

REINFORCEMENT LEARNING & ADVANCED DEEP

M2 DAC

TME 7. Continuous Actions

Ce TME a pour objectif d'expérimenter l'approche DDPG pour environnements à actions continues.

DDPG

Implémenter l'algorithme DDPG suivant:

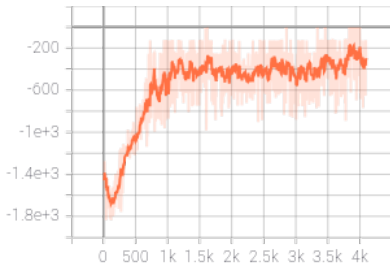
Algorithm 1 Deep Deterministic Policy Gradient

- 1: Input: initial policy parameters θ , Q-function parameters ϕ , empty replay buffer \mathcal{D}
 - 2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
 - 3: **repeat**
 - 4: Observe state s and select action $a = \text{clip}(\mu_{\theta}(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$, where $\epsilon \sim \mathcal{N}$ ← Ajout d'un bruit gaussien pour exploration + clip pour rester dans des valeurs admissibles
 - 5: Execute a in the environment
 - 6: Observe next state s' , reward r , and done signal d to indicate whether s' is terminal
 - 7: Store (s, a, r, s', d) in replay buffer \mathcal{D}
 - 8: If s' is terminal, reset environment state.
 - 9: **if** it's time to update **then**
 - 10: **for** however many updates **do**
 - 11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from \mathcal{D}
 - 12: Compute targets
$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$
 ← Utilisation de réseaux cible (à la fois pour Q et pour μ) pour le calcul de la cible de Q
 - 13: Update Q-function by one step of gradient descent using
$$\nabla_{\phi} \frac{1}{|B|} \sum_{(s, a, r, s', d) \in B} (Q_{\phi}(s, a) - y(r, s', d))^2$$
 - 14: Update policy by one step of gradient ascent using
$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi}(s, \mu_{\theta}(s))$$
 ← Gradient similaire à DPG, selon les transitions du batch
 - 15: Update target networks with
$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1 - \rho) \phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta \end{aligned}$$
 ← Mise à jour "soft" des paramètres des réseaux cible
 - 16: **end for**
 - 17: **end if**
 - 18: **until** convergence
-

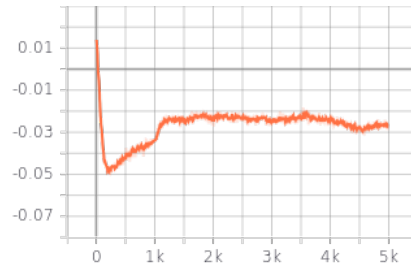
Appliquer l'algorithme aux 3 problèmes suivants:

- MountainCarContinuous-v0
- LunarLanderContinuous-v2
- Pendulum-v0

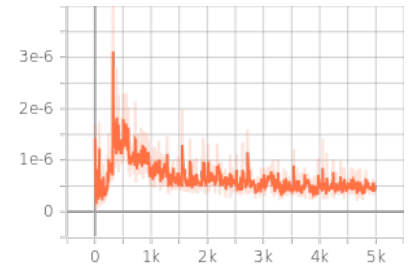
À titre indicatif, voici ce qu'on peut obtenir sur Pendulum avec DDPG utilisant un processus d'exploration de Ornstein-Uhlenbeck avec $\sigma = 0.2$ (voir core.py pour la méthode d'échantillonnage du bruit, à réinitialiser à la fin de chaque trajectoire), effectuant une optimisation de 10 étapes tous les 1000 événements, utilisant deux target networks V et π (mis à jour de manière "soft" à chaque optimisation selon des paramètres $\rho = 0.9$), un discount de 0.95, un pas d'apprentissage de 0.001 pour l'acteur et de 0.003 pour la critique, un batch de 1000 transitions échantillonnées dans un buffer de capacité 1000000 à chaque pas d'optimisation (où chaque reward du buffer à été divisé par 1000), une taille d'épisode maximale de 200 événements en apprentissage (également en test) et selon un réseau de neurones à deux couches cachées de 30 neurones chacune (avec activation leakyRelu sur les couches cachées et batch normalisation avant chaque couche):



(a) Reward en apprentissage
(abscisse: nombre d'episodes)

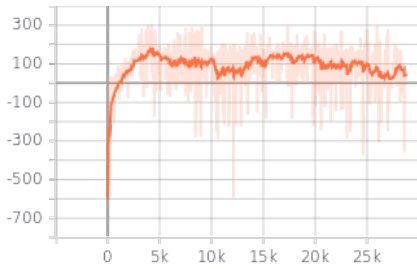


(b) Valeur cible moyenne
(abscisse: nombre d'optimisations)

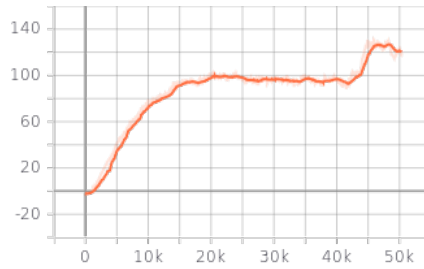


(c) Loss du critique
(abscisse: nombre d'optimisations)

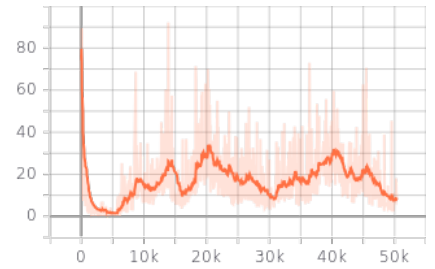
On donne également ci-dessous ce qu'on peut obtenir sur LunarLander avec DDPG, avec les mêmes paramètres que pour Pendulum ci-dessus, mais sans batch normalisation, des batchs de 1280 transitions, un replay buffer de capacité 100000, des réseaux acteurs et critiques à 4 couches cachées (2 de 64 suivies de 2 de 32 neurones), 5 pas de gradients à chaque optimisation (effectuée tous les 1000 événements), sans normalisation des récompenses, un paramètre $\rho = 0.99$ pour la mise à jour soft des paramètres de réseaux cible (via Polyak) et un facteur de discount de $\gamma = 0.99$.



(a) Reward en apprentissage
(abscisse: nombre d'episodes)



(b) Valeur cible moyenne
(abscisse: nombre d'optimisations)



(c) Loss du critique
(abscisse: nombre d'optimisations)