RÉSUMÉ DE PROJET

Reinforcement Learning in Continuous Action Spaces

Team:

Ethan ABITBOL 3804139
Dufourmantelle JEREMY 21104331

Supervisor:

Olivier SIGAUD

1 Introduction

Ce résumé a pour but de présenter notre travail sur l'implémentation des algorithmes et les environnements de tests mentionnés dans le papier **Reinforcement Learning in Continuous Action Spaces** (2007) de Hado van Hasselt et Marco A. Wiering, ainsi que de critiquer la reproductibilité des résultats obtenus concernant une nouvelle famille d'algorithmes d'apprentissage par renforcement appelée **CACLA** (**Continuous Actor Critic Learning Automaton**).

2 Algorithmes implémentés

Tout d'abord, nous avons étudié le papier afin d'extraire un pseudo-code générique de la version en ligne de **CACLA+VAR** (annexe 1). Le pseudo-code de **CACLA+VAR** est en ligne, c'est-à-dire qu'il utilise un exemple à chaque étape d'apprentissage, comme présenté dans le papier. Nous voyons que à la ligne **8**, nous faisons une exploration gaussienne, d'un autre côté, il suffit de faire un tirage aléatoire dans $[0, 1]$ avec une probabilité d'exploration ϵ pour l'approche ϵ-greedy. À chaque étape, nous mettons à jour les poids du critique, mais les poids de l'acteur ne sont mis à jour que si l'erreur de différence temporelle est positive. Enfin, nous faisons un calcul qui utilise un terme de variance et l'erreur temporelle δ pour déterminer le nombre de mises à jour de l'acteur. À savoir que l'approche **CACLA** n'utilise pas ce dernier calcul. De plus, la version **ACLA** est la version discrète de CACLA, c'est-à-dire qu'elle travaille dans un espace d'actions discret. Enfin, la version **CAC** consiste à retirer la condition de la ligne **13** de la mise à jour des poids de l'acteur si l'erreur de différence temporelle est positive. Nous avons implémenté les versions CAC, CACLA et CACLA+VAR avec les méthodes d'exploration gaussienne et ϵ-greedy. Le code à été développé en **Python**.

3 Environnements de tests

Dans un premier temps, nous avons développé l'environnement de test **Tracking** qui comprend un agent qui doit rejoindre une cible en mouvement circulaire dans un environnement d'espace et d'action continu avec un obstacle et une limite de temps (voir annexe 2). Nous avons développé cet environnement en Python avec les informations présentes dans le papier (texte et graphique).
Le second environnement de test est **CartPole Continuous**, où l'agent est une plateforme qui doit faire tenir en équilibre une tige dans un environnement d'espaces et d'actions continu. Comme pour le premier environnement, nous avons utilisé les informations présentes dans le papier. Notre code de CartPole s'est inspiré d'une version sur GitHub que nous avons adaptée à nos besoins.

4 Résultats et comparaisons

Nous avons exécuté nos algorithmes sur les deux environnements de tests avec les deux méthodes d'exploration avec les valeurs d'hyperparamètres données dans le papier. Nous avons comparé nos résultats en termes d'apprentissage sur les récompenses normalisées entre 0 et 1 en termes d'itérations et d'épisodes, nous avons calculé les moyennes et écarts-types ainsi que le taux de réussite des agents après la phase d'apprentissage. Globalement, les résultats semblent cohérents avec ceux du papier et ils nous montrent que les algorithmes **CACLA** réussissent à apprendre dans leurs environnements. De plus, nous avons implémenté une version par **batch** de CACLA et CACLA+VAR qui nous donne des résultats intéressants. Nous avons aussi comparer l'effet de faire les mises à jour de l'acteur et du critique uniquement en fonction du signe de l'erreur de différence temporelle. Enfin, nous avons comparer les résultats d'un **DDPG** avec et sans mise à jour en fonction de l'erreur de différence temporelle et nous avons constaté aussi que l'algorithme **PPO** apprend beaucoup plus rapidement que la famille CACLA.

5 Conclusion

Pour conclure, durant ce projet, nous avons réussi à implémenter CAC, CACLA, CACLA+VAR et les deux environnements de tests. Les résultats obtenus ne sont pas exactement similaires au niveau des chiffres, mais dégagent la même conclusion : CACLA+VAR est meilleur que CACLA. De plus, nous avons également montré que l'exploration gaussienne était plus prometteuse que l'exploration épsilon-greedy et que faire la mise à jour que lorsque l'erreur de différence temporelle est positive est bien meilleure, comme le montrent les comparaisons entre CAC et les autres algorithmes. Aussi, le fait de tester seulement sur 20 simulations nous semble peu et non significatif en termes de performances, car les résultats varient d'une exécution à une autre. Nous avons aussi poursuivie la discussion avec une version des algorithmes par **batch**, tester de mettre à jour l'acteur et le critique en même temps et comparer les résultats avec des algos plus récent de Deep RL comme PPO ou DDPG.

6 Annexe

6.1 Annexe 1

Algorithm 1 CACLA+VAR (version en-ligne)

Require: γ : facteur d'actualisation, σ : ecart-type, var_0 : variance initiale, β ,α_1 : facteur d'apprentissage critique, α_2 : facteur d'apprentissage acteur, M : Nombre d'épisode d'apprentissage, ϵ , ϵ_{min} , ϵ_{decay}

Initialisation critique avec θ_1 noté V
Initialisation acteur avec θ_2 noté AC
for episode de 1 à M **do**
  $var = var_0$
  initialisation du premier état : s_t
  while épisode non fini **do**
    ϵ = max(ϵ_{min} , $\epsilon * \epsilon_{decay}$) {uniquement pour exploration ϵ-greedy}
    action a_t = tirage dans $\mathbf{N}(AC(s_t), \sigma)$ {exploration gaussienne}
    Obtenir l'état suivant s_{t+1} et la reward r en effectuant l'action a_t
    $\delta = r + \gamma * V(s_{t+1}) - V(s_t)$ {Erreur de différence temporelle}
    $\theta_1 = \theta_1 + \alpha_1 \delta \frac{\partial V(s_t)}{\partial \theta_1}$ {Mise à jour des poids du critique}
    $var = (1 - \beta)var + \beta\delta^2$
    if $\delta > 0$ **then**
      $n = \lceil \delta / \sqrt{var} \rceil$ {nombre de mise à jour des poids}
      for itération de 1 à n **do**
        $\theta_2 = \theta_2 + \alpha_2 (a_t - AC(s_t)) \frac{\partial AC(s_t)}{\partial \theta_2}$ {Mise à jour des poids de l'acteur}
      end for
    end if
    $s_t = s_{t+1}$
  end while
end for

6.2 Annexe 2

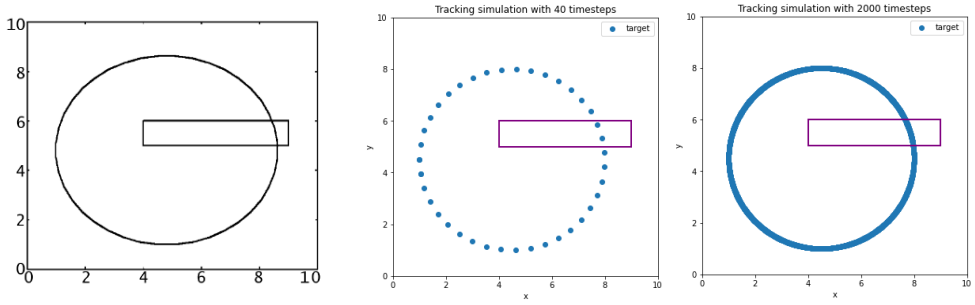

Figure 1: A gauche, l'environnement tracking du papier et à droite, notre environnement avec 40 et 2000 pas de temps.