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

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

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

    end for
  end if
until convergence

Ajout d'un bruit gaussien pour exploration + clip pour rester dans des valeurs admissibles

Utilisation de réseaux cible (à la fois pour Q et pour μ) pour le calcul de la cible de Q

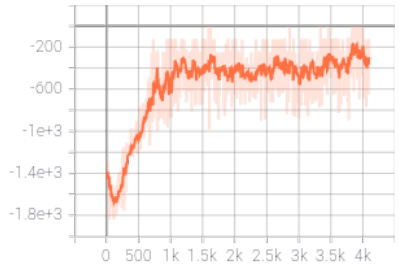
Gradient similaire à DPG, selon les transitions du batch

Mise à jour "soft" des paramètres des réseaux cible

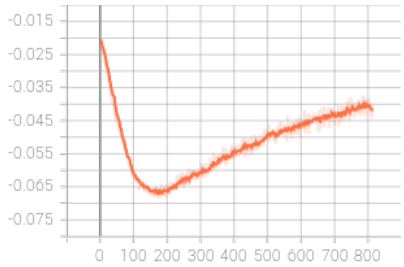
Appliquer l'algorithme aux 3 problèmes suivants:

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

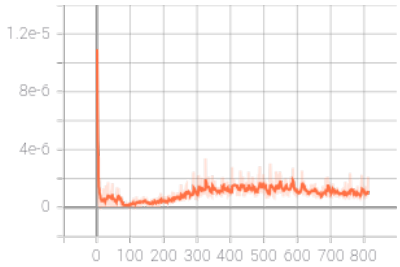
À titre indicatif, voici ce qu'on peut obtenir sur Pendulum avec DDPG utilisant un processus d'exploration de Ornstein-Uhlenbeck avec $\sigma = 0.2$ (voir core.py pour la méthode d'échantillonnage du bruit, à réinitialiser à la fin de chaque trajectoire), effectuant une optimisation de 10 étapes tous les 1000 évènements, 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 évènements en apprentissage (également en test) et selon un réseau de neurones à deux couches cachées de 30 neurones chacune (avec activation tanh sur les couches cachées):



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



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



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