Répondez dans un rapport PDF. Remettez une archive contenant le rapport PDF ainsi que le code Python associé aux différentes questions. Votre rapport doit indiquer clairement quel fichier Python doit être exécuté pour répliquer les résultats présentés dans votre rapport.
Ce devoir contient 4 questions pour un total de 12 points.
Remise : Mercredi 23 novembre à 17h00

Les librairies requises pour ce travail se trouvent dans un fichier `requirements.txt` fourni avec cet énoncé. Vous pouvez les installer avec la commande *pip install -r requirements.txt*. Vous devez aussi suivre les instructions ici pour installer la librairie *box2d*. Enfin, ce travail fait un usage exhaustif des environnements fournis par OpenAI Gym. Pour plus d'informations sur ces derniers, référez vous à leur documentation.

1. (1 point) Soit l'environnement *MountainCar-v0* de OpenAI Gym. Dans cet environnement, l'espace d'état est continu, les actions représentent l'accélération vers la gauche et la droite ainsi que l'absence d'accélération. La récompense est de -1 pour chaque pas de temps effectué, avec une fin d'épisode survenant soit quand 200 pas de temps se sont écoulés ou que la voiture a réussi à gravir la pente avec succès. On utilise alors généralement $\gamma = 1$ et on considère que l'environnement est résolu lorsqu'un agent obtient un retour cumulatif supérieur à -110 en moyenne sur un intervalle de 100 épisodes consécutifs.

   Lorsque l'on utilise des approches approximant les valeurs Q sur cet environnement, il n'est pas nécessaire d'ajouter une exploration forcée à notre algorithme si on initialise $Q(s, a) = 0$ pour tous les états et les actions. Pourquoi? À quel principe tiré des approches par bandits fait-on alors appel?

2. Étudiez l'impact des paramètres sur la stratégie SARSA(λ).

   Utilisez le fichier `sarsa.py` comme point de départ pour les questions suivantes.

   (a) (2 points) Implémentez l'algorithme SARSA(λ) avec caractéristiques binaires et approximation de fonction linéaire tel qu'il est présenté au chapitre 12.7 du livre de Sutton et Barto [1]. Votre implémentation doit inclure les éléments suivants :
   - Une représentation des caractéristiques tirée d'un carrelage sur l'espace joint des états et des actions, qui vous est fournie avec des valeurs par défaut adéquates.
   - Des traces obtenues par remplacement.
   - Aucune exploration forcée (pas de ϵ-greedy).

   (b) (1 point) Montrez que votre implémentation de SARSA(λ) fonctionne en l'appliquant à l'environnement MontainCar-v0 décrit précédemment. Utilisez un taux d'apprentissage $\alpha = 0.1$ et $\lambda = 0.9$. Avec ces valeur, votre algorithme devrait résoudre l'environnement en quelques secondes et moins de 500 épisodes.

   (c) (1 point) Analysez l'impact de la valeur du paramètre λ sur la stabilité de l'apprentissage et la qualité de la solution trouvée. Justifiez les différentes valeurs du paramètre λ investiguées. Rapportez la courbe de performance de votre implémentation pour chaque valeur et expliquez les résultats obtenus.

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

   Deep Q-Learning approxime typiquement la valeur $Q(s, a)$ de l'action a dans l'état s à partir d'une fonction paramétrée $q_\theta(s, a)$, où θ représente les poids d'un réseau de neurones. On cherche alors à minimiser la fonction de perte suivante :

$$\mathcal{L}(\theta) = \frac{1}{b} \sum_{t=1}^{b} \left(r_t + \gamma \max_{a' \in \mathcal{A}} q_\theta(s_{t+1}, a') - q_\theta(s_t, a_t) \right)^2$$

   correspondant à la moyenne empirique du *one-step lookahead target* calculée sur b tuples $(s_t, a_t, r_{t+1}, s_{t+1})$.

   (a) (1 point) Dans les environnements à actions discrètes, il est généralement préférable d'utiliser un modèle d'approximation $q_\theta(s) \in \mathbb{R}^{|\mathcal{A}|}$ permettant de prédire en sortie les valeurs $Q(s, a)$ pour toutes les actions a dans l'état s donné. Quel est l'avantage principal à procéder ainsi plutôt que de spécifier une seule action en entrée?

(b) (1 point) Une astuce pour faciliter l'apprentissage consiste à utiliser un réseau cible q_{θ^-} pour calculer le *one-step lookahead target*, résultant en la fonction de perte suivante :

$$\mathcal{L}(\theta) = \frac{1}{b}\sum_{t=1}^{b}\left(r_t + \gamma \max_{a' \in \mathcal{A}} q_{\theta^-}(s_{t+1}, a') - q_\theta(s_t, a_t)\right)^2.$$

Quel est l'impact d'un tel réseau cible sur la stabilité de l'entraînement? Expliquez votre réponse.

(c) (1 point) Une des heuristiques utilisées pour déterminer le réseau cible consiste à calculer une moyenne exponentielle mobile sur les paramètres:

$$\theta^- = (1-\tau)\theta^- + \tau\theta,$$

où la valeur du paramètre τ est proche de 0. Cette mise à jour est effectuée après chacune des mises à jours des poids θ.

Quel est le dilemme fondamental sur le choix d'une bonne valeur τ? Que se passe-t-il si $\tau = 0$ ou $\tau = 1$?

(d) (1 point) Au fur et à mesure d'un épisode, on entrepose les transitions $(s_t, a_t, r_{t+1}, s_{t+1})$ dans un buffer $\mathcal{D}$ avec une taille maximale fixée. Les poids θ sont mis à jour en pigeant une *minibatch* de manière i.i.d. depuis $\mathcal{D}$.

Quels sont les avantages d'utiliser un *replay buffer*?

(e) (1 point) Dans la régression en apprentissage supervisé, on suppose qu'il existe une distribution $\mathcal{D}$ sur des paires entrées-sorties (x, y) et l'on souhaite apprendre une approximation de fonction f_θ minimisant:

$$\mathcal{L}(\theta) = \mathbb{E}_{x,y\sim\mathcal{D}}\left[(f_\theta(x) - y)^2\right].$$

Cette fonction objectif semble très similaire à celle utilisée dans la mise à jour de Deep Q-Learning. En quoi les deux objectifs sont-ils différents? *Indice :* Quelles sont les différences entre la distribution $\mathcal{D}$ ici et le *replay buffer* $\mathcal{D}$ présenté plus haut?

4. Expérimentez avec Deep Q-Learning.

Utilisez le fichier `deep_q_learning.py` comme point de départ pour les questions suivantes.

(a) (1 point) Implémentez l'algorithme Deep Q-Learning. Votre implémentation doit inclure les éléments suivants :

- Un réseau cible; le code de mise à jour des poids cibles pour un certain τ vous est fourni.
- Une stratégie d'exploration ε-greedy pour la génération de trajectoires.
- Un *replay buffer* pour entreposer les trajectoires, duquel il est possible d'échantillonner une *minibatch*.

(b) (1 point) Montrez que votre implémentation de Deep Q-Learning fonctionne en l'appliquant à l'environnement *LunarLander-v2*. Utilisez les valeurs de paramètres suivantes :

- Fixez le paramètre de mise à jour des poids du réseau cible $\tau = 0.001$ pour commencer et ajustez au besoin.
- Pour la stratégie d'exploration ε-greedy, faites décroître l'exploration au fil des épisodes avec la règle suivante : posez $\varepsilon_0 = 1$ et utilisez $\varepsilon_n = 0.99^{n-1} \cdot \varepsilon_{n-1}$ à l'épisode n.
- Fixez la taille du *replay buffer* à 10^5 pour commencer et ajustez au besoin.
- Fixez la taille de *minibatch* à 64 avec un taux d'apprentissage de 10^{-4}. Ces paramètres sont déjà présents par défaut dans le code fourni.
- Actualisez les récompenses avec un taux $\gamma = 0.99$.
- Effectuez un entraînement du réseau de neurones à chaque N pas dans l'environnement. Pour commencer, utilisez $N = 4$ et ajustez au besoin.

Assurez-vous de bien gérer les états terminaux pour les *one step lookahead targets*. Si l'état s_{t+1} d'une transition est terminal, la cible pour la mise à jour devrait seulement être r_t.

Rapportez sous forme de figures l'évolution de la somme des récompenses et de la fonction de perte de votre modèle en fonction du nombre de pas effectués dans l'environnement. Discutez des différents défis d'implémentation auxquels vous avez faits face.

Note : Pour cet environnement, une trajectoire est considérée comme un succès si la somme des récompenses est de 200 ou plus. Avec les hyperparamètres suggérés, vous devriez atteindre au moins 200 la plupart du temps après vous être entraînés sur environ 600 trajectoires. Ce processus d'entraînement devrait prendre entre 5 et 10 minutes.

References

[1] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.