TME8 RLD

Soft Actor-Critic

Brient, Hein - Décembre 2021

1 Objectifs

Ce TME a pour objectif d'expérimenter l'approche SAC, pour environnements à actions continues, avec maintient d'entropie. Trois environnements différents sont disponibles pour tester cet algorithme :

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

2 Algorithme

Soft Actor Critic (SAC) est un algorithme qui optimise une politique stochastique de manière off-policy, liant l'optimisation de politique stochastique et les approches type DDPG. Un aspect central de l'algorithme SAC est la régularisation d'entropie. La politique est entraînée pour maximiser le compromis entre les récompenses attendues et l'entropie. Intuitivement, augmenter l'entropie revient à explorer plus, ce qui peut accélerer l'apprentissage et éviter la convergence de la politique vers un mauvais minimum local.

L'entropie est une quantité qui donne idée sur à quel point une variable aléatoire est aléatoire. Soit une variable aléatoire x avec une fonction densité de probabilité P, l'entropie H de x est définie par :

$$H(P) = \mathbb{E}_{x\sim P}[-\log P(x)]$$

En apprentissage par renforcement avec régularisation d'entropie, l'agent reçoit une récompense bonus à chaque étape proportionnel à l'entropie de la politique à cet étape. Ainsi, pour ce type de problèmes, l'équation de Bellman pour Q^π est alors de la forme suivante :

$$Q^\pi(s,a) = \mathbb{E}_{s'\sim P, a'\sim\pi}[R(s,a,s')+\gamma\left(Q^\pi(s',a') + \alpha H\left(\pi(\cdot|s')\right)\right)] = \mathbb{E}_{s'\sim P}[R(s,a,s')+\gamma V^\pi(s')]$$

SAC apprend à la fois une politique π_θ et deux Q-fonctions Q_{ϕ_1}, Q_{ϕ_2}. Ces deux Q-fonctions sont entraînés par minimisation MSBE à partir d'une même valeur cible. Cette cible partagée est calculée en utilisant des target Q-networks, qui sont obtenus par polyak averaging durant la phase d'entraînement. Aussi, les next-states utilisés dans la valeur cible viennent de la politique courante au lieu de la politique cible. Par ailleurs, il n'y a pas de lissage de la politique cible puisque le bruit introduit par la stochasticité de la politique à prédire est suffisant pour avoir des résultats similaires. Ainsi, la fonction de perte pour les Q-networks de l'algorithme SAC s'exprime par la relation suivante :

$$L(\phi_i, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \left[\left(Q_{\phi_i}(s,a) - y(r,s',d) \right)^2 \right],$$

Où la cible est donnée par :

$$y(r,s',d) = r + \gamma(1-d)\left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \qquad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

Par ailleurs, la politique à apprendre doit maximiser les récompenses futures attendues ainsi que la future entropie. C'est à dire qu'elle doit maximiser $V^\pi(s)$, qui peut être exprimée de la façon suivante :

$$V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a) + \alpha H\left(\pi(\cdot|s)\right)] = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a) - \alpha \log \pi(a|s)]$$

Pour optimiser la politique, on utilise le "reparameterization trick", avec lequel un échantillon de $\pi_\theta(\cdot|s)$ est réalisé de la façon suivante :

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I)$$

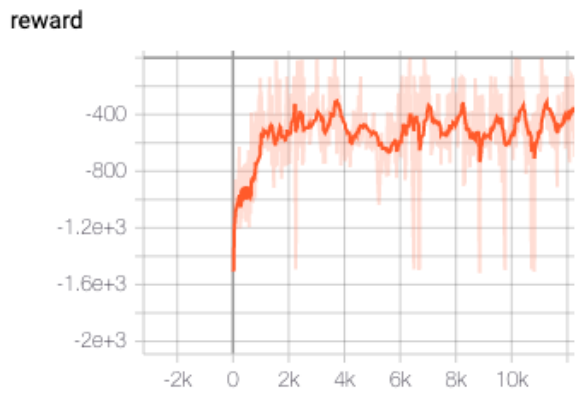
Enfin, pour obtenir la perte liée à la politique, la dernière étape consiste à substituer Q^{π_θ} avec une de nos fonctions d'approximation. SAC utilise $\min_{j=1,2} Q_{\phi_j}$. Ainsi, la politique est optimisée selon :

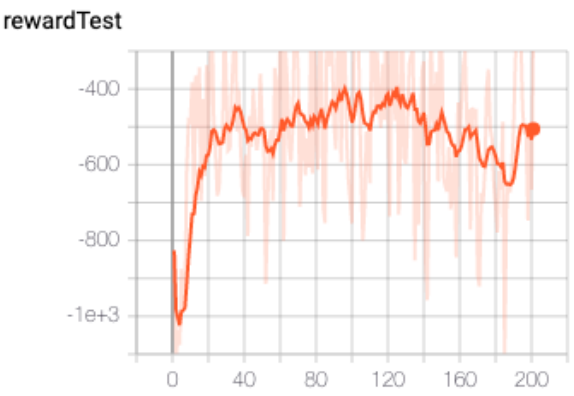
$$\max_\theta \mathbb{E}_{s\sim\mathcal{D},\xi\sim\mathcal{N}}[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)]$$

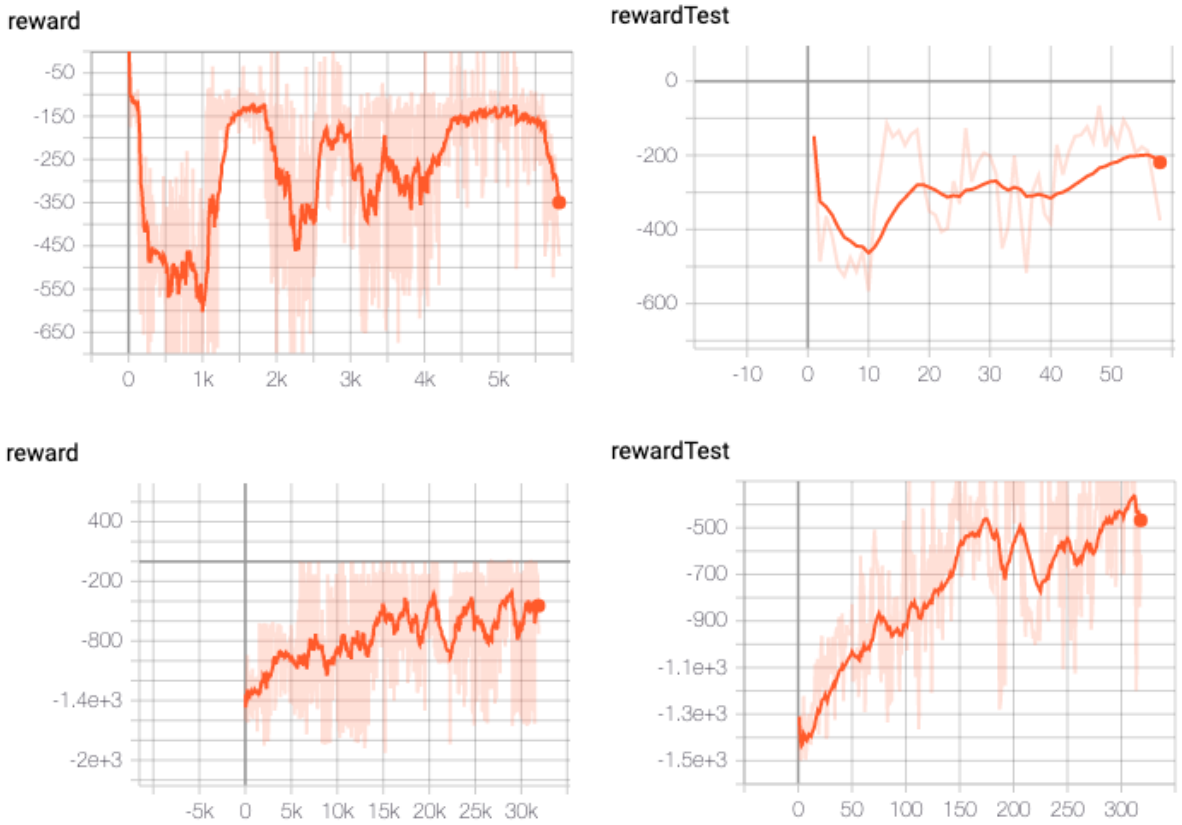
Cette fonction de perte est très ressemblante à celle vue dans l'algorithme DDPG avec quelques légères différences : la stochasiticité, la technique du double Q minimal et le terme entropique.

3 Implémentation et résultats

Nous avons alors implémenté l'algorithme SAC puis l'avons testé sur les trois environnements proposés après une recherche exhaustive des hyperparamètres. On présente comme dans le TP précédent, les résultats pour les trois environnements, repectivement Pendulum, LunarLander et Mountain Car.







On peut voir que l'algorithme SAC apprend mieux que DDPG sur pendulum mais que les performances ne sont pas aussi bonnes que celles présentées dans le sujet, avec des hyper-paramètres très semblables. Le reward semble en effet stagner vers -400 plutôt que -200.

Pour l'environnement LunarLander, il a été de nouveau beaucoup plus difficile de trouver un jeu d'hyperparamètres donnant de bonnes performances. L'agent semble passer par des phases d'apprentissage mais ne réussit pas à atteindre une récompense de plus de -200.

Pour MoutainCar, la convergence était très longue mais l'agent semblait continuellement améliorer son score. Toutefois, en 30000 épisodes, on s'attendrait à obtenir de bien meilleures récompenses via un jeu d'hyperparamètres plus adéquat.

En conclusion de ces deux derniers TP, nous retiendrons qu'à nouveau, la recherche d'hyperparamètres semble être une étape nécessaire pour correctement évaluer les performances des algorithmes. En effet, dans les deux cas, nous ne somme pas parvenus à obtenir de bons scores sur l'environnement LunarLander. Aussi, SAC semble mieux appréhender l'environnement Pendulum que DDPG et on observe le cas contraire pour l'environnement MountainCar. Enfin, la plupart des performances n'ont pas satisfait nos attentes et même si tester différents jeux de paramètres permet de mieux appréhender les influences de chacun, une étude plus poussée et chiffrée sur cet aspect permettrait sûrement, à terme, d'obtenir de biens meilleurs résultats.