# RLD : Compte-rendu des TPs

Arij Riabi et Nicolas buton

15 février 2020

## 1 TME 1 : Problème de bandits

Dans ce premier TP on s'intéresse aux problèmes d'apprentissage par renforcement dans lequel **les actions ne changent pas la configuration de monde** : Bandits manchots. Le problème est le suivant :
On dispose de K machines et on suppose que chacune a une distribution différente et on souhaite obtenir le meilleur gain cumulé au bout d'un certain nombre d'itérations.

Dans le cadre de ce TP, on dispose des taux de clics sur les publicités de 10 annonceurs pour 5000 articles, ainsi que les profils de ces articles (les contextes). L'objectif est de choisir la publicité d'un descde maximiser le taux de clics cumulés sur les 5000 visites.

L'enjeu de ce problème est que l'on ignore les distributions des machines et ce n'est qu'en jouant que l'on peut obtenir des informations à partir des gains reçus. En particulier, la clef de ces méthodes est **le dilemme exploration/exploitation**. D'une part, on veut choisir les machines ayant donné de bonnes récompenses dans le passé, d'autre part une machine peu explorée peut donner de très bons gains, c'est juste que l'on n'a pas eu de chance quand on a joué avec cette machine.

Pour évaluer nos algorithmes, on commence par mettre en place trois baselines, dont deux ne sont pas réalisables si on ne connaît pas au préalable les taux de clics de tous les annonceurs à chaque itération.

Nos **baselines** sont les suivantes :

- **stratégie random** : à chaque itération, on choisit n'importe quel annonceur
- **stratégie StaticBest** : à chaque itération, on choisit l'annonceur avec le meilleur taux de clics cumulés
- **stratégie optimale** : à chaque itération, on choisit l'annonceur ayant le meilleur taux de clics à cette itération.

Voici les stratégies que nous avons implémenté :

**UCB**. La stratégie UCB consiste à former des intervalles de confiance indiquant une zone de gain probable, et à **prendre la distribution ayant la plus grande borne supérieure**, qui est le reward sommée à la moyenne empirique du gain associée à une action.

$$\mu_{i,s} \leq \hat{\mu}_{i,s} + \sqrt{\frac{2 \log t}{s}} \tag{1}$$

avec $s$ le nombre de fois où l'action $i$ a été choisie.

On choisit l'action qui maximise la borne de confiance supérieure :

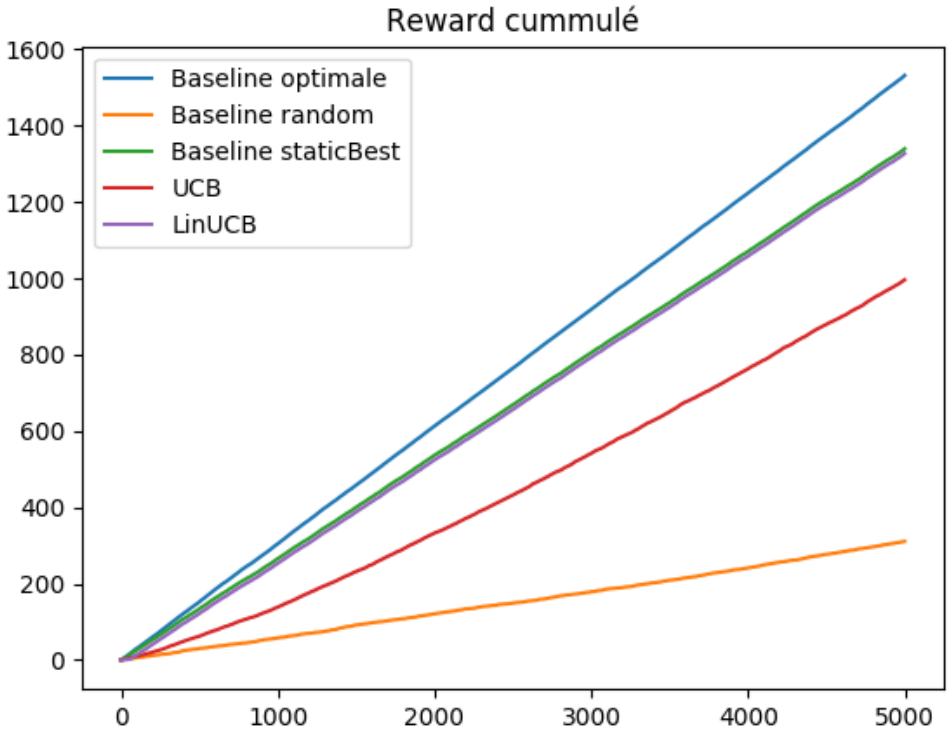
$$\pi_t = \arg\max_i B_{t,T_i(t-1)}(i) \tag{2}$$

**LinUCB**. L'algorithme LinUCB est une **variante de UCB**, où l'on prend en considération le contexte de la décision à chaque instant. L'idée est de faire **une régression sur chaque machine afin d'estimer le gain retourné par la machine pour un contexte donné**. La régression prend en entrée le contexte et on applique une régularisation RIDGE.

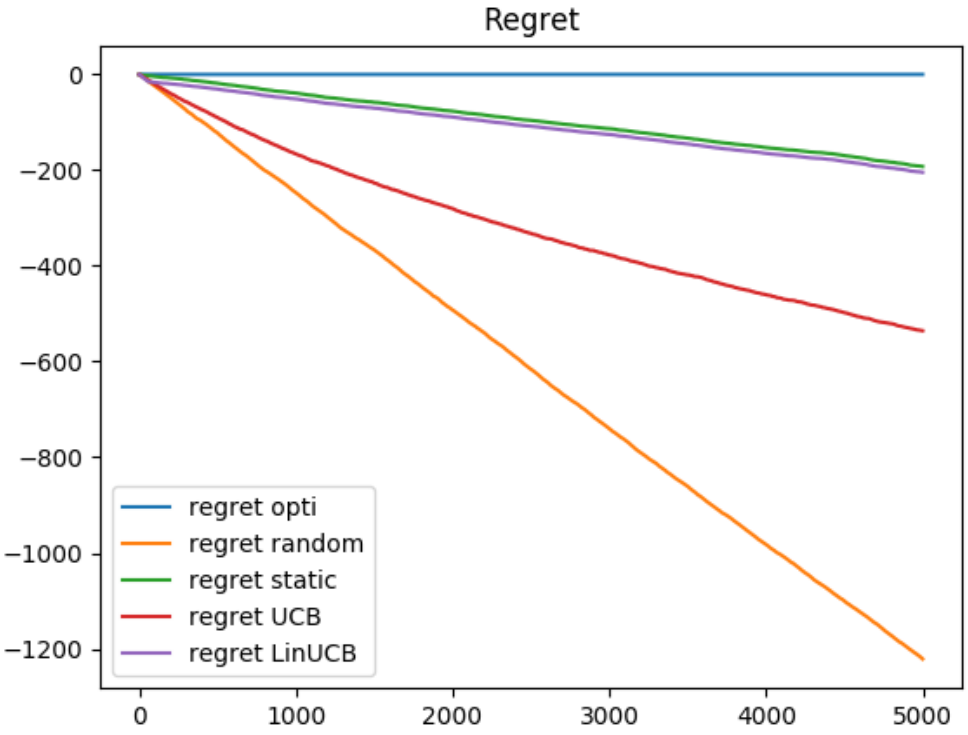
On compare ces baselines avec les stratégies UCB Upper-Confidence Bound et LinUCB Linear Upper-Confidence Bound.

La figure 1 présente des courbes de reward/regret cumulé obtenues à chaque itération. On remarque que la courbe des gains cumulés de LinUCB est bien toujours au dessus de la courbe UCB, elle est même très proche de la baseline staticBest et bien meilleure que la baseline random. Ainsi prendre en compte les contextes permet d'avoir un cumul de gains bien plus important.

Dans la courbe de regret on peut voir que la pires des stratégies testé est la stratégies random et la meilleur est évidement la stratégies optimales avec 0 de regret puisque c'est notre référence(pour le calcul du regret). Ensuite nous avons la stratégies UCB que l'on vois s'améliorer notablement avec le temps, et la stratégies LinUCB qui elle s'améliore plus rapidement que UCB et arrive presque au score de la stratégie statique.




(a) Courbes des gains cumulés obtenus avec chaque stratégie




(b) Courbes des regrets cumulés obtenus avec chaque stratégie

FIGURE 1 – Cumul sur 5000 visites

# 2 TME 2 : Policy and Value iteration

Lorsque l'on a des espaces d'actions et d'états petits ou du moins finis, on cherche à les représenter par une MDP. Un MDP est composé de 4 éléments :

— un ensemble S d'états

— un ensemble A d'actions

— une distribution de probabilité P(s' | s, a) sur les états d'arrivée étant donné un état initial et une action effectuée.

— une espérance de gain R(s, a, s') pour chaque transition possible

Quand **le MDP est connu**, nous pouvons déterminer la politique optimale **sans interagir avec l'environnement**, nous sommes dans un cas d'offline planning. Nous pouvons utiliser les algorithmes **Policy iteration** et **Value iteration**, qui utilisent **l'équation de Bellman** pour permettre de trouver les valeurs optimales V ainsi que la politique optimale associée. La différence entre ces deux algorithmes est que **value iteration commence d'abord par optimiser la fonction qui définit la valeur des états jusqu'à convergence**, puis définit la politique qui en résulte. Tandis que **policy iteration optimise de façon jointe la fonction d'approximation des valeurs et la politique**.

**Résultat sur la carte 0 :**

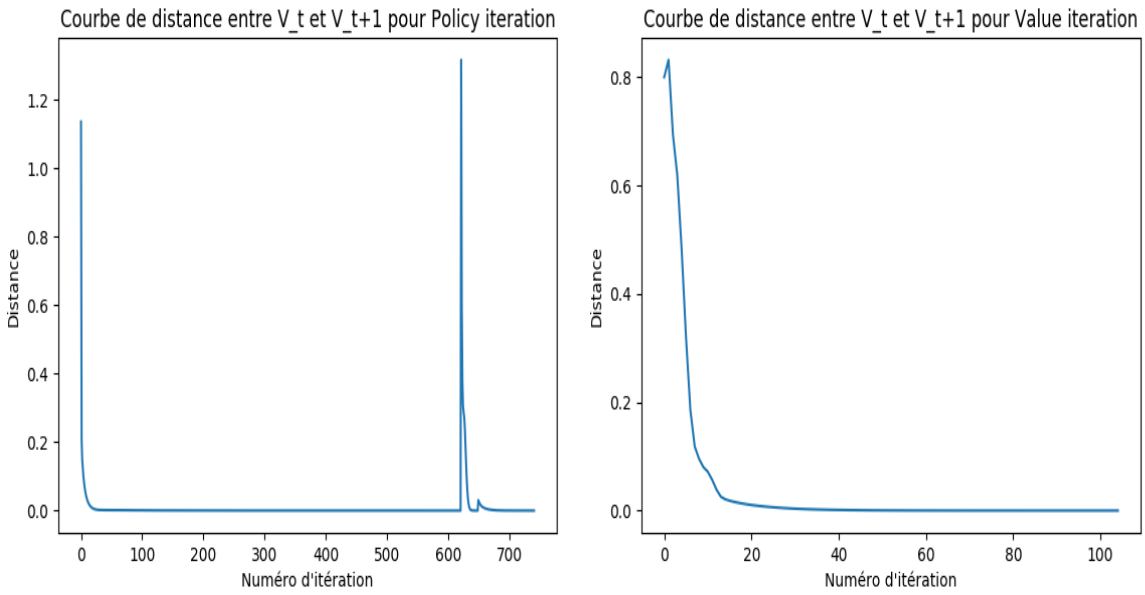



FIGURE 2 – Courbe de différence entre V_t et V_t+1

Hyperparamètres utilisés :

— $\epsilon = 1e - 6$

— $\gamma = 0.99$

On peut remarquer qu'à l'itération 1500 nous avons un pic sur la distance entre $V\_t$ et $V\_t + 1$ pour l'algorithme de Policy iteration, ce qui est normal car cela correspond au moment où l'on change la politique. Sinon, pour les deux courbes on remarque bien une décroissance comme voulu de cette distance.

**Comparaison des deux algorithmes :**

| Algorithme | Temps du fit | nb maj value | nb maj politique | reward moyenne |
|---|---|---|---|---|
| Policy iteration | 0.0137s | 741 | 3 | 0.9809 |
| Value iteration | 4.816e-5s | 105 | 1 | 0.9800 |

On peut voir que dans notre expérience, Policy iteration est un peu plus rapide malgré un nombre de mise a jours plus élevé, ce qui peut s'expliquer par le fait que dans policy iteration nous n'avons pas besoin de faire un max pour mettre a jours les valeurs.

**Test du changement de $\gamma$ :**
On peut remarquer que si l'on diminue le $\gamma$, l'aglorithme va priviligier les chemins courts même si ceux-ci sont dangereux, alors qu'à l'inverse, si $\gamma$ tend vers 1, il va être plus prudent même si la trajectoire est plus longue. Ceci s'explique par le fait que $\gamma$ quantifie à quel point nous prenons en compte les rewards plus loin dans le futur.

**Test changement de carte :**
On effectue des tests sur toutes les cartes et on moyenne la reward sur 100 épisodes.

| Numero plan | Temps P.I. | Reward P.I | Temps V.I. | Reward V.I. |
|---|---|---|---|---|
| 0 | 0.0137 | 0.9809 | 4.81e-5 | 0.9800 |
| 1 | 0.0209 | 1.9794 | 7.152e-07 | 1.976 |
| 2 | 0.07959 | 1.892 | 4.768e-07 | 1.852 |
| 3 | 0.0127 | 0.9941 | 4.768e-07 | 0.9942 |
| 4 | 0.1788 | -2.001 | 4.768e-07 | -2.001 |
| 5 | 0.3724 | 1.943 | 4.768e-07 | 1.943 |
| 6 | 2.292 | 1.940 | 2.3841e-07 | 1.941 |
| 7 | 3.046 | 1.289 | 4.768e-07 | 1.206 |
| 8 | 8.061 | 0.9302 | 4.768e-07 | 0.9305 |

Nous n'avons pas pu tester la carte 9 car l'erreur suivante apparaissait : RecursionError : maximum recursion depth exceeded while getting the repr of an object .

On remarque que value itération est beaucoup plus rapide à performance égale, mais ceci peut être à cause de la façon dont nous avons codé les deux algorithmes. De plus aucun des deux agents n'arrive à apprendre une bonne stratégie sur la carte 4.

# 3 TME 3 : Q-Learning : value-based models

En général, on ne connaît pas le MDP, ainsi l'agent doit interagir avec son environnement pour comprendre le fonctionnement de ce dernier. Deux paradigmes sont possibles :

— l'agent va chercher à reconstruire le monde en collectant des trajectoires → **Model based RL**

— l'agent va directement essayer à maximiser son gain sans chercher à avoir une représentation de monde → **Model free RL**

Les algorithmes étudiés dans ce TP sont :

— **Q-Learning** Méthode indépendante de l'environnement. Seule la connaissance du nombre d'états est nécessaire : on utilise une fonction mesurant la qualité d'une combinaison état-action.

— **Sarsa** La version on-line de Q-learning.

— **Dyna-Q** C'est un algorithme hybride car il cherche à apprendre P et R et qui utilise aussi la Q-value.

Nous testons ces algorithmes sur le problème de gridworld. La figure 3 contient les courbes d'apprentissage sur 1000 épisodes des rewards cumulés calculés globalement pour chaque agent.

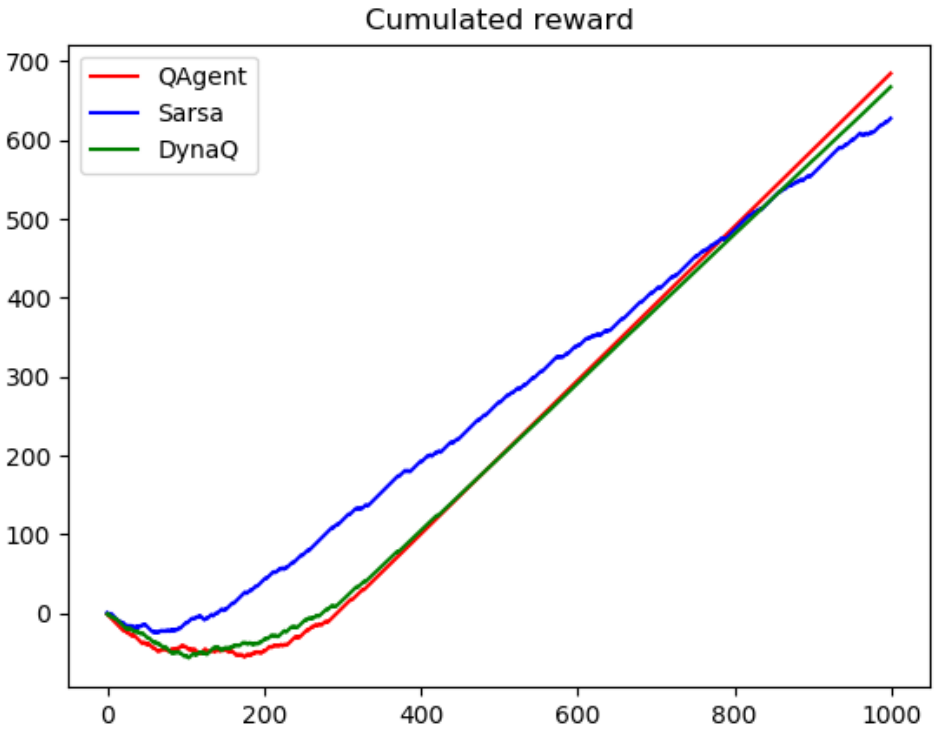



FIGURE 3 – Comparaison de performances des agents

Les performances obtenues par les agents relèvent de l'aléatoire mais la tendance générale obtenue est que l'algorithme Sarsa converge plus rapidement, les deux autres sont presque similaires .

Les hyper-paramètres utilisés sont :

- Le paramètre $\epsilon$ pour l'exploration $\epsilon$-greedy que l'on initialise à 1 et fait décroître exponentiellement (d'un facteur 0.999).
- Le paramètre $\alpha$ des agents que l'on initialise à 0.5 et fait décroître exponentiellement (d'un facteur 0.9995) afin que les politiques convergent .

Les algorithmes étudiés dans ce TP utilise une version tabulaire du monde (des états discrets) et s'appuient sur des estimations individuelles des valeurs pour chaque état-action.

# 4 TME 4 : Deep Q-Learning : problèmes à états continus

Quand les états sont continus, **on ne peut pas apprendre de matrice Q. On peut alors se servir d'un réseau de neurones pour représenter la fonction Q**. Le réseau prend en entrée un vecteur représentant l'état et retourne la valeur de chaque action.

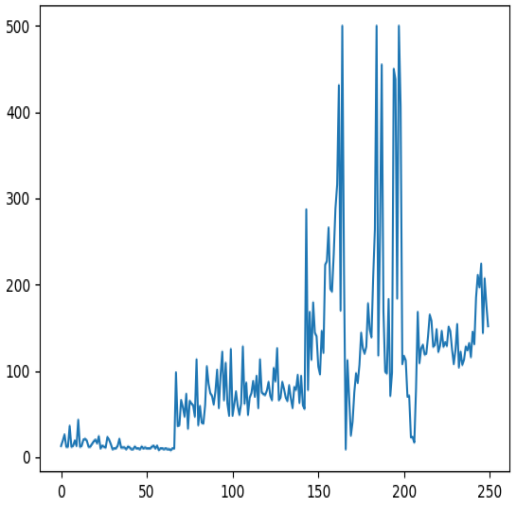
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\ r_t + \gamma\, max_{a' \in A(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a_t)] \quad \text{(Q-Learning)}$$
$$\theta \quad \leftarrow \quad \theta \ - \ \alpha\, \nabla_\theta\ [(r_t + \gamma\, max_{a' \in A(s_{t+1})} Q(s_{t+1}, a') - Q(s_t, a_t))^2] \quad \text{(Deep Q-Learning)}$$

FIGURE 4 – Comparaison entre les algorithmes Q-Learning et DQN

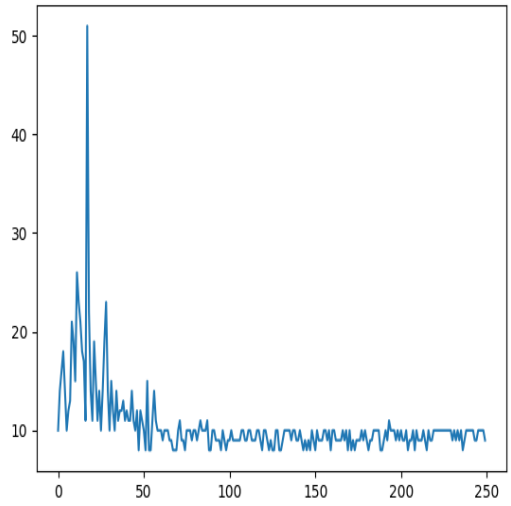
**Test de DQN sur les trois environnements** : GridWorld, Cartpole et LunarLander. Nous allons comparer les courbes moyennes des rewards pour chaque environnement avec les 4 combinaison d'hyperparamètres possibles (avec ou sans l'experience replay et le target network).

Hyperparamètres :

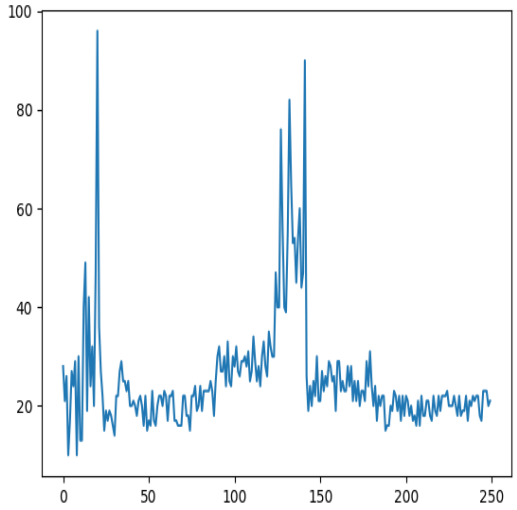
- miniBatchSize = 64
- $\gamma = 0.9$
- stepMAJ=10



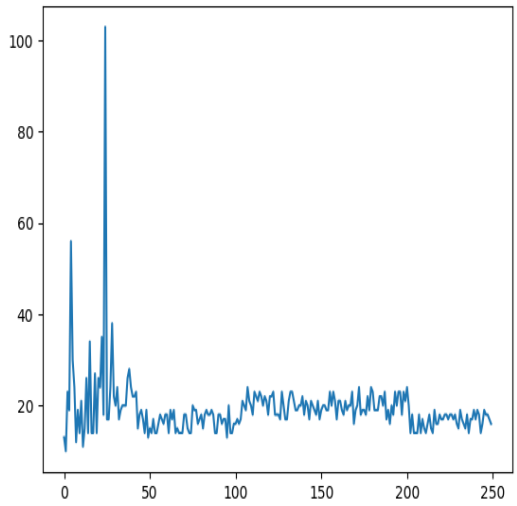

(a) Avec expérience replay et avec target network




(b) Avec expérience replay et sans target network

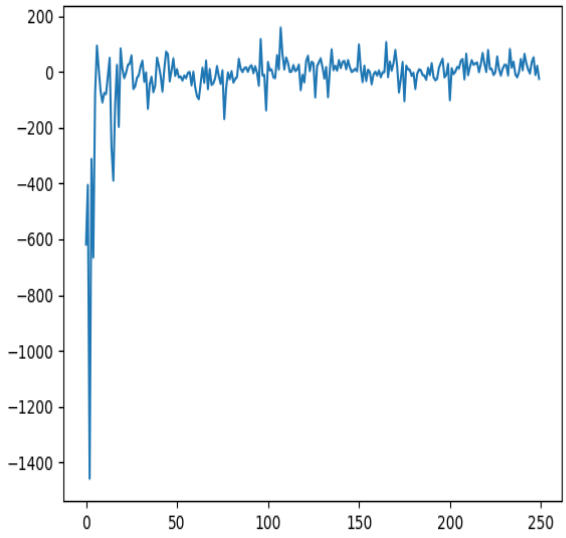



(c) Sans expérience replay et avec target network

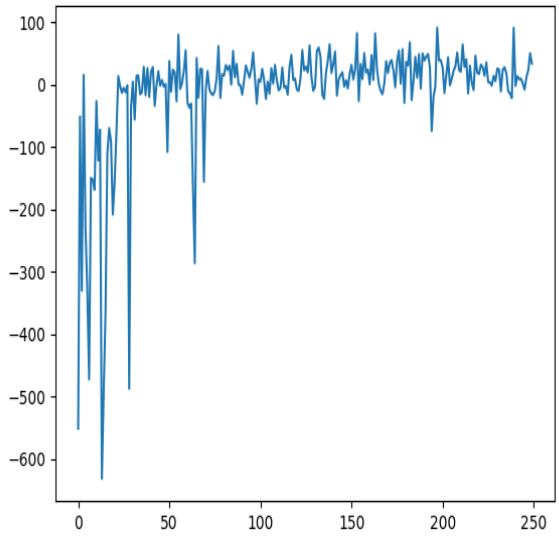



(d) Sans expérience replay et sans target network

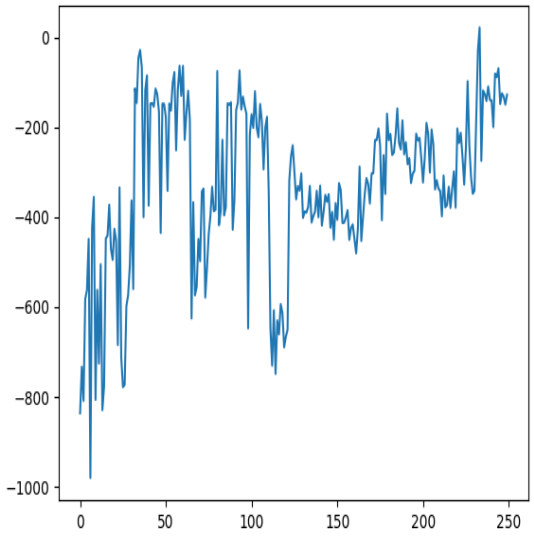
FIGURE 5 – Courbe de reward selon les différentes configurations pour l'environement Cartpole



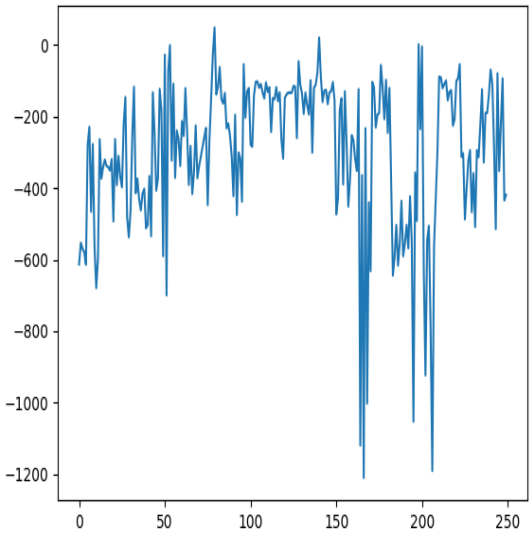

(a) Avec expérience replay et avec target network




(b) Avec expérience replay et sans target network

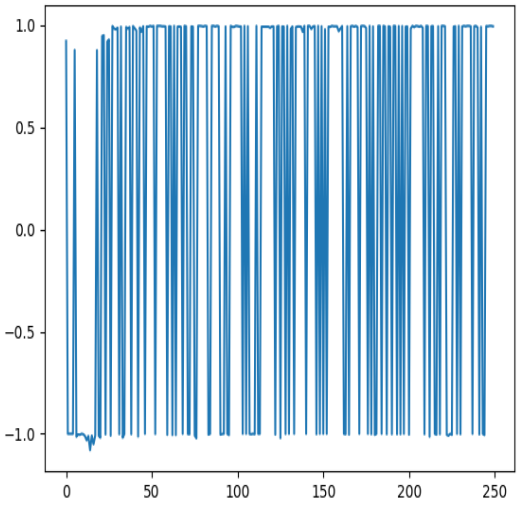



(c) Sans expérience replay et avec target network

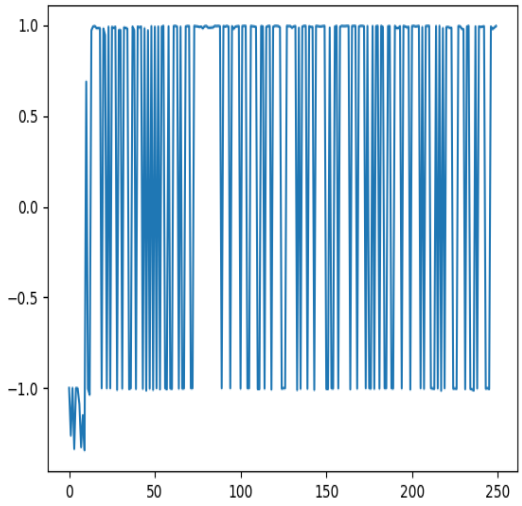



(d) Sans expérience replay et sans target network

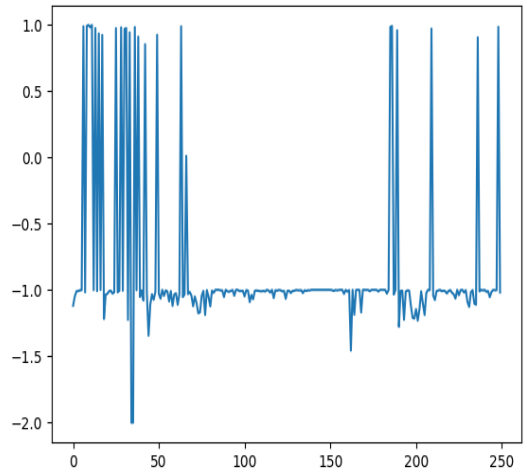
FIGURE 6 – Courbe de reward selon les différentes configurations pour l'environement Lunar Lander



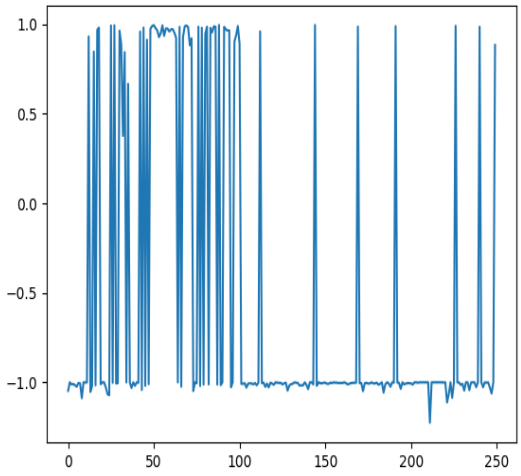

(a) Avec expérience replay et avec target network




(b) Avec expérience replay et sans target network




(c) Sans expérience replay et avec target network




(d) Sans expérience replay et sans target network

FIGURE 7 – Courbe de reward selon les différentes configurations pour l'environement Gridworld

| Environnement/Amélioration | ER/TN | ER | TN | No |
|---|---|---|---|---|
| Cartpole | 500 | 50 | 98 | 102 |
| LunarLander | 198 | 97 | 2 | 2 |
| Gridworld | 1 | 1 | 1 | 1 |

D'après les graphes précédents on peut en déduire que le target network et l'expérience replay sont tout deux nécessaires pour obtenir de bons résultats.

# 5 TME 5 : PolicyGradients avec A2C

Basé sur les méthodes **policy-based** qui cherchent à améliorer la politique $\pi_\theta$, avec $\theta$ l'ensemble des paramètres (généralement un réseau de neurones), l'algorithme *Advantage Actor Critic (A2C)* est composé de :

- **Actor** $\rightarrow$ Policy-based, apprend à prendre les décisions , c'est notre $\pi$
- **Critic** $\rightarrow$ Value-based, émet des avis sur les possibles décision critic ; introduire une baseline qui permet de réduire fortement la variance en ne conservant que l'avantage tiré de l'action choisie.

Les hyper-paramètres utilisés sur Cartpole sont :

- 2000 episodes
- pasMAJ = 100 (la fréquence à laquelle je mets à jour mon réseau qtarget)
- $\gamma = 0.99$
- layers=[30,30]
- lrpi=0.001 (learning rate de $\pi$), lrv=0.001 (learning rate de v)

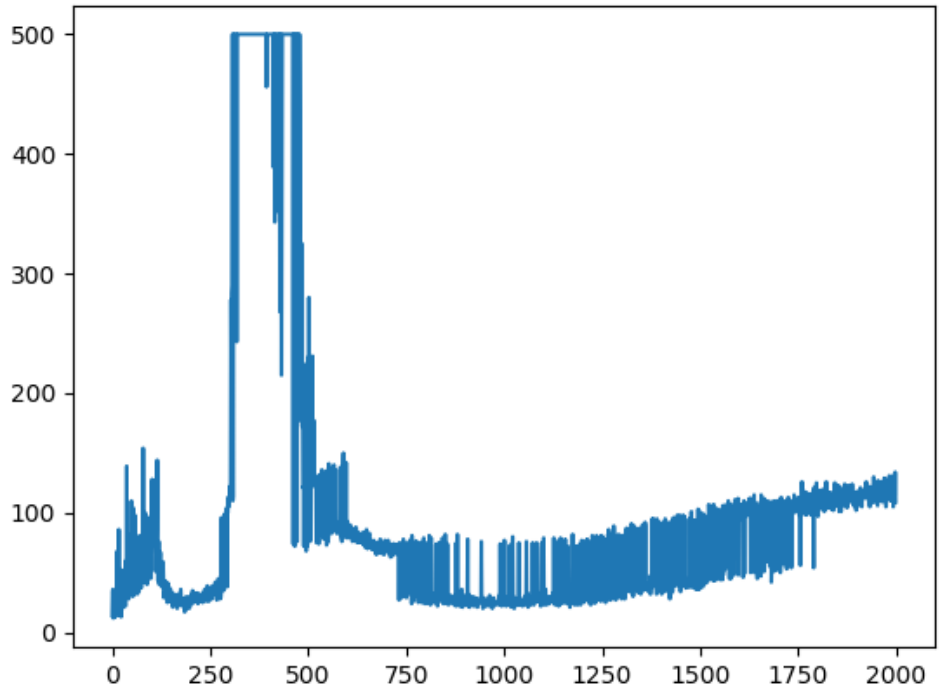



FIGURE 8 – Courbe de reward pour l'environnement Cartpole

Les hyper-paramètres utilisés sur GridWorld sont :

- 10000 episodes sur plan0 avec rewards{0 :-0.001,3 :1,4 :1,5 :-1,6 :-1}
- pasMAJ = 1 (la fréquence à laquelle je mets à jour mon réseau qtarget)

- $\gamma = 0.99$
- layers=[200]
- lrpi=0.001 (learning rate de $\pi$), lrv=0.001 (learning rate de v)

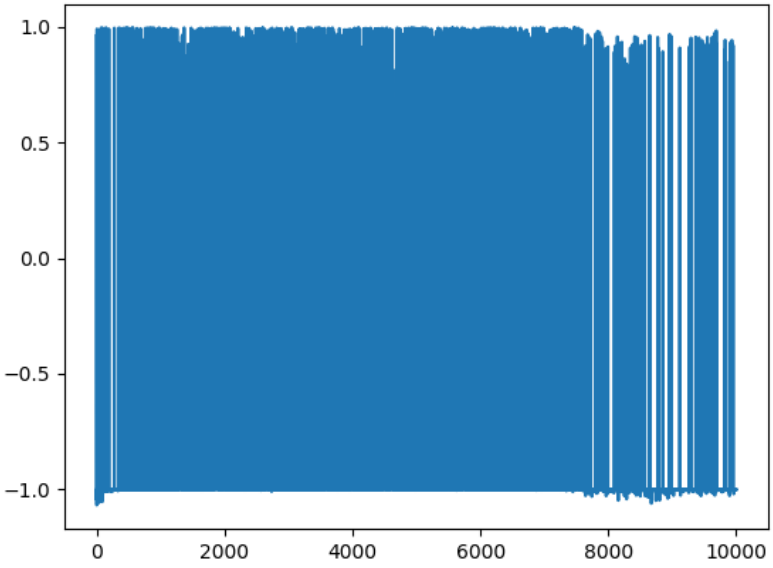



FIGURE 9 – Courbe de reward pour l'environement GridWorld

Les hyper-paramètres utilisés sur LunarLander sont :

- 10000 episodes
- pasMAJ = 1 (la fréquence à laquelle je mets à jour mon réseau qtarget)
- $\gamma = 0.99$
- layers=[30,30]
- lrpi=0.001 (learning rate de $\pi$), lrv=0.001 (learning rate de v)

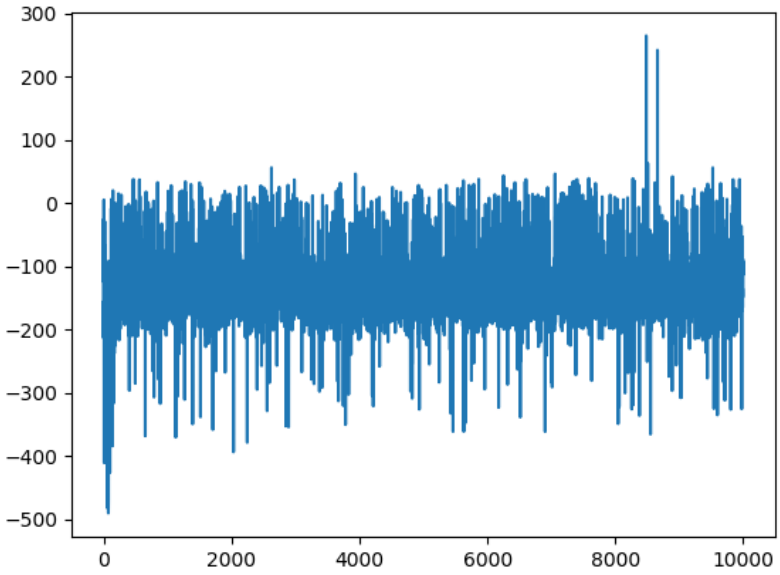



FIGURE 10 – Courbe de reward pour l'environnement LunarLander

L'agent A2C est très instable, il atteint parfois un reward à 500 pour Cartpole, 1 pour gridWorld et 300 pour LunarLander, mais il n'arrive pas à garder ces performances. Cela peut être expliqué par le fait que l'agent n'explore pas les bons endroits.

# 6 TME 6 : Policy gradients : PPO

PPO à été **conçu dans le but d'effectuer le plus d'améliorations possibles à chaque étape de l'algorithme, sans pour autant aller trop loin dans les modifications**, ce qui pourrait causer un effondrement des performances. Son but est le même que TRPO, mais il utilise des méthodes du premier ordre, beaucoup plus faciles à calculer et implémenter. Dans ce TP nous allons tester plusieurs versions de PPO, une premières version avec un coup KL adaptatif, et une seconde avec un objectif tronqué.

**PPO with Adaptive KL Penalty** Une contrainte *soft* est rajoutée et garantit que l'on reste à l'intérieur d'une région de confiance.

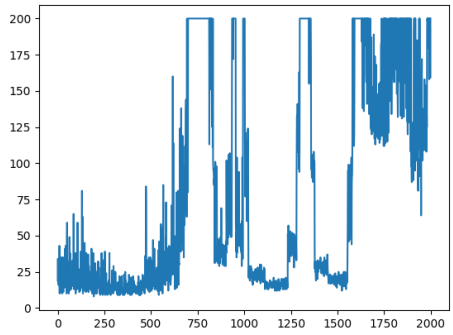
$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \hat{D}_{KL}(\pi_\theta \| \pi_{\theta_k})$$

**PPO with Clipped Objective**

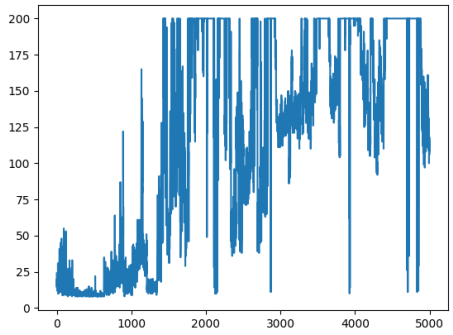
On considère le ratio entre la nouvelle politique et l'ancienne : $r_t(\theta) = \dfrac{\pi_\theta(a|s)}{\pi_{old}(a|s)}$.

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathbb{E}_{\tau\sim\pi_k}\left[\sum_{t=0}^{\tau}\left[min(r_t(\theta)\hat{A}^{\pi_k}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_k}_t)\right]\right]$$

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}^{CLIP}_{\theta_k}(\theta)$$

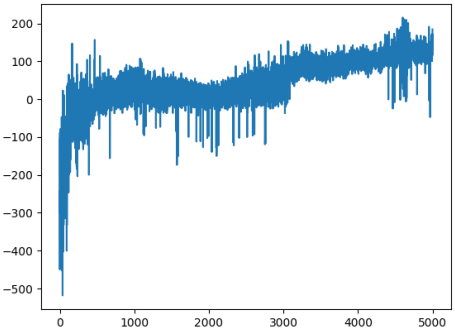



(a) Version clipped PPO

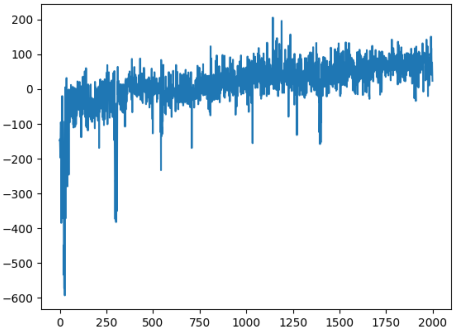



(b) Version PPO classique

FIGURE 11 – Courbe des reward cummulé pour les deux versions de PPO sur cartpole




(a) Version clipped PPO




(b) Version PPO classique

FIGURE 12 – Courbe des reward cummulé pour les deux versions de PPO sur Lunar-Lander

On peut voir que pour les deux algorithmes, PPO classique ou cliped, lorsqu'il a atteint la meilleurs reward il chute parfois assez drastiquement pour ensuite remonter. Il n'est pas stable mais par contre il arrive à atteindre la reward max et est donc meilleurs sur ce point comparé aux autres algorithmes étudié précédemment.

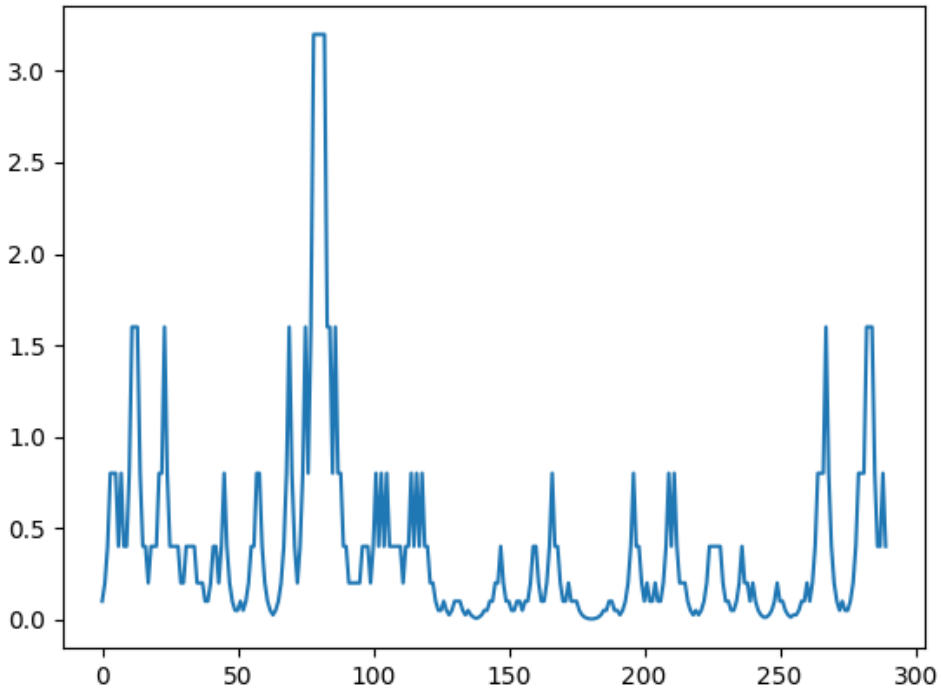



FIGURE 13 – Courbe de la valeur de $\beta$ au cours des itérations sur cartpole

Avec $\beta$ initialisé à 0.2 on peut voir que le $\beta$ reste entre 0 et 3.4, il s'adapte selon la valeurs de la DKL, qui dans certains cas augmente beaucoup, ce qui fait augmenter $\beta$ pendant un certain temps vers l'itération 75, pour pouvoir plus pénaliser la DKL pour ne pas que le réseau diverge et donc les politiques changent moins vite (à cause de l'augmentation de la loss sur la différence des politiques) et donc le DKL diminues et le $\beta$ avec.

# 7 TME 7 : Continuous Actions

Dans certains problèmes, l'espace des actions est continu. Les algorithmes Policy gradient peuvent être étendus pour s'adapter à ce genre de problèmes, mais on n'aura pas de très bons résultats. L'algorithme DDPG **exploite les mécanismes d'expérience Replay et Target Network de DQN en suivant les principes de DPG qui cherche à limiter la variance des trajectoires en laissant tomber l'incertitude**. Finalement DDPG est un algorithme actor-critic off-policy.

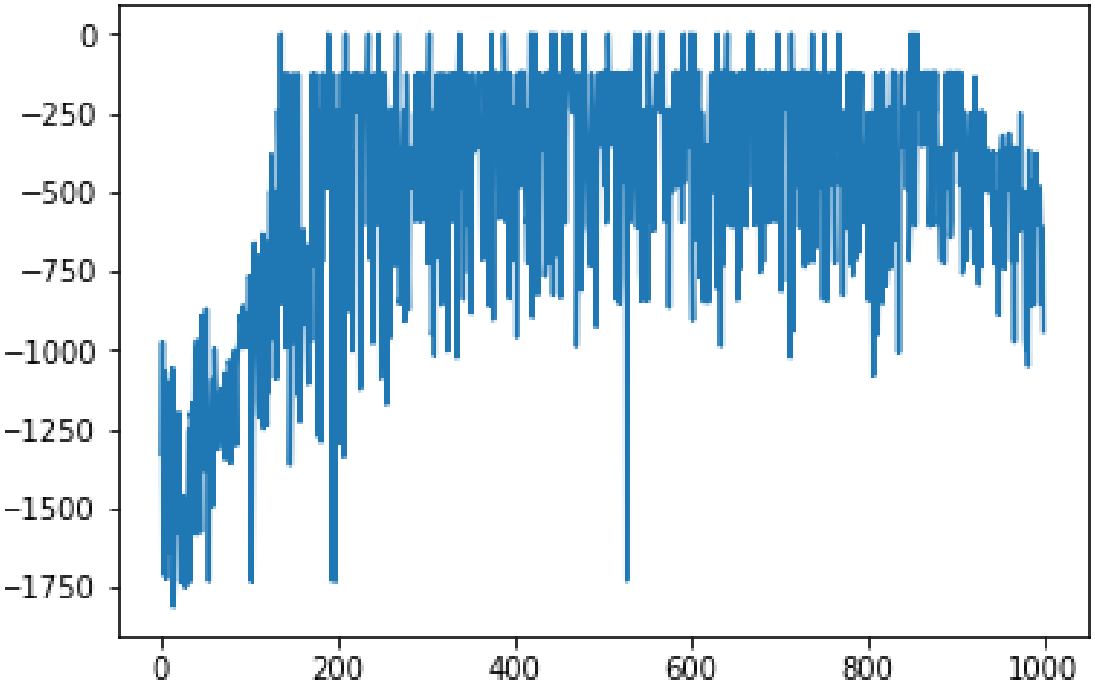



FIGURE 14 – Reward pour l'environement Pendulum

Les scores semblent se stabiliser entre 200 et 800 épisodes, puis ils ont l'air de diminuer à nouveau.
Les hyperparameters :

- replay buffer size = int(1e5)
- minibatch size = 12
- discount factor = 0.99
- TAU(soft update of target parameters)= 1e-3
- learning rate of the actor = 1e-4
- learning rate of the critic = 1e-3

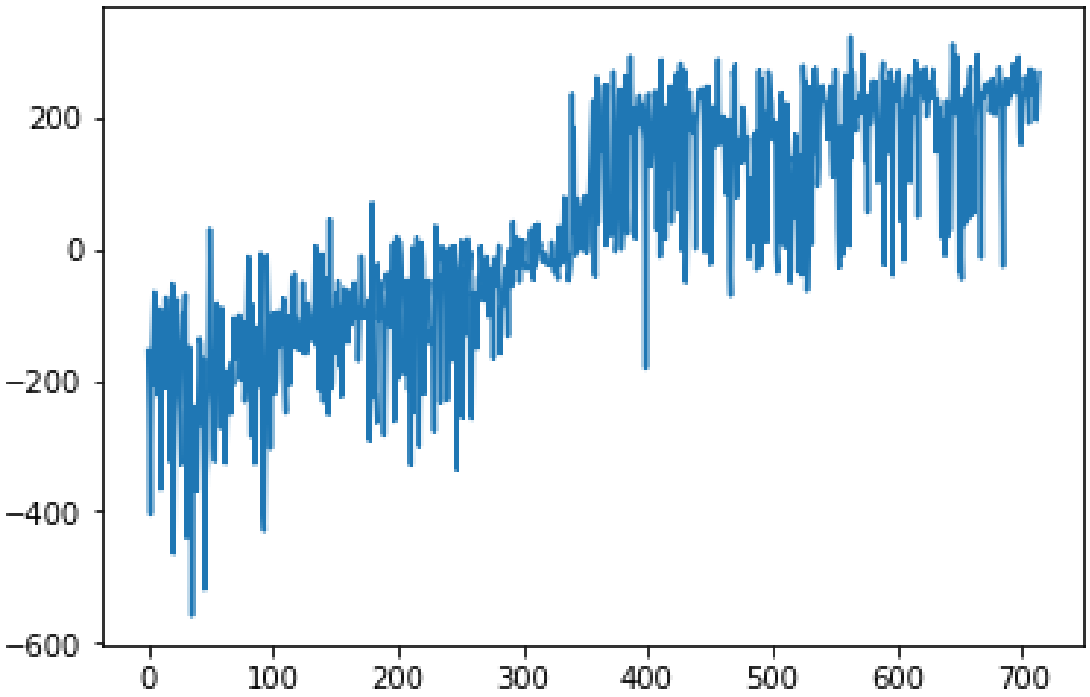



FIGURE 15 – Reward pour l'environement LunarLander

Les hyperparameters :

— replay buffer size = int(1e5)
— minibatch size = 64
— discount factor = 0.99
— TAU(soft update of target parameters)= 1e-3
— learning rate of the actor = 5e-4
— learning rate of the critic = 5e-4

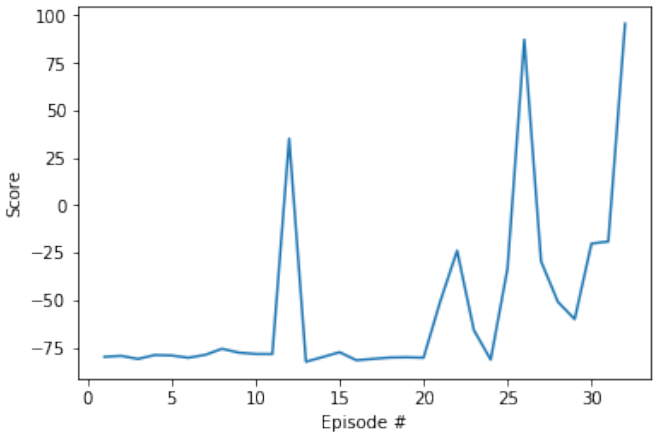



FIGURE 16 – Reward pour l'environement MountainCarContinuous

On arrive pas à stabiliser les score pour MountainCar et par contrainte de temps on a pas pu relancer les expériences.

# 8 TME 8 : Multi Agent RL : MADDPG

Le multi-agent consiste à prendre des décision où plusieurs agents interagissent simultanément avec le même environnement, affectant ainsi le processus d'apprentissage de chacun.
**L'algorithme MADDPG introduit l'idée que, si l'on connaît les actions de chaque agent, alors l'environnement est stationnaire, et ce même si les politiques des agents évoluent.**

MADDPG reprend DDPG en l'étendant au cas multi-agents :

— la critique de chaque agent est apprise en utilisant les actions et observations de l'ensemble des agents

— la politique de chaque agent n'est conditionnée que par sa propre observation

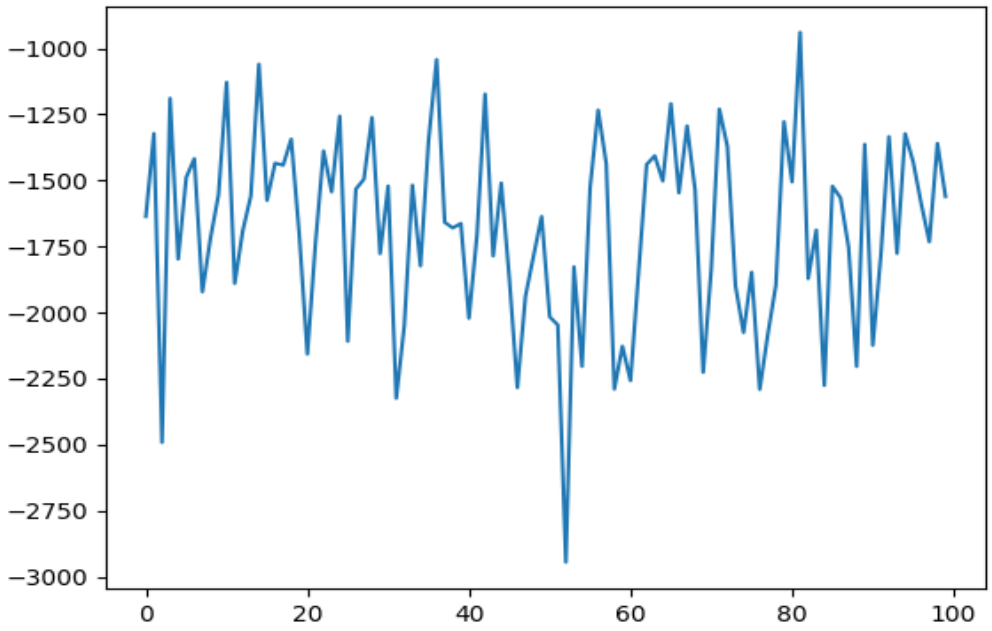



FIGURE 17 – Courbe d'apprentissage pour cooperative navigation

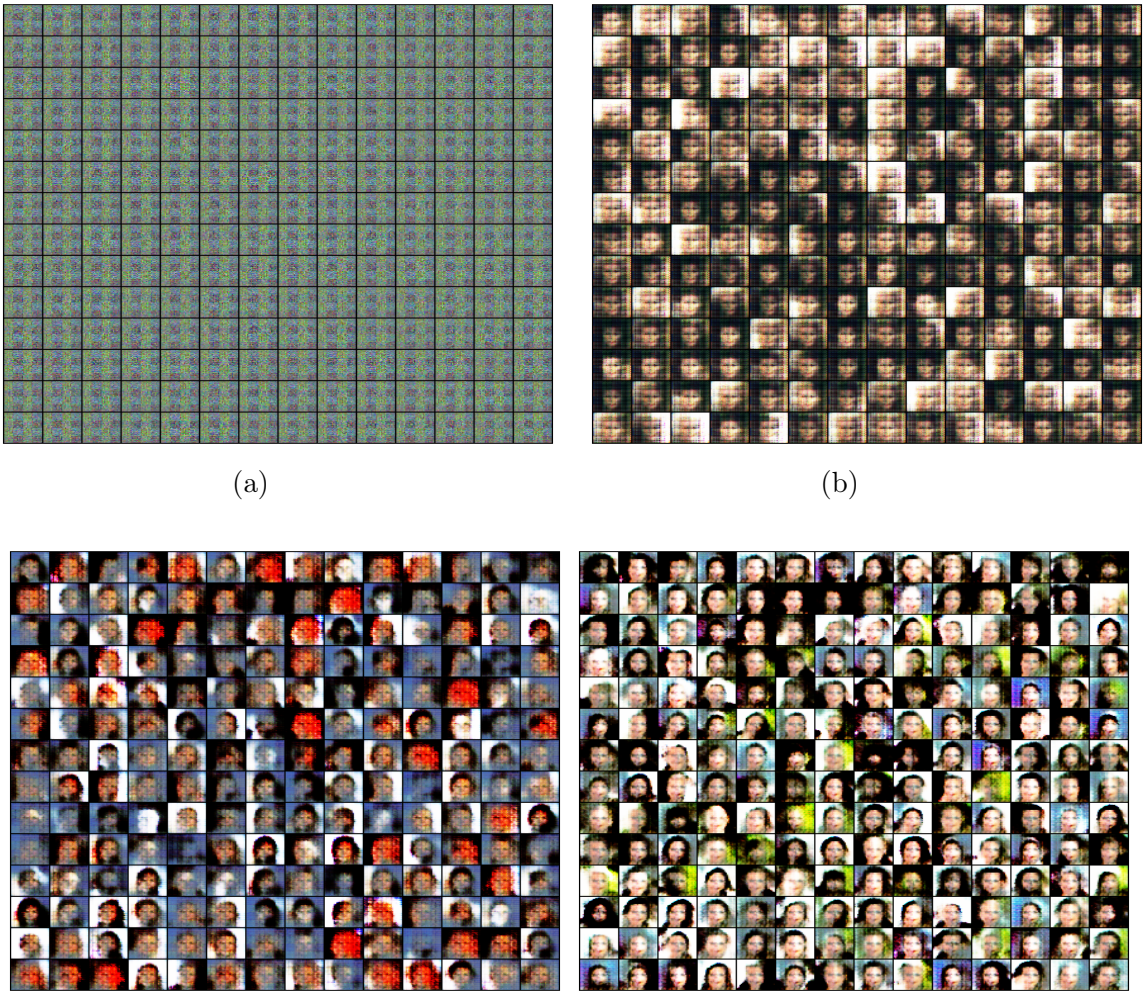
# 9 TME 9 : Generative Adversarial Networks

Un GAN est composé de deux réseaux adverses :

— Le réseau G veut **maximiser la probabilité que D dise que les images générées soient vraies**. La formalisation de son objectif ne considère pas directement les images réelles.

— Le réseau D veut d'une part **maximiser la probabilité de bien classer les images réelles et d'autre part de repérer les images générées par le réseau G**.

**Les deux réseaux doivent progresser parallèlement**. Si nous ne maximisons que le générateur, nous aurions seulement à produire du bruit adverse pouvant certes tromper le discriminateur mais les images générées n'auraient pas de sens. Si nous n'améliorons que le discriminateur, le générateur ne sera pas en mesure de générer des images convaincantes.



(a) (b)

(c) (d)

FIGURE 18 – Evolution image apprentissage GAN

On expérimente les GANs sur un problème de génération de visages, à partir du dataset `CelebA`. Selon un ensemble de visages d'entraînement, il s'agit d'apprendre à générer des visages qui paraissent les plus réalistes possibles tout en conservant une certaine diversité dans les distributions de sortie. On emploie pour cela une architecture DCGAN, qui utilise des réseaux de neurones convolutionnels pour le générateur et le discriminateur.

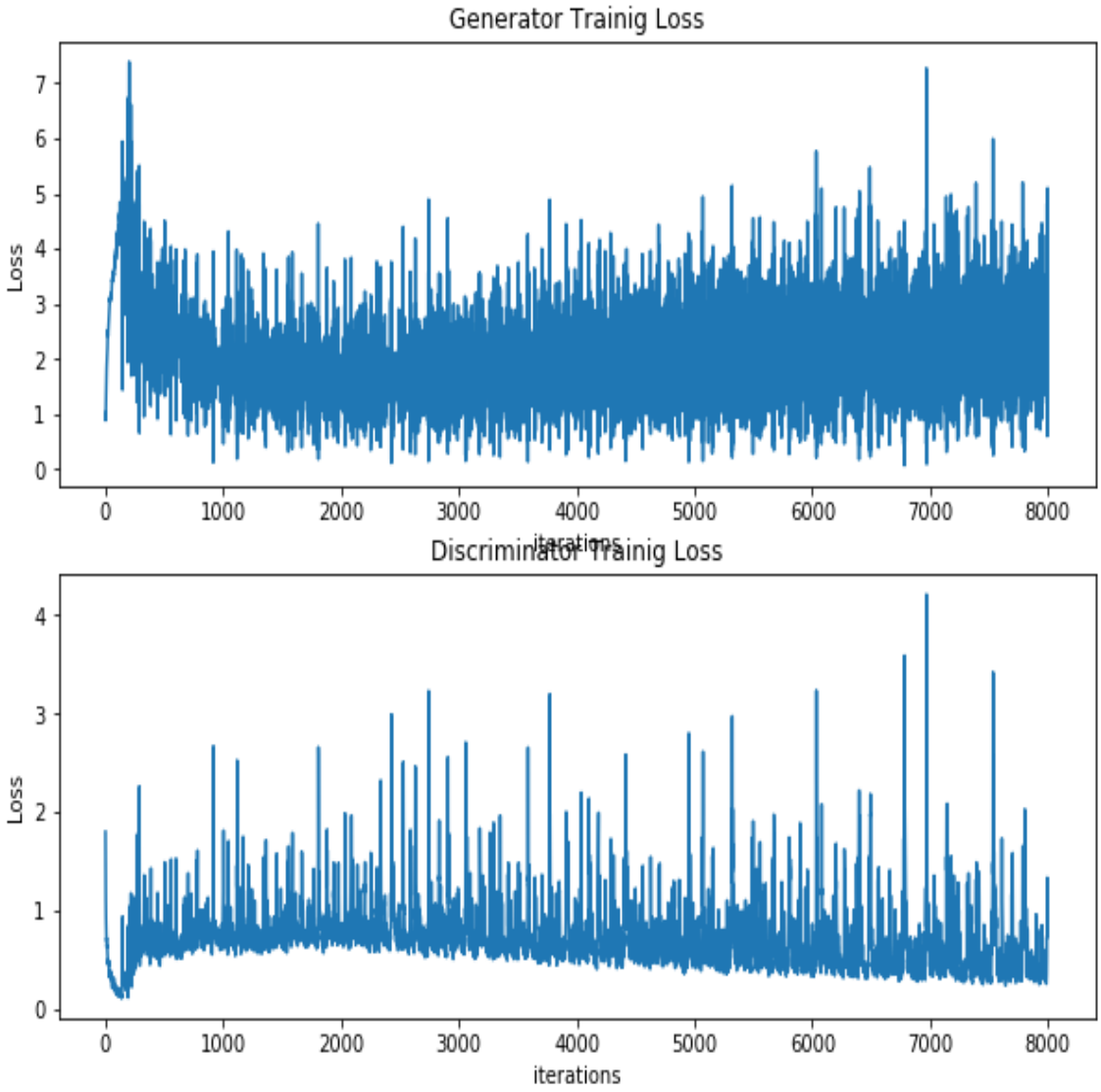



FIGURE 19 – Les loss oscillent sans jamais converger

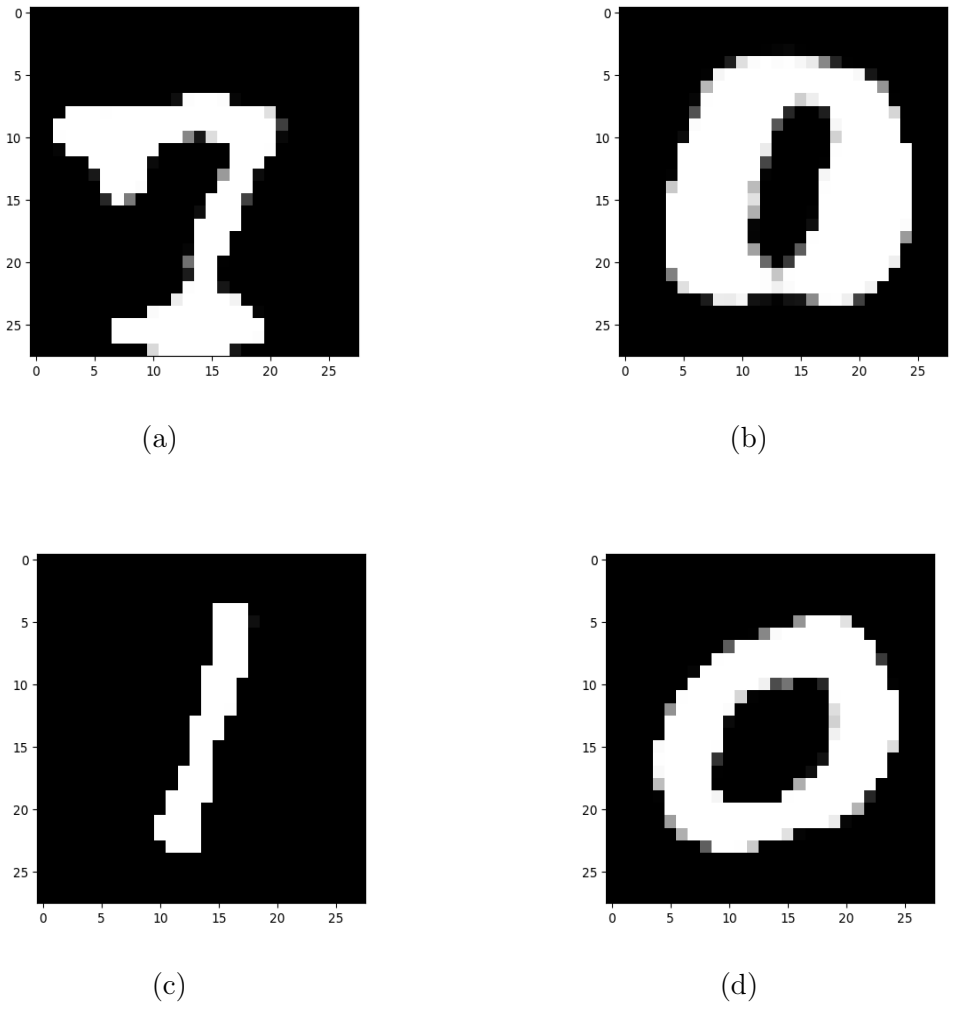
Les **loss oscillent** et ne diminuent pas au fil des itérations car les réseaux ont des objectifs opposés, les loss sont **difficilement interprétables**. Les images obtenues **ressemblent bien à des visages** mais elles ne sont pas de bonne qualité, si on les regarde en détail on parvient aisément à distinguer les vraies images des images générées.

# 10 TME 10 : VAE

Un auto-encodeur variationnel (VAE) est **un modèle génératif** qui reprend le principe d'un auto-encodeur classique en modifiant le processus d'encodage des données.

1. L'entrée $x$ est encodée par une distribution sur l'espace latent, $q_\phi(z|x)$. Cette dernière est une gaussienne permettant d'approximer $p(z|x)$.
2. Le vecteur $z$ est obtenu en échantillonnant selon $q_\phi(z|x)$.
3. Le décodeur produit une distribution $p_\theta(x|z) \sim p(x|z)$ (on utilisera une loi de Bernoulli). En échantillonnant selon cette dernière, on obtient une image.

Ci-dessous les images générer par le VAE en tirant aélatoirement la variable latente après un entraînement de 100 EPOCH. Avec un batch size de 32 et un espace lattent de taille 100 et une dimension intermédiaire de taille 200.



(a) (b)

(c) (d)

FIGURE 20 – Evolution image apprentissage GAN