) et l'exploration (choisir une action aléatoirement).
- La fonction **minibatch_from_memory** permet de tirer de manière aléatoire un batch d'expériences déjà jouées par l'agent.
- La fonction **evaluate_model** nous permet d'évaluer notre modèle sur un certain nombre de parties: elle renvoie le score moyen obtenu.
- La fonction **train_on_game** est la fonction en charge de l'apprentissage, elle prend en comptes les différents hyper-paramètres (fonction d'exploration, l'environnement, epsilon, gamma ...) pour entraîner un agent.

2 Cartpole

Nous avons testé différentes architectures de réseaux entraînés sur 50 000 actions dans l'environnement Cartpole. L'évolution de leur apprentissage est représentée figure 1: pour chaque réseau sont tracés les scores moyens obtenus toutes les 50 parties. Le réseaux utilisés ont tous une architecture à deux couches cachées avec fonction d'activation ReLU et de taille identique: 4, 5, 10, 20, 30 ou 100 neurones.

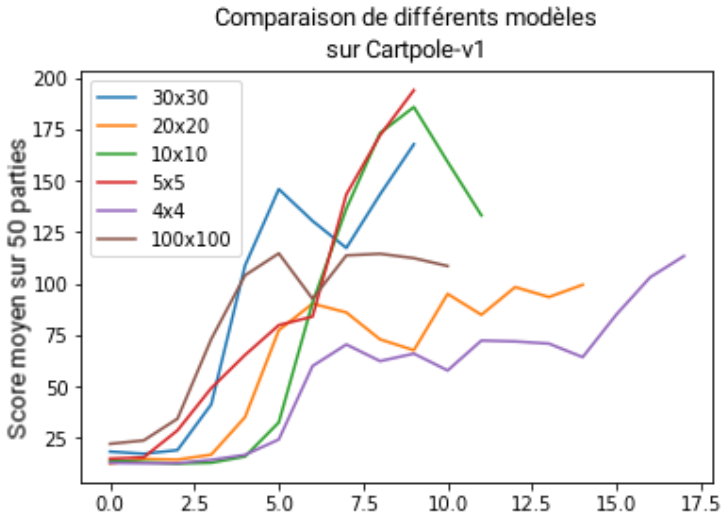

Figure 1: Apprentissage de différents réseaux sur le jeu Cartpole v1

À la fin de l'apprentissage, les agents ont été testés sur 1000 parties, obtenant les scores moyens résumés table 1

Architecture	Score moyen
4x4	145.0
5x5	152.2
10x10	140.2
20x20	119.1
30x30	133.4
100x100	153.5

Table 1: Évaluation des performances des différentes architectures de réseau

L'apprentissage comporte une grande part de stochasticité, que ce soit dans son démarrage, car les actions sont tout d'abord prises aléatoirement, ou dans la fonction d'exploration. Ainsi, il serait bon d'entraîner de nombreuses fois chaque architecture pour être capables d'obtenir des tendances moyennes et d'étudier la significativité des différences de performances entre les réseaux. Cela n'a pas été fait pour des raisons de temps d'apprentissage. Ainsi, les courbes 1 ne sont pas nécessairement très représentatives des performances du réseau tout au long de l'apprentissage. On notera par exemple que le réseau 100x100 semble avoir des résultats assez moyens sur les 50 parties de son apprentissage, mais lorsqu'on

le teste ensuite sur un échantillon de taille plus importante, on remarque qu'il s'agit d'un des meilleurs agents entraînés.

Quoiqu'il en soit, on note clairement un apprentissage effectué par tous les réseaux sur le jeu, puisque toutes les architectures permettent à l'agent d'obtenir un score moyen entre 110 et 160, là où le hasard ne permet pas de dépasser quelques actions avant de laisser tomber le bâton.

3 Breakout

L'apprentissage sur le jeu Breakout étant très long nous avons mis en place la sauvegarde et le chargement du modèle via la fonction **save_model** comme nous utilisons l'environnement Google Colab nous sauvegardons les modèles directement sur Google Drive.

Pour simplifier les spécificités du jeu Breakout nous avons surchargé l'environnement gym avec la classe "BreakoutEnv" pour internaliser certaines actions. Cela permet entre autres la transformation des 4 derniers états dans un seul et même état, la possibilité de sauvegarder la partie en GIF, ou encore de suivre de manière plus simple l'évolution de l'agent, nous avons défini que l'agent ne possède qu'une seule vie, ainsi la variabilité stochastique des scores est réduite.

4 Prioritized experience replay

Nous avons mis en place une amélioration de l'utilisation de l'expérience replay: la *prioritized experience replay*. La fonction **minibatch_prioritized** permet de tirer un batch d'expériences non plus avec une distribution de probabilités uniforme, mais avec une priorité définie pour chaque expérience. La probabilité de tirer une expérience i de priorité u_i est ici:

$$p_i = \frac{u_i}{\sum_{k=1}^{N} u_k}$$

Chaque fois qu'une expérience replay est utilisée pour l'entraînement, on met à jour la priorité qui lui est associée. Sa nouvelle valeur correspond à l'erreur quadratique entre l'estimation de Q et la valeur attendue. Ainsi, plus on se trompe sur une expérience, c'est-à-dire plus elle est surprenante et contient des informations qu'on n'a pas encore apprises, plus on a de chance de la réutiliser dans l'apprentissage. Au contraire, une expérience sur laquelle on n'a fait qu'une erreur très faible est considérée comme acquise et peut diminuer en priorité. Toute nouvelle expérience va se voir attribuer une priorité maximale afin qu'elle soit jouée au moins une fois avant de pouvoir être évaluée comme importante ou non par le procédé précédent.

4.1 Les résultats

Dû à des temps d'entraînement très longs nous n'avons pu aller jusqu'à l'émergence de stratégies avancées dans le comportement de l'agent. Nous pouvons cependant constater une nette amélioration du temps d'apprentissage suite à l'implémentions du prioretized experience replay, il aura fallu pour obtenir un score moyen de 5 une dizaine d'heures sans cette amélioration là ou en 5 heures nous obtenons des résultats similaires avec.

5 Conclusion

Nous avons pu mettre en place les techniques de deep Q learning et constater la progression de l'agent pour jouer à différents jeux: Cartpole et Breakout. On remarque un apprentissage extrêmement long rendant difficile l'obtention et l'exploitation de résultats très significatifs sans une infrastructure et une puissance de calcul adaptée. Cependant, plusieurs heures d'entraînement sur des ordinateurs personnels permettent de vérifier que l'apprentissage se fait effectivement et de noter une progression et un comportement qui ne peut pas être dû au hasard, mais relève d'une certaine adaptation au jeu.