Deep Deterministic Policy Gradient: Applications à un Environnement Personnalisé

Rémi Veillard
2409445
Polytechnique Montréal

Abstract

Ce papier présente une explication détaillée de l'algorithme DDPG (Deep Deterministic Policy Gradient), et ses avantages. Suivra une présentation du projet de recoder DDPG à partir de zéro et de l'appliquer sur un environnement personnalisé simple.

1 Introduction

Le DDPG est un algorithme de Deep Reinforcement Learning qui s'applique à un environnement à valeurs continues. Il est très utile pour contrôler des actions dans un environnement physique, quand les actions et les états ont des valeurs non dénombrables. Pour cela DDPG se base sur l'architechture DPG (Deterministic Policy Gradient) en y apportant des améliorations venant des DQNs (Deep Q-Learning).

1.1 Q-learning et DQN

Le Q-learning est une méthode classique basée sur l'approximation de la fonction de valeur d'action $Q(s, a)$:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$$

pour choisir des actions et apprendre d'elles mêmes sans avoir besoin de connaître le modèle de l'environnement.

DQN introduit plusieurs amélioration pour rendre la méthode plus efficace. Un réseaux de neurones est utilisé pour approximer Q et une la stabilisation est grandement améliorée via 2 pratiques qui seront expliquées plus tard:

- Un buffer de relecture (Replay Buffer)
- Un réseau cible figé (Target Network)

DQN est toutefois limité aux actions discrètes ce qui restreint beaucoup ses utilisations.

1.2 Deterministic Policy Gradient (DPG)

Plutôt que d'apprendre Q pour toutes les actions possibles (comme DQN), l'idée de DPG est d'apprendre directement une politique déterministe qui donne une action continue $\mu(s)$ en maximisant :

$$\nabla_\theta J(\mu_\theta) = E_{s \sim \rho^\mu} \left[\nabla_a Q(s, a)|_{a=\mu(s)} \nabla_\theta \mu_\theta(s) \right]$$

Cela rend DPG adapté pour les espaces d'actions continues de grande dimension.

2 Deep Deterministic Policy Gradient (DDPG)

DDPG combine et améliore les idées de DPG et de DQN pour former une méthode efficace dans un milieu continu. DDPG repose sur un système acteur/critique : un acteur $\mu(s|\theta^\mu)$, qui prédit directement l'action optimale pour un état donné, et un critique $Q(s, a\theta^Q)$, qui évalue la qualité d'une paire état-action. Chacun est réalisé à partir d'un réseau de neurones profond.

Une autre composante importante est le réseau cible (Target Network) composé d'un acteur cible et d'une critique cible, chacun est un réseau de neurones profond. Les réseaux cibles sont là pour stabiliser l'apprentissage en fournissant des cibles "douces" et non chaotiques sur lesquelles l'entrainement pourra s'appuyer.

De plus, un bruit est ajouté pour provoquer l'exploration: le bruit de Ornstein-Uhlenbeck. Le bruit est ici nécessaire pour l'exploration sans quoi l'acteur bouclera sur certaines valeurs d'actions.

Enfin, le Relpay Buffer est ajouté pour garder en mémoire les transitions d'expériences que l'agent accumule au fil du temps. Lorsque l'agent apprend directement à partir des expériences qu'il vient de vivre, les transitions sont fortement corrélées. Cela peut entraîner des mises à jour instables et biaisées des réseaux de neurones.

L'entraînement du DDPG consiste à aligner les réseaux acteur et critique avec leurs cibles respectives, tout en mettant à jour les réseaux cibles de manière lente et stable. Le pseudo-code est détailler dans l'aglorithme 1 ci-dessous.

3 Différences entre DQN, DPG et DDPG

DQN est limité aux espaces d'action discrets et s'appuie sur des valeurs Q, tandis que DPG et DDPG sont adaptés aux espaces d'action continus et se basent sur l'optimisation d'une politique déterministe. DDPG, améliore DPG en utilisant des réseaux profonds et des mécanismes supplémentaires, tels que le buffer de relecture et les réseaux cibles empruntés au DQN. Cela a pour effet de stabiliser et améliorer l'efficacité de l'apprentissage dans des environnements complexes.

Algorithm 1 DDPG Algorithm

Randomly initialize critic network $Q(s, a; \theta_Q)$ and actor network $\pi(s; \theta_\pi)$ with weights θ_Q and θ_π.
Initialize target networks Q' and π' with weights $\theta'_Q \leftarrow \theta_Q$ and $\theta'_\pi \leftarrow \theta_\pi$.
Initialize replay buffer R.

for episode = 1 to M **do**
  Initialize a random process N for action exploration.
  Receive initial observation state s_1.
  for t = 1 to T **do**
    Select action $a_t = \pi(s_t; \theta_\pi) + N_t$ according to the current policy and exploration noise.
    Execute action a_t, observe reward r_t, and observe new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1}) in R.
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R.
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \pi'(s_{i+1}; \theta'_\pi); \theta'_Q)$.
    Update critic by minimizing the loss:
      $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i; \theta_Q))^2$.
    Update actor policy using the sampled policy gradient:
      $\nabla_{\theta_\pi} J = \frac{1}{N} \sum_i \nabla_a Q(s, a; \theta_Q)|_{a=\pi(s_i;\theta_\pi)} \nabla_{\theta_\pi} \pi(s_i; \theta_\pi)$.
    Update the target networks:
      $\theta'_Q \leftarrow \tau\theta_Q + (1 - \tau)\theta'_Q$.
      $\theta'_\pi \leftarrow \tau\theta_\pi + (1 - \tau)\theta'_\pi$.
  end for
end for.

4 Projet : Recodage de DDPG

Le projet vise à recoder DDPG en suivant l'algorithme d'origine à partir de 0:

- Implémentation manuelle de l'acteur, du critique et des réseaux cibles.
- Ajout du bruit OU pour l'exploration.
- Gestion du Replay Buffer.
- Mise à jour des critiques et acteurs.
- Mise à jour douce des cibles.

Cela m'a donné une compréhension approfondie de chacun des composant et m'a permis d'expérimenter tout au long de la construction de l'algorithme.

5 Application : Environnement Personnalisé

Une seconde partie du projet a été de concevoir un environnement simple :

- État : une variable continue.
- Action : un réel borné entre $[-1, 1]$.
- Récompense : fonction qui dépend directement de l'action et facilement modifiable.

On peut choisir une récompense maximum pour une certaine valeur d'action (par exmeple pour 1 ou 0.5) et on peut regarder l'algorithme apprendre à maximiser sa récompense. Le but de cet environnement est de tester le bon fonctionnement de l'algorithme DDPG avant de l'appliquer à des problèmes plus complexes comme `Pendulum-v1`.

6 Hyperparamètres

Afin de mieux comprendre l'impact des hyperparamètres sur les performances de DDPG, plusieurs expériences ont été réalisées en faisant varier systématiquement chacun d'entre eux tout en maintenant les autres constants. Voici les principales observations :

6.1 Taux d'apprentissage (α)

Le taux d'apprentissage des réseaux acteur et critique joue un rôle dans la stabilité de l'entrainement. Lors de nos expériences, un taux d'apprentissage trop élevé (par exemple $\alpha = 0.1$) entraînait des oscillations fortes de la politique et une instabilité générale de l'apprentissage, avec parfois des chutes brutales de la récompense. À l'inverse, un taux trop faible ($\alpha = 10^{-4}$) ralentissait considérablement la convergence, rendant l'entraînement inefficace sur un nombre d'épisodes raisonnable. Un compromis autour de ($\alpha = 5 \times 10^{-3}$) s'est révélé offrir un bon équilibre entre vitesse de convergence et stabilité.

6.2 Taux de mise à jour des réseaux cibles (τ)

Le paramètre τ contrôle à quelle vitesse les réseaux cibles sont mis à jour. Un τ élevé (par exemple $\tau = 10^{-1}$) causait des comportements instables, car les cibles changeaient trop rapidement, annulant l'effet de stabilisation recherché. À l'inverse, un τ très petit (par exemple $\tau = 10^{-4}$) ralentissait la propagation de l'apprentissage vers les réseaux cibles, ce qui rendait l'agent très lent à s'adapter. Le meilleur comportement a été observé pour des valeurs typiques de environ $\tau = 5 \times 10^{-3}$, assurant un transfert progressif d'information tout en stabilisant les mises à jour.

6.3 Facteur d'actualisation (γ)

Le facteur d'actualisation γ influence le poids attribué aux récompenses futures. Des valeurs faibles devraient entrainer des comportements myopes cherchant à maximiser des récompenses immédiates sans se soucier des conséquences à long terme. L'environnement personnalisé qui donne une récompense dépendant directement de l'action effectuée devrait alors être appris très vite avec un $\gamma = 0$. Or ce n'est pas le cas, d'après les expériences il faut toujours un γ valant minimum 0.7 environ pour avoir un réel apprentissage. Une grande valeur ($\gamma = 0.99$) est censé favoriser une stratégie plus long terme, au prix d'une exploration initiale plus difficile. Globalement, pour des grandes valeurs de γ, cette intuition est cohérente avec l'expérience et $\gamma = 0.99$ s'est imposé comme un choix robuste.

6.4 Taille du buffer de relecture

La taille du buffer joue sur la diversité des transitions utilisées pour l'apprentissage. Une taille trop petite entraîne un surapprentissage sur des trajectoires récentes et très corrélées. À l'inverse, un buffer trop grands ralentit l'accès aux transitions récentes et nécessite beaucoup de mémoire.

Dans les tests, un buffer de 1000 est adapté et a offert un bon compromis, permettant un échantillonnage diversifié tout en gardant une certaine fraîcheur des expériences.

6.5 Taille du batch

La taille du batch utilisé pour les mises à jour du réseau affecte la variance du gradient estimé. Des petits batchs (¡32) donnent lieu à des mises à jour très bruyantes et instables qui rendent l'apprentissage innefficace. Des batchs très grands (¿256) augmentent fortemment le temps de calcul nécessaire et sont donc peu interressants. Les meilleurs résultats ont été obtenus avec des batchs de taille 64 ou 128, permettant un bon compromis entre précision et rapidité d'apprentissage.

6.6 Bruit d exploration

Le bruit ajouté aux actions, issu d'un processus d'Ornstein-Uhlenbeck, est essentiel pour favoriser l'exploration dans un environnement continu. Une faible amplitude de bruit (je ne l'avais pas implémenter à l'algorithme au départ) conduit à une exploitation dès le départ, où l'agent reste piégé dans des optima locaux. Il est donc absolument nécessaire pour explorer l'espace. À l'inverse, un bruit trop fort ($\sigma > 0.6$) provoque une exploration excessive est ralenti la stabilisation de la politique.

En manipulant, j'ai pu comprendre l'interêt de ce bruit Ornstein-Uhlenbeck car celui-ci est corrélé dans le temps. Cela évite que l'exploration soit complètement aléatoire et instable à chaque pas. En effet, pour simuler le mouvement physique d'un pendule, une certaine inertie doit être prise en compte et donc il faut plusieurs actions dans la même direction pour avoir un effet visible. D'où la nécessité de la correlation temporelle.

7 Conclusion

Ce travail m'a permis de prendre en main des réelles applications de reinforcement learning, ce que je n'avais jamais fait. Re-coder depuis le début un tel algorithme me permet d'en comprendre finement le fonctionnement et d'avoir pu manipulé plus facilement les hyperparamètres. Au final, mon implémentation personnelle n'a de bons résultats que sur mon environnement simple créé pour l'occasion (qui est vraiment très simple) et des résultats poussifs sur `Pendulum-v1`. Il apprend quand même, mais ne reussi pas à correctement tenir droit. Enfin, avec plus de temps, j'aurai continuer à adapter mon algorithme à d'autres environnements plus complexes en cherchant à comprendre comment amméliorer ses performances.

References

Lillicrap et al., Continuous Control with Deep Reinforcement Learning, arXiv:1509.02971 (2015).

Watkins, C.J.C.H. and Dayan, P. (1992). "Q-learning." Machine Learning, 8(3-4), 279–292. DOI: 10.1007/BF00992698

Silver, Lever, Heess, Degris, Wierstra, Riedmiller, Deterministic Policy Gradient Algorithms, ICML (2014).

Mnih et al., Playing Atari with Deep Reinforcement Learning, arXiv:1312.5602 (2013).