Devoir 2

Travail présenté à Audrey Durand dans le cadre du cours *IFT-7201 : Apprentissage par renforcement*

Travail réalisé par :
NAOUSSI SIJOU, Wilfried Armand - NI : 536 773 538
Jack Goodall - NI : 537 010 048
Hugo Meignen - NI : 537 016 570

Automne 2022

1 MountainCar-v0 OpenAI Gym

Etant donné $r_t = -1 \ \forall t$ et $\gamma = 1 \implies Q(s,a) \leq 0 \ \forall s, a$. Ainsi pour un état donné s, si une action a a déjà été jouée, $Q(s,a) < 0$. L' action sélectionnée est celle qui maximise Q. Soit 0 pour une action qui n'a pas encore été jouée pour un état donné s.

Dans les bandits stochastiques, cela équivaut à jouer chaque action au moins une fois.

2 SARSA(λ)

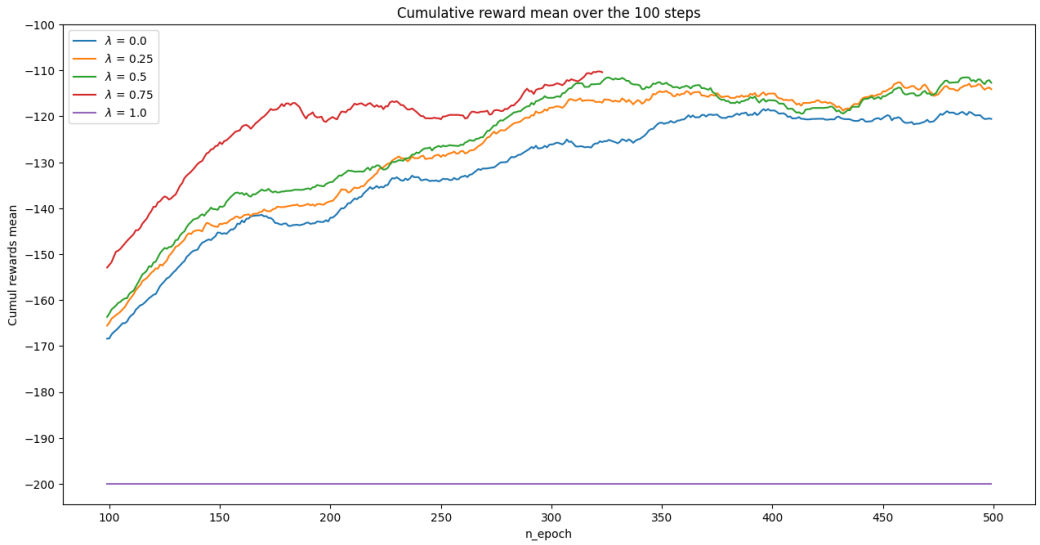

FIGURE 1 – Moyenne cumulatif des 100 derniers épisodes

λ est le rapport de diminution des traces. Il permet de contrôler la quantité d'information des traces (actions sélectionnées antérieures) à considérer pour le calcul de l'erreur. Il joue donc un rôle majeur dans la convergence et la stabilité de Sarsa(λ).

Pour les valeurs de λ égales à 0, 0.25, 0.5 et 0.75, la vitesse de convergence semble diminuer avec la croissance de λ.

Par contre, pour $\lambda = 1$, l'agent n'arrive pas à terminer ne serait-ce qu'un épisode dans le temps imparti (ie $R_T = -200$ à tous les épisodes).

Pour mieux comprendre ce qui se passe nous avons testé avec des valeurs de λ plus proches de 1 (0.9, 0.95, 0.96, 0.97, 0.98, 0.99). On remarque alors que l'algorithme arrive à résoudre MontCar-v0 en moins de 150 époques ce qui constitue meilleure performance enregistrée. Cependant, 0.95 à 0.98 cette performance se détériore en fonction de la grandeur de λ. On observve alors pour 0.99 que

l'algorithme obtient le même score qu'avec 1.

D'après ces observations, la prise en compte de la totalité ou la quasi-totalité des traces pour le calcul de l'erreur entraine une grande instabilité et comme conséquence la divergence de l'algorithme. Ceci pourrait s'expliquer par le fait que l'état actuel et celles antérieurs ont le même poids dans le temps. Ce qui déboussole complètement l'agent car il n'a plus de repère.

3 Motivez les astuces utilisées pour stabiliser l'entraînement d'une stratégie Deep Q-Learning

(a) Principal avantage d'utiliser $q_\theta(s) \in \mathbb{R}^{|\mathcal{A}|}$ dans un modèle d'approximation dans les environnements à actions discrètes :

Réduire la latence grâce à un gain de temps de calcul. En effet, l'apprentissage par lot simplifie les opérations d'aller/retour du réseau de neuronnes (foward propagation/ Backward propagation).

(b) Utilité d'un réseau cible :

En RL, la cible $r_t + \gamma \max_{a' \in A} q_\theta(s_{t+1}, a')$ change continuellement à chaque itération. Ce qui rend l'apprentissage instable avec les réseaux de neuronnes. Pour remédier à cela, une solution est d'utiliser un réseau cible. Ce réseau à la même architecture que le réseau qui approxime la fonction mais avec un paramètre gelé qui est mis à jour à chaque itération. Cela conduit à un entrainement plus stable car la fonction cible reste fixe pendant un certain temps.

(c) Heuristique pour déterminer θ^- :

Soit l'heuristique : $\theta^- = (1-\tau)\theta^- + \tau\theta$. Le choix de τ permet de contrôler la variation du paramètre θ^- en fonction de θ.

Si $\tau = 0$, $\theta^- = (1-0)\theta^- + 0 * \theta \implies \theta^- = \theta^-$
θ^- est constante. l'entrainement du réseau est stable, mais probablement pas adapté car l'écart entre θ et θ^- peut être considérablement grand.

Si $\tau = 1$, $\theta^- = (1-1)\theta^- + 1 * \theta \implies \theta^- = \theta$
le réseau cible utilise la valeur de θ pour calculer la cible. Cela revient pratiquement à ne pas avoir de réseau cible. Par conséquent cause un entrainement instable.

Choisir une bonne valeur de τ dans ce cas revient à déterminer une valeur comprise entre 0 et 1 de telle sorte que l'entrainement du réseau soit le plus stable possible tout en restant le plus proche possible de la paramétrisation estimée ou réelle du problème.

(d) Utilité d'un replay buffer :

En RL nous recevons des échantillons séquentiels provenant des interactions avec l'environnement. Le réseau peut être amené à voir trop d'échantillons d'un même type et à oublier les autres.

Une solution à cela consiste à utiliser un replay buffer.

- Il permet de briser la corrélation temporelle des échantillons d'entraînement et un meilleur comportement de convergence lors de l'apprentissage d'un approximateur de fonction. Cela s'explique en partie par le fait que les données sont plus proches des données i.i.d. supposées dans la plupart des preuves de convergence de la descente de gradient stochastique.

- De plus, Lorsqu'une expérience est coûteuse(transition à éviter) ou rare(manque d'informations), la connaissance des transitions antérieures permet de tirer pleinement parti de celle-ci.

(e) Différence Replay buffer et Apprentissage supervisé :

- La régression logistique sépare $\mathcal{D}$ en train/test/validation tandis que le replay buffer pige de manière i.i.d. dans $\mathcal{D}$ pour entrainer le modèle.

- Dans la regression logistique, la sortie y est connue pour chaque entrée x tandis que le replay buffer se sert d'une estimation de la cible $\hat{y} = r_t + \gamma \max_{a \in A} q_\theta(s_{t+1}, a')$ pour chaque entrée $x = (s, a)$

4 Expérimentations avec Deep Q-Learning

(b) Dificultés rencontrées :

Les principales difficultés rencontrées sont l'intégration du réseau de neurones dans l'environnement ainsi que le temps d'entrainement qui limite la recherche des hyperparamètres.

Grâce aux valeurs données dans l'énoncé, on obtient le 1er succès au bout de plus de 250 itérations. Mais on observe que les valeurs des récompenses cumulées ondulent beaucoup trop.

Pour réduire le temps d'entrainement, nous avions deux possibilités : réduire la taille des lots ou augmenter l'intervalle d'entrainement N.

Pour stabiliser l'entraînement :
- Modifier τ pour que θ^- fit l'évolution de θ (comme mentionné à la question 3).
- Modifier le pas d'apprentissage (learning_rate) qui joue un rôle important dans la descente du gradient stochastique. Si trop petit converge lentement sinon si trop grand, alors il y'a risque de sauter le minimum et osciller autour de cette valeur sans jamais converger.

Enfin, pour s'assurer que notre réseau continue à explorer des solutions une valeur minimale d'epsilon a été également explorée.

A la fin, nous avons pu stabiliser l'apprentissage en utilisant les valeurs suivantes : $\tau = 0.01$, la taille du *replay buffer* à 10^6, et l'intervale d'entraînement $N = 2$.

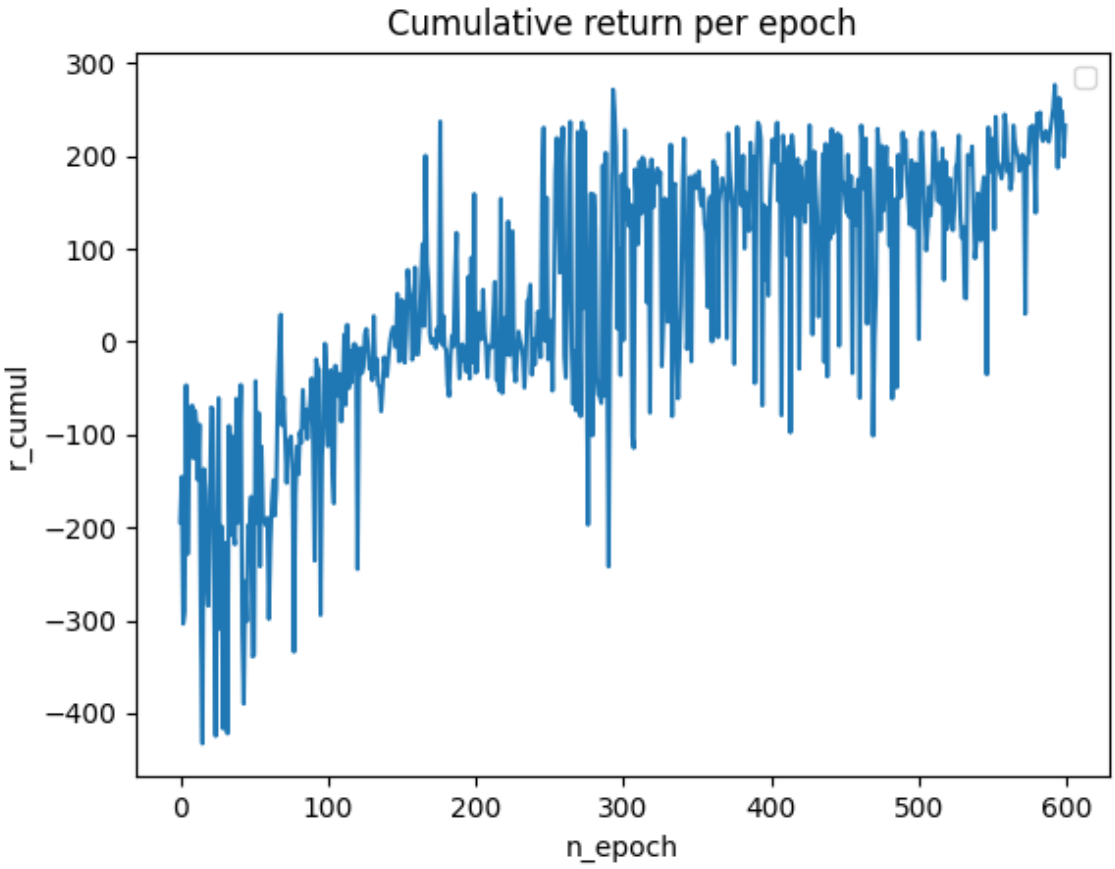

FIGURE 2 – somme des récompenses par époque

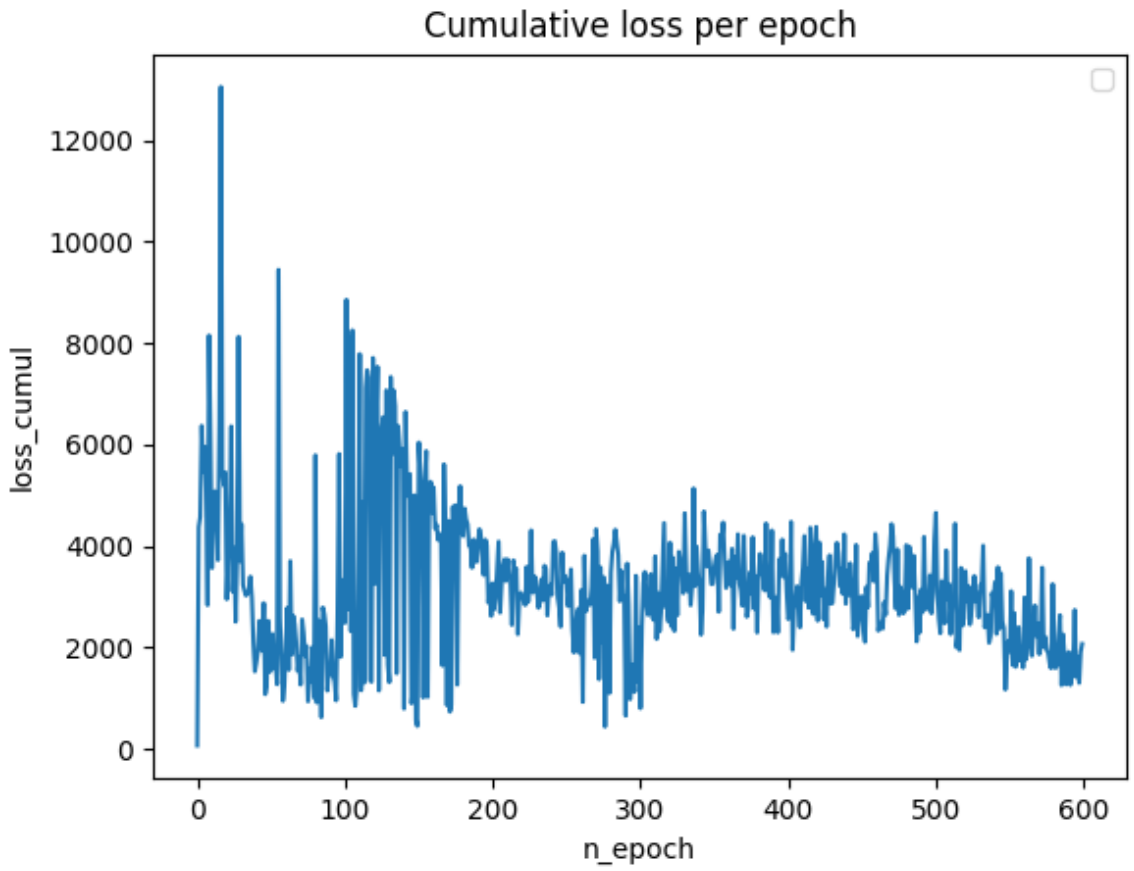

FIGURE 3 – perte cumulative du modèle par époque