TME7 RLD

Continuous Actions

Brient, Hein - Décembre 2021

1 Objectifs

Ce TME a pour objectif d'experimenter l'approche DDPG pour des environnements à actions continues. Trois environnements différents sont disponibles pour tester cet algorithme :

MountainCarContinuous-v0, LunarLanderContinuous-v2 et Pendulum-v0.

2 Algorithme

Deep Deterministic Policy Gradient (DDPG) est un algorithme qui apprend à la fois une Q-fonction et une politique. Il se base sur les équations de Bellman pour apprendre cette Q-fonction, puis l'utilise ensuite pour déterminer une politique performante. Comme pour l'algorithme Q-learning, le but est de trouver les action-value optimales $Q^*(s,a)$, pour ensuite pouvoir, dans chaque état, déterminer l'action optimale à choisir $a^*(s)$ en utilisant la relation :

$$a^*(s) = \arg\max_a Q^*(s,a)$$

L'algorithme DDPG va chercher, comme pour l'algorithme Deep Q-Network, à apprendre un approximateur de la Q-fonction optimale $Q^*(s,a)$. Cependant, DDPG est conçu pour pouvoir appréhender les environnements à actions continues grâce à la façon dont est calculée $\max_a Q^*(s,a)$, une différence majeure par rapport à l'algorithme deep-Q learning. Les deux algorithmes sont complémentaires et leur utilisation repose sur l'environnement à étudier.

Lorsque il y a un nombre discret d'actions possible à un certain état, le calcul du maximum ne pose pas de problèmes, puisque le réseau de neurones peut predire dans un même temps les Q-values pour chacune de ces potentielles actions. On peut alors les comparer facilement pour sélectionner l'action optimale. Lorsque l'espace des actions est continu, cette façon de faire devient archaïque puisque il est impossible d'évaluer à chaque itération de l'agent l'ensemble ou un nombre suffisant d'actions possibles pour déterminer une valeur suffisamment correcte de $\max_a Q^*(s,a)$.

En supposant la fonction $Q^*(s,a)$ différentiable par rapport à l'argument d'action, il est possible de mettre en place une règle d'apprentissage efficace basée sur le gradient, pour une certaine politique $\mu(s)$. Le but est alors, plutôt que de calculer le terme $\max_a Q^*(s,a)$ de façon coûteuse, de l'approximer par $\max_a Q(s,a) \approx Q(s,\mu(s))$.

L'algorithme DDPG approxime la Q-function de la même manière que l'algorithme DQN précédemment étudié (cf TME3 et TME4), et peut aussi se munir de certaines améliorations de ce dernier comme un target network et un replay buffer, dont les principes ont été expliqués dans le rapport du TME4. Une différence toutefois est que l'algorithme DDPG met à jour le target network par "polyak averaging" à chaque mise à jour du current network et non pas de façon directe à une certaine fréquence fixée, ce qui peut faciliter la convergence :

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$$

Avec ρ un hyperparametre entre 0 et 1.

Selon le même principe, DDPG utilise aussi un target policy network pour calculer une action qui approxime le maximum de $Q_{\phi_{\text{targ}}}$. Ce target network est aussi mis à jour par polyak averaging.

Pour résumer, le calcul des Q-values dans l'algorithme DDPG s'éffectue en minimisant la fonction de perte suivante par descente de gradient.

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\text{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right],$$

Où $\mu_{\theta_{\text{targ}}}$ est la politique cible et $\mathcal{D}$ est un batch de transitions issues du replay buffer.

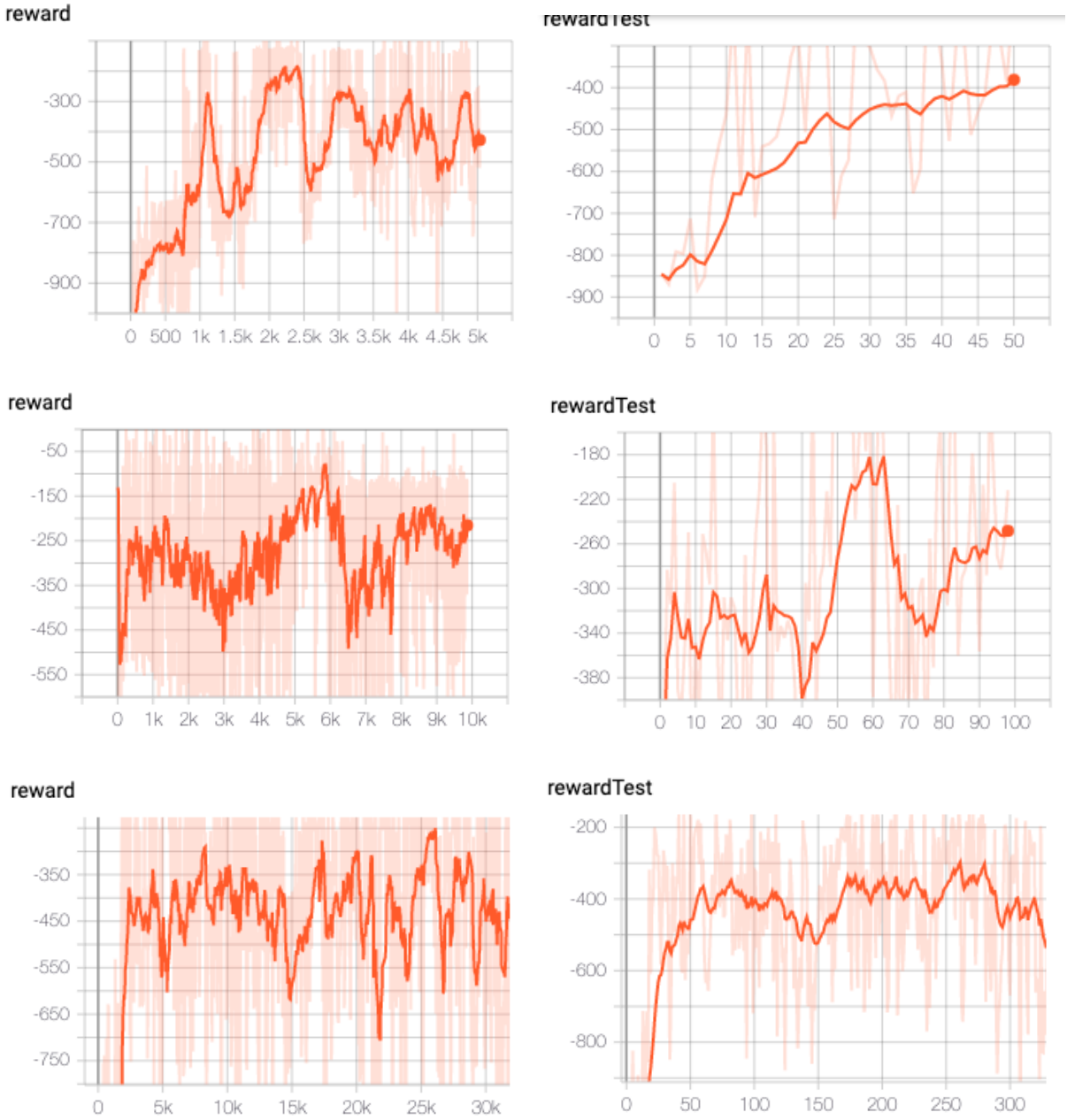
Dans l'algorithme DDPG, on veut aussi apprendre une politique déterministe $\mu_\theta(s)$ qui nous donne l'action maximizant $Q_\phi(s,a)$. Puisque l'espace des actions est continu et que par conséquent on considère la Q-fonction differentiable par rapport à l'action, il est possible de réaliser une montée de gradient pour résoudre :

$$\max_\theta \underset{s\sim\mathcal{D}}{\text{E}}\left[Q_\phi(s, \mu_\theta(s))\right].$$

Par ailleurs, puisque la politique entraînée par DDPG est déterministe, l'agent en l'état a du mal à explorer suffisamment correctement pour trouver des signaux d'apprentissage utiles. Pour améliorer l'exploration, on peut ajouter du bruit aux actions pendant l'apprentissage, mais pas lors de la phase de test. En plus de ça, on peut encore améliorer l'exploration en forcant l'agent à choisir des actions échantillonées aléatoirement selon une distribution uniforme parmi les actions valides, pour un certain nombre d'itérations au début de l'expérience. Une fois fait l'agent retourne à la méthode d'exploration de DDPG.

3 Implémentation et résultats

Nous avons implémenté l'algorithme DDPG puis l'avons testé sur les trois environnements proposés en utilisant, lorsque disponibles, les hyperparamètres présentés dans le sujet. Les graphiques suivants représentent respectivement les résultats sur les environnements Pendulum, LunarLander et Mountaincar.



Pour l'environnement pendulum, on peut voir que les récompenses semblent converger vers les valeurs attendues, environ -400, au bout de 5000 épisodes. Toutefois, la convergence de l'agent, bien que plutôt stable, est beaucoup plus lente que dans le graphique présenté dans l'énoncé.

Ensuite, malgrè de nombreux essais avec des jeux d'hyperparamètres divers et variés, nous ne sommes pas parvenus à obtenir de bonnes performances dans l'environnement LunarLander en un nombre d'itérations relativement limité. En effet, malgré les oscillations, on peut noter une courbe de tendance des récompenses test qui augmente légèrement, mais pas suffisamment vis-à-vis du nombre d'épisodes réalisés. Par conséquent, une analyse plus poussée des hyperparamètres comme nous avons réalisé dans le TME4 pourrait être très bénéfique pour les performances de cet agent et pourrait aider à mieux cerner

les impacts de chacun.

Enfin, pour l'environnement MountainCar, les performances ne sont pas satisfaisantes non plus. On peut voir que l'agent stagne à des récompenses d'environ -400 alors que l'on peut considérer que l'environnement est réussi avec des récompenses stables d'environ -110. En analysant la courbe, on voit que la convergence s'opère relativement vite mais que l'agent stagne pendant un très grand nombre d'épisodes, ce qui est probablement dû aux hyperparamètres choisis dans le fichier config. A nouveau, malgré plusieurs jeux différents, les performances n'étaient pas plus convaincante et une étude plus poussée semble requise.