



SORBONNE UNIVERSITÉ
FACULTÉ SCIENCES ET INGÉNIERIE
MASTER 2 INFORMATIQUE, PARCOURS DAC

Rapport RLD

Etudiants:

Bozhang HUANG
Haoran LI

Tuteur:

Sylvain LAMPRIER
Nicolas BASKIOTIS

Table des matières

1	TME1 Problèmes de Bandits	2
1.1	Intro	2
1.2	Baselines	2
1.3	UCB	3
1.4	LinUCB	3
1.5	Conclusion	4
2	TME2 Programmation Dynamique Value/Policy itération	5
2.1	Intro	5
2.2	Value Itération	5
2.3	Policy Itération	6
2.4	Conclusion	6
3	TME3 Q-Learning	7
3.1	Intro	7
3.2	Test sur plan1	8
3.3	Comparaison sur plan2	10
3.4	Comparaison sur plan5	12
3.5	Conclusion	13
4	TME4 : DQN	14
4.1	Intro	14
4.2	Test	14
4.3	Conclusion	15
5	TME5 : Policy Gradient : Actor-Critic	16
5.1	Intro	16
5.2	CartPole	16
5.3	GridWorld	17
5.4	LunarLander	19
5.5	Conclusion	19
6	TME6 : Policy Gradient : PPO	20
6.1	Intro	20
6.2	CartPole	21

6.3	LunarLander	22
6.4	Conclusion	23
7	TME7 Continuous Action DDPG	24
7.1	Intro	24
7.2	MountainCarContinuous-v0	24
7.3	LunarLanderContinuous-v2	26
7.4	Pendulum-v0	27
7.5	Conclusion	28
8	TME8 Soft Actor-Critic (SAC)	29
8.1	Intro	29
8.2	MountainCarContinuous-v0	29
8.3	LunarLanderContinuous-v2	30
8.4	Pendulum-v0	31
8.5	Conclusion	32
9	Résumé final	33

Introduction

Ce rapport contient tous les travaux pratiques qu'on a fait dans la partie Reinforcement Learning. On présente ici, pour chaque algorithme qu'on a appris, les hyper-paramètres qu'on utilise, les résultats obtenus et éventuellement des comparaisons entre les modèles. A propos de bibliothèques utilisée, on utilise [pytorch](#) pour construire les réseaux de neurones et tester dans les environnements de [gym](#).

Les exemples des environnements qu'on utilise sont :



FIGURE 1 – CartPole

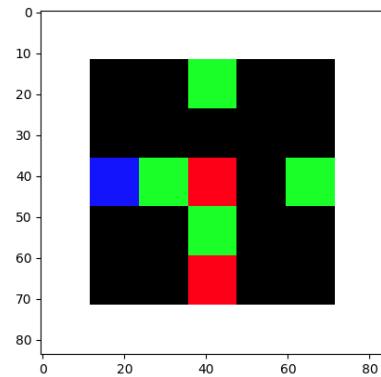


FIGURE 2 – GridWorld

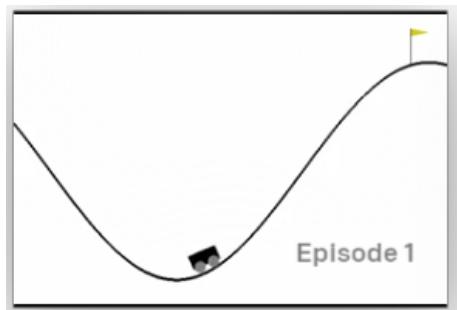


FIGURE 3 – MountainCar



FIGURE 4 – Pendulum

1 TME1 Problèmes de Bandits

1.1 Intro

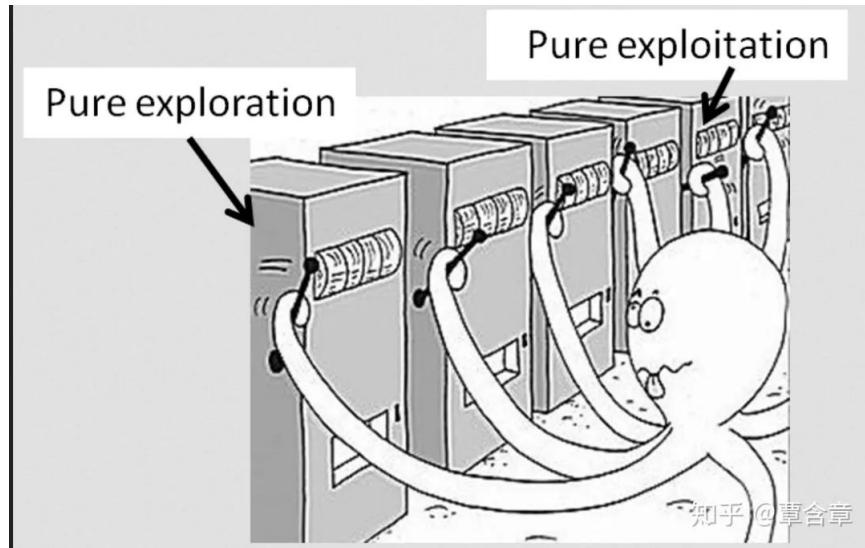


FIGURE 5 – Bandit manchot

C'est une problématique très classique.

Généralement, il y a toujours 2 procédures dans ces modèles :

- **Exploration** : Comprendre et explorer l'environnement.
- **Exploitation** : Utiliser des informations connues.

L'exploitation versus l'exploration est un sujet critique dans l'apprentissage par renforcement. Nous aimerais que l'agent RL trouve la meilleure solution le plus rapidement possible. Cependant, en attendant, s'engager dans des solutions trop rapidement sans suffisamment d'explorations semble assez mauvais, car cela pourrait conduire à des minima locaux ou à un échec total.

1.2 Baselines

On utilise un agent avec (`nb_bras = 10`) et teste les différents baselines.

`times` : Distribution de fréquences pour chaque bras

`rewards` : Distribution de rewards sommés pour chaque bras

Stratégie Random

- `times` : [518 518 504 483 465 499 498 540 496 479]
- `rewards` : [6.99951162 37.79807934 4.06623848 143.90063666 17.01328377 0.89227063 47.47689077 16.90964307 3.55279738 133.39963749]

Stratégie StaticBest

- times : [10 10 10 4910 10 10 10 10 10 10]
- rewards : [6.99951162 37.79807934 4.06623848 143.90063666 17.01328377 0.89227063 47.47689077 16.90964307 3.55279738 133.39963749]

Stratégie Optimale

- times : [7 146 1 2681 13 0 243 6 0 1903]
- rewards : [1.27436352e+00 3.15482431e+01 2.05103396e-01 8.72934461e+02 2.04748753e+00 0.00000000e+00 6.55797526e+01 1.19821485e+00 0.00000000e+00 5.57215610e+02]

1.3 UCB

UCB est **the upper confidence bound algorithm**. Il suit le principe de l'optimisme face à l'incertitude qui implique que si nous sommes incertains au sujet d'une action, nous devons supposer avec optimisme qu'il s'agit de l'action correcte.

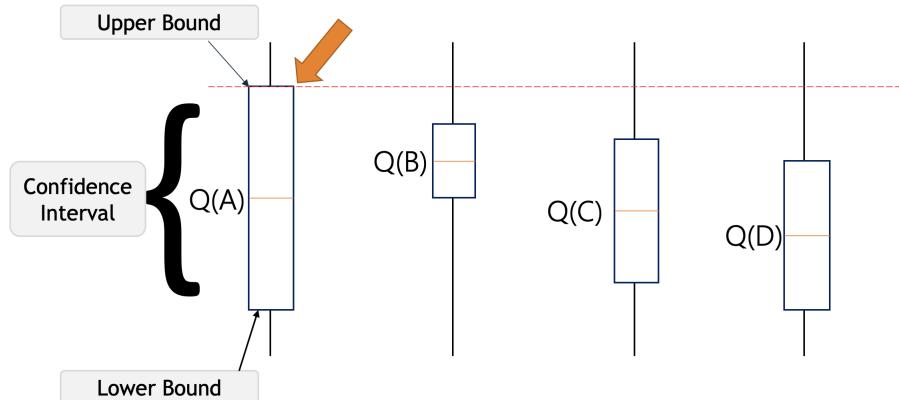


FIGURE 6 – UCB

- times : [5 215 3 3350 6 3 4 809 133 472]
- rewards : [1.43129681e-01 1.63207644e+01 0.00000000e+00 8.99155409e+02 3.27505792e-01 0.00000000e+00 1.94630537e-01 2.72421641e+01 3.86983769e-01 1.16946581e+02]

1.4 LinUCB

Par rapport à UCB, on utilise le contexte dans LinUCB, CAD avec Contextual Bandits.

- times : [8 2 2 2191 52 2 654 5 2 2082]
- rewards : [1.22581751e-01 0.00000000e+00 0.00000000e+00 6.38115428e+02 1.95993327e+00 0.00000000e+00 7.92637545e+01 6.97578580e-02 0.00000000e+00 5.62572528e+02]

1.5 Conclusion

On visualise les rewards sommés pour chaque méthode. Ça marche bien. Stratégie optimale est le meilleur, ensuite c'est staticbest et LinUCB, random est le pire.

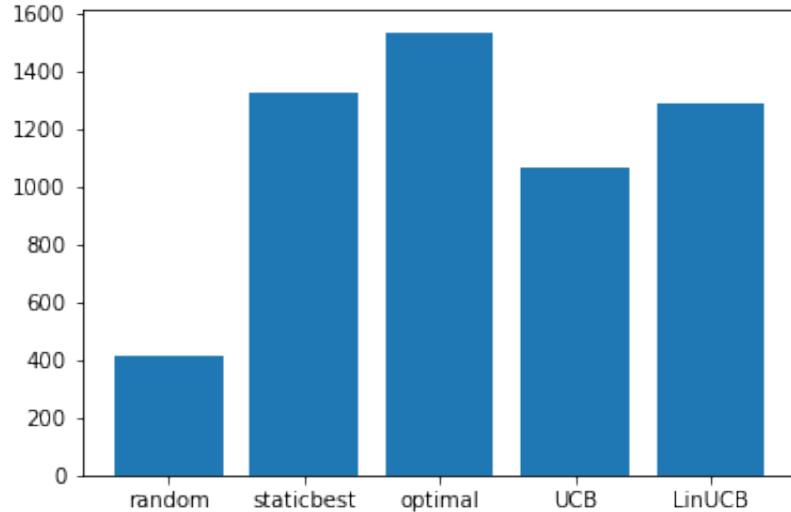


FIGURE 7 – Histogramme de tous

2 TME2 Programmation Dynamique Value/Policy itération

2.1 Intro

Ce TME a pour objectif d'expérimenter les modèles algorithmes de programmation dynamique sur un MDP classique de type **GridWorld**.

Nous utiliserons au cours des TME de RL la plateforme en python **gym** (de open-ai).

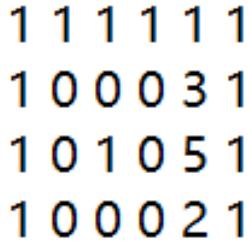


FIGURE 8 – Plan de GridWorld

Environnement de Gridworld 2D avec le codage suivant :

0 : case vide 1 : mur 2 : joueur 3 : sortie 4 : objet à ramasser 5 : piège mortel 6 : piège non mortel

Actions :

0 : South 1 : North 2 : West 3 : East -1 : Fini

2.2 Value Itération

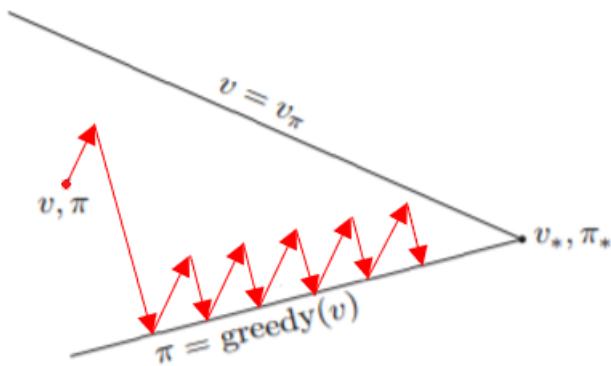


FIGURE 9 – Value Iteration

ValueAgent(env, epsilon = 1e-4, gamma=0.95)

Value Iteration : Round 12

On trouve la politique optimale est : [2. -1. 2. 1. 3. -1. 3. 3. 1. 1. 2.]

2.3 Policy Itération

PolicyAgent(env, epsilon = 1e-4, gamma=0.95)

policy iterations Round Number :3

On trouve la politique optimale est : [2, 1, 2, 1, 3, 1, 3, 3, 1, 1, 2]

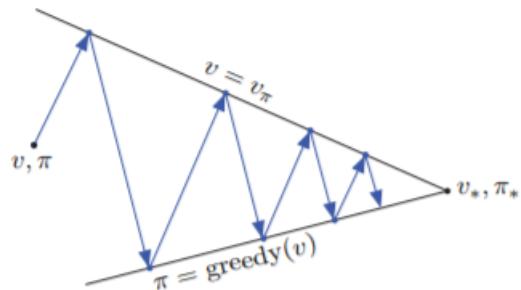


FIGURE 10 – Policy Itération

2.4 Conclusion

Policy itération c'est chaque fois qu'il doit courir à la convergence de value fonction.

Value itération c'est chaque fois qu'il couvre une étape.

Les deux types d'itérations convergeront toujours vers le même comme les 2 graphes ci-dessus.

On le visualise ci-dessous :

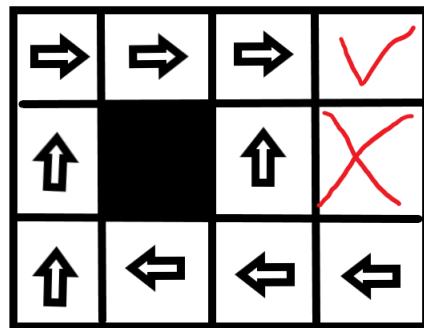


FIGURE 11 – politique

3 TME3 Q-Learning

3.1 Intro

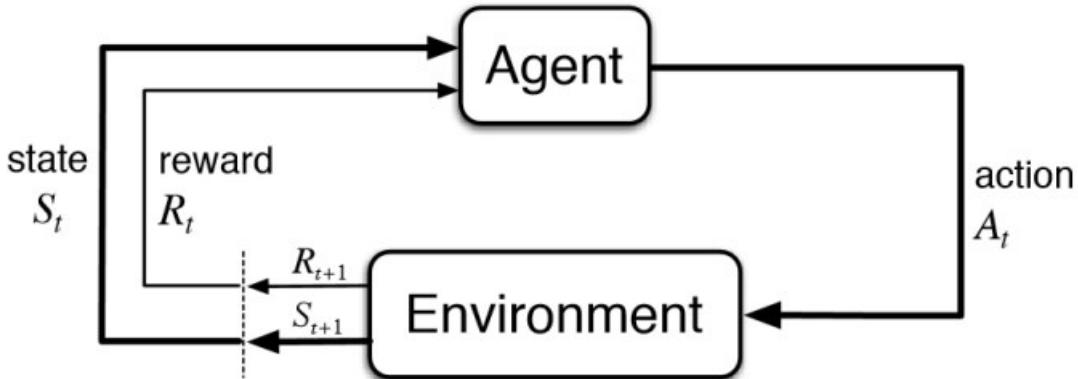


FIGURE 12 – QLearning

Ce TME, on va concentrer sur QLearning.

Il y a 3 méthodes ci-dessous :

- **QLearning classique**

La version la plus basique. C'est off-policy.

- **Version SARSA**

Par rapport à QLearning, c'est on-policy et moins agressif. La stratégie d'objectif et la stratégie de comportement sont la même stratégie.

- **Extension Dyna-Q**

On utilise cette extension avec QLearning. On ajoute un modèle de plus pour se souvenir du passé.

Certaines de leurs idées de code peuvent être un peu différentes, mais l'idée la plus basique est d'utiliser valeur état-action notée Q .

On utilise Gridworld comme environnement.

nbEpisodes : 100000

explo : 0.1 # coefficient d'exploration initial

decay : 0.999 # à la fin de chaque trajectoire, le coefficient d'explo est multiplié par ce facteur

gamma : 0.99 # Facteur de discount

learningRate : 0.1 # Pas d'apprentissage

BULE : joueur

YELLOW : objet à ramasser

GREEN : sortie

RED : piège mortel

3.2 Test sur plan1

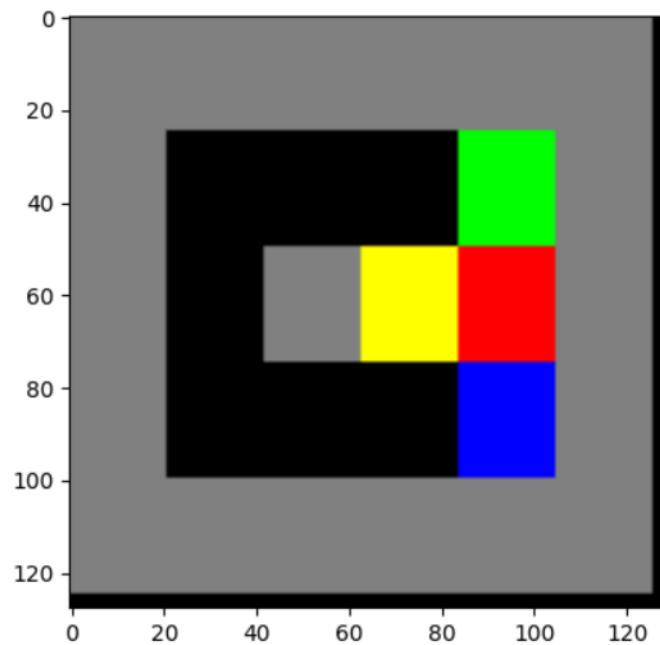


FIGURE 13 – Graphe Plan1

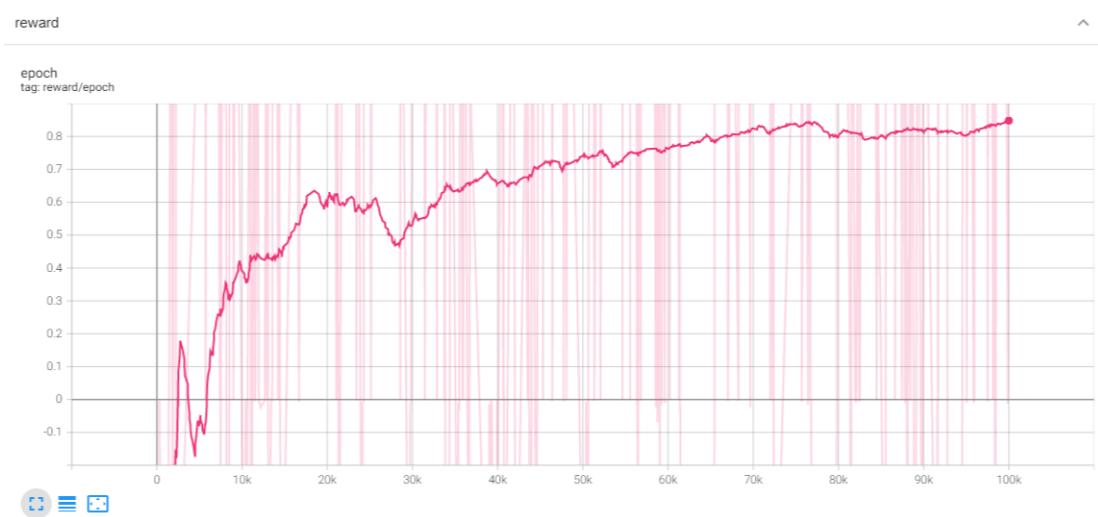


FIGURE 14 – QLearning

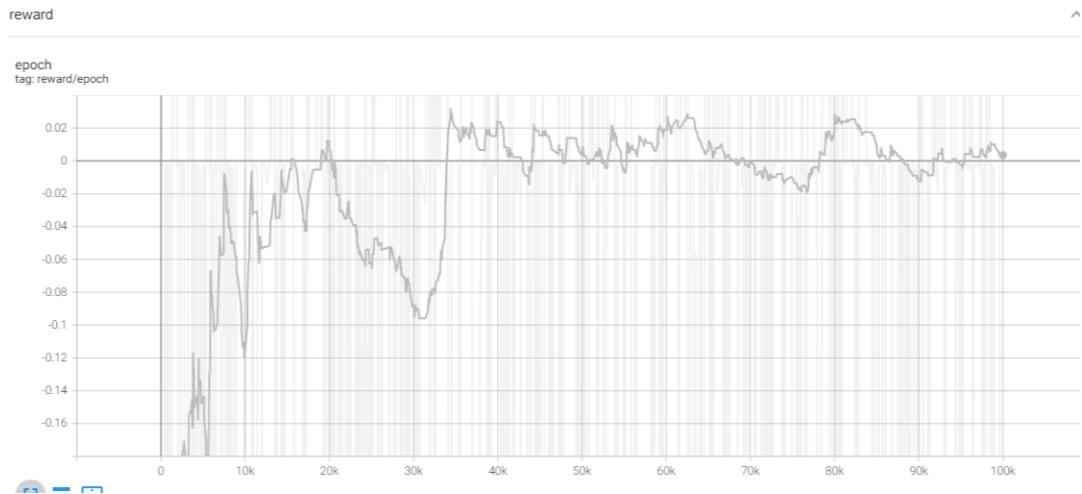


FIGURE 15 – Dyna-Q

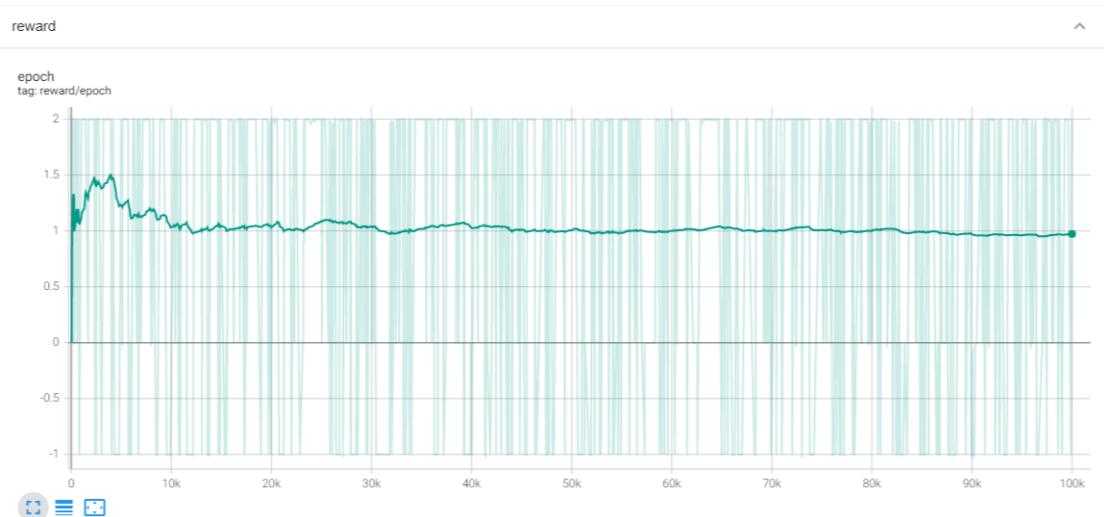


FIGURE 16 – SARSA

•Commentaire :

On voit plan1, la carte est relativement petite et il y a un piège mortel au-dessus du point de départ.

Donc l'espérance de Qlearning et de SARSA est d'environ 1. A propos de Dyna-Q, ce n'est pas bien. Les raisons sont comme nous disons d'avance.

On verra les comparaisons ci-dessous :

3.3 Comparaison sur plan2

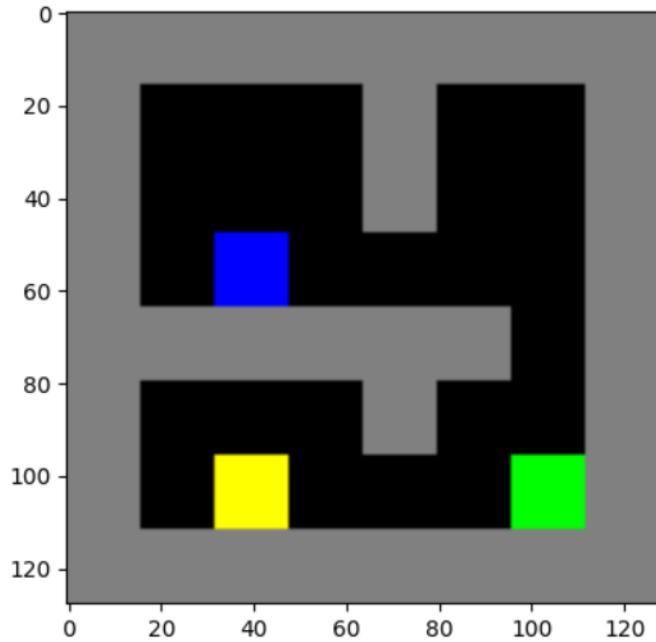


FIGURE 17 – Graphe Plan2

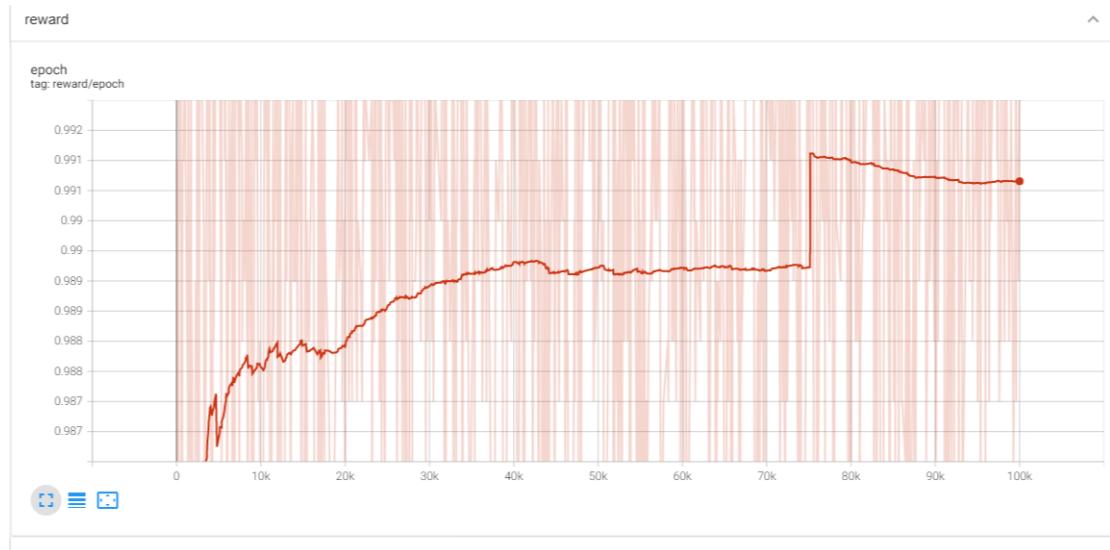


FIGURE 18 – QLearning

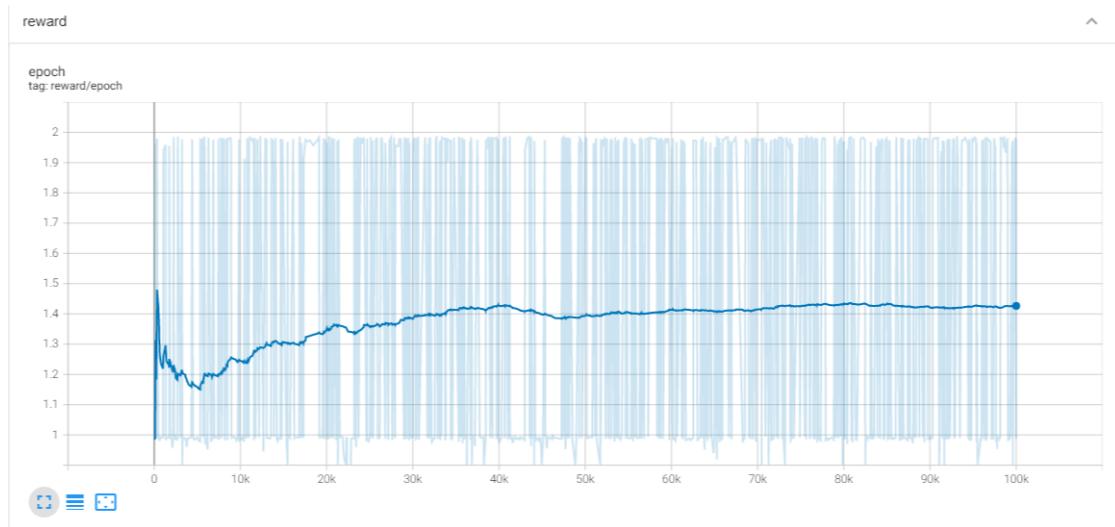


FIGURE 19 – Dyna-Q

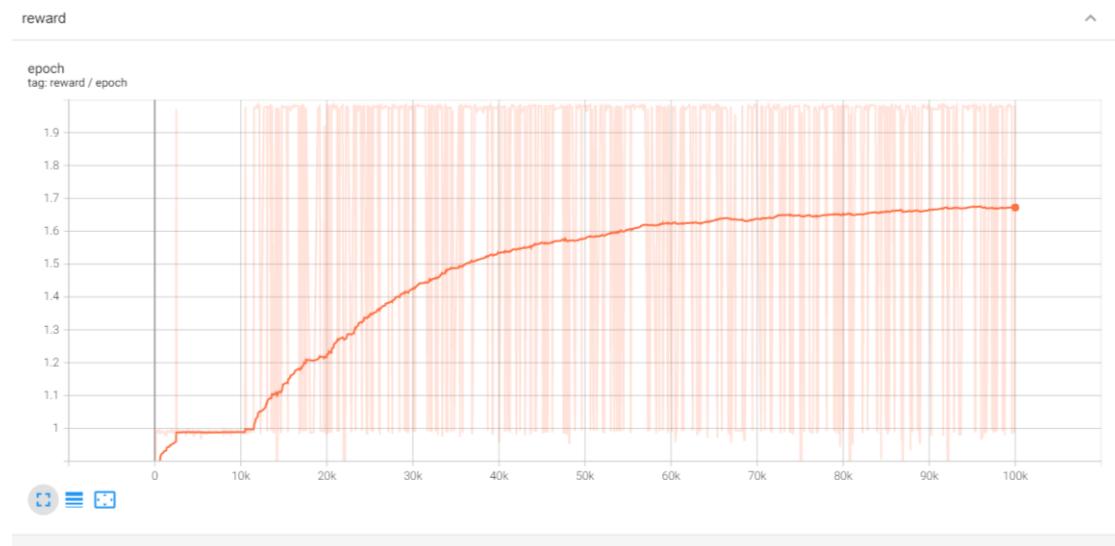


FIGURE 20 – SARSA

- Commentaire :

On voit qu'il marche pas mal cette fois. Le résultat de Dyna-Q est mieux que celui de QLearning. SARSA est le meilleur. On trouve que il y a différentes influences dans différentes cartes pour chaque algorithme.

3.4 Comparaison sur plan5

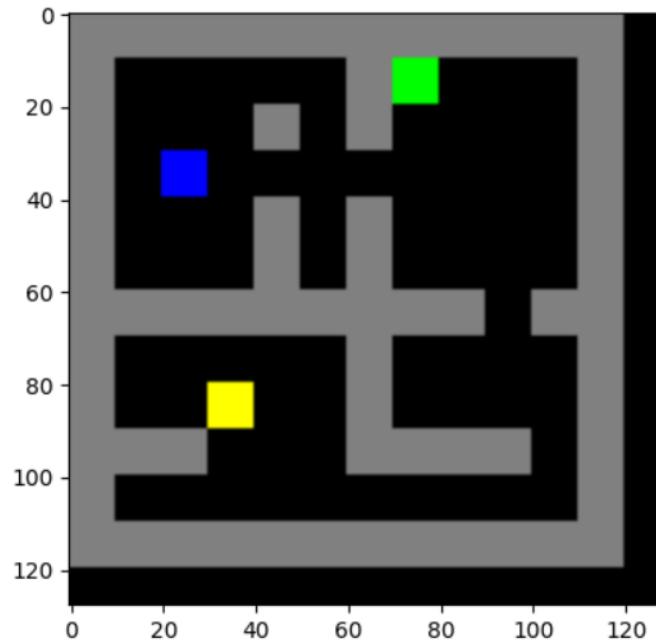


FIGURE 21 – Graphe Plan5

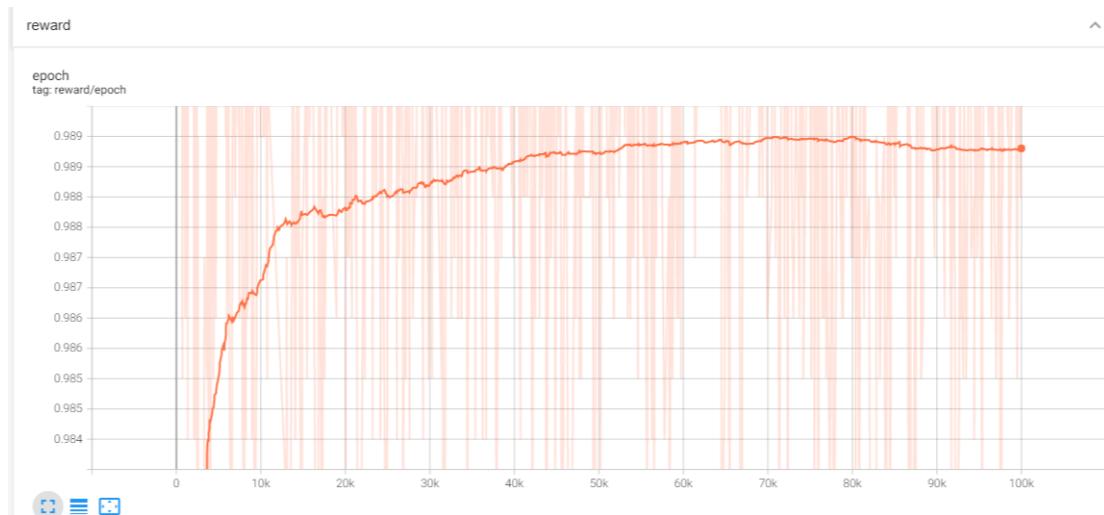


FIGURE 22 – QLearning

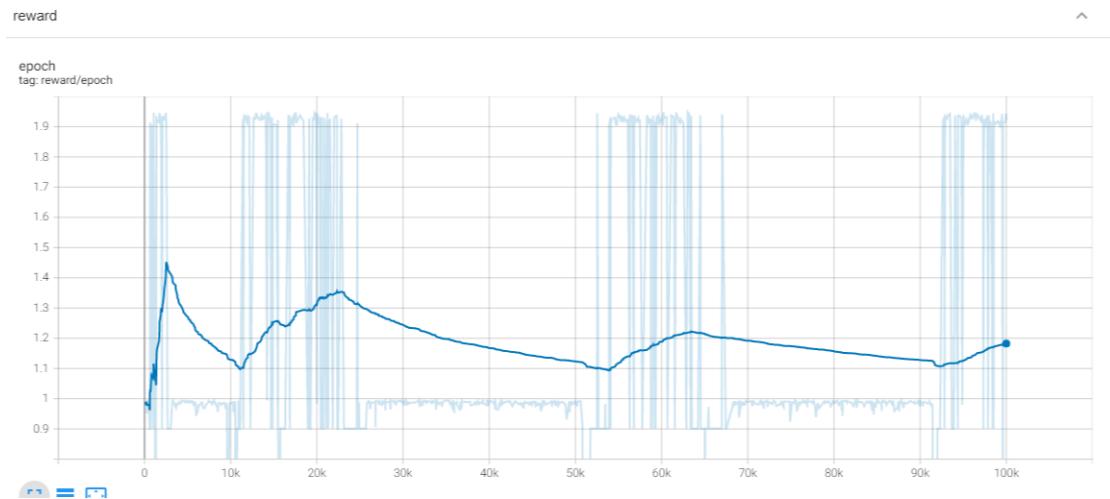


FIGURE 23 – Dyna-Q

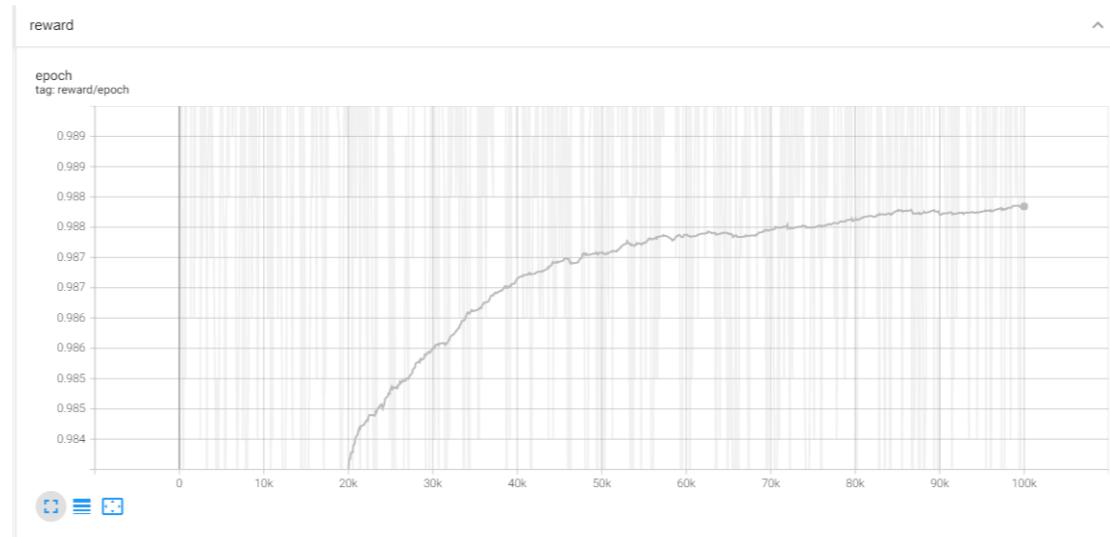


FIGURE 24 – SARSA

- Commentaire :

Le résultat de Dyna-Q est mieux que celui de QLearning. En revanche, SARSA est le pire. C'est aussi différent que les résultats précédents. Donc l'environnement est vraiment important pour l'application d'algorithme.

3.5 Conclusion

Les 3 algorithmes ont performances différentes dans différentes cartes. Bien sûr, cela peut aussi être causé par un réglage insuffisant.

Plan1 est spécial car il y un piège mortel au-dessus du point de départ. Ça marche pas mal sur plan2 et plan5.

4 TME4 : DQN

4.1 Intro

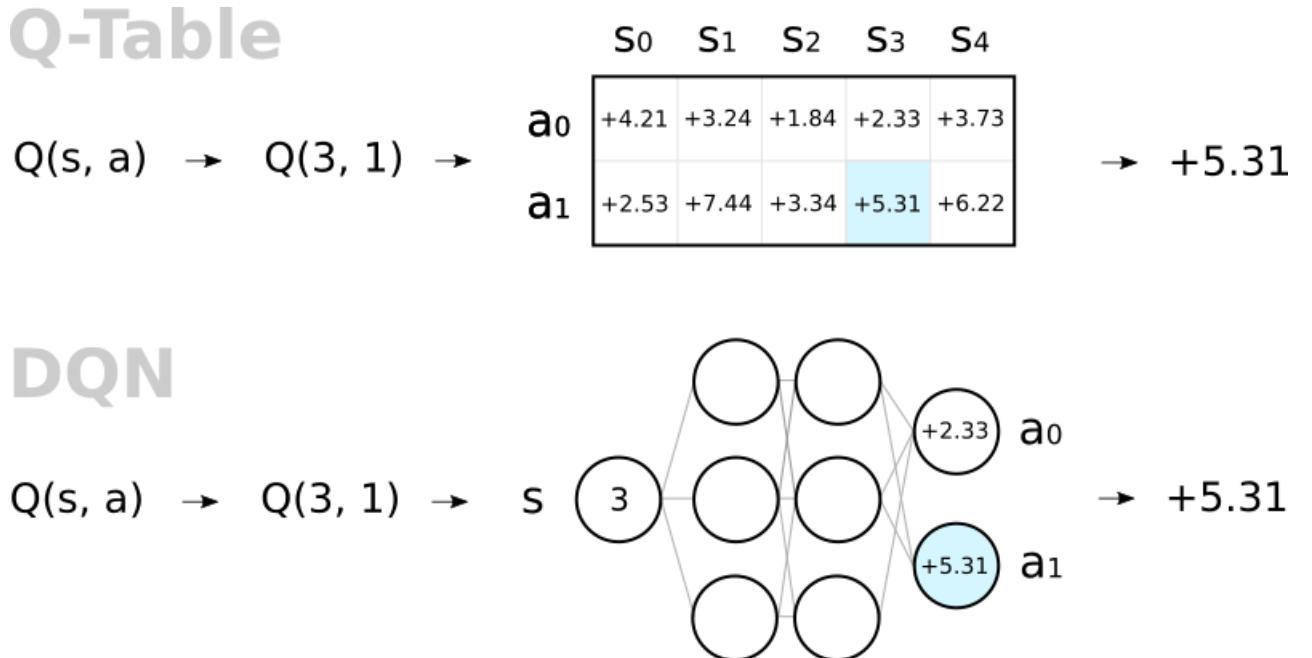


FIGURE 25 – Q-table et DQN

Tout simplement, DQN est la combinaison avec Reinforcement Learning et Deep Learning. On utilise les réseaux pour évaluer les Q values, pas Q-table.

DQN adopts the **Experience Replay** mechanism to store the trained data in the Replay Buffer so that it can be randomly sampled for training. The advantages are :

1. High data utilization ;
2. Reduce the correlation of continuous samples, thereby Reduce variance (variance).

4.2 Test

Nous avons implémenté l'algorithme DQN avec Target network et Prioritized Experience Replay.

Nous utilisons un pas d'apprentissage de 0.001, un disconut de 0.99, un ϵ d'exploration variant avec l'itération, un batch size de 32 pour apprentissage et le target network est mis à jour toutes les 100 itérations. L' ϵ que nous considérons suivant le graphe :

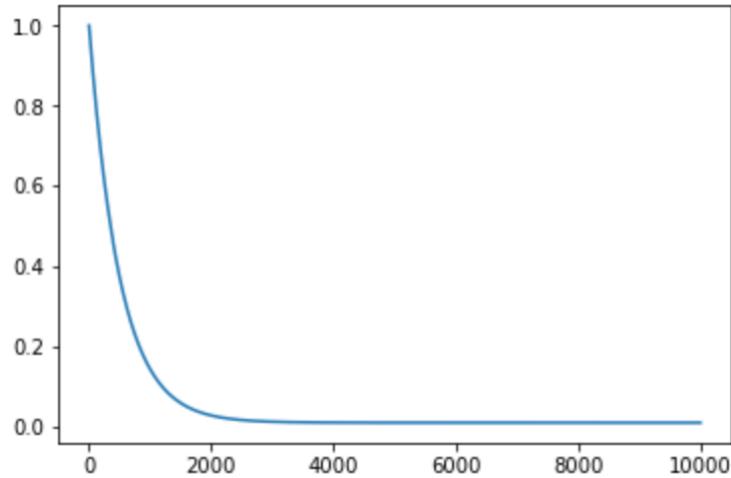


FIGURE 26 – variation de l’ ϵ avec itération, DQN

Nous obtenons ainsi :

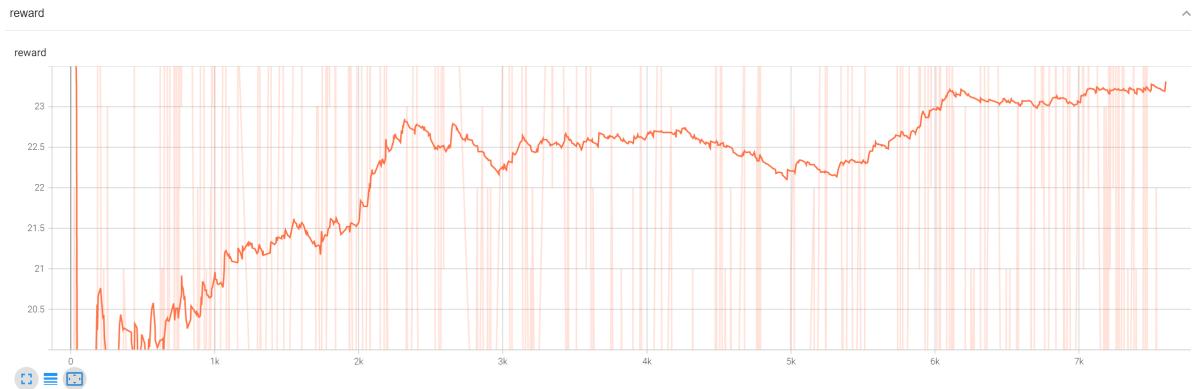


FIGURE 27 – CartPole, rewards, DQN

Nous voyons que le reward augmente mais très lentement.

4.3 Conclusion

On voit les résultats montrés par le prof, il y a 70k itérations totalement. Il y a une augmentation significative à 50k itération. Mais pour nous, 7k itérations est la limite. Car c'est trop lent.

On verra que on teste AC après, ça a montré que utilisation de réseaux de neurones est très utile pour Reinforcement Learning.

5 TME5 : Policy Gradient : Actor-Critic

5.1 Intro

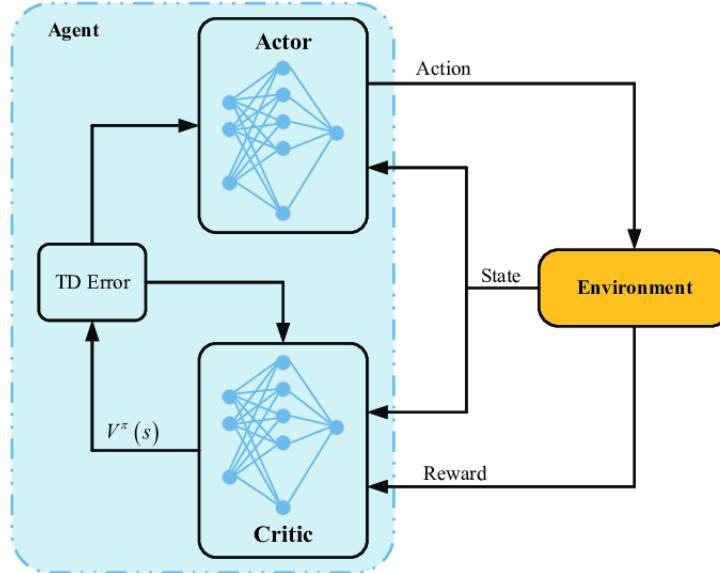


FIGURE 28 – AC

Actor signifie acteur et Critic signifie critique. Comme son nom l'indique, cet algorithme résout le problème de la variance élevée en introduisant un mécanisme d'évaluation. Par conséquent, il existe deux ensembles de paramètres dans l'algorithme Actor-Critic : Critic : mise à jour du paramètre de fonction de valeur d'action ω ; Actor : Mettre à jour les paramètres de stratégie θ dans le sens guidé par le critique.

5.2 CartPole

Nous considérons ici 3 critic différents pour le problème de CartPole :

- \mathbf{V} comparé avec \mathbf{R} , la version Rollout Monte-Carlo
- V_t comparé avec $r_t + \gamma * V_{t+1}$, la version TD(0)
- La version TD(λ)

avec un discount de 0.9, un pas d'apprentissage de 0.001 pour le réseau acteur et 0.01 pour le réseau critic, un paramètre $\lambda = 0.99$ pour la version TD(λ)

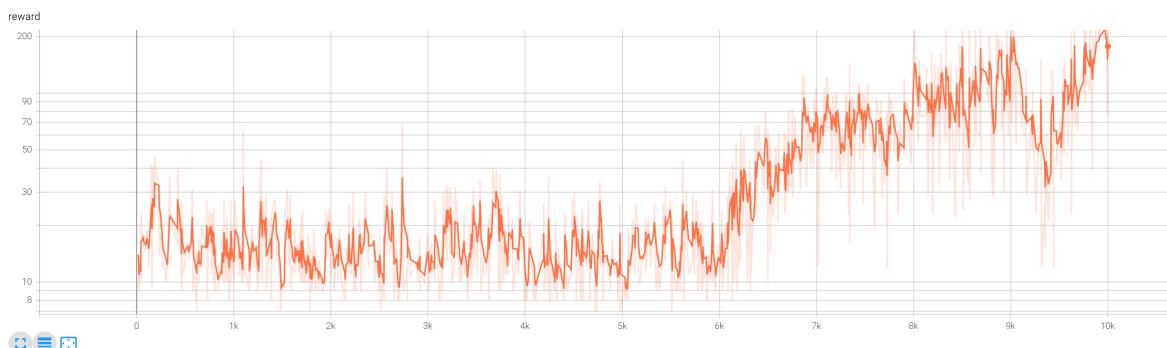


FIGURE 29 – CartPole, rewards, version Rollout Monte-Carlo

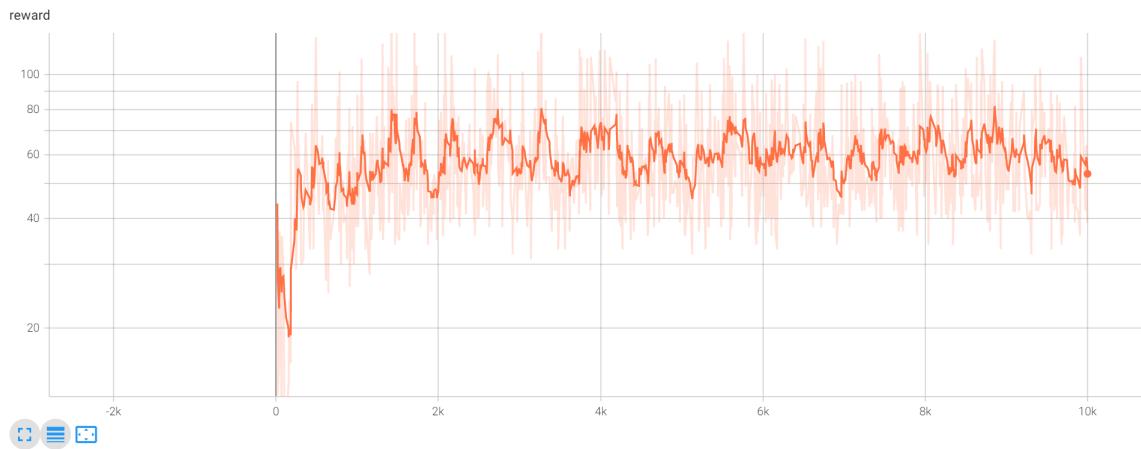


FIGURE 30 – CartPole, rewards, version TD(0)

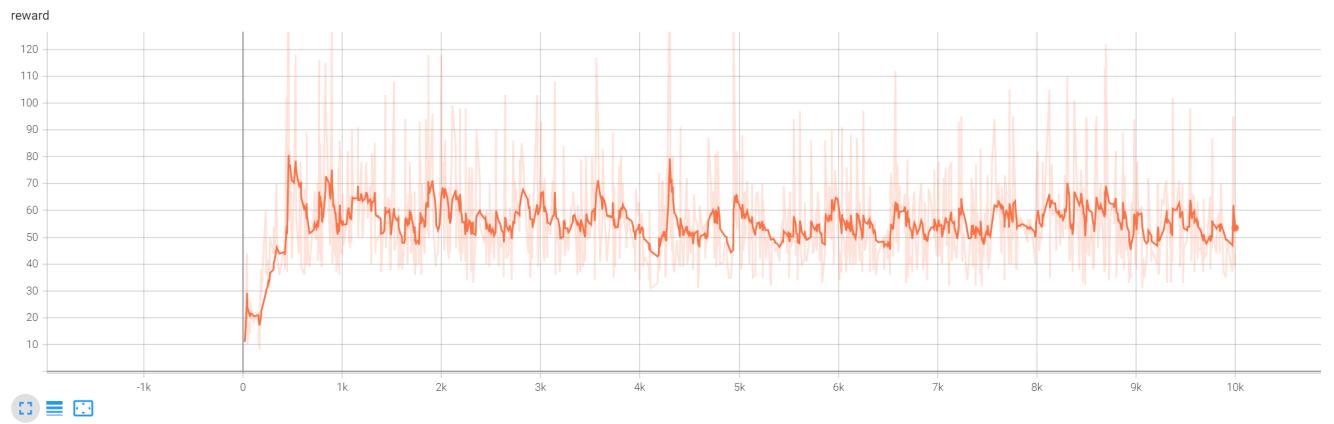


FIGURE 31 – CartPole, rewards, version TD(λ)

En comparant ces trois versions, on voit clairement que la version Rollout Monte-Carlo converge moins vites que les deux autres et que TD(λ) est plus stable par rapport au TD(0)

5.3 GridWorld

Nous considérons pour le problème de GridWorld, la version Rollout Monte-Carlo et la version TD(0) avec un discount de 0.9999, un pas d'apprentissage de 0.001 pour le réseau acteur et le réseau critic.

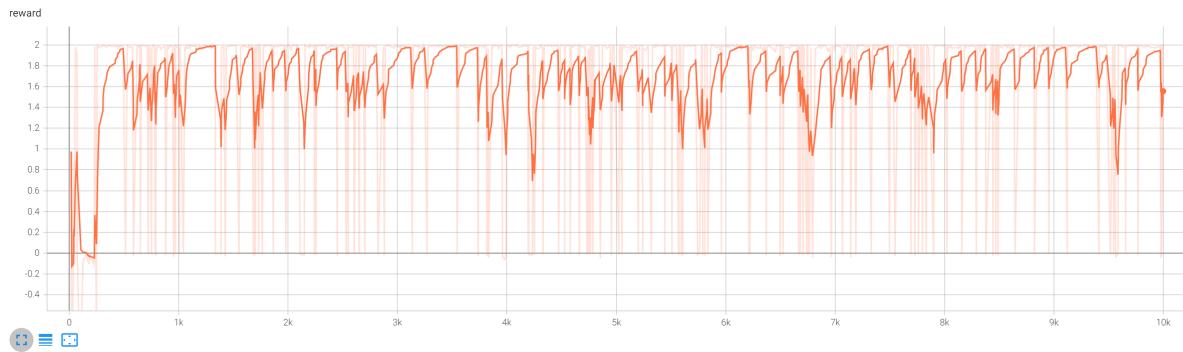


FIGURE 32 – GridWorld, nombre d’étapes, version Rollout Monte-Carlo

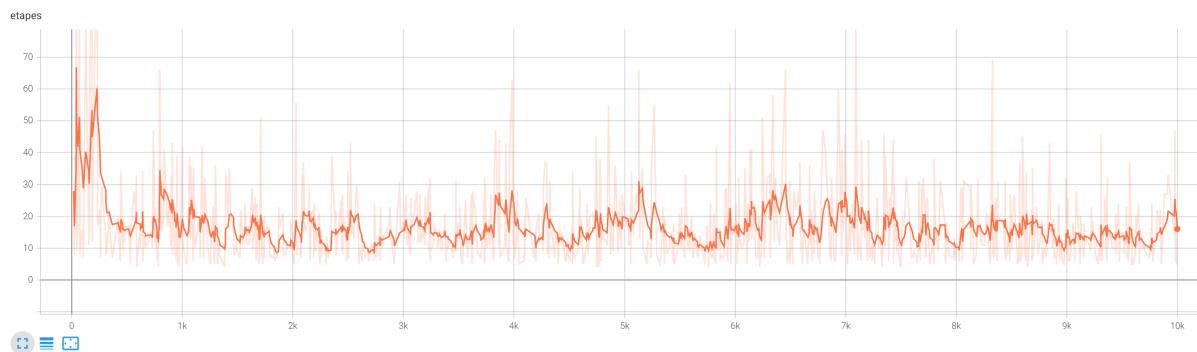


FIGURE 33 – GridWorld, rewards, version Rollout Monte-Carlo

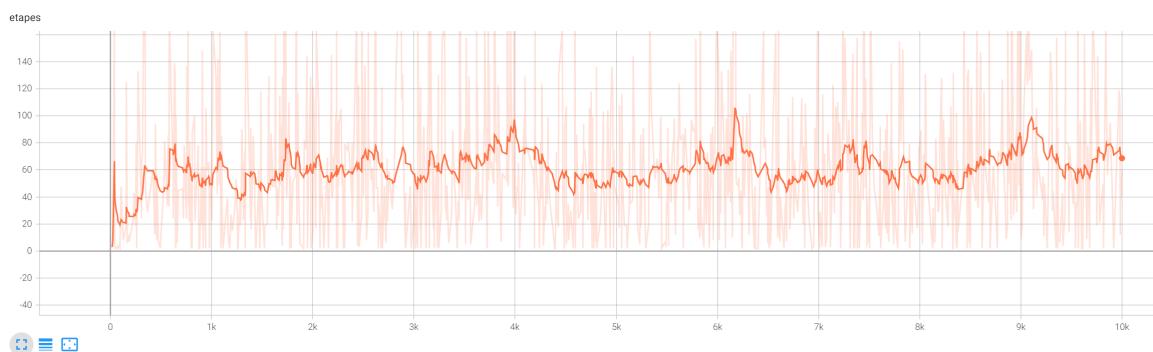


FIGURE 34 – GridWorld, nombre d’etape, version TD(0)

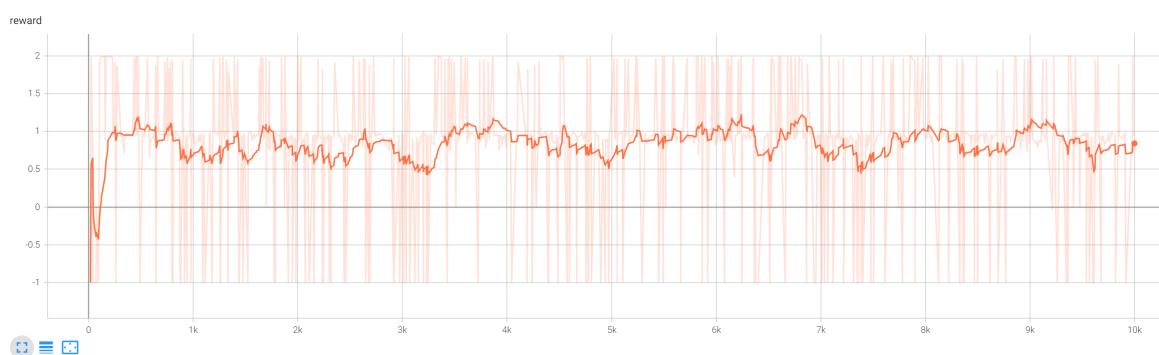


FIGURE 35 – GridWorld, rewards, version TD(0)

Pour la version TD(0), en comparant avec l’autre version, on constate que l’agent tend de

prendre trop de étapes avant d'arriver au point final. Donc on augmente la pénalisation des cases intermédiaire : de -0.001 au -0.01 et on obtient les courbes suivants :

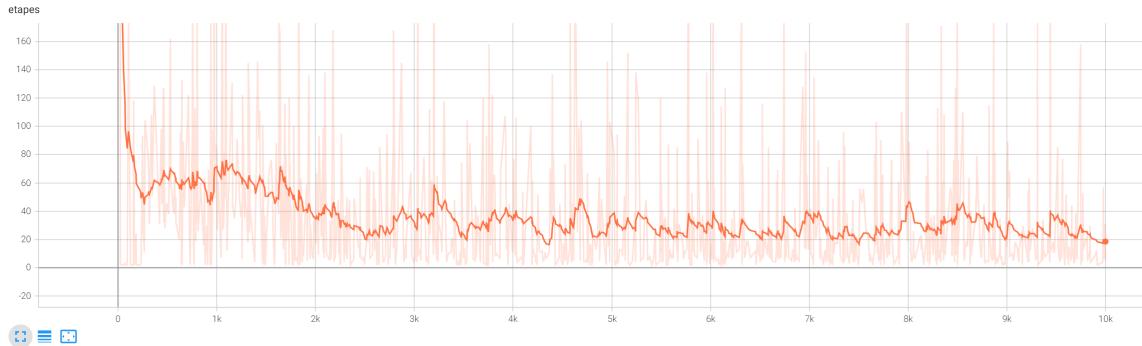


FIGURE 36 – GridWorld, nombre d'étape, version TD(0) modifiée

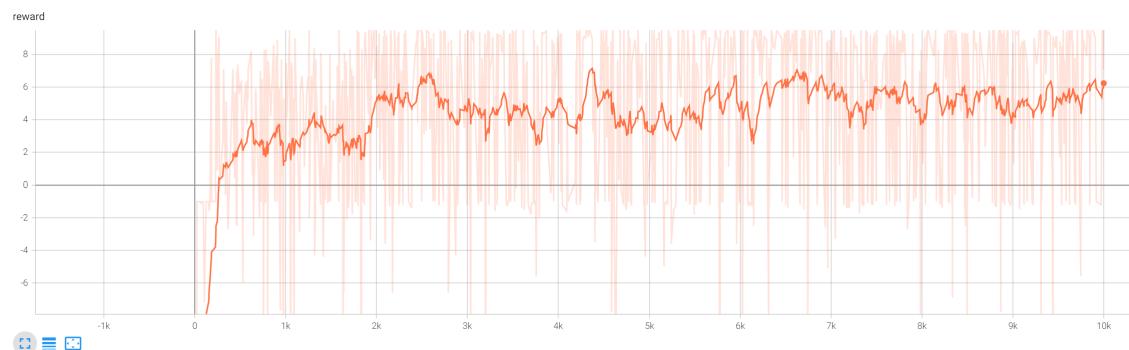


FIGURE 37 – GridWorld, rewards, version TD(0) modifiée

5.4 LunarLander

Pour le problème de LunarLander, nous avons testé la version la version Rollout Monte-Carlo et nous obtenons :



FIGURE 38 – LunarLander, rewards, version Rollout Monte-Carlo

5.5 Conclusion

D'après nos expériences sur ces trois jeux, nous constatons que la performance de modèle dépend des hyper-paramètres choisis et parfois, nous devons modifier les rewards dans les jeux pour rendre notre agent plus intelligent selon les situations

6 TME6 : Policy Gradient : PPO

6.1 Intro

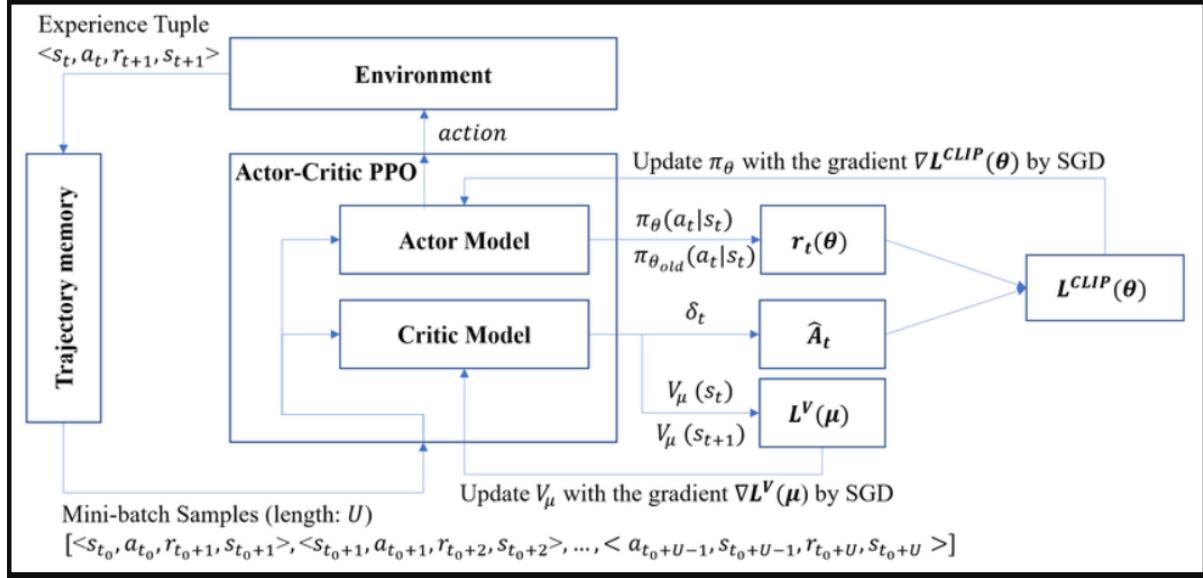


FIGURE 39 – PPO

PPO est basé sur l'architecture AC, CAD que PPO dispose également de deux réseaux, Actor et Critic.

Pour mettre à jour le réseau politique, ou pour utiliser la méthode de descente de gradient pour mettre à jour le réseau, nous devons avoir une fonction objective. Pour les réseaux stratégiques, la fonction objective est en fait relativement simple à donner, elle est très simple, et le résultat final ! C'est-à-dire Loss fonction est :

$$L(\theta) = \mathbb{E}(r_1 + \gamma r_2 + \gamma^2 r_3 + \dots | \pi(\cdot, \theta))$$

FIGURE 40 – PPO Loss Fonction

Au premier passage de boucle comment la distribution à évaluer et la distribution de trarget sont les mêmes, donc la divergence KL vaut 0. Si on prend K = 1, c'est la méthode policy gradient classique.

No clipping or penalty:

$$L_t(\theta) = r_t(\theta) \hat{A}_t$$

Clipping:

$$L_t(\theta) = \min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta)), 1 - \epsilon, 1 + \epsilon) \hat{A}_t$$

KL penalty (fixed or adaptive)

$$L_t(\theta) = r_t(\theta) \hat{A}_t - \beta \text{KL}[\pi_{\theta_{old}}, \pi_\theta]$$

FIGURE 41 – 3 versions PPO

6.2 CartPole

Pour le jeu de CartPole, nous avons testé les 3 versions différentes :

- La version sans KL ni clipped
- La version KL
- La version clipped

nous prenons un pas d'apprentissage pour le réseau actor 0.0001 et 0.0003 pour le réseau de critic. un batch de la taille du buffer, un paramètre de discount de 0.98, un paramètre de λ de 0.99, un paramètre K = 10, le target de la divergence KL de 0.01 et un paramètre $\epsilon = 0.1$ pour la version clipped.

et nous obtenons :

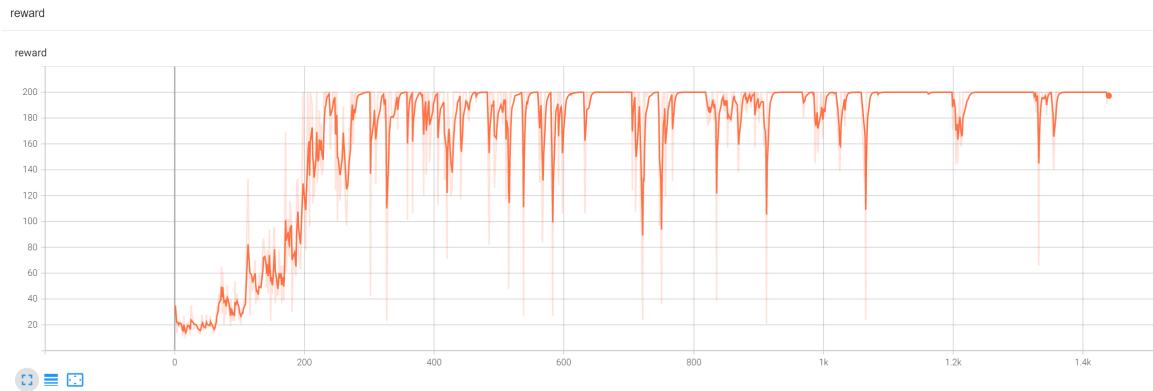


FIGURE 42 – Cartpole, rewards, version sans KL ni clipped

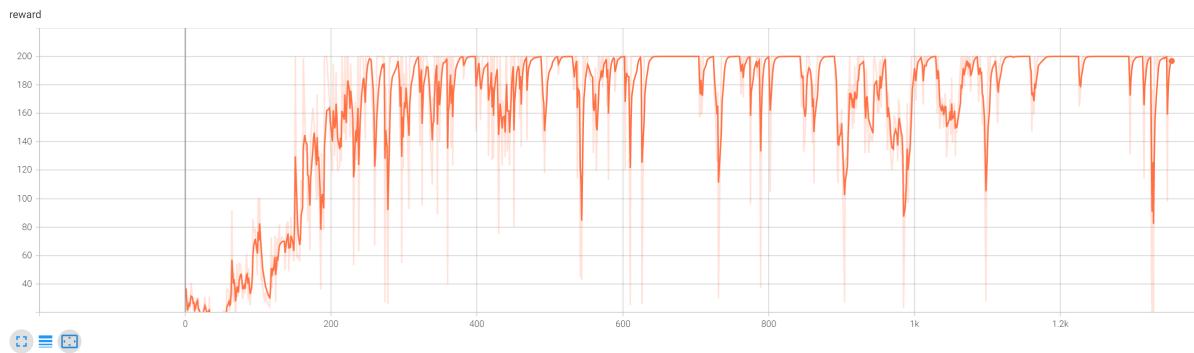


FIGURE 43 – Cartpole, rewards, version KL

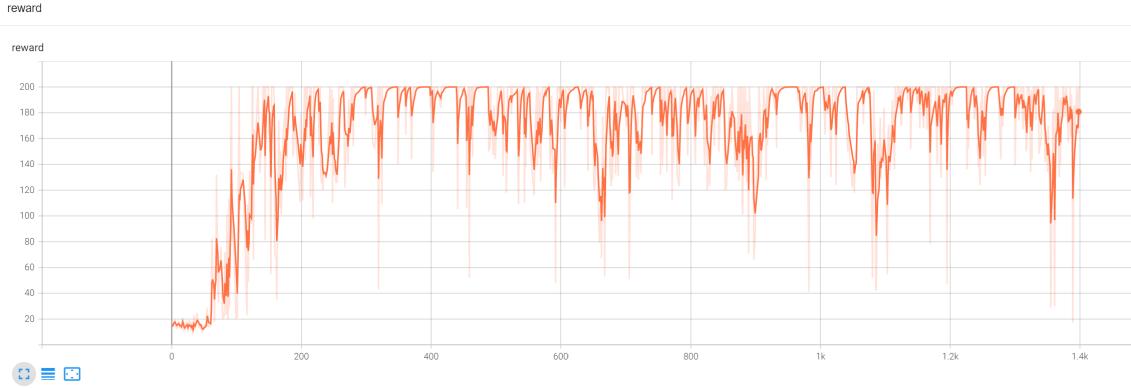


FIGURE 44 – Cartpole, rewards, version clipped

Nous constatons que, la version clipped converge le plus vite, la version KL est plus lisse pour converger et la version sans KL ni clipped est plus stable après la convergence

6.3 LunarLander

Pour le jeu de LunarLander, nous considérons aussi les 3 versions comme précédemment. Nous prenons un pas d'apprentissage pour le réseau actor 0.0002 et 0.0004 pour le réseau de critic. un batch de la taille du buffer, un paramètre de discount de 0.98, un paramètre de λ de 0.99, un paramètre $K = 3$, le target de la divergence KL de 0.01 et un paramètre $\epsilon = 0.1$ pour la version clipped. Nous obtenons :

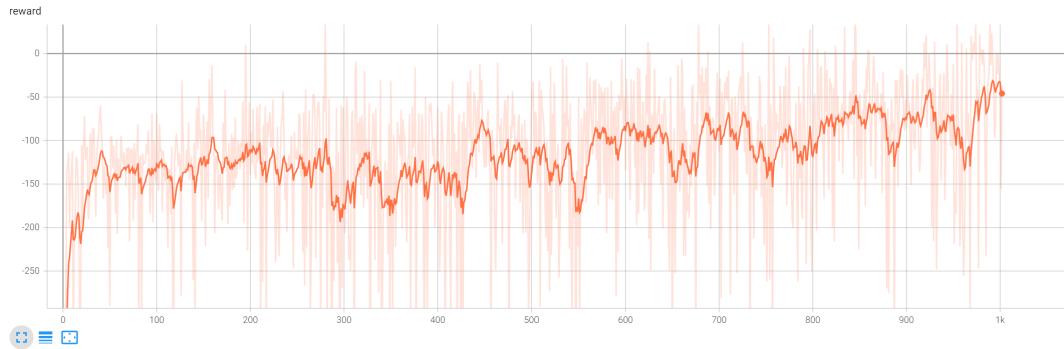


FIGURE 45 – LunarLander, rewards, version sans KL ni clipped

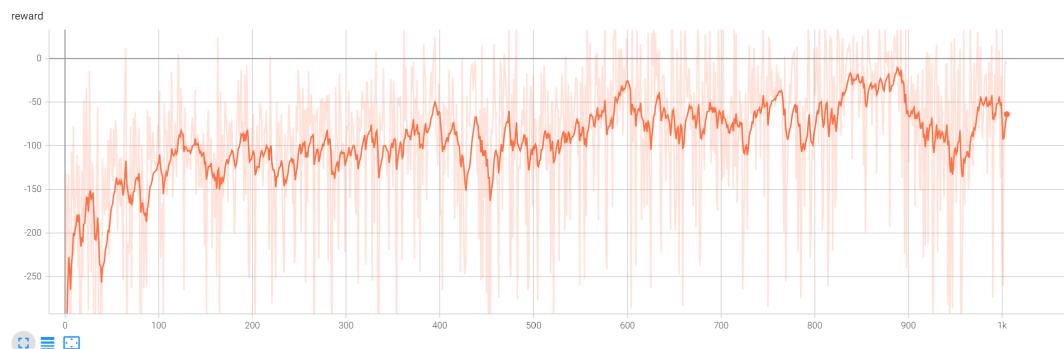


FIGURE 46 – LunarLander, rewards, version KL

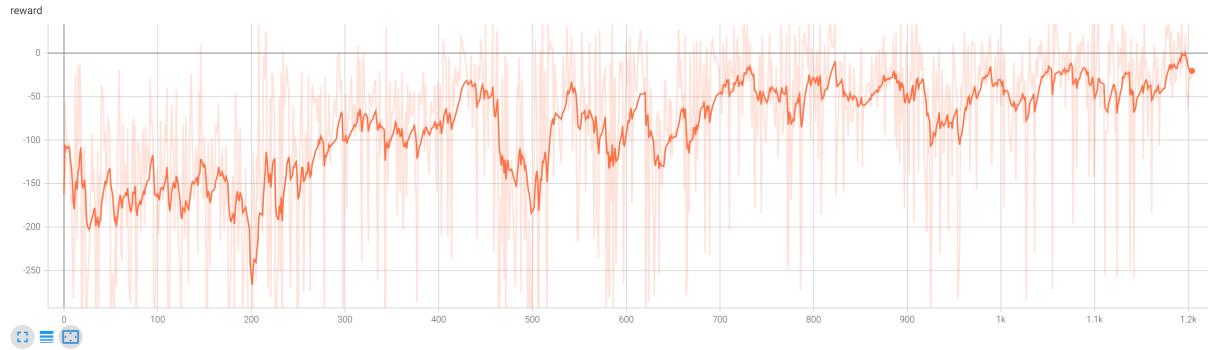


FIGURE 47 – LunarLander, rewards, version clipped

Nous constatons que pareil que le jeu de Cartpole, la version clipped a tendance de converge plus vite que les deux autres, celle sans KL ni clipped augmente les rewards le plus lentement.

Avec l'aide de la visualisation de l'environnement nous voyons que pour tous ces trois versions, au bout de 500 itérations,l'argent savoir bouger ces positions petit à petit au lieu de tomber par terre directement.

6.4 Conclusion

Le choix de la version d'erreur appris par le réseau de critic et ce qui est transmis entre le réseau de critice et le réseau d'actor affecte largement sur le comportement des courbes de rewards dans ces trois jeux.

7 TME7 Continuous Action DDPG

7.1 Intro

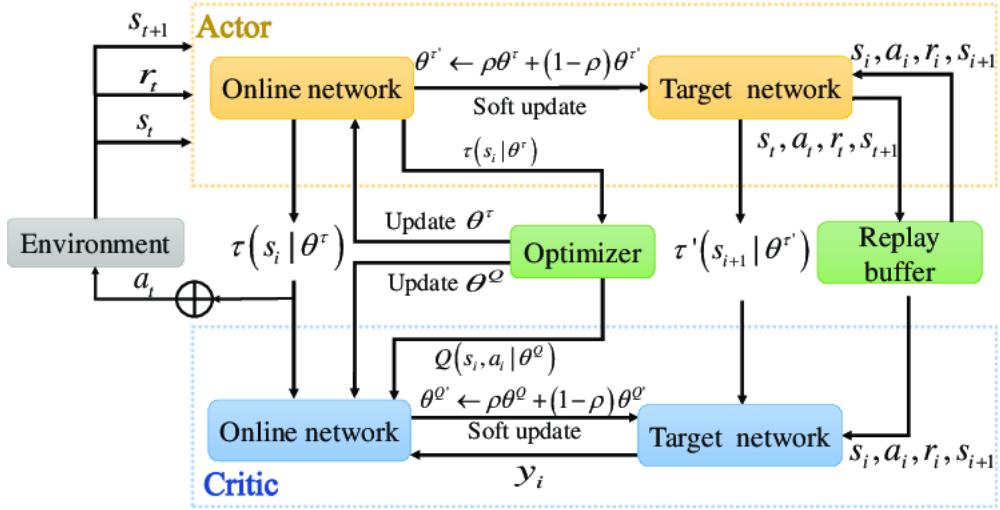


FIGURE 48 – DDPG

Ce TME a pour objectif d’expérimenter l’approche DDPG pour environnements à actions continues.

Le nom complet de l’algorithme d’apprentissage par renforcement DDPG est Deep Deterministic Policy Gradient, qui est essentiellement un algorithme d’apprentissage par renforcement du cadre AC.

En conclusion, ActorNet et CriticNet sont très importants. En DDPG, il y a NET avec Target_NET pour Actor et Critic.

7.2 MountainCarContinuous-v0

```

ENV_NAME = 'MountainCarContinuous-v0'
MAX_EPISODES = 200
MAX_EP_STEPS = 500/2000
LR_A = 0.001 # learning rate for actor
LR_C = 0.003 # learning rate for critic
GAMMA = 0.9 # reward discount
TAU = 0.01 # soft replacement
MEMORY_CAPACITY = 10000
BATCH_SIZE = 32

```

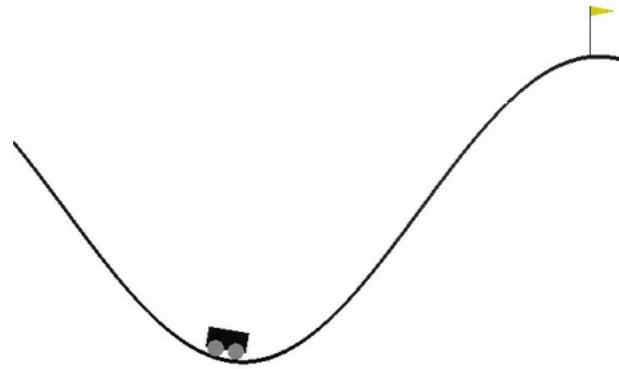


FIGURE 49 – Environnement MountainCarContinuous

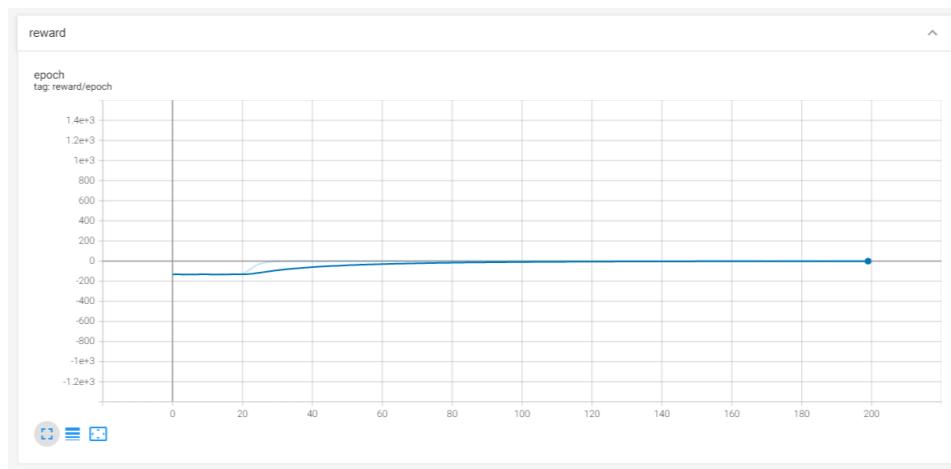


FIGURE 50 – Courbe Reward step 500

•Commentaire :

Ici, avec `max_step = 500`, on voit que le reward converge vers 0. CAD Car n'arrive pas dans le point final pendant l'exploration. S'il ne trouve pas toujours le point final. Afin d'éviter reward « 0 », il y a une tendance à rester immobile.

On verra le cas différent ci-dessous :

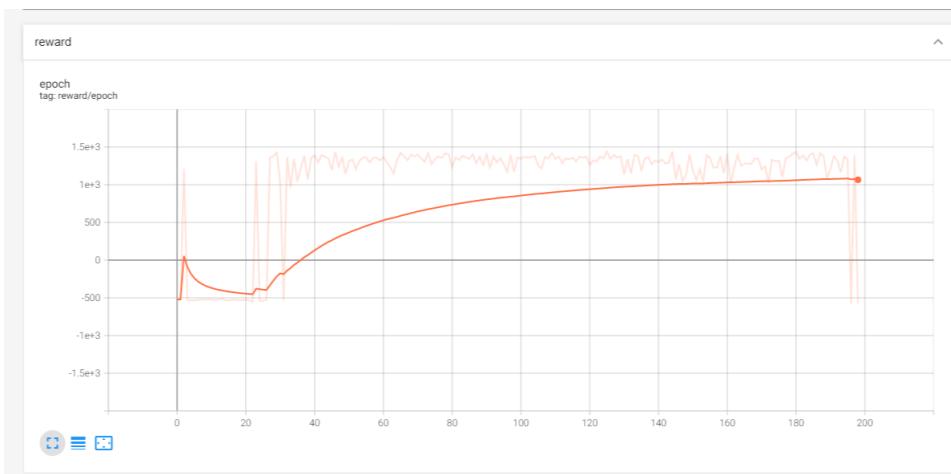


FIGURE 51 – Courbe Reward step 2000

•Commentaire :

Cette fois, le courbe converge à 1500, parce qu'on fixe reward == 1500 au point final. La chose la plus importante est d'attendre le point final pendant l'exploration. Sinon, le risque est de rester en place comme l'avance.

7.3 LunarLanderContinuous-v2

```
ENV_NAME = 'LunarLanderContinuous-v2'
MAX_EPISODES = 200
MAX_EP_STEPS = 2000
LR_A = 0.001 # learning rate for actor
LR_C = 0.002 # learning rate for critic
GAMMA = 0.9 # reward discount
TAU = 0.01 # soft replacement
MEMORY_CAPACITY = 10000
BATCH_SIZE = 32
```

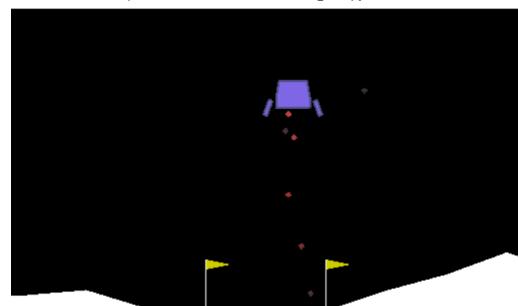


FIGURE 52 – Environnement Pendulum

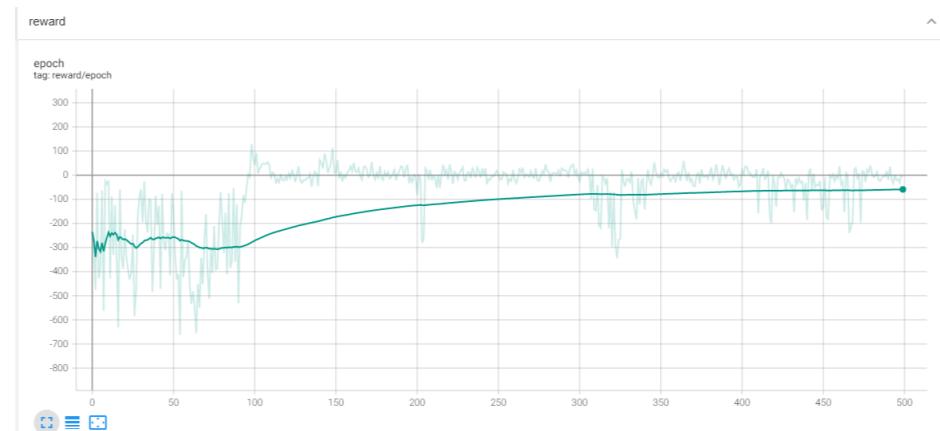


FIGURE 53 – Courbe Reward

- Commentaire :

Cette tâche est la plus difficile. Mais notre résultat a empêché le crash du vaisseau spatial.
Ce n'est pas mal.

7.4 Pendulum-v0

```
ENV_NAME= 'Pendulum-v0'
MAX_EPISODES = 200
MAX_EP_STEPS = 200
LR_A = 0.001 # learning rate for actor
LR_C = 0.002 # learning rate for critic
GAMMA = 0.9 # reward discount
TAU = 0.01 # soft replacement
MEMORY_CAPACITY = 10000
BATCH_SIZE = 32
```



FIGURE 54 – Environnement Pendulum

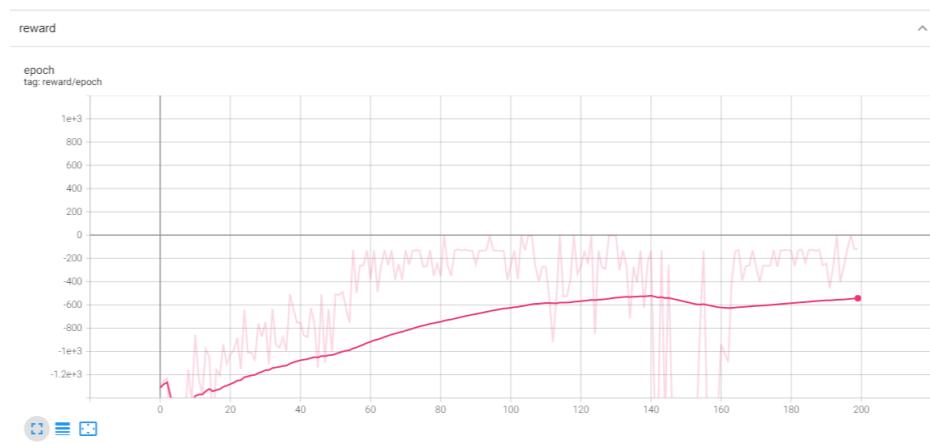


FIGURE 55 – Courbe Reward

- Commentaire :

Le résultat n'est pas mal, parce que reward = 0 c'est la limite dans cet environnement.

7.5 Conclusion

Nous avons testé dans trois environnements, les résultats sont pas mal en regardant les Courbes Reward. On trouve que les **hyperparamètres** sont beaucoup importants dans les différents environnements.

Par exemple, dans MountainCarContinuous-v0, si `max_step` est petit. Donc pendant l'exploration, c'est difficile d'attendre le terminal. Il choisit de rester immobile.

En revanche, s'il peut savoir il y a un terminal avec gros reward, il essaie toujours d'aller au terminal. C'est comme on a parlé avant.

8 TME8 Soft Actor-Critic (SAC)

8.1 Intro

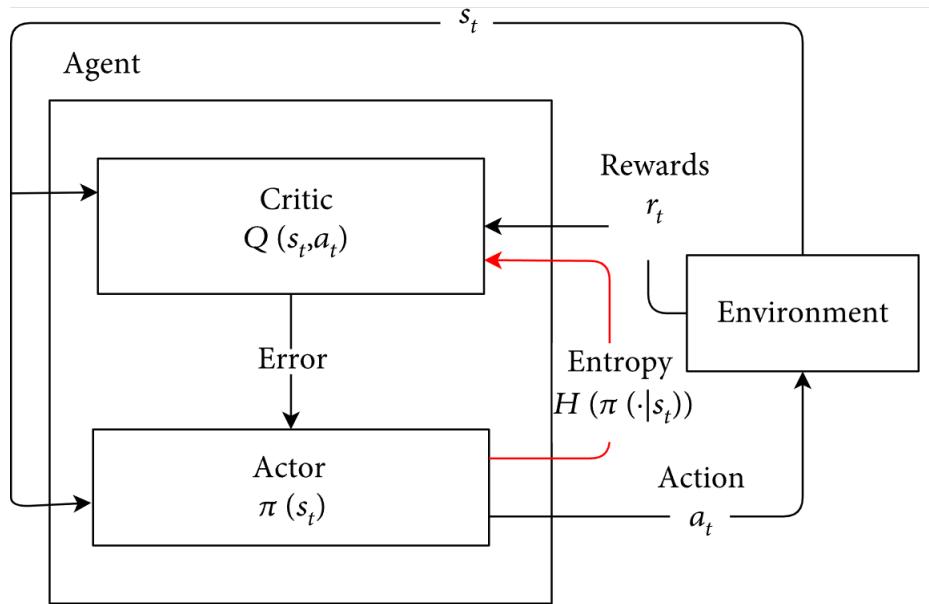


FIGURE 56 – SAC

Ce TME a pour objectif d’expérimenter l’approche SAC, pour environnements à actions continues, avec maintien d’entropie. Il est basé sur DDPG. En effet, on ajoute plus de réseaux et utilise l’idée d’entropie.

On résume :

Actor-critic framework (approximate policy and value function/ q-function by two networks)

Off-policy (Improve the efficiency of sample usage)

Entropy to ensure stable and exploration

```

gamma=0.9
lr=1e-3
alpha=0.2
MAX_EPISODE = *** #dependant d'environnement
MAX_STEP = *** #dependant d'environnement
update_every = 100 # fréquence replay_buffer
batch_size = 50 # size of replay_buffer

```

8.2 MountainCarContinuous-v0

Comparer avec 7.2 de DDPG

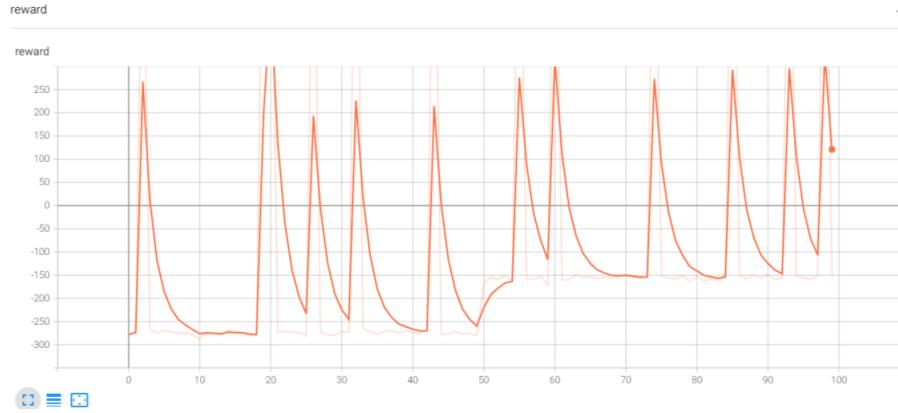


FIGURE 57 – Courbe Reward

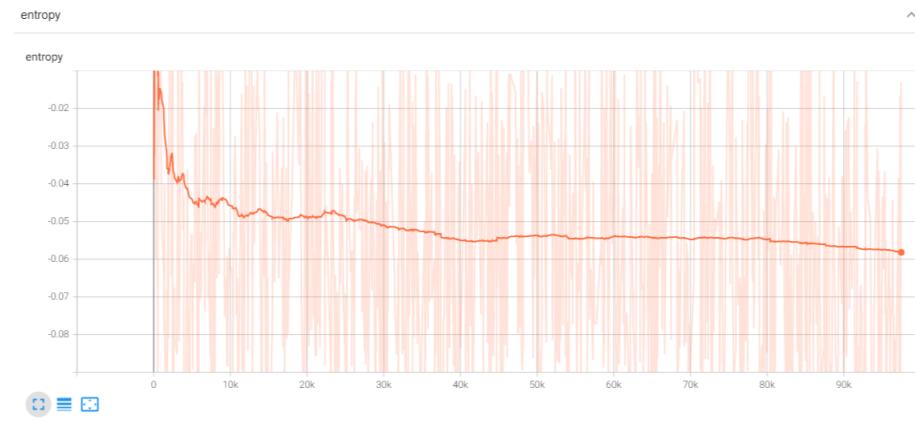


FIGURE 58 – Courbe policy loss

•Commentaire :

Par rapport à DDPG, ce n'est pas bien. Parce que cet environnement est particulier. Il a besoin de savoir qu'il y a un terminal avec gros reward, donc MAX_STEP est grand. Mais il y a plusieurs NETS dans SAC, puis on trouve que l'exécution est trop lente. Mais nous voyons probablement que la tendance est bonne.

On verra les 2 autres environnements ci-dessous, ça marche mieux.

8.3 LunarLanderContinuous-v2

Comparer avec 7.3 de DDPG

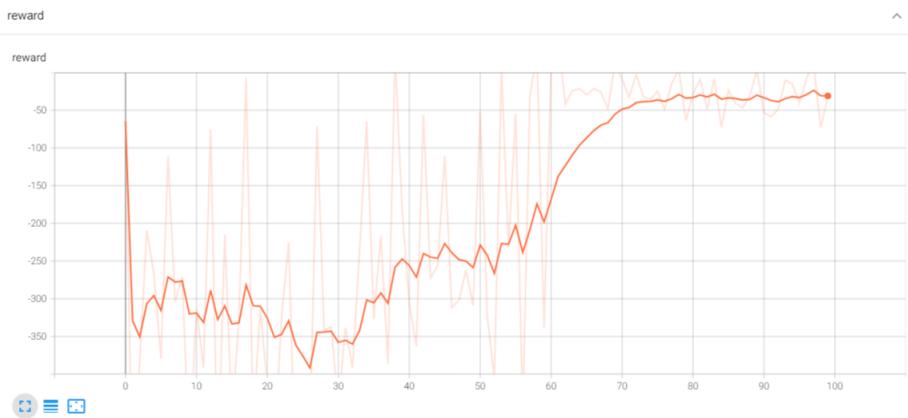


FIGURE 59 – Courbe Reward



FIGURE 60 – Courbe policy loss

•Commentaire :

Ça marche pas mal. Cette tâche est plus difficile. Comparé avec DDPG, c'est mieux. Même si nous n'avons couru que 100 époques, la tendance de courbe reward est plus évident que celui de DDPG.

8.4 Pendulum-v0

Comparer avec 7.4 de DDPG

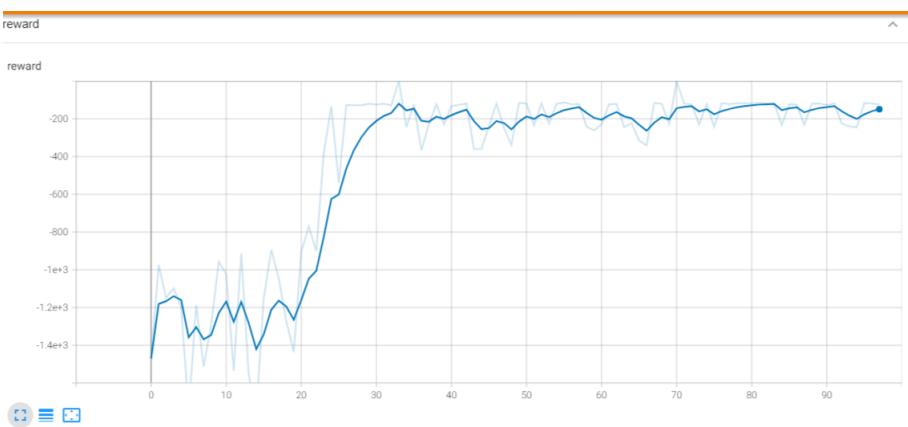


FIGURE 61 – Courbe Reward

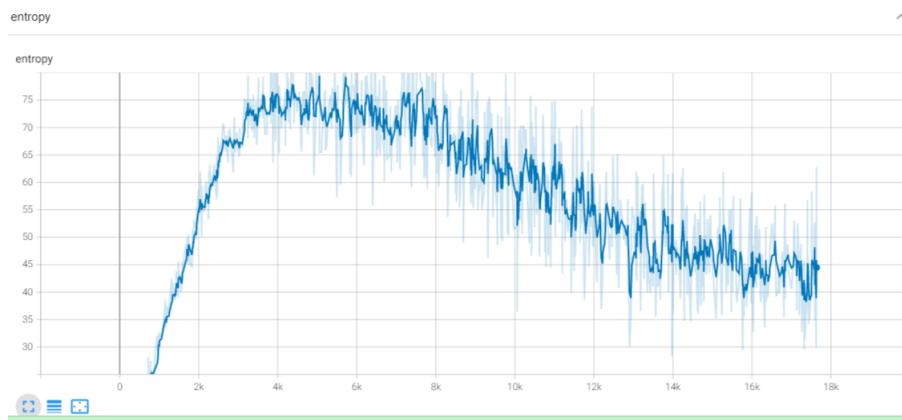


FIGURE 62 – Courbe policy loss

•Commentaire :

Dans ce cas-là, la tâche et l'environnement ne sont pas très compliqués et MAX_STEP peut être ajusté pas grand.

Par rapport à DDPG, SAC est beaucoup mieux cette fois.

8.5 Conclusion

SAC est le modèle le plus compliqué. Dans Pendulum-v0, SAC a montré de bons résultats. Nos résultats sont pas mal.

Mais à cause du temps et de la limite de matériel informatique, il reste beaucoup à améliorer.

9 Résumé final

Ces TMEs nous permet d'avoir la première expérience dans le monde de Reinforcement Learning et nous donne la chance d'implémenter par nous même les state-of-art dans ce domaine.

Enfin, grâce à des informations sur l'internet et les aides de prof Sylvain Lamprier et Nicolas Baskiotis, nous pouvons finalement finir les travaux.

Ce qui reste à faire dans notre futur développement c'est de prendre plus de temps sur les jeux relativement difficiles pour voir si son comportement est comme ce que nous voyons par les premières itérations et de développer éventuellement les autres variantes de notre méthode.

Nous vous remercions beaucoup.

