UE

Rapport Projet : Deep Reinforcement Learning

Doriand PETIT
Rabah MOULAI
ISI

Introduction

Dans ce mini-projet, nous avons dû entraîner un agent à résoudre l'environnement LunarLander-v2 avec des algorithmes de Deep Reinforcement Learning. Ce rapport expliquera succintement le déroulement de nos recherches et il présentera ensuite les résultats obtenus.

Méthodes

DRQN

Tout d'abord, nous avons essayé d'utiliser l'algorithme DRQN, avec la commande suivante :

```
!python train.py --algo qrdqn --env LunarLander-v2 --n-timesteps 100000
```

Les résultats semblaient assez décevants (140 de reward moyen) même avec un haut nombre de steps. Le git Stable-baseline3-zoo fournissait un benchmark récapitulatif des différentes méthodes sur les différents environnements, ce qui nous a permis de voir quels algorithmes étaient optimaux pour cette tâche.

ppo	LunarLander-v2	242.119	31.823	1M	149636	369
ppo	LunarLanderContinuous-v2	270.863	32.072	1M	149956	526
ppo	MountainCar-v0	-110.423	19.473	1M	149954	1358

Nous sommes donc passés à un deuxième algorithme qui semblait beaucoup plus efficace : Proximal Policy Optimization (ppo).

PPO classique

Nous avons donc testé d'entraîner un agent avec PPO avec cette commande :

```
!python train.py --algo ppo --env LunarLander-v2 --n-timesteps 100000
```

Au bout d'un million de steps, le reward moyen continuait à croître (pour un reward final de environ 145, ce qui est similaire des résultats qrdqn). Nous avons donc décidé, pour obtenir des résultats optimaux, d'enchaîner les entraînements sur PPO, afin de voir le reward moyen maximum que l'on peut obtenir.

PPO en réentrainant sur des modèles pré-entraînés

C'est avec cette méthode que nous avons obtenus nos meilleurs résultats. En utilisant la commande ci-dessous, nous avons pu réentraîner notre agent, en utilisant à chaque fois le meilleur modèle de l'entraînement précédent.

```
!python train.py --algo ppo --env LunarLander-v2 -n 1000000 --eval-freq 10000
--eval-episodes 10 --n-eval-envs 1 -i logs/ppo/LunarLander-v2_1/best_model.zip
```

Plus précisément, nous avons utilisé des entraînements de 500 000 ou de 1 000 000 de steps dans la série. Cela permettait d'effectuer des entraînements beaucoup plus longs, qui plus est à chaque fois en ne gardant que les meilleurs modèles.

Nous tracerons les résultats dans la partie ci-dessous.

Optimisation des paramètres

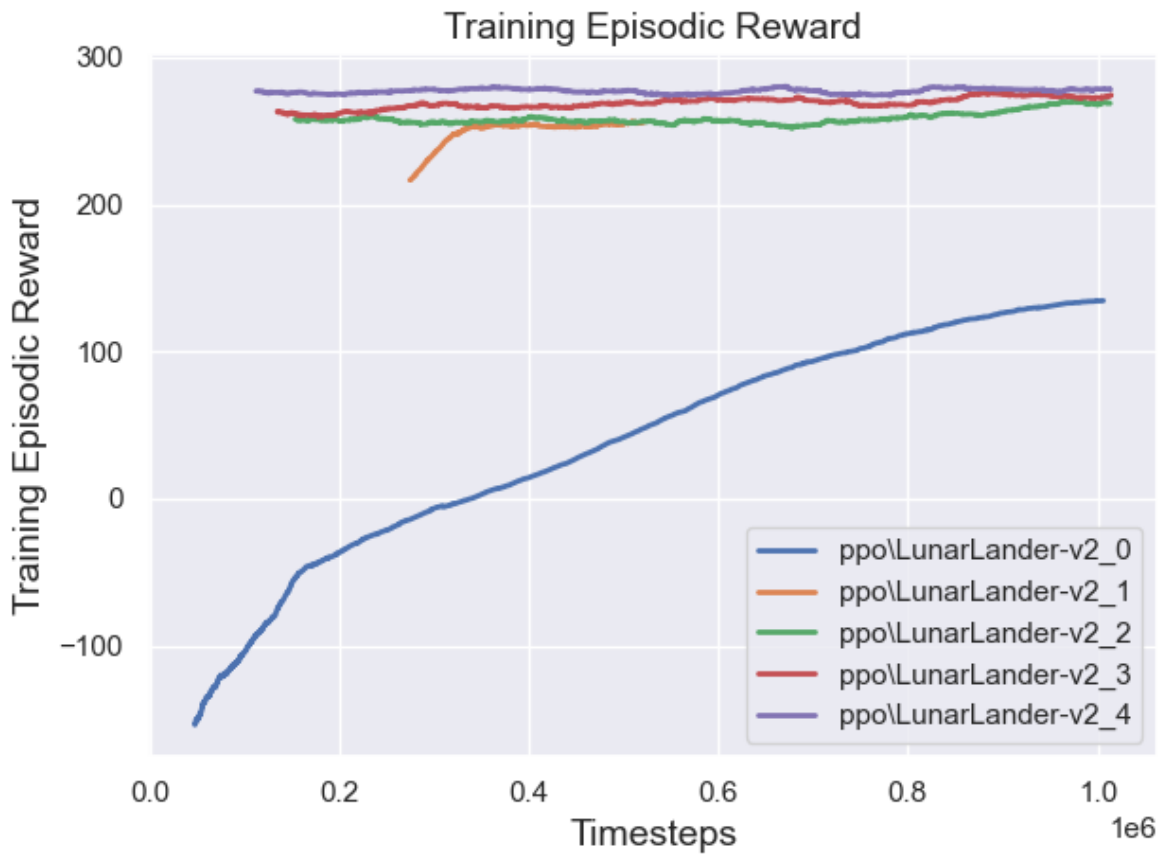
Au bout d'un certain temps, l'algorithme d'apprentissage semble atteindre ses limites malgré de très grand nombres d'étapes. Il paraissait alors logique de chercher à optimiser nos hyperparamètres avec optuna. Une première étape a été d'appliquer la commande basique ci-dessous, mais l'inconvénient de celle-ci était qu'elle utilise des agents non-entraînés et elle prend donc énormément de temps, à un point où le Google Colab s'arrêtait seul avant même d'atteindre la fin des calculs.

```
!python train.py --algo ppo --env LunarLander-v2 -n 1000 -optimize --n-trials 500
--sampler tpe --pruner median
```

La solution serait d'optimiser les hyperparamètres à partir d'un agent pré-entraîné. Cependant, la bibliothèque n'est pas codé de telle sorte que nous pouvons faire ceci, et il faut donc modifier une grande partie du code. Nous n'avons pas eu le temps de réaliser ces changements, et nous avons préféré nous focaliser sur le plot des résultats, qui posait aussi problème, de par l'utilisation des scripts de plot dans Colab.

Résultats

Voici ci-dessous la courbe d'évolution du reward moyen par rapport aux timesteps. Les différentes courbes correspondent à chaque entraînement séparé, dans l'ordre. Certaines courbes sont plus courtes (orange par exemple) car ont duré moins longtemps que un million de steps. Nous pouvons notamment observer que à partir de la fin du deuxième entraînement, la croissance du reward moyen est beaucoup plus faible, voir négligeable (la fin deux ou trois derniers entraînements ont un reward moyen équivalent).



Nous les avons obtenu à partir de cette commande :

```
!python sb3_evaluator.py --algo ppo --env LunarLander-v2
```

Il semble qu'à partir de x steps, le ré-entraînement n'a plus beaucoup d'intérêt si ce n'est un aléatoire supplémentaire (sur la politique et l'initialisation de l'épisode), qui peut donner un meilleur reward moyen avec de la chance. On a cependant atteint une limite de la croissance du reward moyen.

Finalement, à partir de sb3 evaluator, nous avons obtenu un reward moyen de : 281,82. Il est possible que de résultats un petit peu meilleurs soient obtenables en continuant le training, par l'aléatoire évoqué précédemment.

```
!python sb3_evaluator.py --algo ppo --env LunarLander-v2
```

```
LunarLander-v2#ppo#PETIT_nom.zip
Hall of fame
Environment : LunarLander-v2
team:  PETIT_nom                algo: ppo              mean score:  281.82065917 std:  20.34485771418111
Time : 48s 493ms
```

Conclusion

Pour conclure, nous avons donc obtenu des résultats satisfaisants. Il semble que nous ayons atteint la limite de la méthode du "ré-entraînement", et l'optimisation des hyperparamètres semble essentielle pour obtenir des résultats encore meilleurs (le maximum semblant être à une dizaine de reward supplémentaire).