SORBONNE UNIVERSITE
Faculté des Sciences et Ingénerie

Licence Informatique - L3

PROJET RECHECHE

Combiner apprentissage par renforcement profond et méthodes évolutionnaires

Algorithme CEM-ERL

Rapport final

Auteurs

Ben Kabongo Buzangu
Théo Charlot
Wassim Ouni

Encadrant

Olivier Sigaud

Janvier 2022 - Mai 2022

1 Introduction

Dans le cadre de l'unité d'enseignement **Projet Recherche**, encadrés par M. Olivier Sigaud, nous avons travaillé sur le projet intitulé **Combiner apprentissage par renforcement profond et méthodes évolutionnaires : l'algorithme CEM-ERL**. L'objectif de ce projet a été de concevoir un nouvel algorithme, **CEM-ERL**, qui fusionne les principes de trois autres algorithmes existants qui sont **CEM**, **CEM-RL** et **ERL** ; et ensuite d'effectuer des comparaisons des performances des algorithmes **CEM**, **CEM-RL** et **CEM-ERL**.

Dans la suite de ce rapport, nous allons commencer par définir les différents concepts clés du sujet que nous traitons. Nous aborderons ensuite le sujet de l'apprentissage par renforcement profond et les méthodes évolutionnaires. Puis nous présenterons les algorithmes **CEM**, **CEM-RL**, **ERL** et **CEM-ERL**. Nous présenterons par la suite les résultats des comparaisons des différents algorithmes. Enfin nous conclurons et aborderons quelques points d'ouvertures suite à ce projet.

2 Définitions

Nous commençons par définir quelques notions importantes utilisées dans ce rapport :

- **Politique** ou stratégie : est un comportement décisionnel de l'agent, une fonction qui à tout état s associe l'action à exécuter. L'apprentissage par renforcement a pour but la recherche des politiques optimales.
- **Critique** : donne, pour chaque couple (état s, action a), une indication sur l'espérance des récompenses futures de l'agent s'il choisit l'action a depuis l'état s.
- **Acteur** : donne, pour chaque état s, une action a.
- **Descente de gradient** : est un algorithme d'optimisation permettant de trouver le minimum d'une fonction en convergeant progressivement vers ce dernier. Ce procédé est utilisé dans les différents algorithmes pour mettre à jour les poids (les paramètres) des agents.
- **Expérience** : définie par un tuple (état courant s, action a, état suivant s', récompense r), qui est une transition de l'agent de l'état s à l'état s', après avoir effectué l'action a, qui lui a rapporté la récompense r.
- **Replay buffer** : tampon, mémoire dans laquelle sont stockées les expériences.
- **Episode** : succession d'expériences d'un individu de la population jusqu'à un critère d'arrêt.
- **Score** : le score d'un individu correspond à la somme cumulée des récompenses obtenues à la suite d'interactions avec l'environnement au cours d'un épisode.
- **Evaluation** : calcul du score des individus de la population.
- **Elites** : une proportion donnée d'individus d'une population aux scores les plus élevés après évaluation.

Les algorithmes **CEM-RL**, **ERL** et **CEM-ERL** sont tous une combinaison de l'apprentissage par renforcement profond et des méthodes évolutionnaires.

Ils commencent par initialiser une population d'individus et des agents d'apprentissage par renforcement profond (critiques et acteurs). A la suite d'épisodes et d'évaluations, une élite est sélectionnée afin de constituer la prochaine génération d'individus de la population.

A chaque épisode, les expériences des individus sont enregistrées dans un replay buffer. Les agents d'apprentissage par renforcenement utilisent les informations contenues dans ce dernier afin de se mettre à jour. À une certaine fréquence et selon l'algorithme, les informations (direction du gradient, poids du réseaux de neurones, etc.) sont copiées ou injectées dans une partie ou la totalité de la population.

3 Apprentissage

L'apprentissage par renforcement consiste, pour un agent, à apprendre par le biais d'interactions avec un environnement les actions à réaliser dans chaque état, afin de maximiser la somme des récompenses au cours du temps Sutton et Barto 2018.

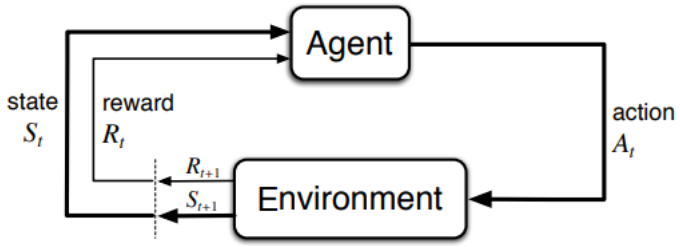


Figure 1 – Apprentissage par renforcement (tiré de Sutton et Barto 2018

L'agent (un robot, un réseau de neurones, etc.) est plongé au sein d'un environnement et décide de l'action à effectuer depuis son état courant. En réponse à son action, l'environnement lui renvoie une récompense, positive ou négative. Ainsi, au travers d'actions itérées, l'agent cherche une politique optimale afin de maximiser ses récompenses cumulées.

Dans **l'apprentissage par renforcement profond**, les agents sont des **réseaux de neurones artificiels**.

Dans notre projet, le principal algorithme d'apprentissage par renforcement profond que nous avons utilisé est **TD3**.

4 Méthodes évolutionnaires

Les méthodes évolutionnaires sont une famille d'algorithmes qui s'inspirent de la théorie de l'évolution pour résoudre divers problèmes. Le principe est de faire évoluer un ensemble de solutions du problème à résoudre, afin de trouver les meilleurs résultats.

Les différentes solutions constituent les individus de la population. A chaque individu est associé un score, qui correspond à l'appréciation qualitative de la solution. Plus le score d'un individu est élevé, plus grandes sont les chances que ses caractéristiques soient répandues au sein de la population.

Deux familles d'opérateurs principaux définissent la façon de manipuler les individus. Il s'agit des opérateurs de **sélection** qui définissent les procédures de sélection et de remplacement des individus au sein de la population ; et des opérateurs de **variation** qui définissent les mutations et les croisements que peuvent subir les individus de la population.

Les méthodes évolutionnaires commencent par initialiser une population d'individus. Puis elles itèrent jusqu'à un certain critère d'arrêt, à définir, différentes étapes sur la population. Ces étapes sont : la sélection d'une partie de la population (aléatoirement ou sélection de l'élite), reproduction des individus sélectionnés, différents croisements et mutations, évaluation des individus, et constitution de la nouvelle population.

5 CEM : Cross-Entropy-Method

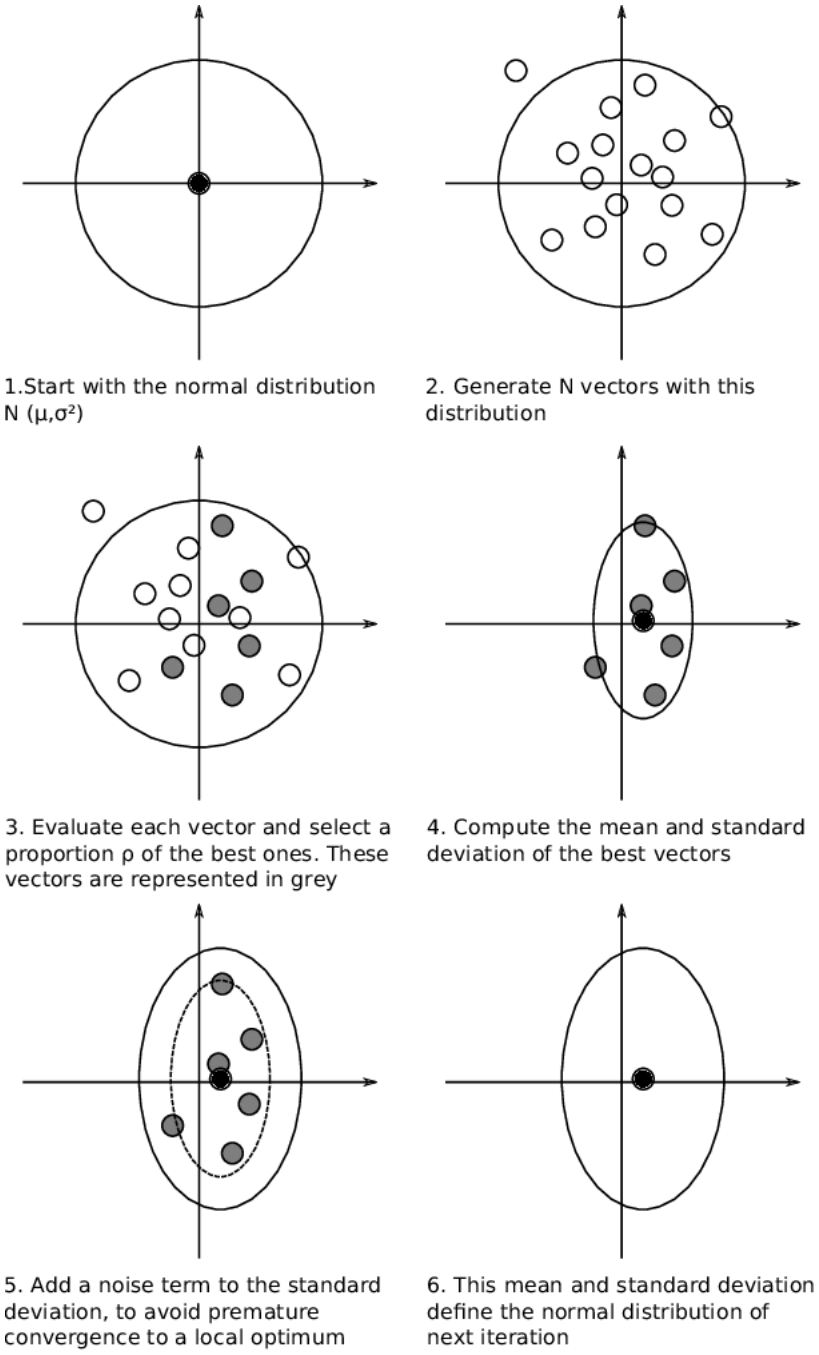


FIGURE 2 – CEM

Il existe deux principales approches pour la recherche de politiques optimales : **les méthodes de gradient de politique** et **les méthodes de recherche directe de politiques**. Il existe de nombreuses méthodes de recherche directe de politiques, dont la **Cross Entropy Method (CEM)**, qui est une méthode simple et efficace SIGAUD et STULP 2019 DUAN et al. 2016

Dans ce projet, nous nous sommes interéssés à une version de CEM que l'on combine à des

réseaux de neurones. Ces derniers nécessitent en entrée un vecteur de paramètres qui correspond à l'état courant de l'agent et fournissent en sortie une action.

Soit θ le vecteur de paramètres représentant les poids et les biais des différents réseaux de neurones. La modification des paramètres change la politique. L'objectif de la méthode est de trouver les paramètres qui correspondent à une politique optimale. Cependant, un problème auquel se heurte CEM est que le nombre de paramètres est souvent trop important. La figure 2 illustre l'algorithme avec des paramètres sur deux dimensions.

La matrice de covariance est une matrice carrée qui donne la covariance entre chaque paramètre.

5.1 Algorithme

La Cross Entropy Method est un algorithme qui se déroule en six étapes :

1. **Initialisation** : l'algorithme commence par initialiser une distribution représentée par une matrice de covariance à partir de laquelle une population d'individus (des paramètres ou des solutions) sera tirée ; généralement une distribution normale dont la moyenne et variance sont données.
2. **Génération de la population** : suivant la précédente distribution, N individus sont générés. Ils correspondent en effet à différentes politiques dont on donne les paramètres θ.
3. **Evaluation et sélection** : les individus sont évalués et une élite est sélectionnée.
4. **Sélection des élites** : on ne conserve que l'élite et on calcule une nouvelle matrice de covariance en fonction de cette dernière.
5. **Ajout du bruit** : on rajoute du bruit à la précédente distribution.
6. **Nouvelle distribution** : de la précédente étape, on obtient une nouvelle distribution, pour la génération de la prochaine population.

6 CEM-RL

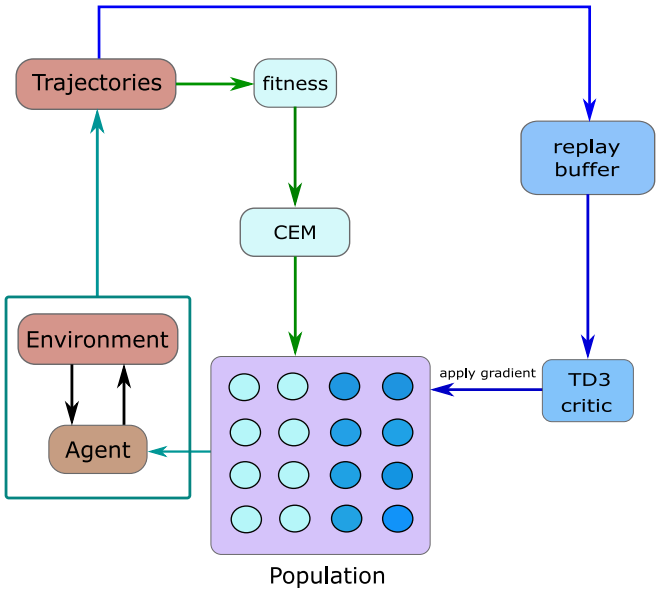


FIGURE 3 – CEM-RL

Comme ERL et CEM-ERL, **CEM-RL** POURCHOT et SIGAUD 2018 combine **CEM** avec des algorithmes d'apprentissage par renforcement profond tels que **DDPG** LILLICRAP et al. 2015 et **TD3** FUJIMOTO, HOOF et MEGER 2018.

L'algorithme commence par initialiser aléatoirement les individus de la population CEM ainsi qu'un critique pour l'apprentissage par renforcement.

Selon la matrice de covariance courante et à chaque itération, une population d'individus est échantillonnée par CEM, avec du bruit gaussien autour de la moyenne.

Le critique est mis à jour à l'aide des données générées par tous les individus. Puis il est utilisé pour mettre à jour la moitié des individus de la population en leur appliquant la direction de son nouveau gradient pour un nombre d'étapes fixe. Ensuite, tous les individus de la population sont évalués.

Dans les cas où l'application du gradient augmente la performance de l'acteur, CEM profite de cette augmentation. En revanche, dans les cas où les étapes de gradient diminuent les performances, les acteurs résultants sont ignorés par CEM, qui se concentre plutôt sur l'élite.

Cette approche génère un flux d'informations bénéfique entre la partie apprentissage et la partie évolution. D'autre part, de bons acteurs trouvés en suivant l'actuel critique améliorent directement la population. Et, les bons acteurs trouvés par évolution remplissent le replay buffer à partir duquel l'algorithme d'apprentissage s'entraîne.

Enfin, les nouveaux paramètres de CEM sont calculés à partir de la moitié d'individus la plus

performante de la population obtenue.

7 ERL

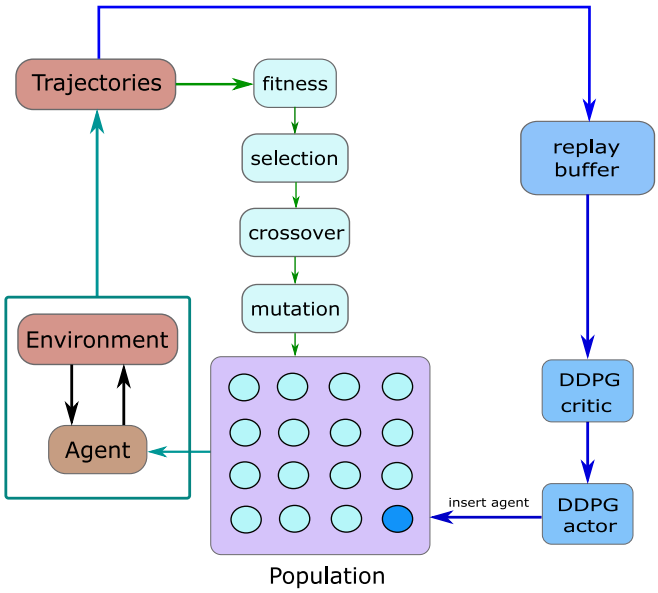


FIGURE 4 – ERL

L'algorithme ERL (KHADKA et TUMER 2018) initialise une population d'individus, par initialisation Glorot. Il s'agit d'un procédé d'initialisation de réseaux de neurones permettant d'obtenir des meilleures performances.

L'algorithme initialise également un critique et un acteur d'apprentissage par renforcement.

Au fil des épisodes, on sélectionne les élites. Ces derniers sont préservés, et le reste des individus sélectionnés subissent mutations et croisements. Les individus obtenus et les élites constituent les individus de la prochaine génération. Egalement, le critique et l'acteur se mettent à jour.

Périodiquement, les poids de l'acteur sont copiés dans un individu de la population en évolution, afin de tirer parti des informations apprises par la méthode de descente de gradient et de stabiliser l'apprentissage. Si la politique apprise est bonne, elle sera sélectionnée et étendue dans les générations suivantes. Sinon, elle ne sera pas sélectionnée. Cela garantit que les informations obtenues de l'acteur soient constructives.

8 CEM-ERL

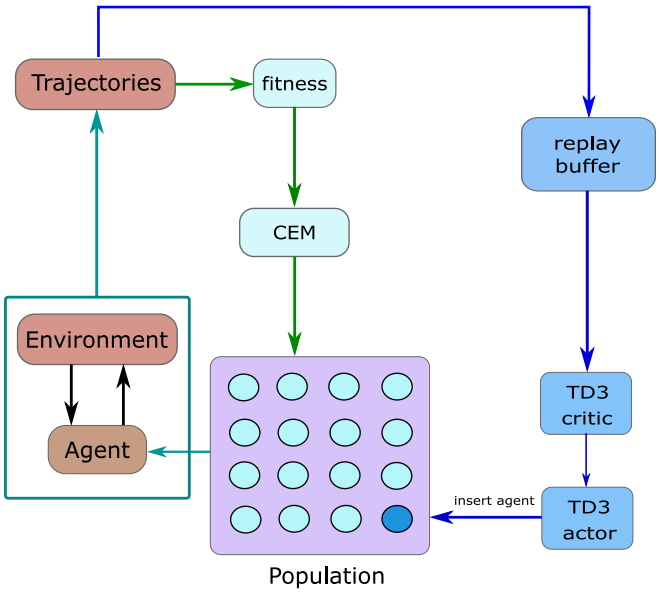


FIGURE 5 – CEM-ERL

8.1 Présentation

CEM-ERL est l'algorithme que nous devons implémenter pour notre projet, pour ensuite comparer ses performances à celles de CEM-RL et ERL, dont il fusionne les principes fondamentaux.

L'algorithme initialise une population d'individus CEM, un acteur et un critique pour l'apprentissage par renforcement.

Selon la matrice de covariance et à chaque iteration, CEM échantillonne une population. Périodiquement, l'acteur est rajouté aux individus de la population CEM. A la suite d'épisodes et d'évaluations, les élites sont sélectionnées. Ils déterminent les nouveaux paramètres de la distribution de la prochaine génération.

8.2 Algorithme

Algorithm 1 CEM-ERL

Require: max_steps the maximum number of steps in the environment
τ_{CEM},σ_{init}, pop_size and $elite\_size$, hyper parameters of the CEM algorithm
γ,τ,lr_{actor},lr_{critic}, hyperparameters of TD3

Initialize a random actor π_μ, to be the mean of the CEM algorithm
let $\Sigma = \sigma_{init} I$ be the covariance matrix of the CEM algorithm
Initialize critic networks $\mathcal{Q}_1$, $\mathcal{Q}_2$ and actor network π_{TD3}
Initialize an empty cyclic replay buffer $\mathcal{R}$

$total\_steps, actor\_steps = 0, 0$
while $total\_steps < max\_steps$: **do**

  Draw the current population pop from $\mathcal{N}(\pi_\mu, \Sigma)$ with importance mixing
  Train $\mathcal{Q}_1, \mathcal{Q}_2$ for $2 * actor\_steps$ / pop_size mini-batches
  Train π_{TD3} for $actor\_steps$ mini-batches
  Insert π_{TD3} in pop

  $actor\_steps = 0$
  for $i \leftarrow 1$ to $pop\_size/2$ **do**
    Set the current policy π to $pop[i]$
    (fitness f, steps s) $\leftarrow$ evaluate(π)
    Fill $\mathcal{R}$ with the collected experiences
    $actor\_steps = actor\_steps + s$
  end for
  $total\_steps = total\_steps + actor\_steps$

  Update π_μ and Σ with the top $elite\_size$ of the population
end while

9 Expériences et résultats

Dans cette section nous étudions CEM-ERL pour répondre à la question :

- Quelle est la performance de CEM-ERL comparée à celles de CEM-RL, CEM et TD3 pris séparément ?

9.1 Processus Expérimental

Pour étudier cette question nous évaluons ces différents algorithmes dans deux environnements gym continus, CartPoleContinuous-v1 ainsi que Pendulum-v1. Nous avons implémenté CEM-ERL avec la librairie Salina Denoyer et al. 2021, une surcouche de la librairie PyTorch. Notre code s'inspire de l'implémentation CEM-RL en Stable-Baselines-3 d'A.Pourchot ainsi que de l'implémentation CEM multi-processus en salina de L.Denoyer.

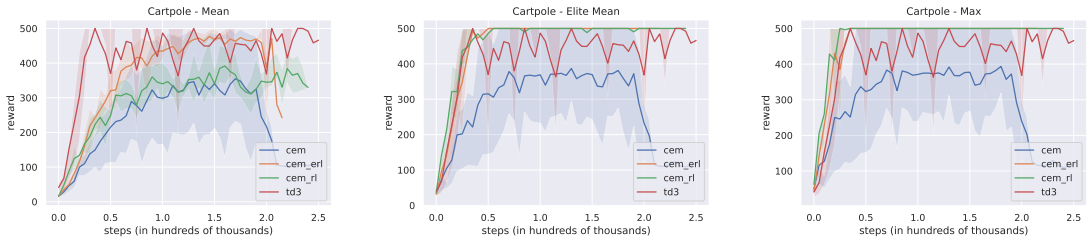
Pour diminuer le temps d'exécution de la partie TD3 nous avons divisé la taille des réseaux par 10. L'éxecution de CEM étant assez peu coûteuse nous avons augmenté la taille de la population

à 15 agents tout en gardant la sélection des 3 meilleurs pour l'élite nous ramenant à un ratio de séléction des 0.2 meilleurs de la population. Hormis les éléments venant d'être cités, l'architecture de nos réseaux et nos paramètres expérimentaux sont identiques à ceux d'A.Pourchot dans CEM-RL.

Chaque courbe représente respectivement, la moyenne de la population, la moyenne des élites sélectionnées ainsi que le score maximum. Ces courbes ont été obtenues en faisant la moyenne et les quartiles (25%,75%) sur 10 expérimentations différentes avec pour CartPole-v1 et Pendulum-v1 respectivement, 200.000 et 500.000 steps.

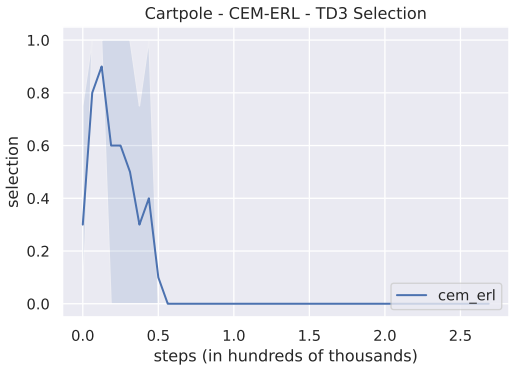
TD3 étant composé d'un seul agent, il ne possède pas de population, sa moyenne et son élite étant lui-même il est donc représenté avec la même courbe sur les différentes figures.

9.2 Cartpole-v1



(a) (b) (c)

FIGURE 6 – Courbes d'apprentissage de CEM, TD3, CEM-RL et CEM-ERL sur CartPole-v1



(a)

FIGURE 7 – Sélection de l'acteur TD3 dans l'élite

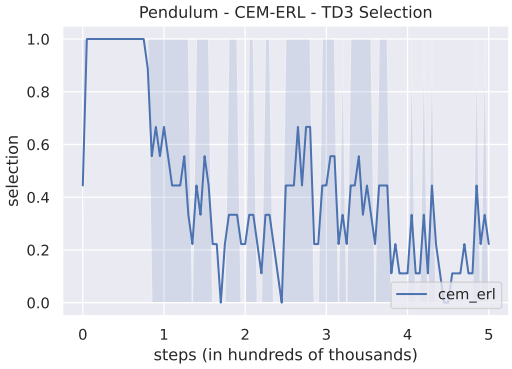
D'après la figure 6a on peut voir que CEM-ERL est plus performant que CEM-RL et est à la hauteur de TD3, les élites elles convergent de la même manière (Figure 6b). D'après la figure 7 l'acteur TD3 semblerait donc permettre de converger rapidement vers la solution optimale dans un premier temps. Une fois atteint un certaine seuil, CEM réussit à trouver continuellement le résultat

optimal. Ceci peut être dû à la simplicité de CartPole-v1 Legros et Kohler 2021 et expliquerait la différence de score avec CEM-RL.

9.3 Pendulum-v1

(a) (b) (c)

Figure 8 – Courbes d'apprentissage de CEM, TD3, CEM-RL et CEM-ERL sur Pendulum-v1



(a)

Figure 9 – Sélection de l'acteur TD3 dans l'élite

Sur Pendulum, d'après la figure 8a CEM-ERL est de pair avec CEM-RL, sur cet environnement plus compliqué, TD3 converge plus rapidement et est légèrement meilleur que CEM-ERL et CEM-ERL. D'après la figure 9 TD3 continuerait à contribuer à l'élite même après avoir atteint le début de la convergence. Ceci suit les affirmations d'A.Pourchot où TD3 accélèrerait la convergence vers de meilleurs acteurs.

10 Conclusion

Nous avons proposé dans ce rapport une évolution de l'algorithme CEM-RL, CEM-ERL qui s'est montré être la hauteur. Nous avons pu montré que l'agent TD3 contribue continuellement à l'apprentissage de l'algorithme. Un seul agent TD3 indépendant apporte autant d'information à la population tout en utilisant moins de ressources que CEM-RL.

L'implémentation de cet algorithme a par ailleurs soulevé de nombreuses questions et idées, durant ce projet nous avons essayé d'implémenter la partie TD3 sur gpu avec l'interface cuda

proposée par torch. Par manque de temps nous avons été contraints d'abandonner cette idée. Il serait donc intéressant d'avoir la partie CEM sur cpu ainsi que la partie TD3 sur gpu où CEM pourrait être exécuté plusieurs fois. Cette implémentation séquentielle, k-CEM-ERL, où CEM s'exécuterait K fois avant que l'agent RL ne soit executé permettrait d'augmenter l'exploration et les performances en calculant plus d'itérations de CEM le tout bénéficiant aussi à TD3 lors de ses mises à jour. De cette piste découle l'idée d'une implémentation asynchrone qui aurait comme bénéfice d'apporter une plus grande part d'exploration Lee et al. 2020, les agents CEM pouvant s'exécuter indépendamment de l'agent TD3, celà permettrait de calculer plus d'itérations de CEM, tout en optimisant le temps de calcul de l'agent TD3 qui est actuellement sur cpu.

11 Remerciements

Nous remercions Olivier Sigaud de nous avoir encadré tout au long du semestre autour de ce projet ainsi que de nous avoir donné l'opportunité de nous familiariser avec l'apprentissage par renforcement. Nous remercions Olivier Serris de nous avoir aidé sur le côté technique de l'implémentation de CEM-ERL.

Références

[1] Ludovic Denoyer et al. "SaLinA : Sequential Learning of Agents". In : *arXiv :2110.07910* (2021).

[2] Yan Duan et al. "Benchmarking Deep Reinforcement Learning for Continuous Control". In : *arXiv :1604.06778* (2016).

[3] Scott Fujimoto, Herke van Hoof et David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In : *arXiv :1802.09477* (2018).

[4] Shauharda Khadka et Kagan Tumer. "Evolutionary reinforcement learning". In : *arXiv preprint, arXiv :1805.07917* (2018).

[5] Kyunghyun Lee et al. "An Efficient Asynchronous Method for Integrating Evolutionary and Gradient-based Policy Searchy (KAIST)". In : (2020).

[6] Damien Legros et Hector Kohler. "CEM vs PG". In : *https ://github.com/DamienLegros/PANDROIDE* (2021).

[7] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In : *arXiv :1509.02971* (2015).

[8] Aloïs Pourchot et Olivier Sigaud. "CEM-RL : Combining evolutionary and gradient-based methods for policy search". In : *arXiv :1810.01222* (2018).

[9] Olivier Sigaud et Freek Stulp. “Policy search in continuous action domains : An overview”. In : *Neural Networks, Elsevier* 113 (2019). arXiv :1803.04706, p. 28-40.

[10] Richard S. Sutton et Andrew G. Barto. *Reinforcement Learning, second edition : An Introduction*. Sous la dir. de MIT Press. 2018. isbn : 9780262039246552.