IAR

Rapport

Tuning TD3 for LunarLanderContinuous-v2

Réalisé par :
Yanis DJERMOUNI (21304195)
Malo THOMASSON (28710401)

Encadrés par :
Olivier Sigaud

22 Avril 2024

Table des matières

1 Introduction

Le *tuning* désigne l'ensemble des méthodes permettant d'optimiser les hyper-paramètres d'un modèle.

TD3 (ou *Twin Delayed DDPG*) est un algorithme d'apprentissage par renforcement profond, *off-policy*, qui vise à résoudre le problème du biais de sur-estimation présent dans DDPG.

LunarLanderContinuous-v2 est une tâche où un robot doit alunir. Les actions dans cette tâche sont continues (d'où le terme *Continuous*).
Le GitHub est à l'URL suivante : https://github.com/ElDjeee/Micro-Projet-IAR.

2 Objectifs

L'objectif de ce micro-projet est d'évaluer l'algorithme TD3 sur la tâche *LunarLanderContinuous-v2* et de comparer notre implémentation à celle d'autres librairies. Nous prêterons une attention particulière aux hyper-paramètres propres à chaque implémentation, tout en veillant à éviter tout biais lié aux *seeds* aléatoires (en configurant le hasard pour garantir une évaluation juste).

Dans un premier temps, nous expliquerons le fonctionnement de l'algorithme TD3, suivi par la présentation des résultats obtenus à partir de notre implémentation. Ces résultats seront ensuite comparés à ceux des autres librairies. Enfin, nous étudierons l'influence des hyper-paramètres sur la performance lors des différents tests.

3 Explication de TD3

Le *Twin Delayed Deep Deterministic Policy Gradient* (TD3) est un algorithme d'apprentissage par renforcement qui améliore la stabilité et la performance de l'algorithme *Deep Deterministic Policy Gradient* (DDPG)[2]. TD3 est conçu pour résoudre des problèmes à espace d'actions continu et repose sur des politiques déterministes, où la fonction d'action est directement apprise à partir des états observés.

3.1 Problème de sur-estimation des Q-valeurs

Dans DDPG, un seul estimateur de Q-valeur est utilisé pour évaluer les actions. Cela peut entraîner une sur-estimation des valeurs Q lorsque des approximations imparfaites sont utilisées dans les réseaux de neurones. Ce phénomène dégrade la qualité de l'algorithme, notamment en cas de politiques complexes.

Pour résoudre ce problème, TD3 introduit deux estimateurs de Q-valeurs distincts (double Q-learning) et utilise la plus petite des deux pour calculer la cible de mise à jour des Q-valeurs. Cela réduit la sur-estimation systématique en garantissant que la mise à jour de la fonction de valeur Q soit plus conservatrice.

3.2 Fonctionnement de TD3

L'algorithme TD3 suit plusieurs étapes clés au cours de chaque itération :

1. **Exploration et collecte de données :** Un agent exécute sa politique dans l'environnement en ajoutant du bruit pour favoriser l'exploration des actions. Les transitions (s, a, r, s') sont stockées dans une mémoire de répétition.
2. **Mise à jour des Q-valeurs :** À chaque pas de temps, les deux réseaux critiques sont mis à jour en minimisant l'erreur quadratique entre les Q-valeurs prédites et les cibles Q calculées en prenant le minimum des Q-valeurs des deux réseaux critiques.
3. **Mise à jour différée de la politique :** La politique de l'acteur est mise à jour moins fréquemment, en maximisant les valeurs Q prédites par l'un des deux réseaux critiques (par exemple, Critique 1).
4. **Mise à jour douce des cibles :** Les paramètres des réseaux critiques et de la politique sont copiés vers les réseaux cibles à l'aide d'une mise à jour "soft", où une petite partie des nouveaux paramètres remplace ceux des anciens réseaux cibles.

3.3 Pseudocode de TD3

Algorithm 1 Pseudocode de TD3

Initialiser les paramètres des réseaux critiques Q_{θ_1} et Q_{θ_2} et de l'acteur μ_ϕ, ainsi que leurs cibles $Q_{\theta'_1}$, $Q_{\theta'_2}$ et $\mu_{\phi'}$
for chaque épisode **do**
  for chaque étape de temps t **do**
    Sélectionner une action avec du bruit $a_t = \mu_\phi(s_t) + \epsilon$
    Observer la récompense r_t et le nouvel état s_{t+1}
    Stocker la transition (s_t, a_t, r_t, s_{t+1}) dans la mémoire de répétition
    if toutes les N étapes **then**
      Mettre à jour les critiques :
      $y_t = r_t + \gamma \min_{i=1,2} Q_{\theta'_i}(s_{t+1}, \mu_{\phi'}(s_{t+1}))$
      Mettre à jour θ_1 et θ_2 en minimisant l'erreur $(Q_{\theta_i}(s_t, a_t) - y_t)^2$
      if étape de mise à jour différée **then**
        Mettre à jour l'acteur : $\phi \leftarrow \arg\max_\phi Q_{\theta_1}(s, \mu_\phi(s))$
        Mettre à jour les cibles avec une mise à jour "$soft$" : $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
      end if
    end if
  end for
end for

En résumé, TD3 améliore DDPG en réduisant la sur-estimation des valeurs Q, en lissant la politique cible, et en retardant les mises à jour de la politique. Ces améliorations permettent à l'algorithme d'apprendre de manière plus stable et efficace dans des environnements complexes et continus.

4 Implementation et comparaison avec des environnements existants

Nous avons implementé TD3 avec BBRL. Pour voir si l'implementation est correcte, nous la comparerons à des bibliothèques existantes telles que Stable-Baselines3 et CleanRL.

Afin de nous rapprocher de la performance des autres libraires, nous avons analysé leurs codes sources afin de comprendre les nuances avec notre version. Nous avons également comparer notre code, avec le pseudo-code de l'article proprosant, initialement, TD3[1].

4.1 Comparaison des résultats

Nous avons pour cela effectué de nombreuses statistiques, qui présentent :

* La somme de la loss des 2 critiques
* La loss des politiques
* La reward

Nous effectuons une statistique sur 10 seeds des parametres ci-dessus pour Stable-Baselines3, on obtient les graphiques suivant :

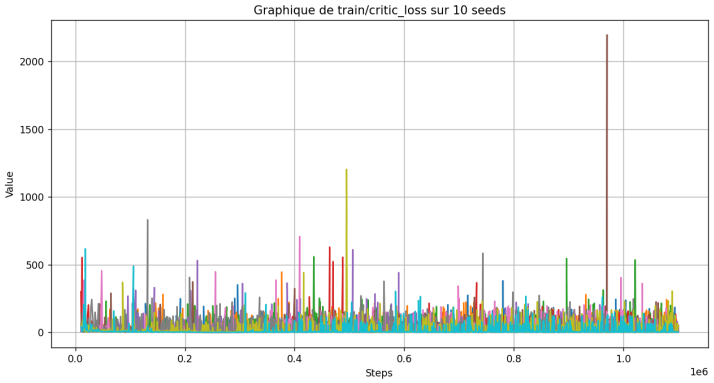

FIGURE 1 – La somme de la loss des 2 critique pour Stable-Baselines3

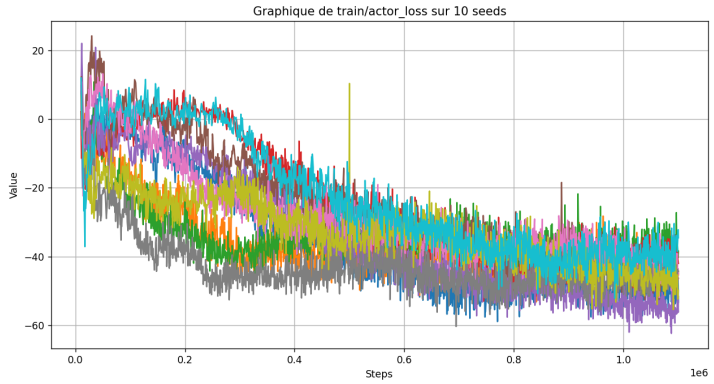

FIGURE 2 – La loss des politiques pour Stable-Baselines3

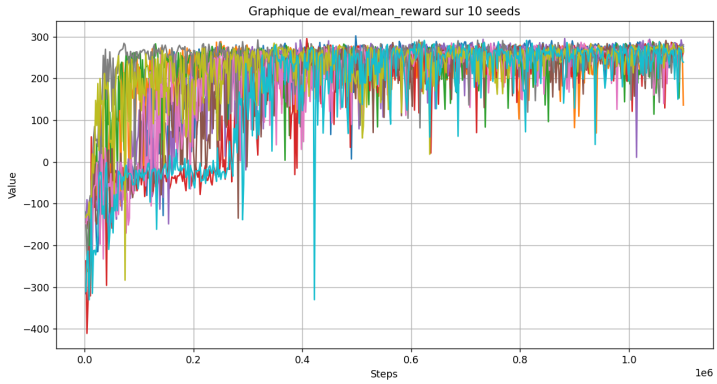

FIGURE 3 – La reward pour Stable-Baselines3

Nous effectuons la meme chose pour BBRL avec les parametres :

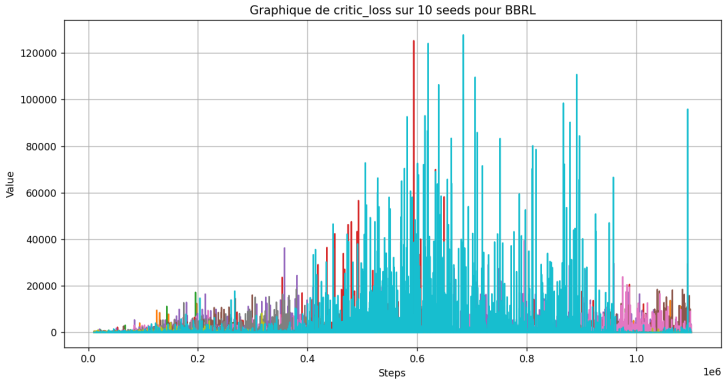

FIGURE 4 – La somme de la loss des 2 critique pour BBRL

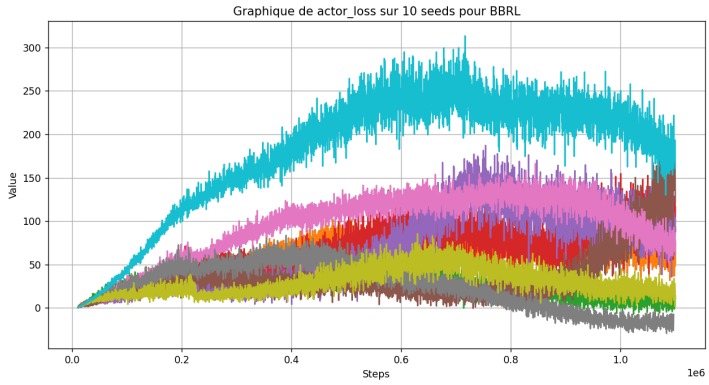

FIGURE 5 – La loss des politiques pour BBRL

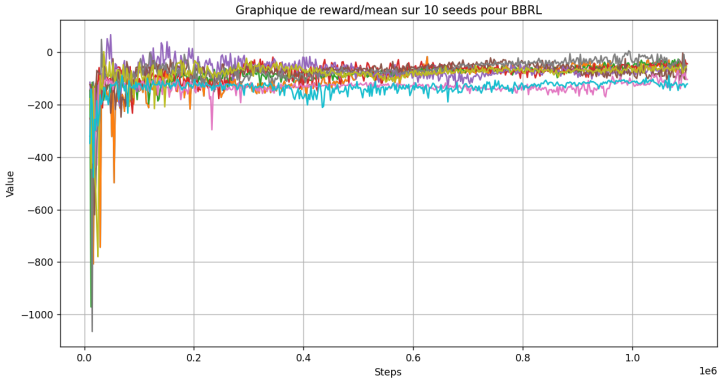

FIGURE 6 – La reward pour BBRL

Et enfin pour CleanRL :

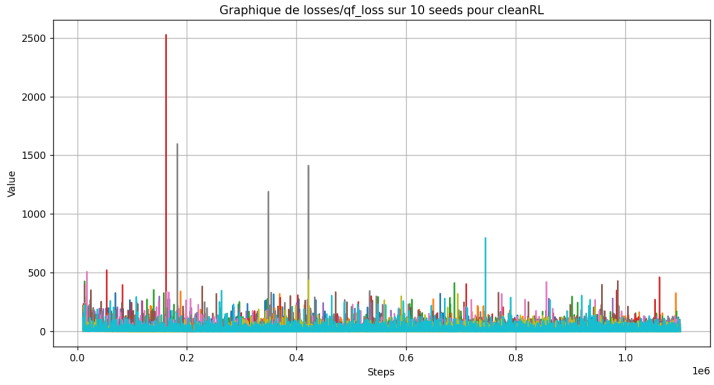

FIGURE 7 – La somme de la loss des 2 critique pour CleanRL

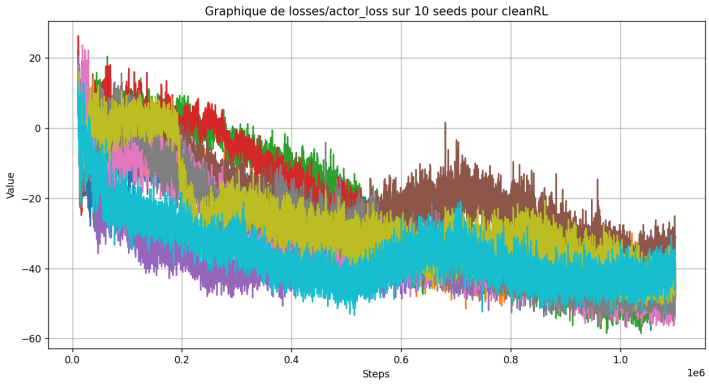

FIGURE 8 – La loss des politiques pour CleanRL

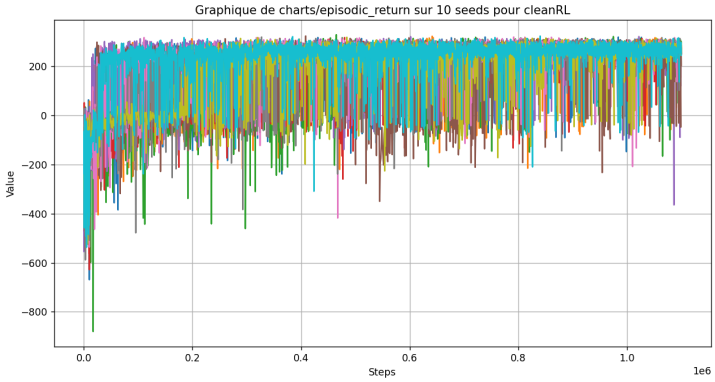

FIGURE 9 – La reward pour CleanRL

Nous constatons, grâce au WelchTTest, que les récompenses obtenues avec CleanRL et SB3 sont comparables, tandis que celles de BBRL se situent à un niveau inférieur. Cela est également visible dans le graphique ci-dessous :

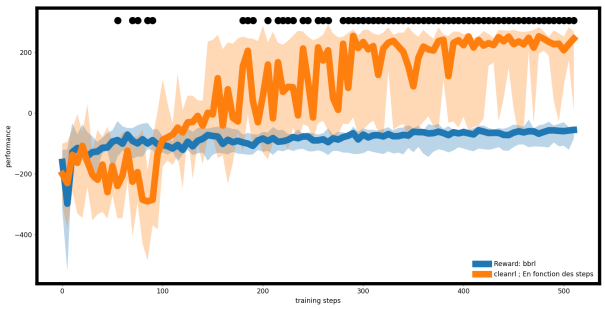

FIGURE 10 – WelchTest de la reward entre BBRL et CleanRL, en fonction des steps

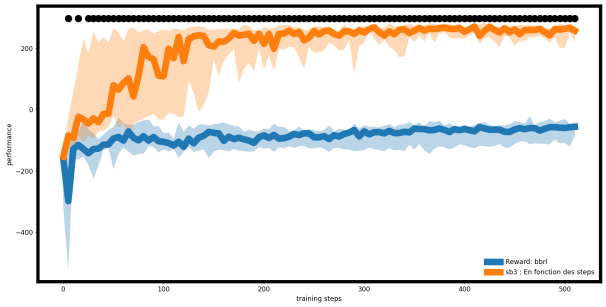

FIGURE 11 – WelchTest de la reward entre BBRL et SB3, en fonction des steps

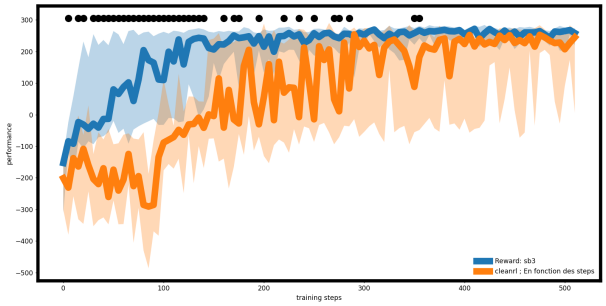

FIGURE 12 – WelchTest de la reward entre SB3 et CleanRL, en fonction des steps

Voici les 3 graphiques representant la comparaison des actor des environnements 2 à 2 :

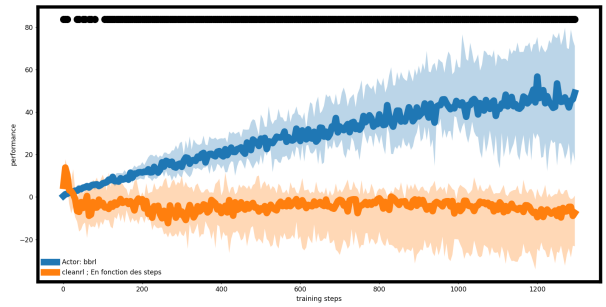

FIGURE 13 – WelchTest des Actor entre BBRL et CleanRL, en fonction des steps

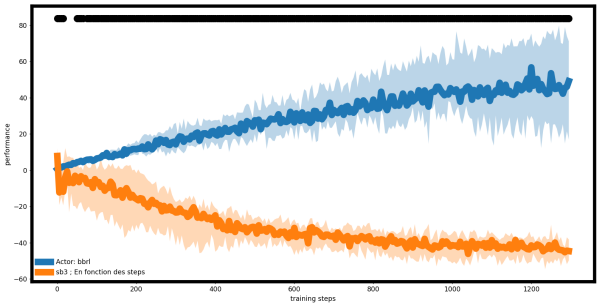

FIGURE 14 – WelchTest des Actor entre BBRL et SB3, en fonction des steps

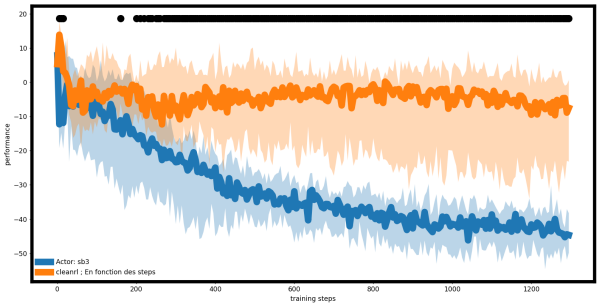

FIGURE 15 – WelchTest des Actor entre SB3 et CleanRL, en fonction des steps

Nous constatons que BBRL est clairement décalé par rapport au reste, notamment sur la reward, et il a été jugé non pertinent d'ajouter les critics.

Nous remarquons que l'implémentation présente certaines lacunes, car elle a du mal à converger vers une reward de 200.

5 Conclusion

Ce projet a permis d'explorer l'algorithme TD3 dans le cadre de la tâche *LunarLanderContinuous-v2*. À travers une implémentation et des comparaisons avec des librairies établies telles que Stable-Baselines3 et CleanRL, nous avons pu analyser les performances de TD3 en termes de loss et de récompense.

Les hyper-paramètres jouent un rôle crucial dans la performance des algorithmes. Dans notre étude, nous avons veillé à les ajuster de manière à ce qu'ils se rapprochent autant que possible de ceux décrits dans l'article Fujimoto et al. (2018) [1].

Références

[1] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *Proceedings of the 35th International Conference on Machine Learning*, 80 :1587–1596, 2018.

[2] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. pages 387–395, 2014.