

Rapport RLD

Dao THAUVIN & Filipe Lauar

TME1

Dans le TME1 nous avons implémenté les algorithmes UCB et linUCB. Nous avons comparé leur performance avec 3 stratégies : Random, Static Best et Optimale sur le dataset CTR, qui a 5 dimensions d'entrée et le taux de clics sur les publicités de 10 annonceurs différents.

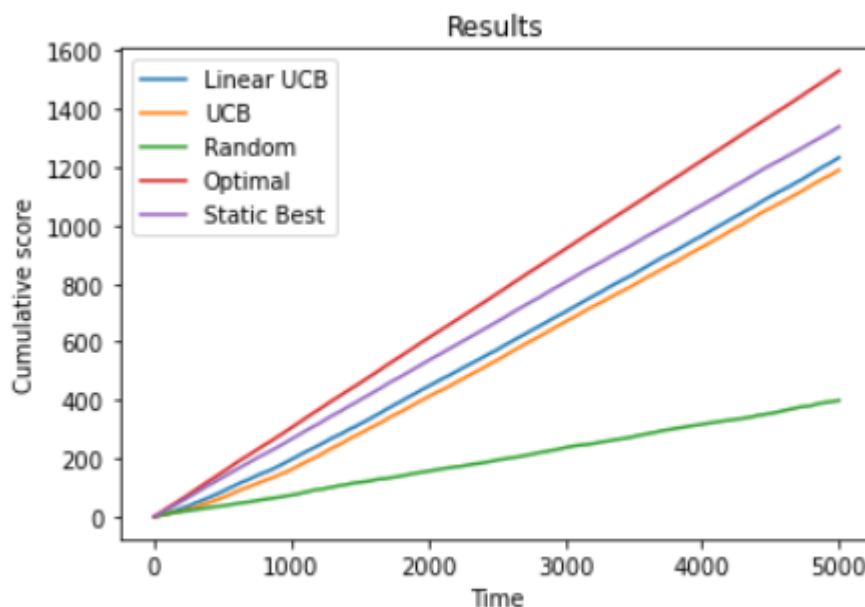
Stratégie Random : A chaque itération, on choisit n'importe quel annonceur;

Stratégie Static Best : A chaque itération, on choisit l'annonceur avec le meilleur taux de clics cumulés;

Stratégie Optimale : A chaque itération, on choisit l'annonceur qui a le meilleur taux de clics à cette itération.

Le but des algorithmes est d'avoir une performance meilleure que les baselines Random et Static Best et arriver le plus proche possible du Optimal.

Ci-dessous on a les résultats pour les 5 différentes stratégies.



On peut voir que la stratégie LinUCB est meilleure que la stratégie UCB simple mais elles ne sont pas meilleures que Static Best.

TME2:

Dans ce TME on a implémenté et testé les méthodes Value Iteration et Policy Evaluation sur le dataset GridWorld. Pour commencer, on va évaluer le temps de convergence de chaque méthode par rapport aux hyper-paramètres gamma (discount) et epsilon (critère d'arrêt) pour l'environnement gridworld-v0.

Nombre d'itérations pour la convergence avec la méthode Value Iteration:

Gamma/Epsilon	0.001	0.01	0.1
0.999	51	31	13
0.99	46	27	13
0.9	16	12	9

Nombre d'itérations pour la convergence avec la Policy Evaluation:

Gamma/Epsilon	0.001	0.01	0.1
0.999	4	4	100
0.99	3	3	100
0.9	3	3	3

On peut voir que pour la méthode Value Iteration, le choix des hyperparamètres a eu beaucoup d'influence dans la convergence, mais dans la méthode Policy Evaluation ça n'a pas trop influencé, juste pour epsilon = 0.1 avec gamma = 0.999 ou gamma = 0.99 où l'algo ne converge pas.

Maintenant on va analyser les résultats des méthodes pour chaque jeu d'hyperparamètres. Pour avoir la récompense moyenne et le nombre d'actions moyen on a fait tourner l'algorithme 1000 fois.

Récompense moyenne/ nombre d'actions moyen avec la méthode Value Iteration.

Gamma/Epsilon	0.001	0.01	0.1
0.999	0.98/20	0.98/20	0.74/10
0.99	0.98/20	0.98/20	0.77/10
0.9	0.54/4	0.56/4	0.47/4

Récompense moyenne/ nombre d'actions moyen avec la méthode Policy Evaluation.

Gamma/Epsilon	0.001	0.01	0.1
0.999	0.98/19	0.98/20	0.79/10
0.99	0.98/20	0.98/20	0.76/10
0.9	0.58/4	0.52/4	0.56/4

On s'attend à ce que le nombre d'actions et la récompense cumulée soient plus petits quand gamma est petit. Un discount plus grand veut dire que cela coûte beaucoup de rester dans un même endroit beaucoup de temps, donc c'est mieux de bouger vite vers la case avec reward 1 même avec une chance d'aller vers la case avec reward -1. Si le discount n'est pas si grand, c'est mieux de prendre plus de temps qu'avoir la chance d'aller vers la case de reward -1.

Pour une même valeur de gamma, on peut voir qu'un epsilon plus petit (modèle avec plus d'itérations), la récompense cumulée est plus grande qu'avec un epsilon plus grand, du coup on peut dire que la politique a été mieux apprise.

Maintenant on va essayer de résoudre les environnements plus difficiles avec ces deux méthodes. Dans ce cas, on va juste utiliser les paramètres $\gamma = 0.999$ et $\epsilon = 0.001$, qui ont donné les meilleurs résultats pour le jeu de nombre 0. Les résultats affichés sont la récompense moyenne et le nombre d'actions moyen après 1000 itérations.

Récompense moyenne/ nombre d'actions moyen

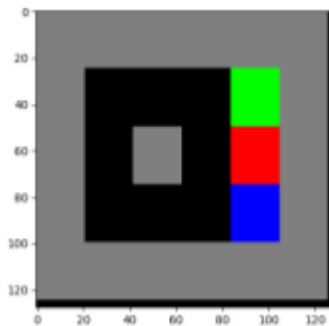
Jeu/méthode	Value Iteration	Police Iteration
0	0.98/20	0.98/20
1	1.97/24	1.97/24
2	1.96/42	1.85/18
3	0.99/6	0.99/6
4	-0.03/32	-0.03/32
5	1.94/59	1.94/59
6	1.94/61	1.92/80
7	2.67/75	2.13/168
8	0.93/71	0.92/76
9	env error	env error
10	0.95/43	0.95/43
Average	1,43/43	1,36/52

On peut voir que pour ce jeu de paramètres la méthode Value Iteration a eu un meilleur résultat en moyenne que la méthode Police Itération et a eu aussi un nombre moyen d'actions plus petit.

TME3

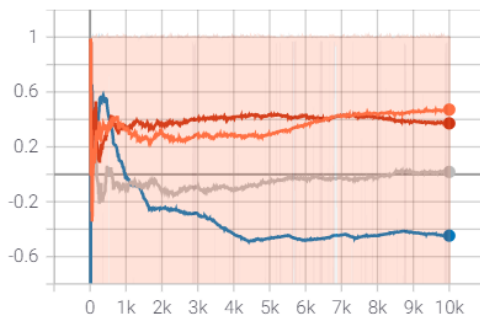
Dans ce TME nous allons étudier la méthode Q-Learning et ses autres versions et extensions SARSA, Dyna-Q et l'implémentation avec traces d'éligibilité. On va évaluer ces méthodes dans différents environnements gridworld. Comme je n'ai pas une GPU, Dyna-Q prend trop de temps donc nous avons utilisé $k=10$ au lieu de 100 pour que la méthode soit plus rapide.

Plan0:

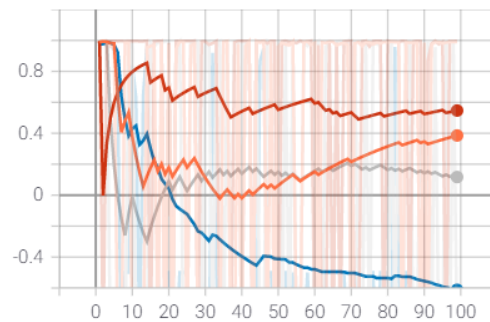


Pour le plan0 (ci-dessus), la méthode SARSA (rouge) a eu le meilleur résultat en test suivi par Q-Learning (orange). Dans ce plan, la méthode Dyna-Q (gris) n'a pas eu un bon résultat et les traces d'éligibilité ont eu des récompenses négatives en train et en test (bleu) comme on peut voir ci-dessous.

reward

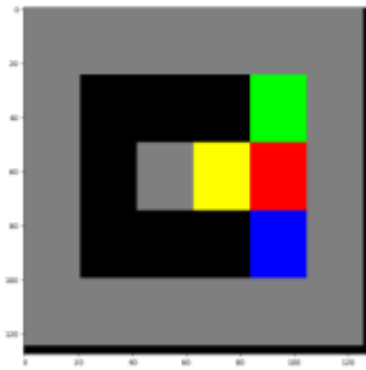


rewardTest



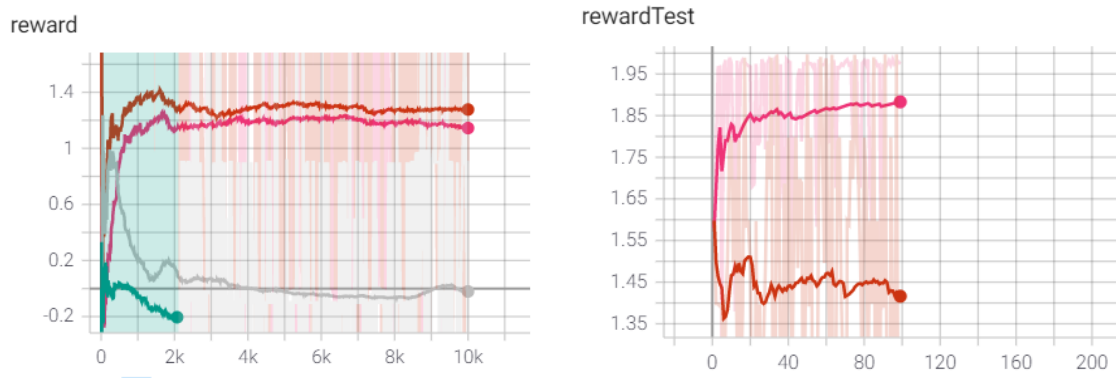
L'expérience a été faite avec le schéma de reward par défaut et les hyperparamètres epsilon initial = 0.1, avec decay de 0.999 à la fin de chaque trajectoire, un discount de 0.99 et un pas d'apprentissage de 0.1.

Plan1:



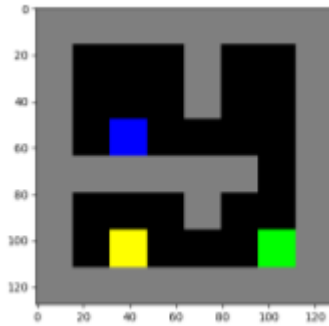
Le plan1 est plutôt le même que le plan0, la seule différence est une casse jaune à côté de la casse rouge.

Dans ce plan, pour les mêmes hyperparamètres du plan précédent, la méthode Q-Learning a eu le meilleur résultat suivi de Sarsa. Dyna-Q et traces d'éligibilité ont eu un mauvais reward et ont pris beaucoup plus temps. On peut voir dans les résultats affichés ci-dessous que la méthode SARSA (rouge) a eu un résultat mieux dans l'entraînement mais la méthode Q-Learning (rose) a eu un résultat beaucoup mieux dans le test.



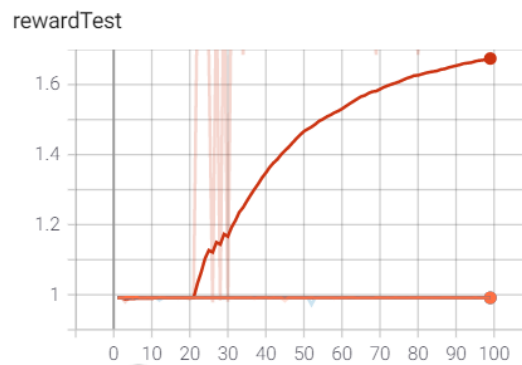
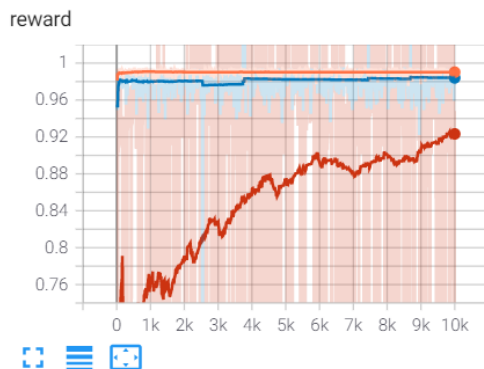
Quand on utilise les traces d'éligibilité dans ce plan, l'agent récupère la récompense jaune et quitte la zone où il y a des casses rouge et verte. Pour que l'agent se dirige vers la case verte, il faut augmenter le facteur de discount (diminuer gamma), mais dans ce cas il arrive la plupart des fois dans la case rouge. Pour ce problème, utiliser les traces d'éligibilité n'a pas aidé l'agent.

Plan2:

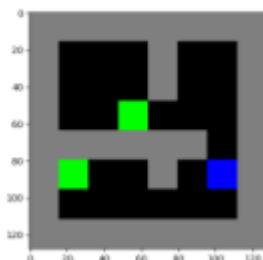


Le plan2 est un problème d'exploration. Notre but ici est de trouver une politique allant vers la case jaune puis retournant à la case verte au lieu de seulement atteindre la case verte (qui est optimale avec les paramètres de l'environnement initiaux). Pour trouver une telle politique optimale, il faut donc augmenter le taux d'exploration de nos méthodes. Q-Learning a été la seule méthode qui ai arrivée vers la politique optimale, les autres n'y sont pas arrivées. Dyna-Q, SARSA et Q-Learning avec traces d'éligibilité ont pris trop de temps pour tourner, nous ne les prenons pas en compte.

Ci-dessous, nous avons le résultat de Q-Learning avec mode d'exploration = 0.1 (orange), 0.5 (bleu) et 0.8 (rouge). On peut voir très bien qu'avec un taux plus grand d'exploration, l'algorithme arrive vers la politique optimale.

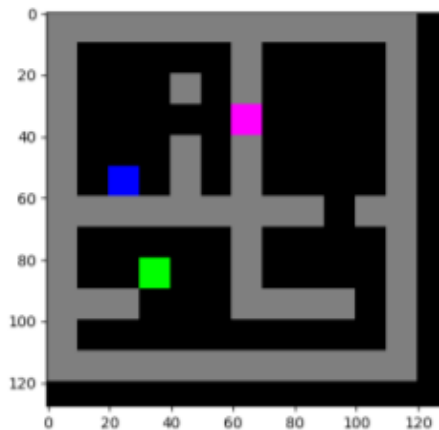


Plan3:



Le plan3 est très simple et toutes les méthodes l'ont bien géré. Nous avons décidé de ne pas expérimenter le plan plus en détail.

Plan4:

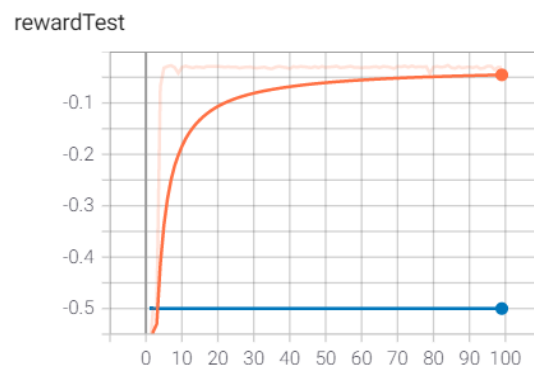
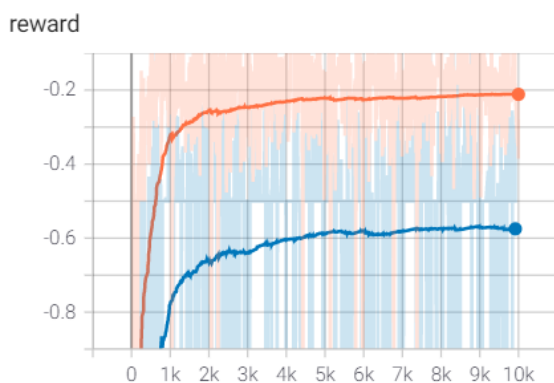


Le plan4 est aussi un plan où il est nécessaire d'explorer la carte mais il y a aussi le problème de la case rose, qui fait que l'agent gagne une récompense négative. Une bonne stratégie est d'avoir plus d'itérations dans un même épisode d'entraînement ($\text{maxLengthTrain} = 500$) et un grand facteur d'exploration ($\text{explo} = 0.8$) pour que l'agent puisse explorer la carte même avec la récompense négative au début. Finalement, l'algorithme préfère la récompense négative -1 puis la récompense positive +1 et finir vite plutôt que de rester toujours dans le même endroit en perdant 0.01 à chaque itération.

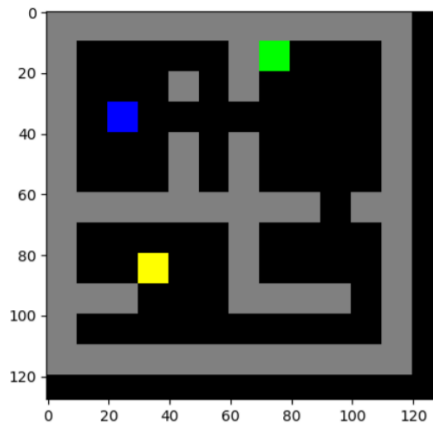
Dans ce problème on peut aussi avoir une autre stratégie qui est d'augmenter la récompense de la case verte. Cela aide à la convergence car cette grande récompense va rétro-propager les valeurs plus vite, mais ça c'était pas nécessaire, juste augmenter le paramètre d'exploration suffit pour converger.

Pour interpréter les résultats, l'objectif est d'atteindre un reward plus proche de 0 (ça veut dire que l'agent est arrivé plus vite vers le résultat). Si on a un reward de -0.5, l'agent n'est ni arrivé à la case verte ni passé par la case rose. Si le reward est inférieur à -0.5, l'agent est passé par la case rose mais n'est pas arrivé à la case verte.

Ci-dessous on peut voir que le Q-Learning (orange) arrive dans la case verte en 50 steps à 10k épisodes. En bleu nous avons la méthode SARSA qui n'arrive pas à la politique optimale, le reward en test reste toujours proche de -0.5.

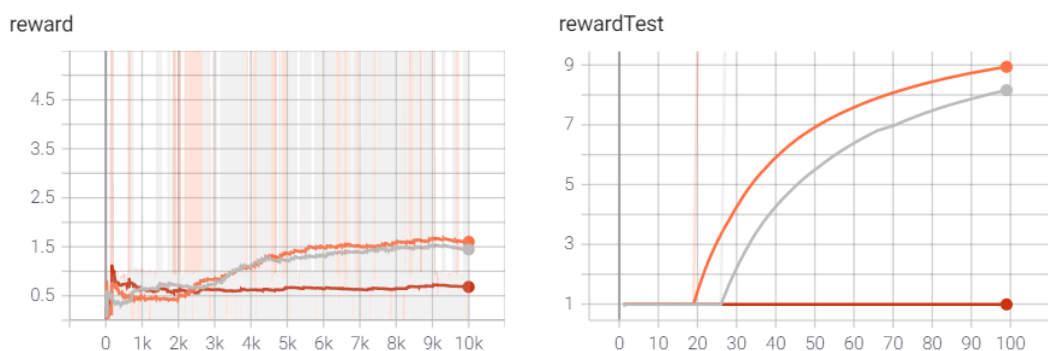


Plan5:

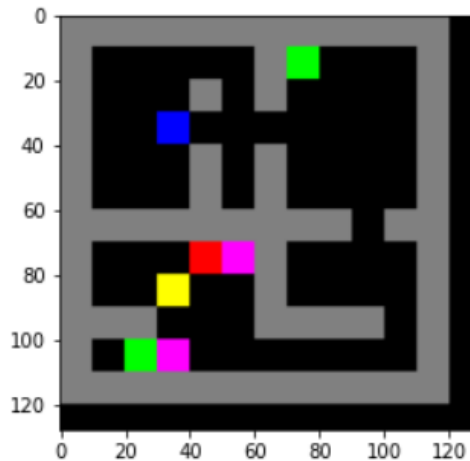


Le plan5 est un problème d'exploration comme le plan2 mais la case jaune est beaucoup plus difficile à atteindre, car la case jaune est très loin de la case verte, et la case verte est très proche du début. Pour atteindre la case jaune, nous avons choisi un taux d'exploration vraiment grand ($\text{explo} = 0.99$) et nous avons aussi augmenté la récompense de la case jaune à 10 au lieu de 1 (car sinon l'atteindre n'a pas d'intérêt à cause de la longueur du chemin nécessaire pour l'atteindre). Avec ça l'agent bouge aléatoirement dans le plan pour trouver la case jaune, quand il l'a trouvée, avec une très grande récompense, il commence à y aller. Le problème de cette approche est de retourner vers la case verte. Pour le résoudre, nous avons augmenté la décroissance du paramètre d'exploration ($\text{decay} = 0.95$ au lieu de 0.999).

Ci-dessous on peut voir les résultats de Q-Learning (en gris), Q-Learning avec une plus grande décroissance (en orange), et SARSA (en rouge). On peut voir que SARSA n'arrive pas vers la politique optimale. Dyna-Q et les traces d'éligibilité sont très lents, nous avons décidé de ne pas les prendre en compte ici.



Plan6:

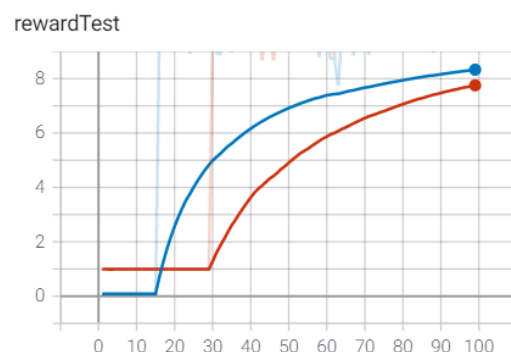
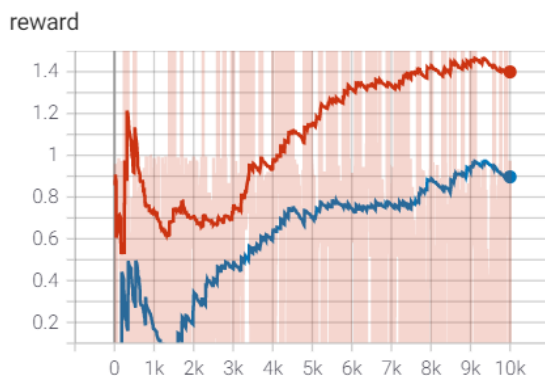


Le plan6 est une variante du plan5 où atteindre la case jaune est plus difficile, car on a les cases rouges et roses proches de la case jaune. Dans ce cas, nous avons diminué la récompense négative de ces cases et avec ça, l'agent est arrivé vers une politique optimale passant par la case jaune. Cependant, parfois l'agent allait vers la case verte en haut directement, donc nous avons diminué la récompense des cases vertes. Ainsi l'agent a convergé plus vite vers la politique optimale en passant par la case jaune.

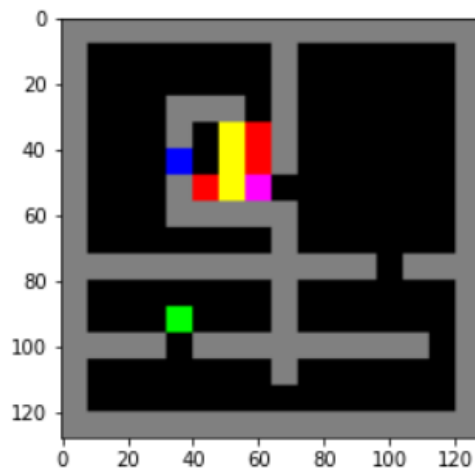
Ci-dessous on peut voir les résultats, le Q-Learning avec récompense = 1 pour les cases vertes (courbes en rouge) et avec récompense = 0.1 (courbes en bleu).

Pour comparer les rewards, il faut considérer que la courbe rouge a un avantage qui est la case verte qui vaut 0.9 plus qu'avec la courbe bleu, on observe tout de même une convergence plus rapide et on obtient un reward plus important en test après 10k itérations.

La méthode SARSA n'a pas réussi à atteindre la case jaune, ce qui est attendu car elle n'avait pas réussi dans le plan5 qui est proche de ce plan. Ici aussi Dyna-Q et l'utilisation de traces d'éligibilités prennent trop de temps et ne sont pas pris en compte.



Plan7:

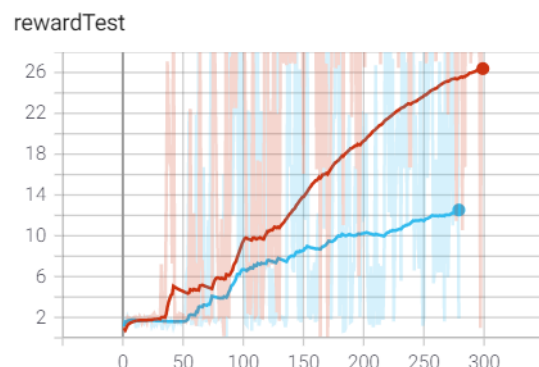
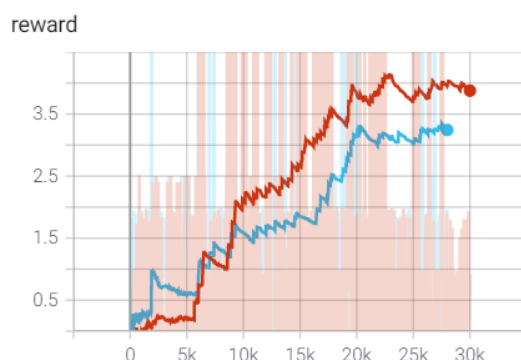


Dans le plan7, il est encore plus dur d'atteindre la case verte. Dans ce plan, la case verte est très loin, il y a 3 cases rouges que l'agent peut rencontrer lors de son passage et aussi une case rose dans le seul endroit où l'agent peut passer pour atteindre la case verte. Notre objectif ici est de faire en sorte que l'agent récupère les cases jaunes puis aille vers la case verte.

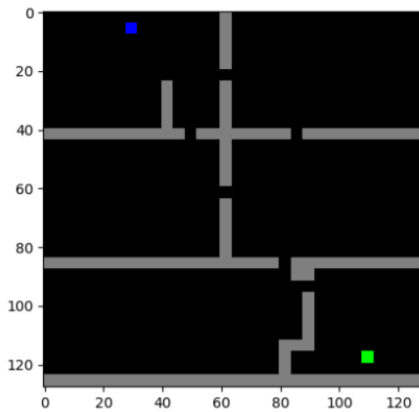
Nous nous sommes ici concentré sur l'algorithme Q-Learning et SARSA. Pour le résoudre, nous avons mis un facteur exploration à 0.8, on ne peut pas utiliser une valeur trop forte car cela augmenterait la probabilité de mourir dans les cases rouges et donc réduire les chances de découvrir la case verte. Nous augmentons le nombre de steps possibles (maxLengthTrain = 1000) et le nombre d'épisodes d'entraînement à 30k. Nous baissons à -0.1 la récompense négative des cases roses et nous augmentons aussi la récompense de case verte à +50.

Le but de ces paramètres est de faire en sorte que l'agent arrive bien dans la case verte sans trop mourir dans les cases rouges au début de l'apprentissage. Après atteinte de la case verte, sa grande récompense va rétro propager dans la table Q, augmenter le reward de cette case permet d'augmenter plus rapidement l'intérêt de l'agent d'y arriver.

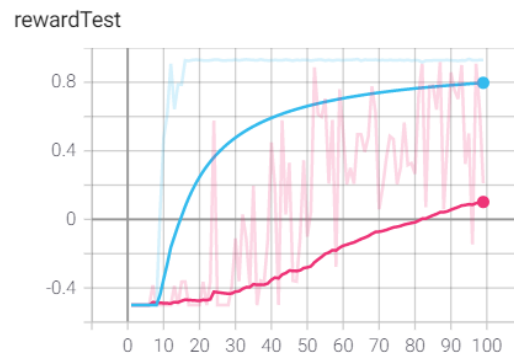
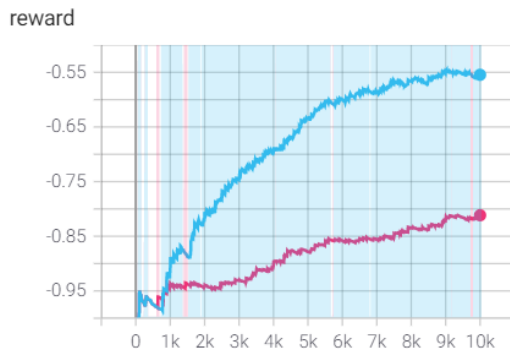
Ci-dessous on peut voir le résultat du jeu de paramètres expliqués au dessus pour Q-Learning (rouge) et SARSA (bleu). On peut voir que Q-Learning converge beaucoup plus vite que SARSA.



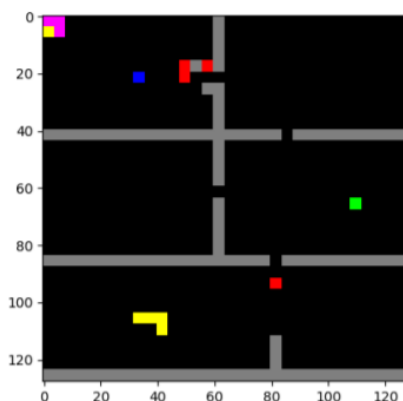
Plan8:



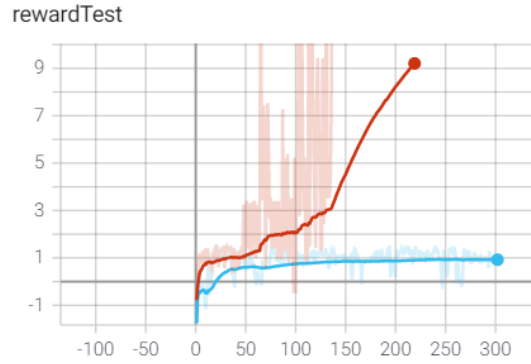
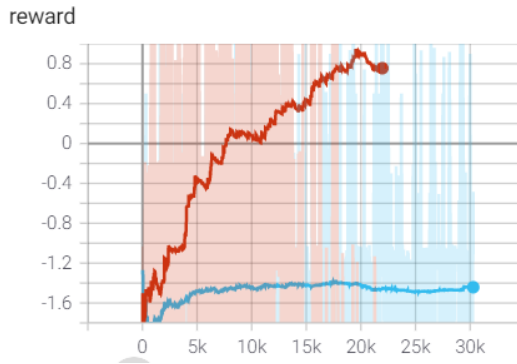
Le plan8 est un plan très grand, nécessitant beaucoup d'exploration, arrivé à la case verte est lent mais pas difficile. Il faut juste augmenter le paramètre d'exploration, nous utilisons 0.9 et aussi le nombre d'itérations (steps possibles par épisode = 1000). Avec cette configuration, Q-Learning (bleu) arrive vite vers la politique optimale et SARSA (rose) semble arriver mais beaucoup plus lentement.



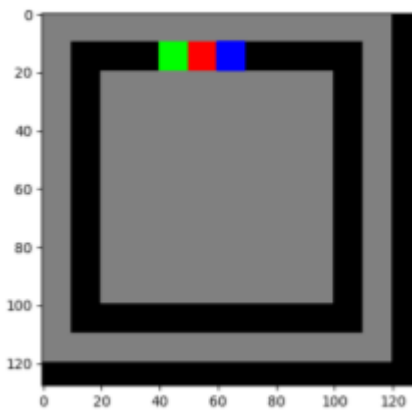
Plan9:



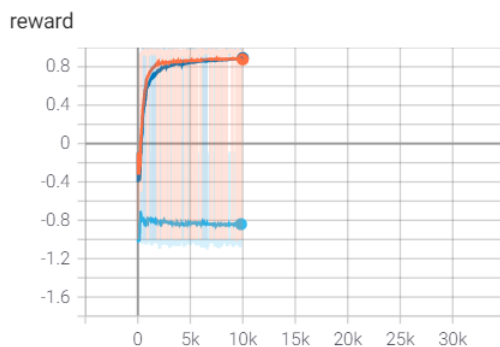
Le plan9 est le plan le plus grand. Nous n'avons pas réussi à faire que l'agent prenne les cases jaunes en bas. Mais nous avons réussi à aller vers la case jaune en haut à gauche puis aller vers la case verte avec Q-Learning. Pour cela, nous avons mis comme paramètre $\text{explor}=0.8$, 20k épisodes et maximum de 1000 itérations par épisode (rouge). Nous avons essayé d'augmenter le paramètre d'exploration mais avec 30k d'itérations l'agent n'a pas convergé (bleu).



Plan10:



Pour le plan10, Q-Learning (orange) et SARSA (bleu foncé) ont réussi à trouver la politique optimale qui est juste d'aller vers la case verte avec les paramètres standards (proposés au début). Par contre, Dyna-Q (bleu clair) et les deux méthodes avec traces d'éligibilité n'ont pas réussi.



Conclusion:

Dans ce TME3 nous avons vu les méthodes Q-Learning, SARSA, Dyna-Q avec et sans trace d'éligibilité. Nous avons vu qu'ajouter les traces d'éligibilité augmente beaucoup le temps de chaque épisode et aussi le temps de convergence. Entre Q-Learning et SARSA, pour les exemples étudiés, Q-Learning a eu un meilleur résultat, surtout si c'est un plan qui demande beaucoup d'exploration.

On peut retenir aussi que le jeu de paramètres a un rôle très important, surtout le paramètre d'exploration et le nombre d'itérations (steps possibles) de chaque époque. Parfois changer les récompenses peut beaucoup aider dans la convergence de l'agent vers la politique optimale.

TME4

Dans ce TME, nous testerons différentes versions de DQN sur les environnements Cartpole et LunarLander. Nous testons tout d'abord les différents effets de l'utilisation d'un replay buffer, de l'ajout d'une priorisation sur celui-ci et l'utilisation d'un réseau cible et leurs différentes combinaisons.

Puis, nous testerons aussi l'effet de la recherche aléatoire, de la fréquence d'optimisation du réseau cible et du discount sur l'apprentissage.

Protocole expérimentale

Le nombre de pas de temps maximum sera de 200 pour CartPole et 500 pour LunarLander. Le discount utilisé est de 0.99, avec un critère d'exploration de 0.1 (choix aléatoire de l'action avec probabilité 0.1).

Le réseau est constitué d'une couche cachée de 128 neurones avec activation tanh, le learning rate appliqué est de 0.001 (avec un optimiseur Adam) et le réseau est mis à jour à chaque action.

Le buffer utilisé à une capacité de 1000 et nous utilisons des batchs de taille 32 à chaque pas d'apprentissage.

Le réseau cible utilisé est mis à jour tous les 100 actions réalisées pour CartPole et 50 pour LunarLander.

Pour chaque environnement, nous obtiendrons un reward final obtenu en réalisant une moyenne des rewards sur 100 épisodes sur la meilleure sauvegarde (réalisée tous les 100 épisodes sur 300 épisodes d'apprentissage pour CartPole et tous les 100 épisodes avec 500 épisodes d'apprentissage pour LunarLander), nous indiquerons aussi le nombre d'épisodes d'apprentissage de la sauvegarde ayant atteint ce score le plus tôt.

Ce score final nous permettra de comparer les performances avec les différents paramètres. Pour les expériences concernant le critère d'exploration, la mise à jour du réseau cible et la suppression du discount, nous analyserons la loss et les rewards obtenue au cours de l'apprentissage. Comme dans toutes les courbes des TMEs suivants, l'axe x utilisé pour les figures est le nombre d'épisodes (les rewards correspondent donc à la somme des rewards de l'épisode).

Résultats

résultats finaux:

Buffer	oui	non	oui	oui	non
Buffer priorisé	oui	non	non	oui	non
Réseau cible	oui	oui	oui	non	non
Score final sur CartPole (avec écart type)	200+/-0	13.17+/-1.65	31.18+/-5.43	9.99+/-1.22	9.99+/-1.38

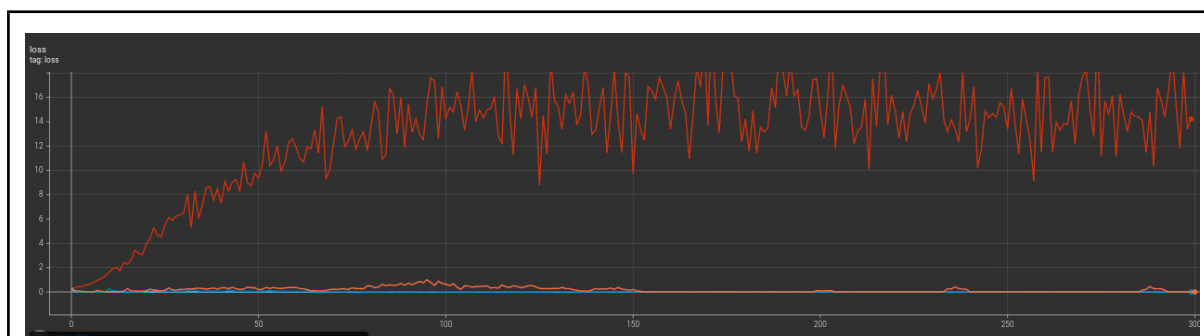
Nombre d'épisodes d'entraînement utilisés sur CartPole	200	300	200	300	300
Score final sur LunarLander (avec écart type)	153.02+/-73.64	-25.53+/-98.30	71.59+/-101.43	121.25+/-94.40	-26.10+/-64.82
Nombre d'épisodes d'entraînement utilisés sur LunarLander	500	500	500	500	400

On observe une perte importante des rewards par épisode en enlevant n'importe quel paramètre pour CartPole. Sans buffer ou réseau cible, le réseau ne semble pas apprendre correctement (avec un rewards de 10 en moyenne, ce qui est très proche du score avant apprentissage). Dans le cas de la suppression de la priorisation du buffer, on observe une perte de performance (de 170 en rewards) mais le réseau arrive à apprendre mais beaucoup plus lentement.

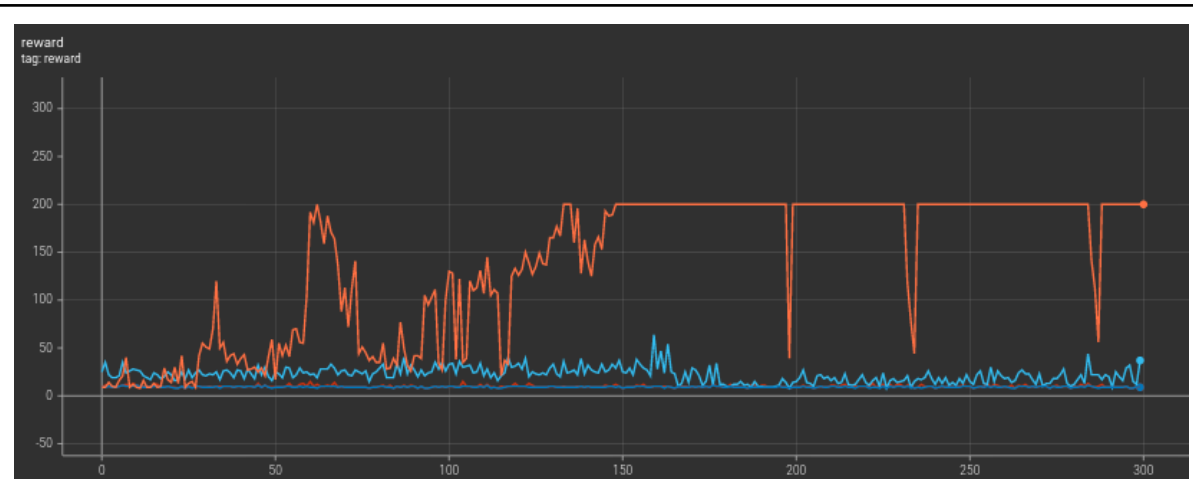
Dans le cas de LunarLander, on observe bien un apprentissage pour toutes les variantes. Supprimer le buffer réduit grandement les rewards. Supprimer le réseau cible à un effet beaucoup moins fort qu'avec CartPole et est moins impactant que l'utilisation d'un buffer non priorisé (on observe une perte de 30 rewards en moyenne pour 80 avec suppression de la priorisation du buffer).

La variante ayant le meilleurs score est cette fois-ci la suppression du réseau cible suivi par la suppression de la priorisation du buffer.

Dans les expériences suivantes , nous utilisons un buffer priorisé avec réseau cible.



Evolution des loss au cours de l'apprentissage Pour CartPole



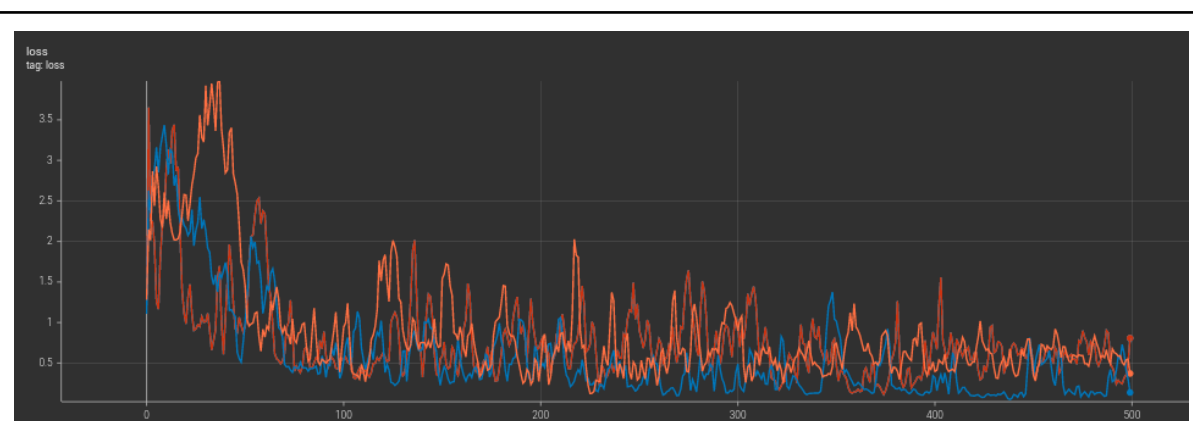
Evolution des rewards au cours de l'apprentissage pour CartPole

Voici l'évolution des rewards et de la loss en apprentissage en fonction du nombre d'épisodes d'apprentissage pour les expériences de suppression de l'exploration, de modification de mise à jour du réseau et de suppression du discount.

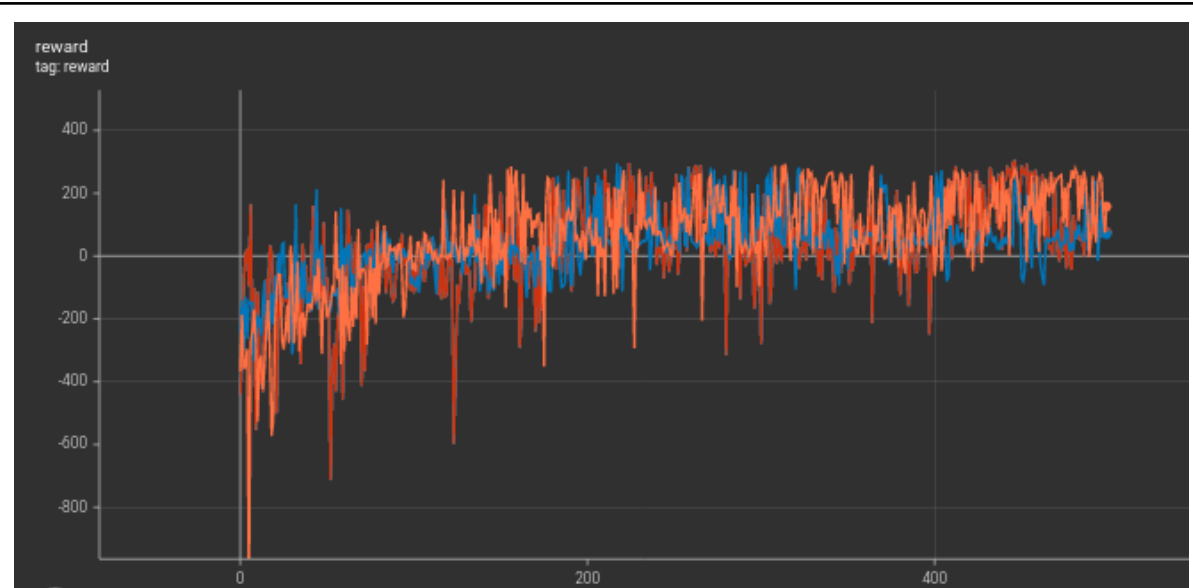
Dans ces images, les courbes oranges correspondent à la performance de l'algorithme de base (avec buffer priorisé et réseau cible). Les courbes bleues correspondent à la suppression de l'exploration. Les courbes rouges, la mise à jour du réseau cible toutes les 10 actions et les courbes bleues claires, l'utilisation d'un discount de 0.

Il est intéressant de remarquer que modifier le gamma ou le critère d'exploration permet toujours de converger mais avec de très mauvais rewards. Modifier la cible pour ne pas prendre en compte les actions futures semble rendre les cibles moins pertinentes. Ceci montre aussi l'importance de l'exploration. Dans le cas du changement de la fréquence de la mise à jour du réseau cible, on observe que l'algorithme n'arrive plus à converger, il semble que les cibles soient modifiées trop rapidement et l'algorithme n'arrive jamais à les atteindre.

Voici, les courbes obtenues pour Lunar Lander pour les mêmes expériences:



Evolution des loss au cours de l'apprentissage Pour LunarLander



Evolution des rewards au cours de l'apprentissage pour LunarLander

Voici les courbes obtenues pour l'environnement LunarLander. Les résultats sont très proches pour les différents réseaux qui semblent réduire légèrement les rewards mais n'ont pas un impact visible sur ces courbes.

La baisse de performance est plus visible sur les scores finaux:

Scores finaux/ Paramètres modifiés	Score final sur CartPole	Nombre d'épisodes en entraînement du meilleur score sur CartPole	Score final sur LunarLander	Nombre d'épisodes en entraînement du meilleur score sur LunarLander
Aucun	200+/-0	200	153.02+/-73.64	500
critère d'exploration nul (pour 0.1 précédemment)	9.89+/-1.24	300	117.34+/-99.76	400
mise à jour du réseau cible tous les 10 actions (au lieu de 100 et 50)	9.93+/-1.38	200	100.02+/-28.53	400
discount à 0 (pour 0.99 précédemment)	43.38+/-16.71	300	100.80+/-31.59	400

On observe tout de même une perte de 50 en moyenne sur le score final environ pour chaque modification de paramètre. On observe tout de même une réduction plus faible avec critère d'exploration nul mais avec une variance beaucoup plus forte.

TME5

Dans ce TME nous testerons l'algorithme batch actor-critique sur les mêmes environnements que précédemment. Nous testerons les versions TD(0), TD(1) et TD(λ) de l'avantage. Nous testerons aussi l'utilisation d'un terme d'entropie en plus dans la loss.

Protocole expérimentale

Les réseaux sont mis à jour à chaque épisode avec 1000 actions réalisées avant tout apprentissage.

Dans le cas du calcul de l'avantage, le λ utilisé pour TD(λ) est de 0.99.

De même, le discount utilisé est de 0.99 dans toutes les expériences.

Pour la mise à jour de V, nous utilisons un TD(0) avec un discount de 0.99.

Les réseaux de neurones utilisés sont composés de deux couches cachées de 30 neurones avec comme fonction d'activation tanh, et on ajoute un softmax à la fin de l'acteur, celui-ci ayant en sortie des probabilités. Le learning rate des deux réseaux est de 0.001.

Pour l'ajout du terme d'entropie, nous utilisons un terme d'entropie multiplié par 0.2 (augmenter ce ratio ne semble pas améliorer les résultats).

Les paramètres sont les mêmes pour les 2 environnements.

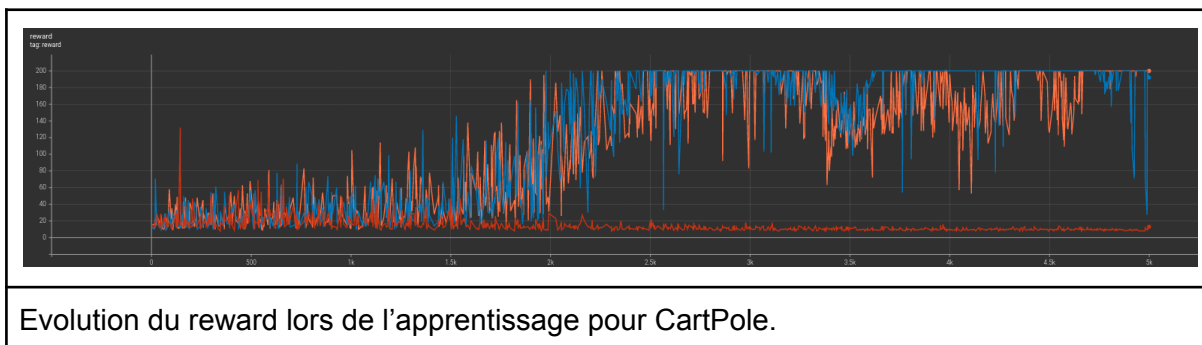
Dans nos différentes expériences, nous observerons ici:

- le divergence de Kullback-Leibler de la politique sur les batchs appris avant et après un pas d'apprentissage permettant d'observer la stabilité de la politique durant l'apprentissage.
- Le reward au cours de l'apprentissage indiquant le temps pris pour obtenir les meilleurs résultats de façon plus précise que le score final.
- Les loss des acteurs et critiques pour observer la convergence.

Les paramètres sur l'environnement et l'obtention du score final sont les mêmes que dans le TME précédent à la différence que nous réalisons 5000 et 7000 épisodes d'apprentissage respectivement pour les environnements CartPole et LunarLander.

Résultats

Tout d'abord commençons par comparer TD(0), TD(1) et TD(λ) pour le calcul d'avantage.

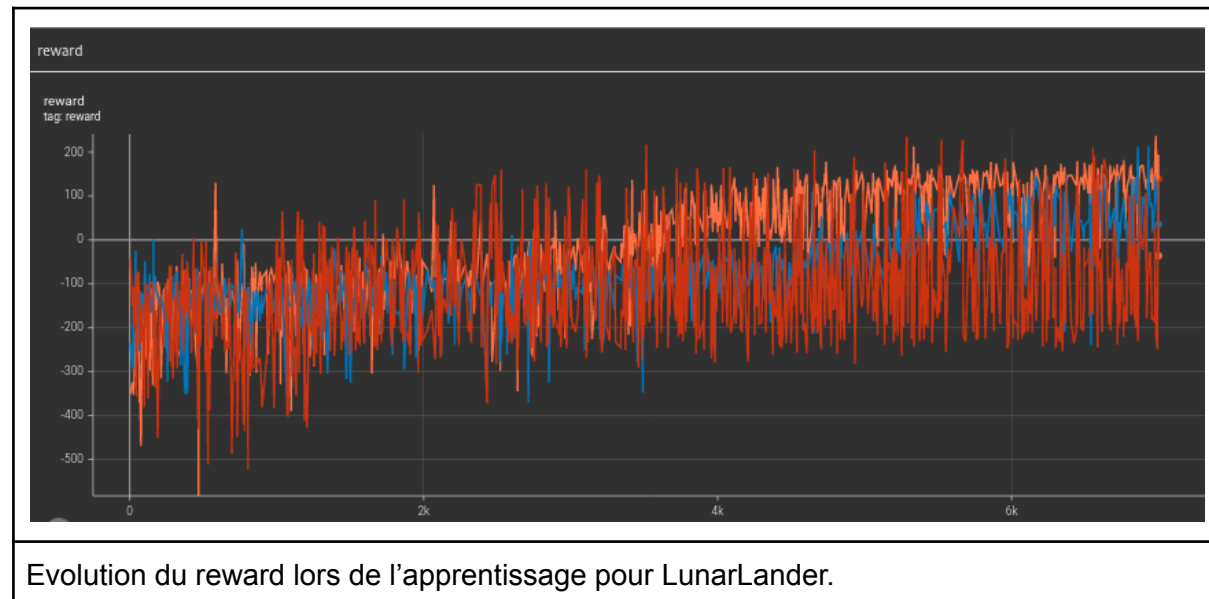


Dans cette image et dans le reste des images de cette partie, la courbe rouge correspond à l'algorithme avec calcul de l'avantage avec TD(0), la courbe orange avec TD(λ), la courbe bleue, TD(1).

Ici l'image correspond à l'évolution du reward au cours de l'apprentissage avec l'environnement CartPole.

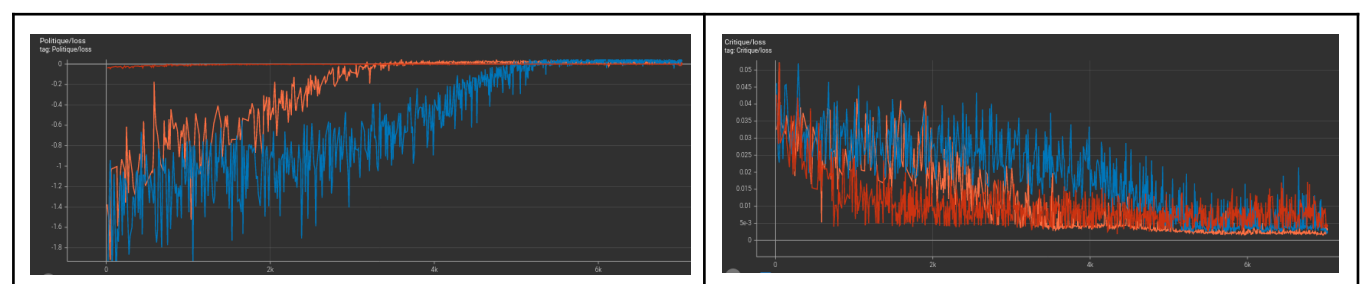
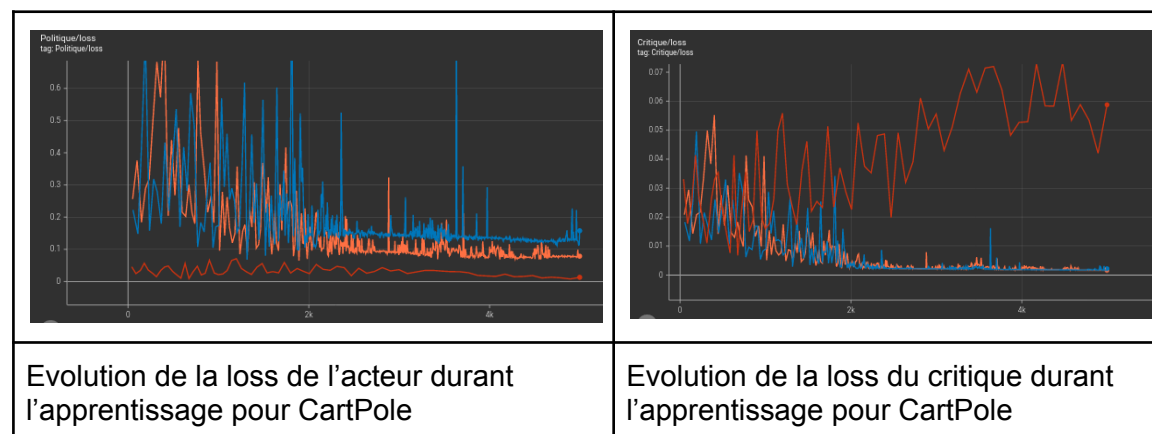
On observe que TD(1) et TD(λ) arrivent au score maximum environ au même moment, ce qui semble cohérent car notre λ est proche de 1. Les rewards finaux (voir en fin de partie)

indiquent tout de même une forte variance des rewards même après avoir atteint le score maximum. Pour nos paramètres, TD(0), lui, n'arrive pas à faire augmenter le reward. Continuons avec l'analyse des rewards en apprentissage avec l'environnement LunarLander:



Dans l'environnement LunarLander, TD(λ) semble converger plus vite que TD(1). Dans le cas de TD(0) le reward semble augmenter mais reste très faible par rapport aux résultats des autres algorithmes, de plus on observe une variance plus importante de ceux-ci (ce qui se confirme avec les rewards finaux, avec un écart-type doublé par rapport aux autres expériences).

Pour étudier plus en profondeur la convergence des algorithmes, analysons maintenant les loss:



Evolution de la loss de l'acteur durant l'apprentissage pour LunarLander

Evolution de la loss du critique durant l'apprentissage pour LunarLander

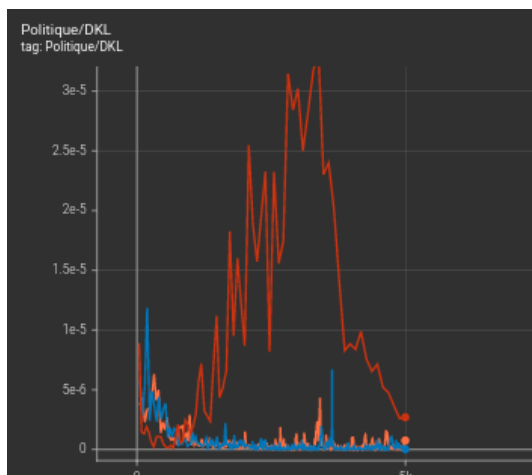
Ici les résultats sont très similaires pour les 2 environnements.

Pour TD(λ) et TD(1), nous observons que la loss du critique et de l'acteur converge vers 0 lors de l'apprentissage avec une amélioration plus rapide et est plus stable pour TD(λ) dans le cas de LunarLander, ce qui confirme le fait que l'algorithme converge plus vite.

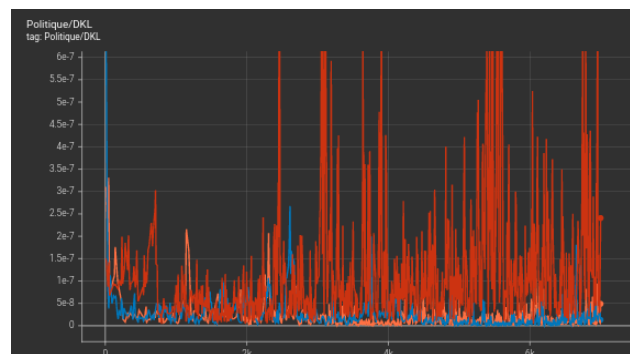
Dans le cas de TD(0) et Cartpole, la loss de l'acteur est déjà très proche de 0 et stagne alors que le critique à une loss qui augmente. Cela semble montrer que le critique n'arrive pas à suivre l'acteur durant l'apprentissage et confirme la non convergence observée plus tôt dans les rewards.

Dans le cas de LunarLander, la loss du critique diminue bien et même plus rapidement que les 2 autres algorithmes en début d'apprentissage mais la convergence réduit rapidement, ce qui explique que les rewards arrivent bien à augmenter cette fois-ci.

Pour finir, voici les divergences de Kullback-Leibler obtenues:

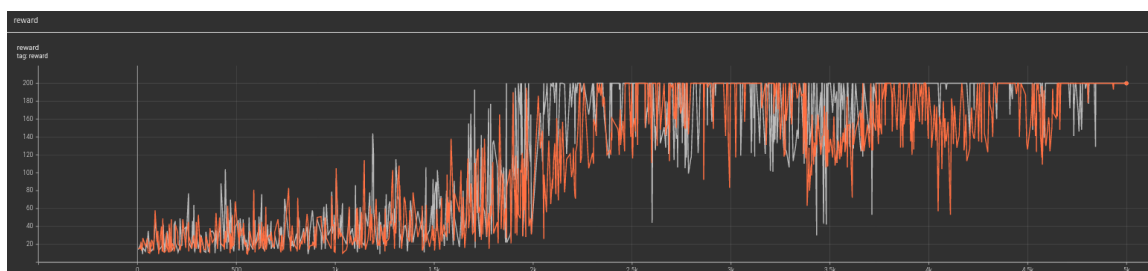


Divergence de Kullback-Leibler de la politique avant et après une phase d'entraînement pour CartPole

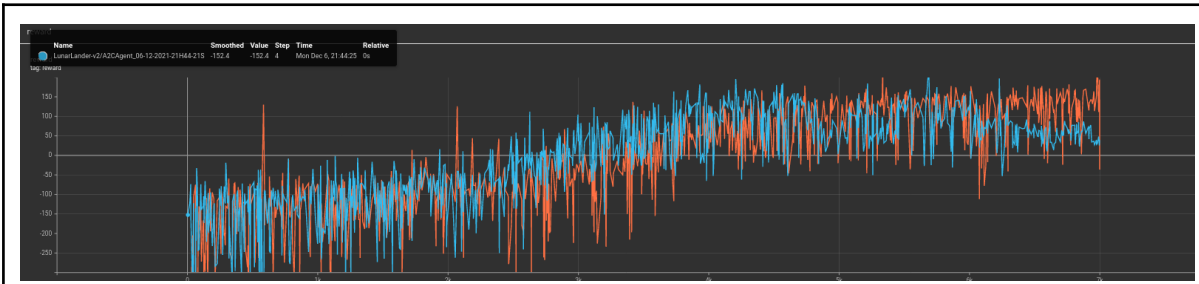


Divergence de Kullback-Leibler de la politique avant et après une phase d'entraînement pour LunarLander

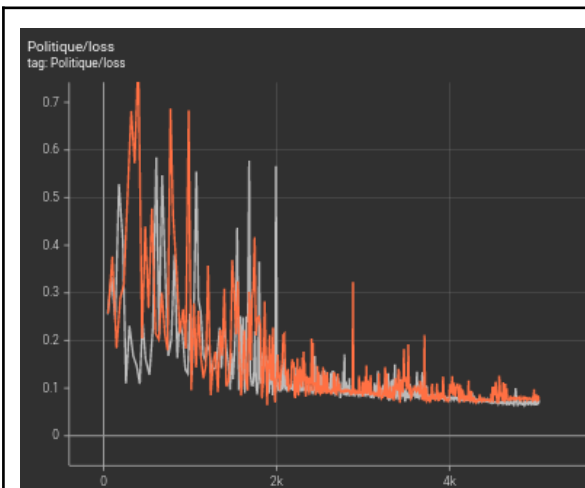
On observe que la politique se stabilise pour TD(1) et TD(λ) lors de l'apprentissage. Pour TD(0), on observe une variation très forte de la politique, expliquant la difficulté à converger. Nous avons ensuite ajouté un terme d'entropie à la loss à maximiser pour TD(λ) pour essayer d'améliorer les résultats finaux.



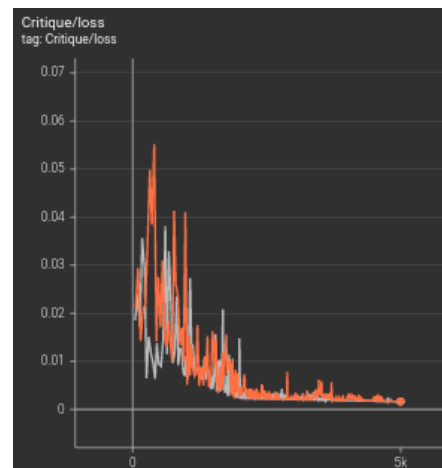
Évolution du reward lors de l'apprentissage pour CartPole.



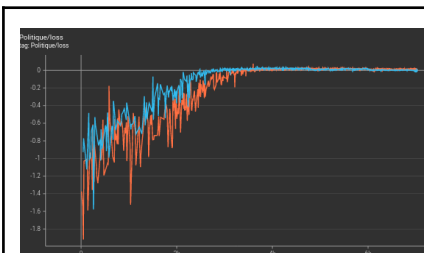
Évolution du reward lors de l'apprentissage pour LunarLander.



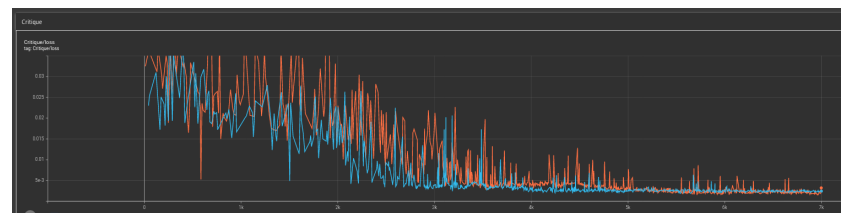
Evolution de la loss de l'acteur durant l'apprentissage pour CartPole



Evolution de la loss du critique durant l'apprentissage pour CartPole



Evolution de la loss de l'acteur durant l'apprentissage pour LunarLander



Evolution de la loss du critique durant l'apprentissage pour LunarLander

Dans ces images nous affichons les loss des réseaux de neurones et les rewards durant apprentissage pour TD(λ) sans (en orange) et avec terme d'entropie (en gris et bleu clair). Ici l'amélioration est peu visible dans le cas de CartPole, on observe tout de même une amélioration en variance de score que l'on voit surtout dans le score final (voir ci-dessous), ce qui pourrait être dû à une convergence plus rapide.

Dans le cas de LunarLander, l'accélération de la convergence est visible dans la loss de l'acteur qui prend en compte l'entropie et dans les rewards qui convergent plus vite vers le

résultat final obtenu sans entropie. Les résultats finaux sont sinon très proches des résultats précédents. On observe tout de même une baisse de performance avec l'entropie sur LunarLander, surement dû à une plus forte exploration.

Scores finaux:

Scores finaux/ Algorithmes utilisés pour le calcul de l'avantage	Score final sur CartPole	Nombre d'épisodes en entraînement du meilleur score sur CartPole	Score final sur LunarLander	Nombre d'épisodes en entraînement du meilleur score sur LunarLander
TD(0)	22+/-11.06	0	-57.97+/-129.37	7000
TD(1)	199.9+/-0.99	3000	68.04+/-62.44	7000
TD(λ)	198.79+/-9.52	5000	122.81+/-66.34	7000
TD(λ) avec entropie	200+/-0	5000	89.74+/-60.38	4000

TME6

Dans cette partie nous comparerons 3 versions de PPO: une version de PPO sans KL ni clipped objective, une version KL et une version clipped objective.

Nous testerons aussi les versions avec KL inversé des deux premières versions et finalement l'ajout d'un terme entropie pour chaque version.

Protocole expérimentale

Pour CartPole, nous utilisons les paramètres suivants:

- un discount et lambda de 0.99 pour TD(λ)
- nous réalisons K=5 pas de gradient à chaque fois.
- 2 réseaux à 2 couches cachés de 30 neurones chacun, avec fonctions d'activation tanh. Pour l'acteur, le réseau fini avec un softmax (celui-ci calculant une probabilité). Le learning rate des 2 réseaux est de 0.001 (avec un optimiseur Adam)
- Des batches de taille 1000
- Un facteur d'exploration epsilon de 0.1
- Un seuil δ de 0.001 (pour le PPO KL)

Pour LunarLander, nous modifions quelques paramètres:

- K à 10
- epsilon à 0.3

Le facteur d'entropie est fixé à 0.005 lorsqu'il est utilisé (l'augmenter ne permettant pas d'améliorer les scores).

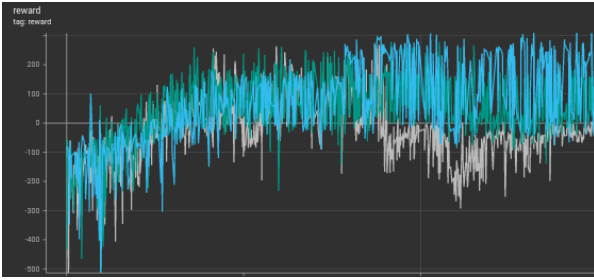
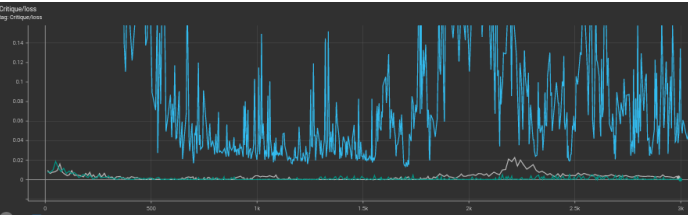
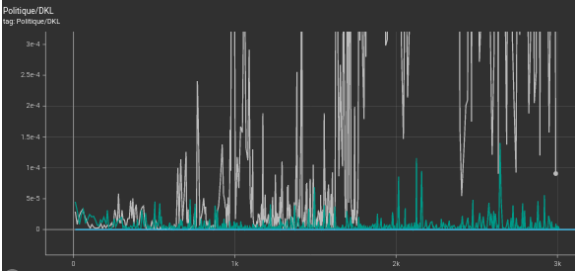
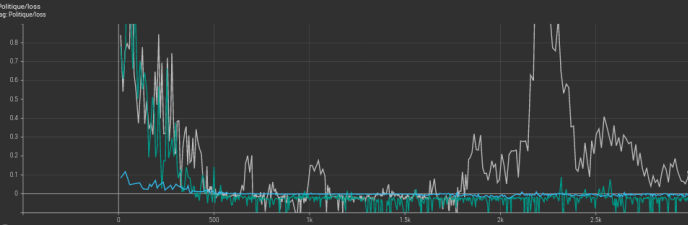
Nous réalisons 2000 épisodes d'apprentissage pour l'environnement CartPole avec une sauvegarde tous les 200 épisodes et 3000 épisodes d'apprentissage pour l'environnement Lunar avec une sauvegarde tous les 300 épisodes. Le calcul des rewards finaux est le même que précédemment avec les sauvegardes réalisées lors de l'apprentissage.

Nous observerons les rewards, la divergence de Kullback-Leibler avant et après apprentissage et les loss au cours de l'apprentissage pour certaines variantes.

Résultats

Score finaux	PPO sans DKL ni clipped objective				PPO avec DKL				PPO avec clipped objective	
Entropie	oui	non	oui	non	oui	non	oui	non	oui	non
Reversed DKL	oui	oui	non	non	oui	oui	non	non		
Score final sur CartPole (avec écart type)	174.19 +/- 22.64	200 +/- 0.0	199.03 +/- 5.72	190.45 +/- 9.63	133.55 +/- 47	200 +/- 0.0	164 +/- 35.39	200 +/- 0.0	103 +/- 47.54	199.45 +/- 3.90
Nombre d'épisodes d'entraînement utilisés sur CartPole	1000	1200	1600	800	1800	1200	2000	1600	1600	2000
Score final sur LunarLander (avec écart type)	-136.67 +/- 37.65	137.46 +/- 64.26	-128.92 +/- 18.34	134.33 +/- 25.05	-175.84 +/- 105.14	116.02 +/- 94.10	-81.50 +/- 28.59	144.53 +/- 57.22	-69.99 +/- 30.71	189.57 +/- 100.95
Nombre d'épisodes d'entraînement utilisés sur LunarLander	1500	1200	2700	1500	0	1200	1500	1800	3000	3000

On observe que chaque algorithme arrive à converger correctement pour chaque problème mais les scores finaux ne sont pas les mêmes pour les différents algorithmes (sauf pour CartPole où les résultats sont parfois très proches, sûrement car il s'agit d'un problème très simple). Pour en apprendre d'avantage, nous observons les loss et rewards au cours de l'apprentissage (sans reversed DKL ou Entropie) sur LunarLander:

	
<p>Evolution des rewards au cours de l'apprentissage</p>	<p>Evolution de la loss du critique au cours de l'apprentissage</p>
	
<p>Evolution de la divergence de Kullback-Leibler avant et après apprentissage</p>	<p>Evolution de la loss de l'acteur au cours de l'apprentissage</p>

On observe que l'algorithme clipped (en bleu) a une loss d'acteur beaucoup plus stable que les autres algorithmes mais beaucoup moins stable pour le critique.

On observe aussi que l'algorithme basique (en blanc) à une loss d'acteur variant beaucoup plus par rapport aux deux autres.

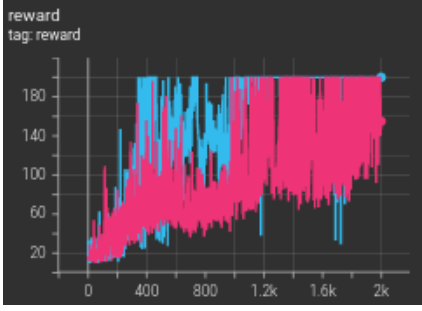
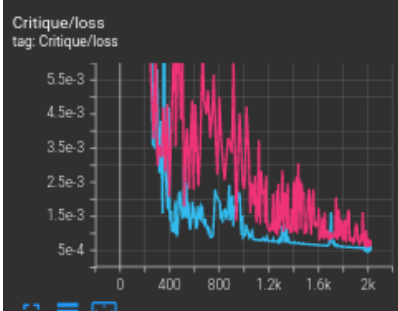

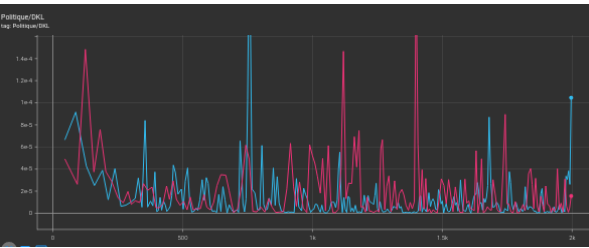
L'algorithme basique et celui utilisant DKL (en vert) semble avoir une loss de critique n'évoluant quasi pas.

Toutes ces instabilités sur les différentes loss sont sûrement dues au fait que nous essayons de peu modifier la politique au lieu de faire des petites modifications de paramètres.

Il semble donc plus intéressant d'observer la divergence de Kullback-Leibler (DKL), correspondant aux différences de politique des batchs avant et après une boucle d'apprentissage.

On observe que l'algorithme de base est très instable et que l'utilisation d'un clipped objective est plus stable qu'avec un DKL adaptatif. Cela semble correspondre aux différences de rewards observées, l'algorithme basique semblant converger vers des valeurs plus faibles que l'algorithme avec DKL adaptatif et celui-ci semblant converger vers des valeurs plus faibles que l'algorithme avec clipped objective.

Une première expérience que l'on peut réaliser est l'ajout d'un terme d'entropie dans la loss dont voici les courbes sur CartPole:

	
<p>Evolution des rewards au cours de l'apprentissage</p>	<p>Evolution de la loss du critique au cours de l'apprentissage</p>
	
<p>Evolution de la loss de l'acteur au cours de l'apprentissage</p>	<p>Evolution de la divergence de Kullback-Leibler avant et après apprentissage</p>

On observe les loss des deux réseaux et les rewards au cours de l'apprentissage pour l'algorithme KL Adaptatif avec (en rose) et sans (en bleu) entropie.

Le comportement observé est à peu près le même sur toutes les expériences, ajouter un terme d'entropie amplifie la variance des loss et des rewards obtenues et rend l'algorithme beaucoup plus instable ce qui semble parfois l'empêcher de converger. Pour la divergence de KL, on n'observe pas de changement notable.

Ce qui confirme les observations du TME précédent sur la baisse de performance observée en utilisant un terme d'entropie. Ici, on ne note aucune amélioration de performance (telle que la vitesse d'apprentissage) avec le terme d'entropie. Pour nous, cela est dû au fait de la simplicité des environnements, l'algorithme arrivant à atteindre la politique optimale rapidement, le terme d'entropie n'améliore pas vraiment les performances.

Pour l'utilisation d'un DKL inversé, nous n'observons des baisses de performances sur LunarLander, notamment une baisse de 30 en score final si on ne met pas un terme d'entropie.

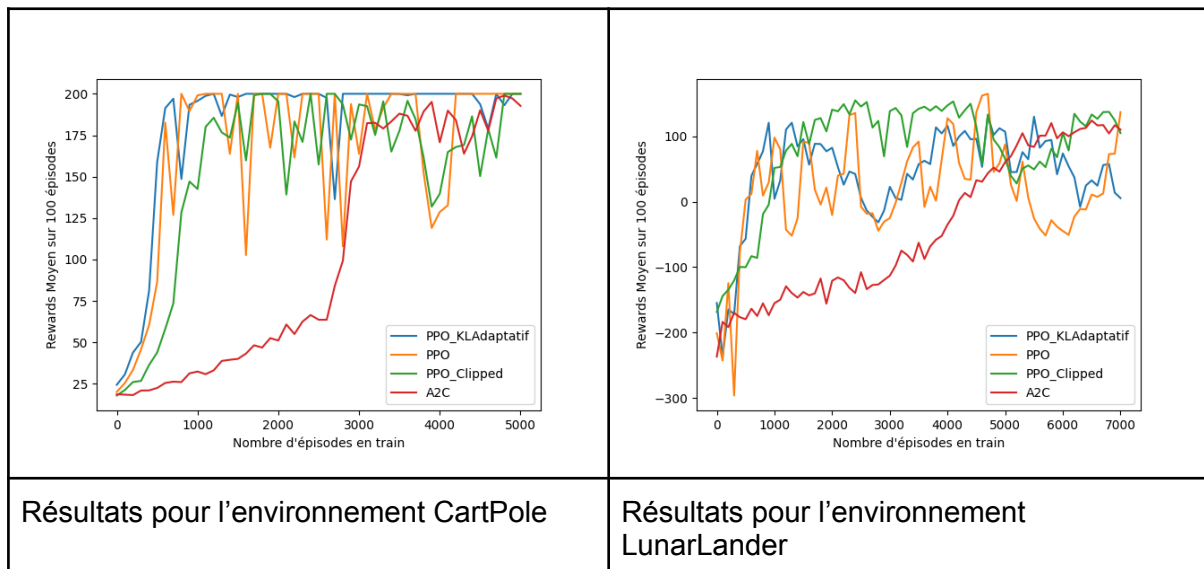
Comparaison des algorithmes

Nous réalisons maintenant une comparaison des algorithmes de ce TME avec le TME précédent. Pour cela, nous réalisons 100 épisodes de test tous les 100 épisodes d'apprentissage pour les algorithmes:

- PPO classique (sans entropie, ni inversé KL)
- PPO avec KL Adaptatif
- PPO avec clipped objectif
- A2C avec TD(λ)

Les paramètres utilisés sont les mêmes que dans les TMEs précédents à la différence que pour les tests nous enlevons le choix aléatoire (paramètre epsilon mis à 0).

Voici les courbes obtenues:



Ici il y a beaucoup de bruit car on n'utilise qu'un apprentissage. On observe tout de même que A2C est beaucoup plus lent pour converger que les autres algorithmes.

Pour l'environnement CartPole, on observe que PPO avec KL adaptatif est plus stable.

Pour l'environnement LunarLander, on observe que PPO clipped est plus stable.

On observe en général que PPO est très instable avec des pertes importantes de rewards en test fréquentes au cours de l'apprentissage.

TME7

Dans ce TME, nous montrerons les résultats que nous avons réussi à atteindre sur l'algorithme de DDPG.

Protocole expérimentale

Pour l'environnement Pendulum, nous utilisons:

- des batchs de taille 128
- Nous réalisons 10000 actions avant apprentissage
- Un processus d'exploration de Ornstein-Uhlenbeck avec $\sigma = 0.1$ et $\theta = 1$, le bruit est alors normalisé pour être dans l'intervalle d'action (en considérant le bruit entre -1 et 1).
- Nous utilisons un discount de 0.99
- Des réseaux à 4 couches cachés de 64 puis 64 puis 32 puis 32 neurones respectivement. Les fonctions d'activations utilisées sont des ReLU. Pour l'acteur nous ajoutons un tanh en fin de réseau avec une normalisation dans l'intervalle d'action. Les learning rates sont de 0.001 pour le critique et 0.0003 pour l'acteur. Nous réalisons un pas de gradient à chaque action.
- La mise à jour des réseaux cibles se fait avec un paramètre p de 0.995

Pour l'environnement LunarLander, nous modifions quelques paramètres:

- Nous passons de batchs de taille 128 à 1280
- Les learning rates sont passé à 0.003 pour le critique et 0.001 pour l'acteur.
- Nous réalisons 5 mises à jour de gradients par optimisation, une optimisation ayant lieu tous les 1000 actions après une exploration aléatoire de 5000 actions (réalisé en faisant tourner le réseau non appris)
- La mise à jour des réseaux cibles se fait avec un paramètre p de 0.99

- Nous utilisons un discount de 0.999

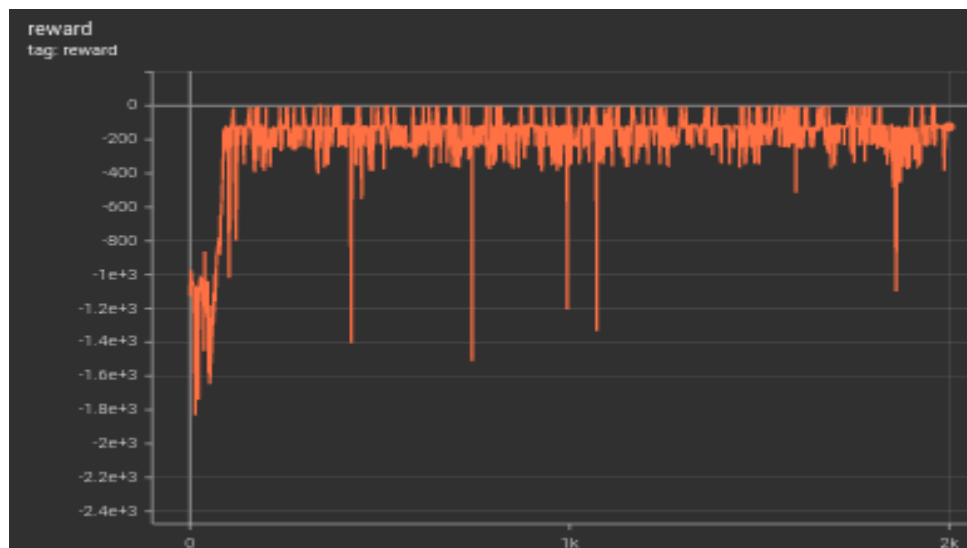
Pour l'environnement MountainCarContinuous, nous modifions quelques paramètres par rapport à Pendulum:

- Des batchs de 1024
- Un processus d'exploration de Ornstein-Uhlenbeck avec $\sigma = 0.25$ et $\theta = 0.05$
- Un discount de 0.999
- La mise à jour des réseaux cibles se fait avec un paramètre p de 0.99
- Le critique à comme learning rate 0.0001 et l'acteur, 0.001

Nous réalisons 2000 épisodes d'apprentissage pour l'environnement Pendulum avec une sauvegarde tous les 200 épisodes et 3000 épisodes d'apprentissage pour l'environnement Lunar avec une sauvegarde tous les 300 épisodes. Pour MountainCar, nous réalisons 1000 épisodes d'apprentissage avec une sauvegarde tous les 100 épisodes.

Résultats

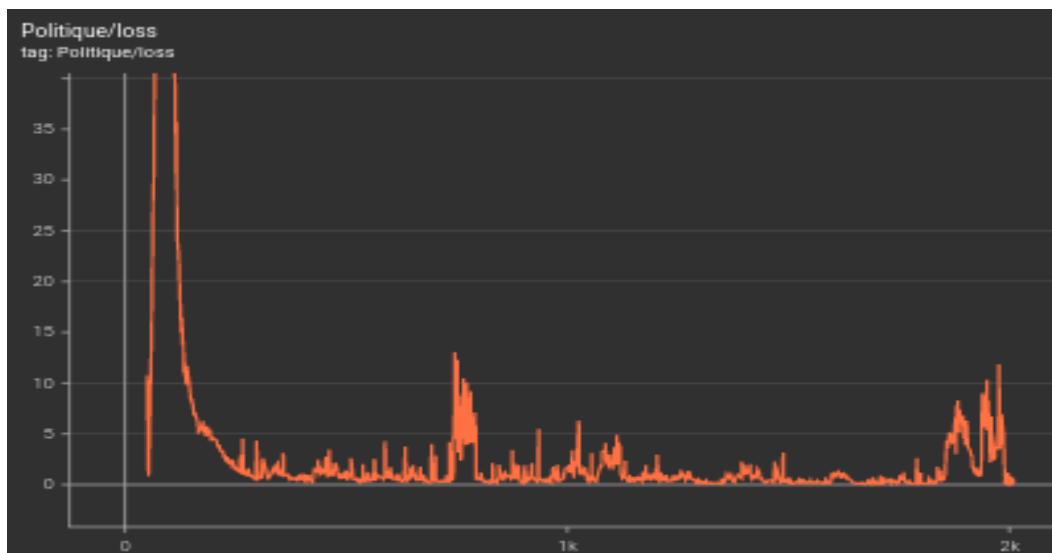
Pour l'environnement Pendulum le score final est de -144.76 ± 89.88 au bout de 400 itérations, voici les courbes associées:



Evolution des rewards au cours de l'apprentissage pour l'environnement Pendulum

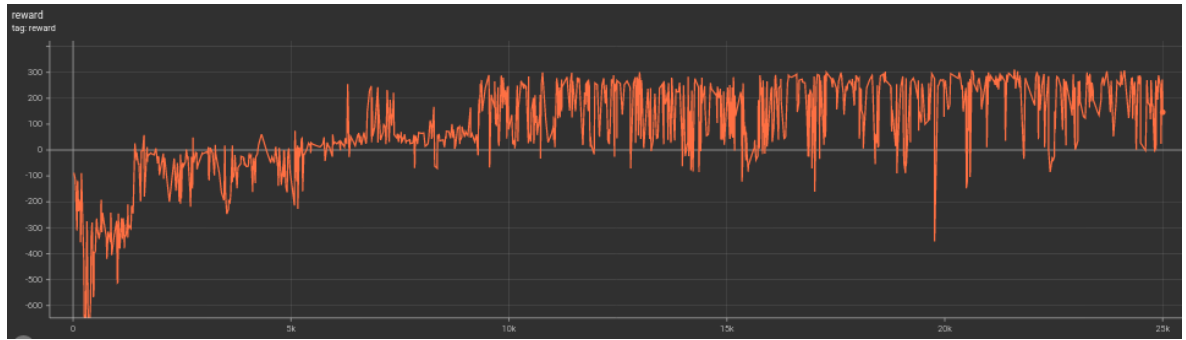


Evolution de la loss du critique au cours de l'apprentissage pour l'environnement Pendulum



Evolution de la loss de l'acteur au cours de l'apprentissage pour l'environnement Pendulum

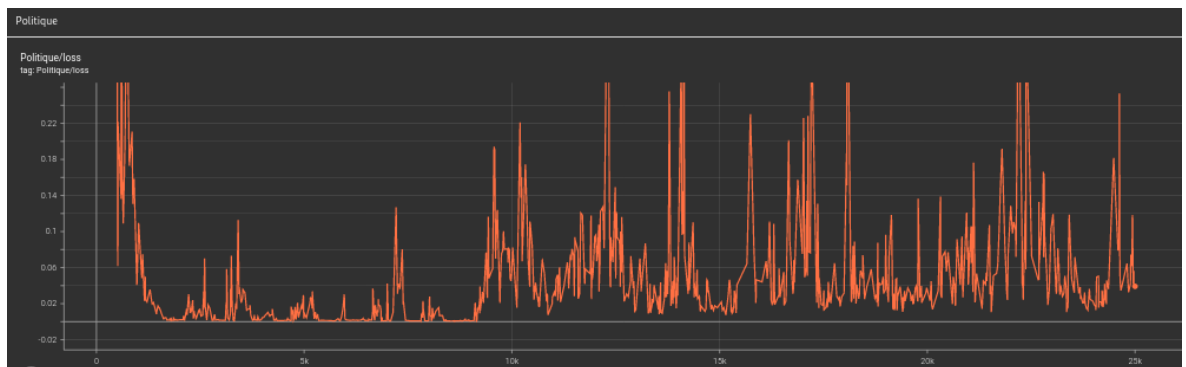
On observe une convergence à environ 200 itérations.
 Pour Lunar Lander on obtient un score final de 142.32+/-66.55 au bout de 25 000 itérations.
 Voici les courbes obtenues:



Evolution des rewards au cours de l'apprentissage pour l'environnement LunarLanderContinuous



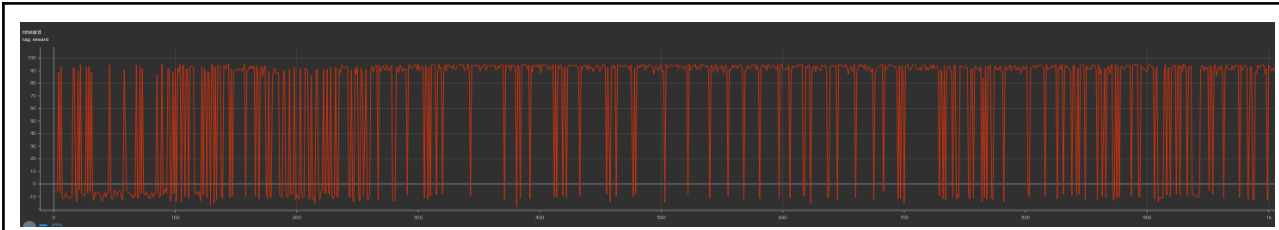
Evolution de la loss du critique au cours de l'apprentissage pour l'environnement LunarLanderContinuous



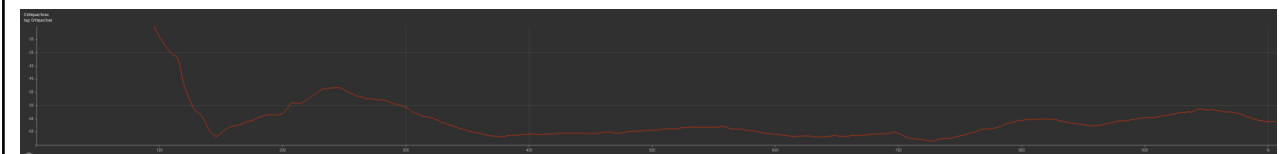
Evolution de la loss de l'acteur au cours de l'apprentissage pour l'environnement LunarLanderContinuous

L'algorithme semble converger vers 10 000 itérations. On observe tout de même des fortes variations des loss du critique et de l'acteur au cours de l'apprentissage. On observe aussi que la loss de l'acteur n'arrive pas à atteindre 0 alors que la loss du critique tend vers des valeurs négatives.

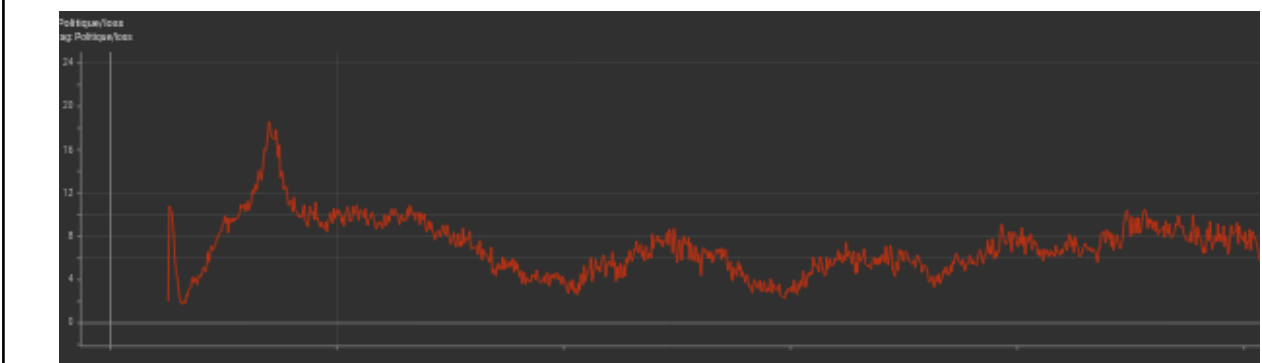
Pour l'environnement ContinuousMountainCar, le score final est de 88.17+/-22.71 atteint après 600 épisodes d'apprentissage, voici les courbes d'apprentissage:



Evolution des rewards au cours de l'apprentissage pour l'environnement MountainCarContinuous



Evolution de la loss du critique au cours de l'apprentissage pour l'environnement MountainCarContinuous



Evolution de la loss de l'acteur au cours de l'apprentissage pour l'environnement MountainCarContinuous

Il est difficile de voir le temps de convergence de l'algorithme dans ce cas mais on observe bien une fréquence de plus en plus importante du nombre d'atteinte de la sortie. La loss du critique tend vers des valeurs négatives et l'acteur semble avoir du mal à converger. On observe tout de même un bon score, celui-ci arrivant à atteindre la sortie mais cela semble indiquer qu'il est possible d'obtenir de meilleurs scores.

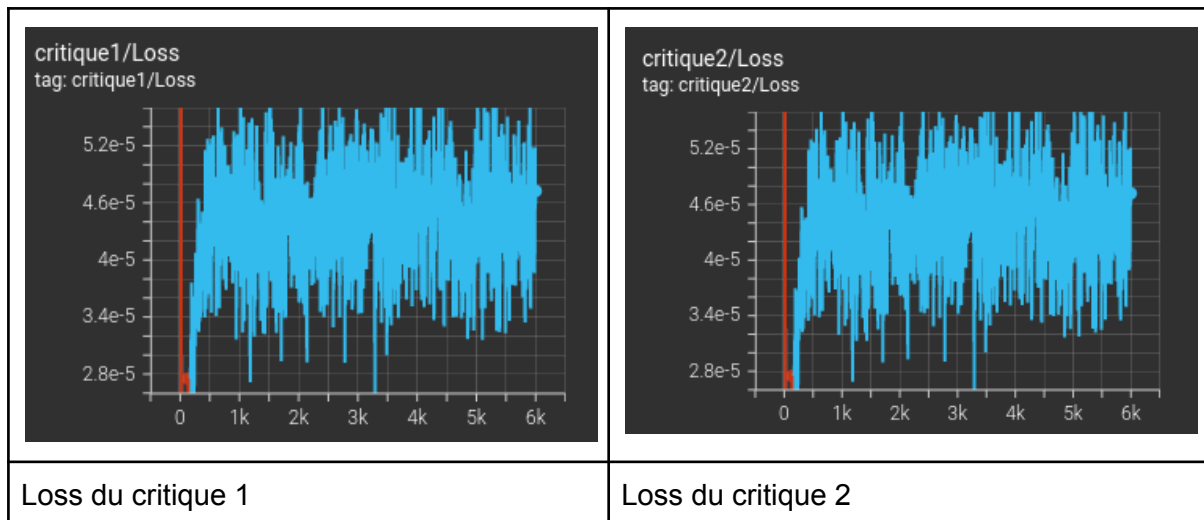
TME8

Nous n'avons malheureusement pas réussi à faire converger ce dernier algorithme. Nous avons testé de nombreux paramètres mais impossible d'obtenir une convergence.

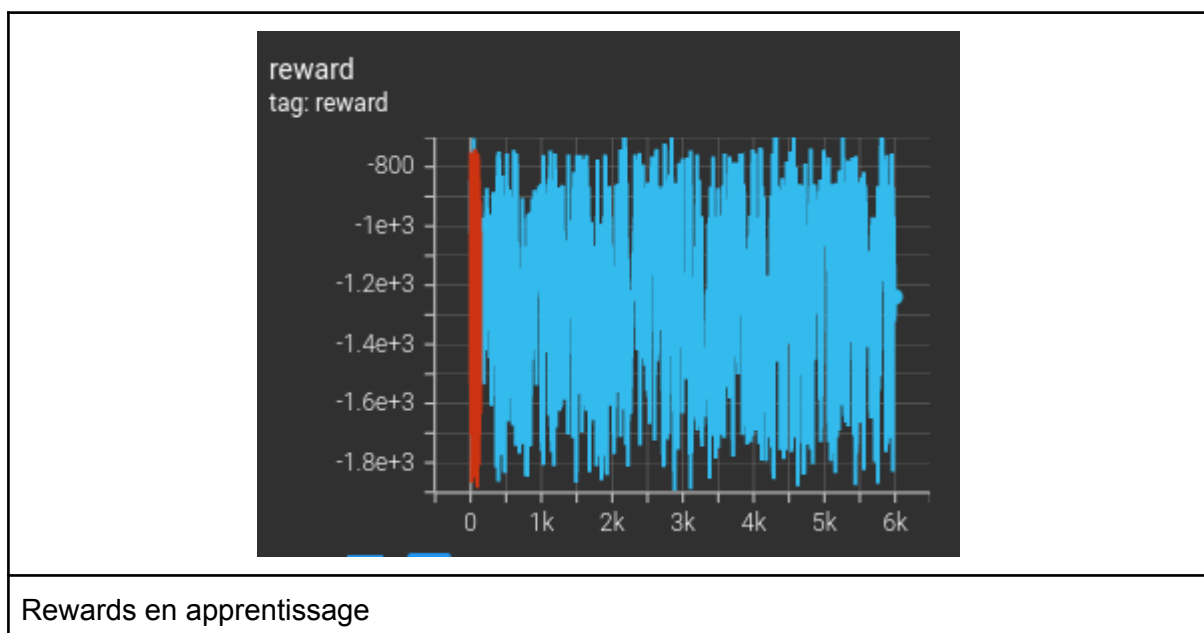
Le code est disponible même si celui-ci ne fonctionne pas.

Voici tout de même les courbes obtenues pour les paramètres indiqués dans le sujet pour Pendulum.

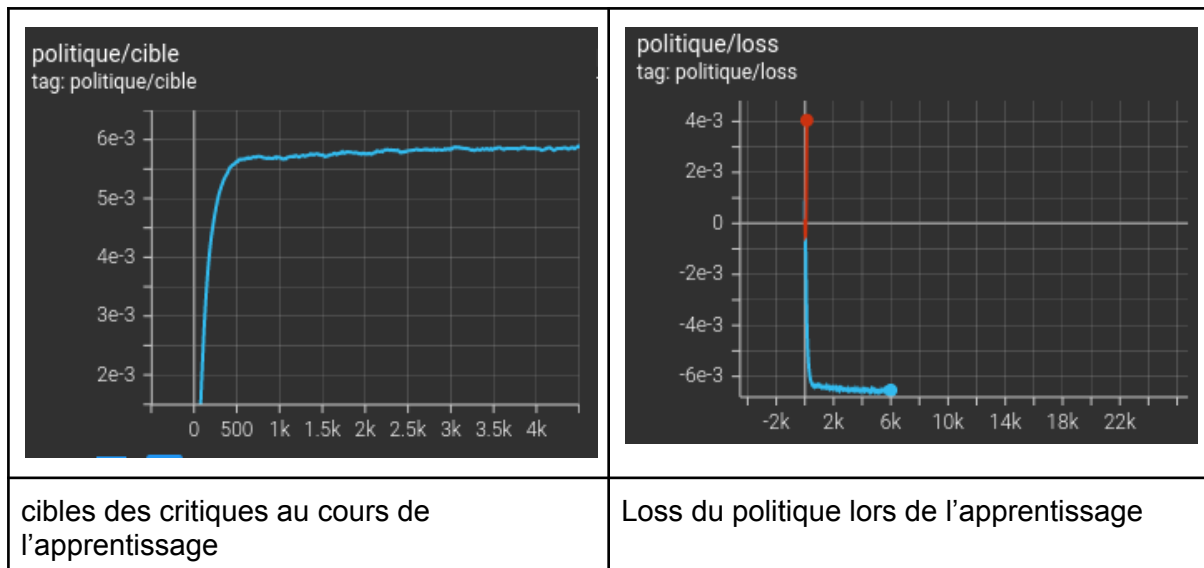
Ici la courbe rouge correspond à l'expérience avec alpha adaptatif et bleu sans alpha adaptatif



Dans le cas de la courbe rouge, on observe une explosion des gradients qui mène à l'apparition de nan en sortie, ce qui stop l'exécution. Dans le cas de la courbe en bleu, on ne le voit pas mais la loss diminue au départ puis ré-augmente pour stagner. Voici les rewards associés:



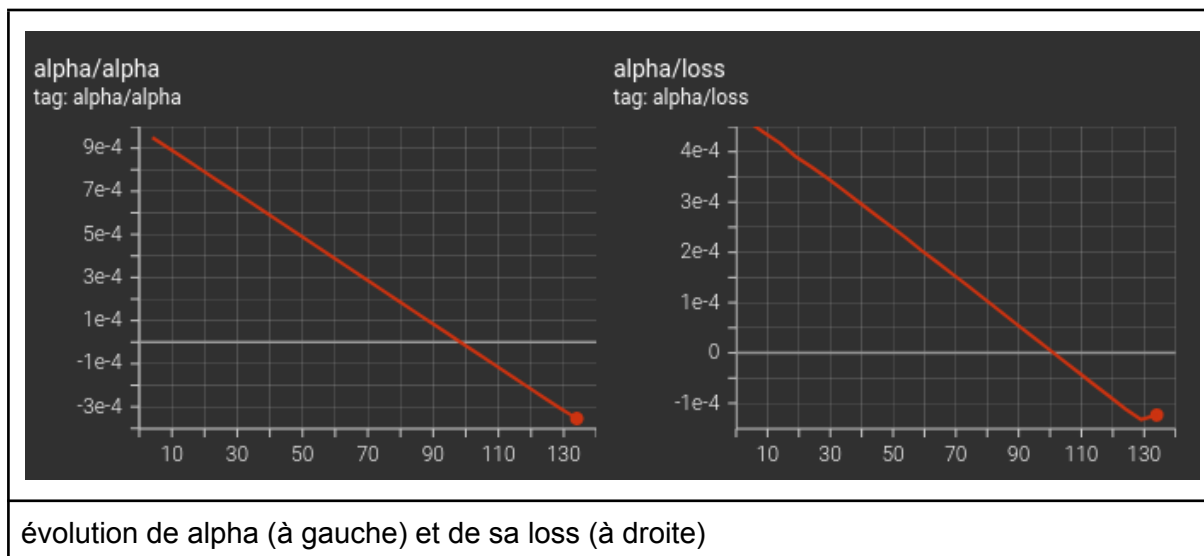
On observe une forte variance des rewards, ceux-ci semblant stagner avec des résultats que l'on pourrait attendre de l'aléatoire. On peut aussi analyser l'évolution de la cible et de la loss du politique:



Ici les résultats semblent correspondre aux résultats attendus mais l'évolution des cibles semble très rapide. Dans le cas de la loss du politique, elle diminue rapidement pour atteindre des valeurs négatives. Dans le cas de l'alpha adaptatif, on observe une explosion de la loss ici aussi.

Pour analyser plus en profondeur le cas de l'alpha adaptatif, on peut observer la loss et l'évolution de la valeur de alpha.

Voici les courbes obtenues:



Ici on observe un apprentissage qui mène vers un alpha négatif.

Pour conclure, il semble qu'un premier problème vient des critiques qui ne se comportent pas correctement mais il est difficile de savoir si cela vient vraiment du critique ou s'il s'agit d'une conséquence d'un problème venant d'ailleurs. Pour régler ce problème, nous avons joué avec les taux d'apprentissage mais sans succès.

Il semble que l'écart type lors de la génération d'actions est très fort (ce qui explique le manque d'apprentissage), ce qui empêche d'apprendre correctement, nous avons notamment essayé de clipper celui-ci et de mieux initialiser les poids pour éviter un trop grand écart type mais cela fait exploser le gradient comme avec l'alpha adaptatif ou fait diverger les critiques.

Nous n'avons pas essayé de debugger l'alpha adaptatif sachant que l'algorithme basique ne fonctionnait pas.

Merci à Nicolas Baskiotis pour son aide pour le debuggage de ce TME.