Lunar Lander

Hector KOHLER
Yannis ELRHARBI-FLEURY
M2 ANDROIDE 08/10/2021

Cross Entropy Method (CEM):

On choisit de représenter la politique de contrôle du Lunar Lander par un réseau de neurones d'une couche cachée de taille 32 avec une fonction d'activation de type Rectified Linear Unit et une sortie de type Logit.

On entraîne ce réseau avec la CEM. Le réseau considéré ici est petit comparé aux réseaux utilisés avec les méthodes de DRL classiques qui ont souvent deux couches cachées. En plus de pouvoir apprendre des petits réseaux la CEM nécessite peu, voire pas de parameters tuning. En effet, il n'y a que 4 paramètres à choisir.

Pour l'environnement LunarLander-v2 voici les valeurs des paramètres que nous avons choisi:

- Nombre de politique échantillonnée par itération: 50
- Nombre d'évaluation par politique échantillonnée: 20
- Bruit : 0.2 (valeur par défaut)
- Proportion des politiques échantillonnées gardée pour calculer la politique apprise: 0.2 (valeur par défaut)

La CEM est une méthode de type Estimation de Distribution: à chaque itération de la méthode on échantillonne plusieurs politiques (ici les poids du réseau de neurones décrit au-dessus) et on les évalue toutes.

On peut donc découvrir de très bonnes politiques qui ne sont pas sur la trajectoire d'exploration de l'espace des politiques. En effet la politique apprise n'est que la moyenne des politiques échantillonnées.

La figure 1 représente l'évolution des récompenses obtenues par la politique apprise à chaque itération. La figure 2 représente l'évolution des récompenses obtenues par la meilleure politique échantillonnée à chaque itération.

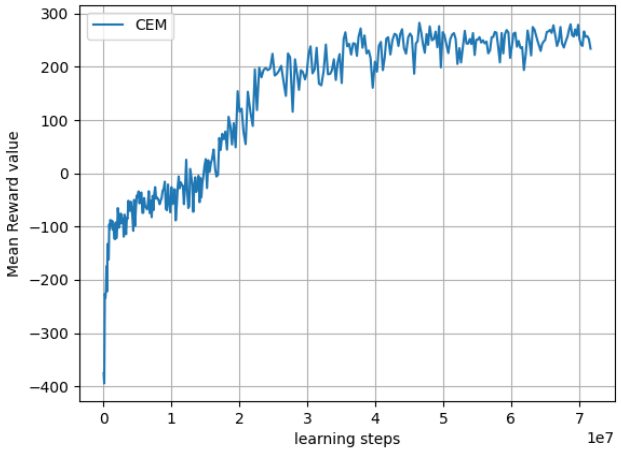

Figure 1: évolution des récompenses obtenues par la politique apprise (apprentissage).

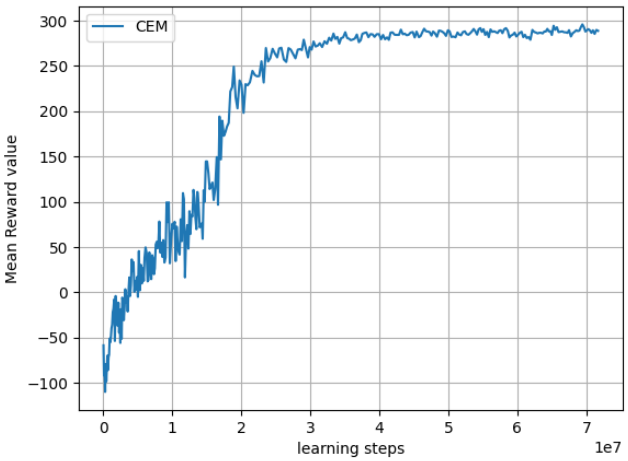

Figure 2: évolution des récompenses obtenues par la meilleure politique échantillonnée durant l'apprentissage.

Dans les figures 1 et 2, on observe qu'il est plus intéressant de choisir la politique de contrôle finale parmi les meilleures politiques échantillonnées plutôt que parmi les politiques apprises.

La meilleure politique obtenue parmi les meilleures politiques échantillonnées donne **282 +- 18** en évaluation.

Visualisation de la politique obtenue :

On cherche à étudier comment le Lander se maintient dans l'axe de la piste lors de la descente.

Dans l'environnement Lunar Lander, l'action 0 signifie "ne rien faire", l'action 1 signifie "activer le réacteur droit", l'action 2 signifie "activer le réacteur principal", et l'action 3 signifie "activer le réacteur gauche".

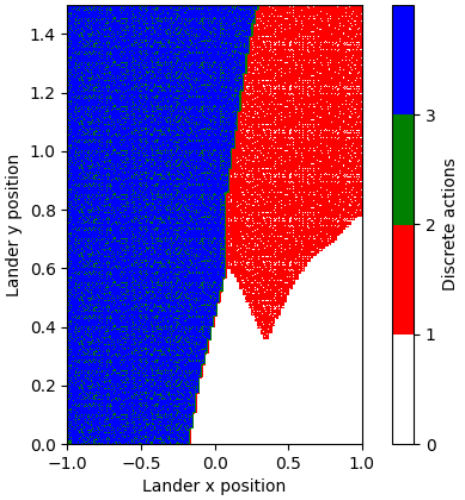

Figure 3: visualisation de l'action effectuée par la politique obtenue avec CEM en fonction des états x et y quand les autres etats sont fixés à 0.

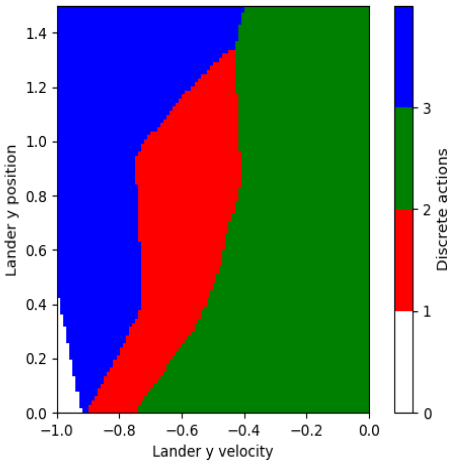

Figure 4: visualisation de l'action effectuée par la politique obtenue avec CEM en fonction de l'altitude et de la vitesse y quand les autres états sont fixés à 0 et que le Lander se trouve à mi hauteur dans l'axe de la piste.

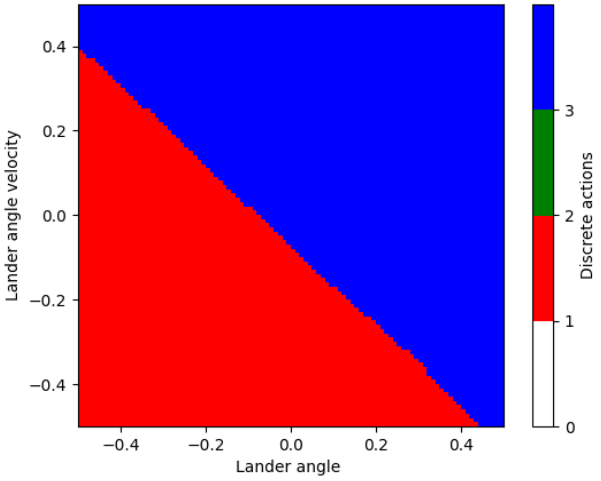

Figure 5: visualisation de l'action effectuée par la politique obtenue avec CEM en fonction de l'angle et de la vitesse angulaire quand les autres états sont fixés à 0 et que le Lander se trouve à mi hauteur dans l'axe de la piste.

Ainsi, sur la figure 3 on devine un comportement efficace: quand le Lander est à gauche de la piste il pousse vers la droite et vice-versa.

Dans la figure 4, on représente l'action effectuée en fonction la position y et de la vitesse y quand les autres états sont fixés à 0. On observe que le Lander active son moteur principal quand il a une basse altitude et une forte vitesse de descente.

Enfin sur la figure 5 on représente l'action effectuée en fonction de l'angle et de la vitesse angulaire quand le robot est à mi hauteur au-dessus de la piste d'atterrissage et les autres vitesses sont nulles. Le Lander essaie toujours d'avoir un angle de 0.

On suppose que cela est dû à la récompense négative lorsque l'acteur n'atterrit pas sur ses deux trains d'atterrissage. De plus, lorsque le l'angle de l'acteur augmente, son moteur est moins efficace pour contrer l'action de la gravité, et induit une déviation par rapport à la piste d'atterrissage.

Comparaison avec DQN :

À titre de comparaison, nous utilisons maintenant l'algorithme DQN avec les paramètres optimaux donnés dans *stable_baselines_3_zoo* (voir Annexe). On obtient une récompense inférieure : 246 +/- 40 en évaluation.

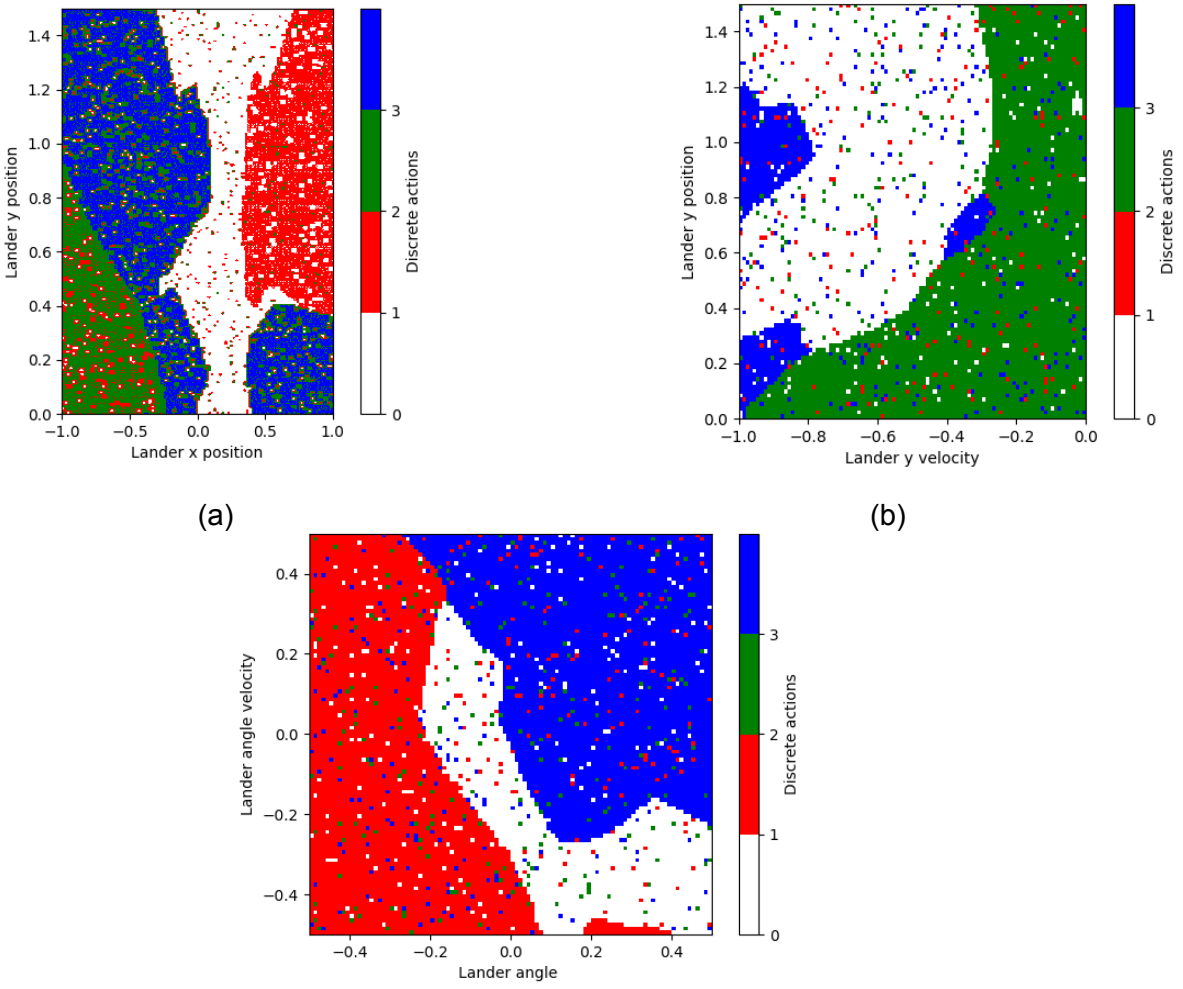

Figure 6: Visualisation d'une politique obtenue avec DQN en utilisant les hyperparamètres du ZOO de SB3 pour LunarLander-v2.

On observe sur la figure 6 une politique différente que pour CEM. La différence(c) majeure est l'utilisation de l'action "ne rien faire" par le Lander entraîné avec DQN. Cela semble logique de ne rien faire quand le lander est dans l'axe de la piste et qu'il n'a pas de vitesse comme sur la figure 6(a) car il peut bénéficier de la gravité pour descendre sans utiliser ses moteurs (utiliser les moteurs ajoute une récompense négative). De même, le Lander entraîné avec DQN ne fait rien quand il a une forte altitude et peu de vitesse y (figure 6(b)). Le Lander entraîné avec CEM ne fait rarement rien: il régule tout le temps son état. Une interprétation de ce résultat pourrait être que le Lander entraîné avec CEM estime qu'il est préférable de tout le temps agir pour contrôler son état plutôt que de ne parfois rien faire pour éviter les pénalités d'utilisation des moteurs.

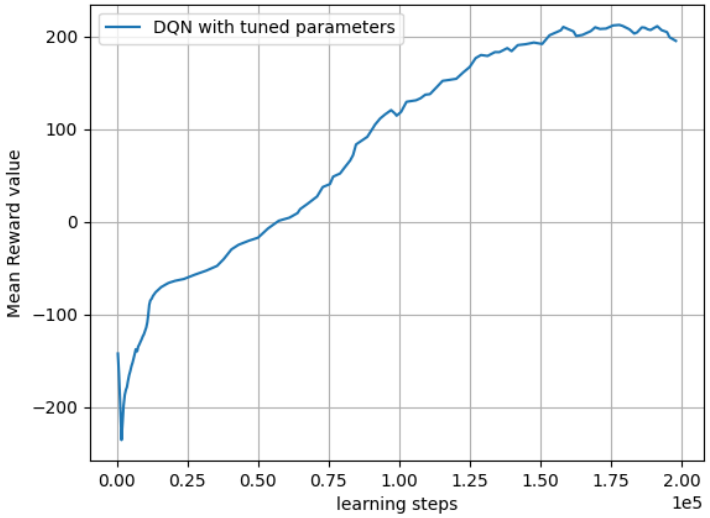

Figure 7: courbe d'apprentissage de DQN avec les hyperparamètres optimaux du ZOO.

Il faut noter que pour obtenir une politique pouvant résoudre l'environnement LunarLander-v2 (récompense supérieure à 200), il faut 100 fois plus de pas de simulation de l'environnement en apprentissage (2 x 10 ^5 pour DQN sur la figure 7, contre 2 x 10^7 pour CEM sur la figure 1).

Tentative d'amélioration DQN:

Pour tenter d'améliorer DQN, nous ajoutons un wrapper à l'environnement. Celui-ci permet d'ajouter une pénalité ou un avantage à chaque action.

Notre objectif est de décomposer l'apprentissage pour que le modèle se concentre sur l'optimisation d'une tâche à la fois. On conjecture que cela améliorera la vitesse de convergence.

On distingue quatre type d'avantages :

- "Nothing" consiste à avantager l'action de ne rien faire
- "Engine" consiste à avantager l'action du réacteur principal
- "Direction" consiste à avantager l'action des réacteurs de direction
- "All" consiste à ne rien avantager

Notre algorithme consiste à tirer à une certaine fréquence un nouvel avantage à appliquer à l'environnement. Lorsqu'une pénalité est tirée, on garde en mémoire sa contribution pour l'apprentissage de l'agent.

$$contribution = r_{tirage} - r_{moyenne\ sur\ le\ tirage}$$

Enfin, on actualise à une fréquence moins grande la distribution de probabilité du tirage en fonction de la contribution de chacune des actions.

```
self.distribution = self.contributions - np.min(self.contributions)
if np.sum(self.distribution) != 0:
      self.distribution= self.distribution / np.sum(self.distribution)

self.distribution = self.distribution + self.min_proba * np.ones(len(self.sampled_list))
self.distribution = self.distribution / np.sum(self.distribution)
```

Avec *min_proba* la probabilité minimale de tirer un avantage.

Pour l'apprentissage on a choisit :

- min_proba : 5%
- durée d'une séquence : 500 pas
- durée d'une distribution : 2000 pas
- probabilité initiale : [Nothing: 1/20, Engine: 1/20, Direction: 0, All: 18/20]
- avantages : Nothing: 0.3 Engine: 0.3 Direction: 0.15 All: 0

La meilleure politique obtenue donne **264 +/- 50** en évaluation, ce qui est meilleur que DQN classique.

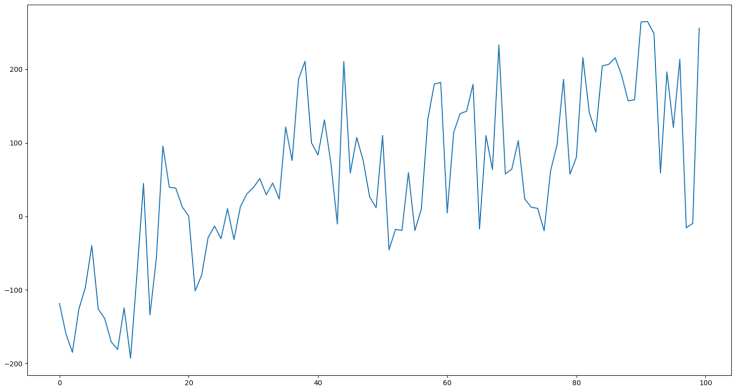

Figure 8: courbe d'apprentissage de DQN avec avantage

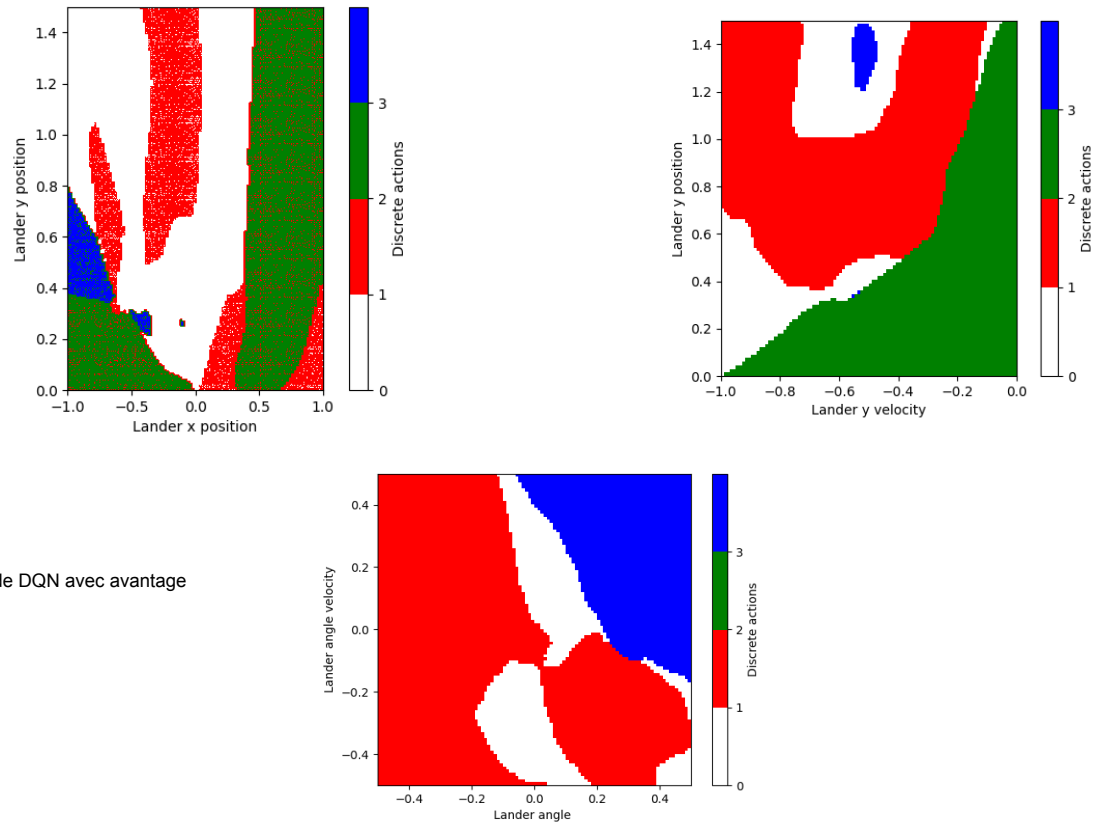

Figure 9: politiques de DQN avec avantage

Dans la figure 8, on remarque immédiatement que l'apprentissage est plus instable, cependant il nous permet quand même d'obtenir de bonnes politiques.

Dans la figure 9, on observe des politiques différentes de celles obtenues précédemment. De plus, on n'a plus de bruit, les zones sont uniformes. On suppose que cela est dû au fait qu'une fois une pénalité choisie, une stratégie va s'imposer aux autres.

Annexe:

Hyperparamètres de SB3 ZOO pour DQN sur LunarLander-v2:

```
  n_timesteps:  1e5
  policy: 'MlpPolicy'
  learning_rate: !!float 6.3e-4
  batch_size: 128
  buffer_size: 50000
  learning_starts: 0
  gamma: 0.99
  target_update_interval: 250
  train_freq: 4
  gradient_steps: -1
  exploration_fraction: 0.12
  exploration_final_eps: 0.1
  policy_kwargs: "dict(net_arch=[256, 256])"
```

(https://github.com/osigaud/rl-baselines3-zoo/blob/master/hyperparams/dqn.yml)