FACULTÉ DES SCIENCES ET INGÉNIERIE

SORBONNE UNIVERSITÉ

REINFORCEMENT LEARNING AND ADVANCED DEEP LEARNING

RLADL

Rapport de TMEs

Auteur :
Ahmed Tidiane BALDÉ

Encadrants :
Sylvain LAMPRIER
Nicolas BASKIOTIS

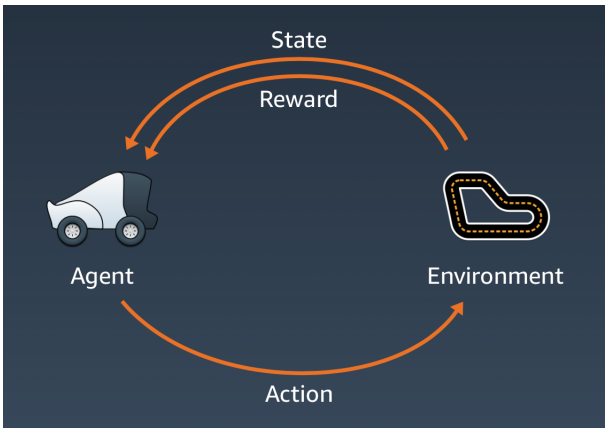
24 février 2020

TABLE DES MATIÈRES

TMEs

Le *reinforment Learning* est un domaine attractif dont le but est la résolution des jeux ou de problèmes. Certaines techniques ont fait leur preuve notamment, le Q-Learning, Value Iteration, Policy Iteration et bien d'autres. Mais avec l'essor du *Deep Learning*, plusieurs autres techniques combinant les deux mondes ont été mises au point, c'est le cas du Deep Q-Learning(DQN), Actor-critic (A2C ou A3C), Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization(TRPO), etc... Aujourd'hui, grâce à ces avancées, ces algorithmes à l'état de l'art arborent des résultats performants dans la plupart des problèmes posés. Nous avons par ailleurs assisté à la domination de l'IA dans certains jeux comme le jeu de go, où le champion du monde Lee Se-dol a été battu. C'est également le cas pour les jeux d'échecs et de shogi (la variante japonaise). Dans cette UE, nous avons alors eu la chance d'appréhender ce domaine ainsi qu'une grande variété des algorithmes à l'état de l'art.

En général, un problème de reinforcement learning peut se définir comme ci-dessous.



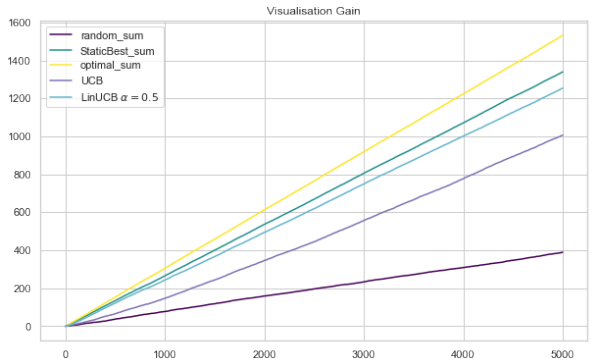
Une action sur l'environnement qui génère un nouvel état du jeu et puis, suite à quoi, nous obtenons le reward correspondant à cette action et défini en amont. L'agent apprend alors la tâche qui lui est spécifiée en explorant l'environnement et en prenant les actions conduisant à un bon résultat plutôt que celle avec un mauvais. Il apprend de ces expériences et au cours du temps, il pourra alors identifier quelles actions mènent au meilleur gain. Il y a également différents types de modèles de reinforcement leearning, le *model-based* qui consiste tout simplement à avoir une connaissance du monde dans lequel nous évoluons, notamment tous les états ainsi que toutes les probabilités de transitions d'un état à un autre. Cependant, nous pouvons énumérez bien des cas où il est très difficile voir impossible d'avoir un modèle de notre jeu déjà défini, peut-être parce qu'il y a un ensemble d'états assez large et que le monde est assez complexe. Dans de pareils cas, les *model-free* algorithms pourraient être les plus adaptés, car inutile d'apprendre un modèle, mais il s'agit ici plutôt d'apprendre directement une policy.

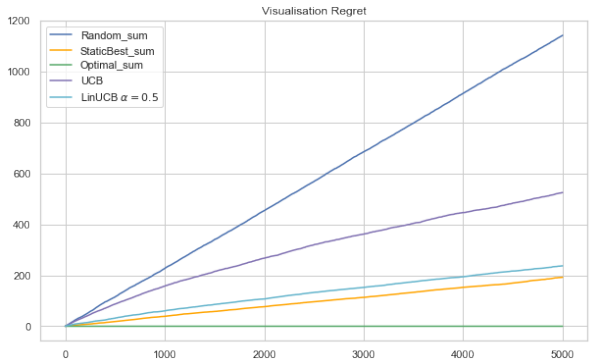
PROBLÈMES DE BANDITS

L'étude des problèmes de bandits remontent à très longtemps, notamment en 1933 où le problème à résoudre consistait en un joueur faisant face à plusieurs machines à sous et cherchant laquelle tenter pour maximiser ses gains. Aujourd'hui le problème est vu sous un autre angle grâce à l'avènement d'internet et des systèmes de recommandations qui fortement relancé l'étude.

Plusieurs stratégies ont été étudiées durant ce TME, dont les **baselines** notamment, *Random*, *StaticBest* et *Optimale*. Cependant, il a été montré que ces stratégies ne sont pas les bonnes. Ce n'est souvent pas la meilleure idée de se fier aux récompenses obtenues par le passé pour construire des politiques.
D'où la naissance de nouveaux algorithmes tels que : *UCB* et *LinUCB*, qui eux, se basent sur une borne supérieure de confiance, plutôt qu'à la performance estimée.

Il convient de souligner que la performance d'une politique est mesurée par le *regret R*, qui correspond à la différence moyenne entre les récompenses qui auraient été accumulées jusqu'au temps $t = n$ si le meilleur bras était connu à l'avance et celles obtenues par la politique durant cette même période de temps.





On note que *LinUCB* et la méthode *StaticBest* sont assez concurrentes quand à la minimisation du Regret. De même pour la maximisation du Gain, on a que ces deux méthodes sont concurrentes.

Toutefois d'un point de vue connaissance et apprentissage c'est *LinUCB* qui donne les meilleures performances. En effet cette méthode apprend au fur et à mesure et ajuste ses décisions en fonction de sa connaissance. Elle donnera de bonnes performances peu importe la base qu'elle doit apprendre.

La différence principale entre *UCB* et *LinUCB*, réside dans le fait que cette dernière prend en compte le contexte grâce donc à la régression linéaire appliquée sur les données collectées.

DYNAMIC PROGRAMMING VALUE ITERATION AND POLICY ITERATION

Les deux (2) méthodes fondamentales pour résoudre les problèmes de Markov Decision Process sont *Value Iteration* et *Policy Iteration*. Chacune de ces deux considèrent que le MDP est connu pour le monde dans lequel on travaille, notamment les probabilités de transitions d'un état à un autre ainsi que les récompenses. D'où le nom qu'on leur attribue le plus souvent (offline) plan.

VALUE ITERATION

L'algorithme *Value Iteration* calcule itérativement la valeur optimale des états en améliorant l'estimation des valeurs de $V(s)$.

Initialize $V(s)$ to arbitrary values
Repeat
    For all $s \in S$
        For all $a \in \mathcal{A}$
            $Q(s,a) \leftarrow E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V(s')$
        $V(s) \leftarrow \max_a Q(s,a)$
Until $V(s)$ converge

POLICY ITERATION

Policy Iteration quant à lui, redéfinie la politique à chaque itération et calcule les valeurs selon cette nouvelle politique jusqu'à la convergence de cette dernière.

Initialize a policy π' arbitrarily
Repeat
    $\pi \leftarrow \pi'$
    Compute the values using π by
        solving the linear equations
            $V^{\pi}(s) = E[r|s,\pi(s)] + \gamma \sum_{s' \in S} P(s'|s,\pi(s))V^{\pi}(s')$
    Improve the policy at each state
        $\pi'(s) \leftarrow \arg\max_a (E[r|s,a] + \gamma \sum_{s' \in S} P(s'|s,a)V^{\pi}(s'))$
Until $\pi = \pi'$

Nous avons testé ces algorithmes sur quelques plans de l'environnement gridworld notamment le plan 0, 1 et 5 puis avons observé les résultats qui suivent.

Quant à la rapidité de l'exécution de ces deux algorithmes, nous observons que *Value Iteration* est un peu plus rapide que *Policy Iteration*. Ce qui nous surprend car de coutume, ce dernier prend moins de temps à converger que le premier, tout simplement parce que la politique peut arriver à convergence avant les valeurs.

gridworld	Value Iteration	Policy Iteration
plan 0	0.97 $\pm$0.013	0.97 $\pm$0.014
plan 1	1.34 $\pm$0.009	1.34 $\pm$0.009
plan 5	1.92 $\pm$0.01	1.92 $\pm$0.01

TABLE 1 – Mean of cumulated reward per episode

Q-LEARNING

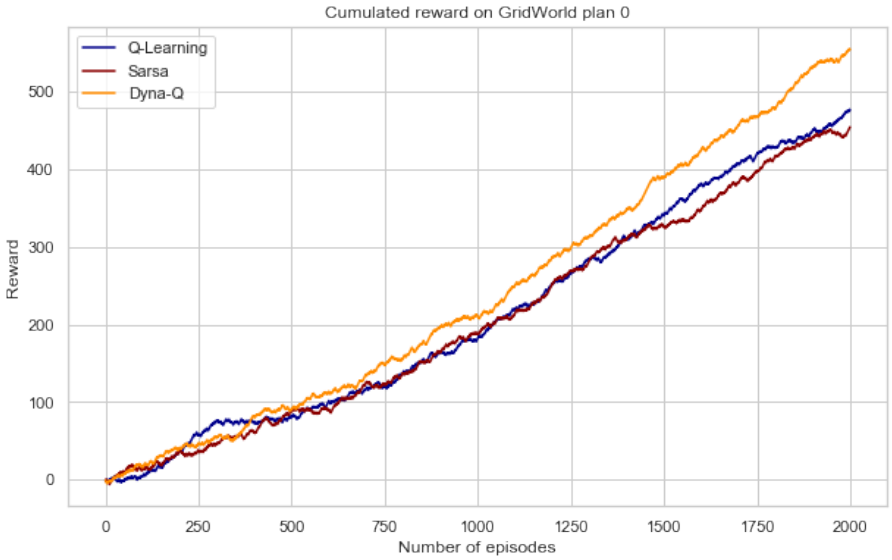
Dans les deux algorithmes précédents, nous avons considéré que nous disposions d'informations a priori, notamment les probabilités de transitions d'un état à un autre et les récompenses. Maintenant, comme nous l'avions souligné dans l'introduction, il est parfois difficile d'approximer un modèle de l'environnement dans lequel nous évoluons, d'où le *Q-Learning*.

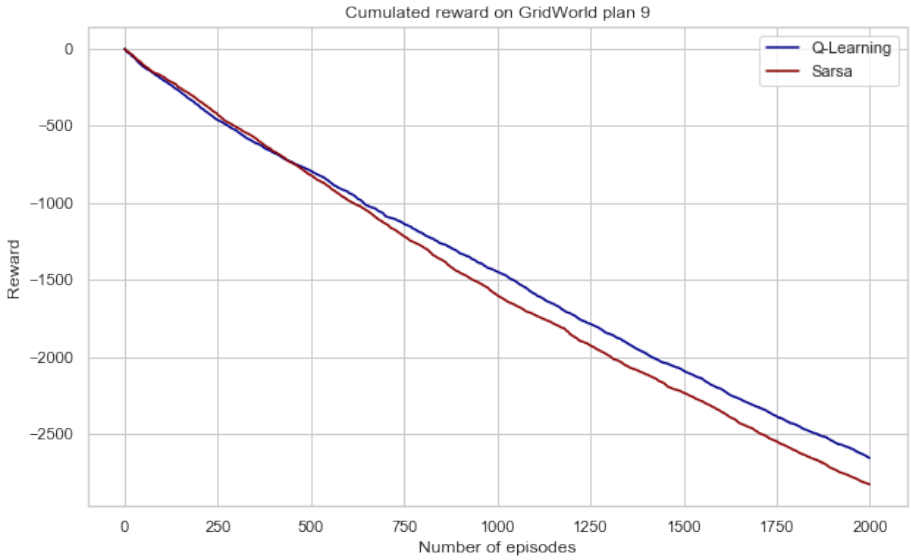
Cet algorithme est considéré comme *model-free* learning car il découvre l'environnement au fur et à mesure et ne dispose d'aucune information a priori. L'idée étant d'approximer $Q(s,a)$ en fonction des différentes interactions que nous faisons avec l'environnement.

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, Q_{obs}(s,a)$$
$$\text{where, } Q_{obs}(s,a) = r(s,a) + \gamma \max_{a'} Q(s',a')$$

Q-Learning algorithm, Time Difference Approach

Le plus souvent, ε qui est la probabilité de choisir l'approche ε-greedy (une action au hasard), diminue tout au long de l'estimation des valeurs de Q, car nous supposons que l'estimation est en elle-même plus précise.
D'autres extension tels que *Sarsa* et *Dyna-Q* furent également testés.





Nous avons testé notre implémentation sur gridworld, notamment les plans 0, 7, 9 et observons les résultats ci-dessus.

Dans la plupart des cas, *Sarsa* montre de meilleures performances que *Q-Learning*. D'un autre côté, *Dyna-Q* qui fait du *planning* qui est, pourrait-on dire une sorte d'expérience replay, justifie de résultats stables et performants.

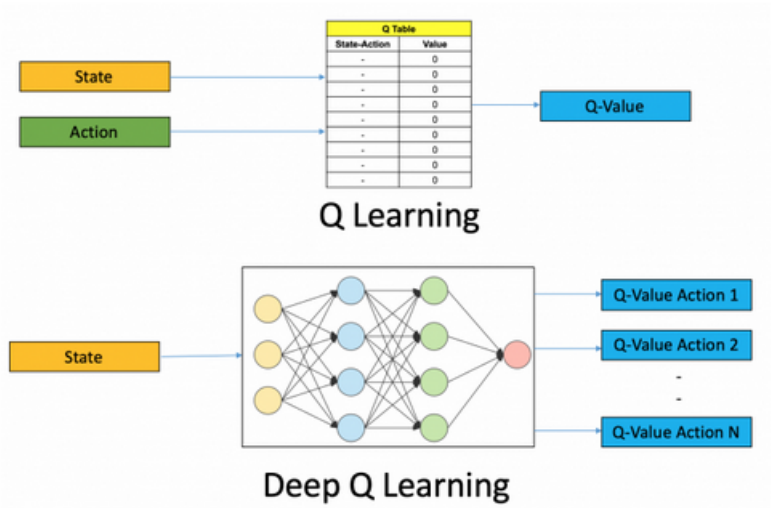
Dans le dernier plan que nous avons testé, notamment le *9*, la complexité de ce dernier se fait sentir sur les résultats, nous n'avons pas pu obtenir les résultats avec le *Dyna-Q*, faute de temps de computation énorme. Cela dit, les deux autres algorithmes ont pratiquement les mêmes résultats.

DEEP Q LEARNING

Bien que le Q-Learning soit un algorithme assez puissant, certains problèmes se posent quand nous nous retrouvons dans un environnement assez vaste notamment avec quelques milliers d'états et d'actions, dont :

— La quantité de mémoire requise pour mettre à jour la table augmente simultanément avec le nombre d'états et le temps pour ainsi faire serait également énorme.

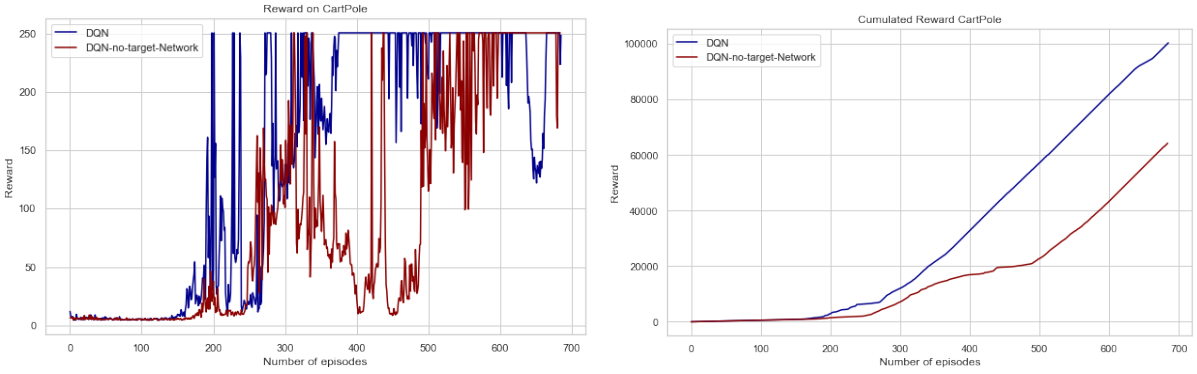
C'est ainsi que *DeepMind* eu l'idée d'approximer les $Q-value$ avec un réseau de neurones.



Notons également qu'il est préférable d'utiliser un autre réseau de neurone pour estimer la *cible* qui est définie comme suit :

$$r + \gamma \, max_{a'} Q(s', a'; \theta_i^-)$$

Avec des paramètres qui sont frisés et qui sont copiés des paramètres du réseau de prédictions chaque C itérations.



On parle également d'*expérience replay*, une technique qui consiste à stocker les expériences de l'agent pour en choisir un certain nombre au hasard durant l'apprentissage, ce qui permettrait de réduire la corrélation qui existe entre les exemples.

Nous avons réalisé les expériences sur l'environnement de *CartPole* et observons les résultats ci-dessus. Comme nous nous y attendions, l'effet positif du *TargetNetwork* est observé. Quand bien même les récompenses obtenues fluctuent beaucoup, en général le *DQN* s'en sort bien sur cet environnement.

POLICY GRADIENTS

Les *policy-based* algorithmes sont d'une philosophie simple : " Je suis dans une situation similaire à une autre que j'ai déjà rencontrée par le passé, voyons voir ce qui se passe si je choisis la même action que j'ai choisi autrefois "
Cela dit, il se peut que ce ne soit pas la meilleure action à choisir dans cette nouvelle situation, il faut alors que l'agent fasse une certaine rétrospective sur la qualité de cette action par le passé.

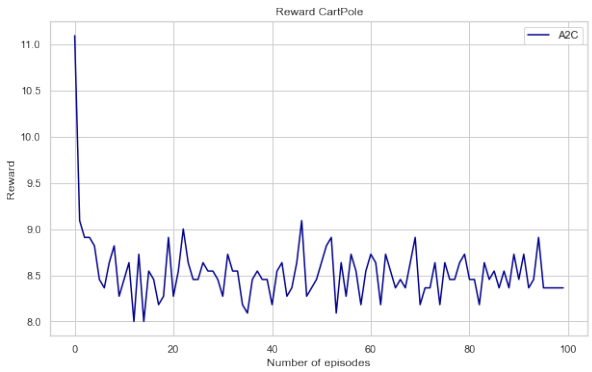
A2C

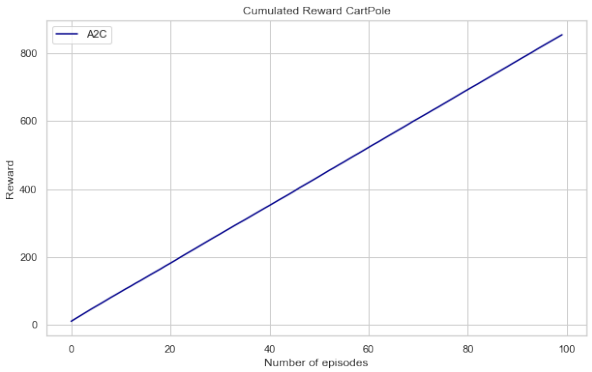
L'algorithme *Actor-Critic* consiste alors à maximiser le gradient de $J(\theta)$ défini comme ci-dessous.

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{T} \nabla_\theta \underbrace{\log \pi_\theta(a_t|s_t)}_{\text{Actor}} \underbrace{(Q(s_t,a_t) - V_\phi(s_t))}_{\text{Critic}}$$

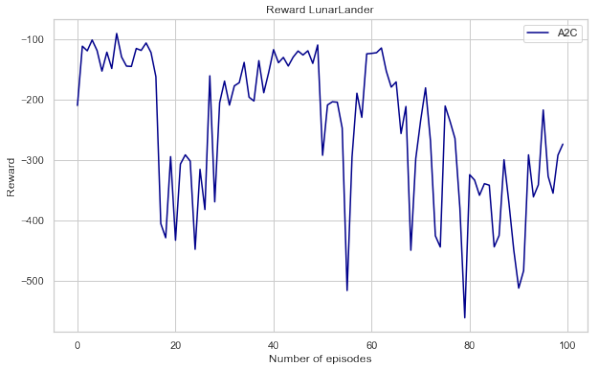
Notons que, et l'*actor* et la *critique* sont calculés par des réseaux de neurones :

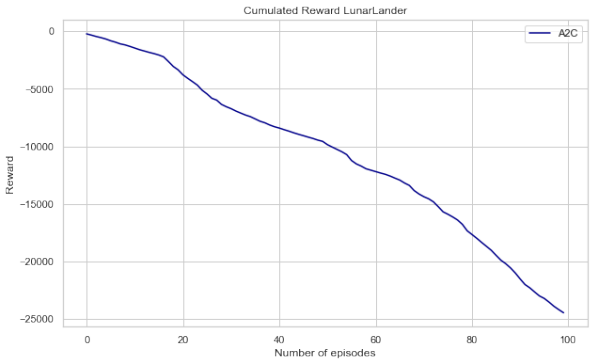
- Actor : $\pi(a|s)$ détermine l'action a à effectuer quand je suis à l'état s, par une probabilité ;
- Critic : nous dit combien ce choix nous est précieux, avec $Q(s_t, a_t)$ étant la valeur de l'action selon l'état et $V(s)$, la moyenne des valeurs des états, ou encore la baseline des récompenses obtenues à cet état.





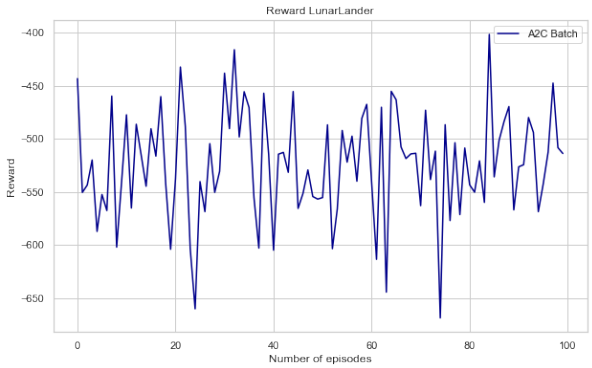
Les résultats avec le *a2c* que nous obtenons sont en général vraiment moyens, autant sur le *CartPole* que sur *LunarLander*. Nous remarquons que l'algorithme est extrêmement instable.

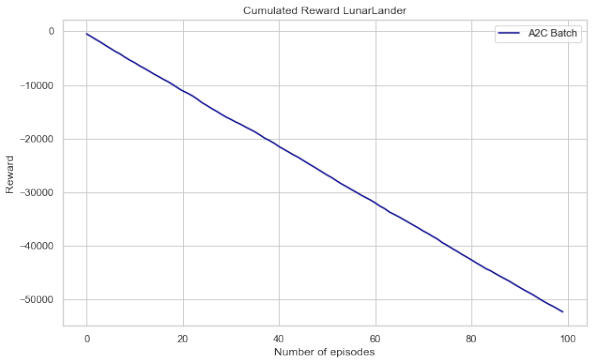




BATCH A2C

Ici, comme son nom l'indique, plutôt que de mettre à jour à chaque action, nous optimisons seulement à la fin d'une trajectoire.





Les résultats sont encore plus instables le *A2CBatch*. Nous imaginons que c'est parce que ici l'optimisation dépend de plusieurs facteurs, donc par conséquent l'algorithme a une plus forte variance contrairement au simple *A2C* qui optimise après chaque itération. Nous n'avons pas d'explications plausibles quant aux performances médiocres obtenues.

CONTINUOUS ACTIONS : DDPG

Deep Deterministic Policy Gradients est un algorithme qui suit à peu près la même logique qu'*Actor-Critic*. Il est utilisé pour la résolution de problèmes disposant d'un nombre d'actions assez vaste, autrement dit, des actions continues. Sa structure est composée de 4 réseaux de neurones dont, la politique, qui est cette fois-ci déterministe, c'est à dire que la sortie du réseau de neurones est tout simplement la meilleure action et non un ensemble de probabilités autour de l'ensemble des actions. Puis nous avons le réseau des Q valeurs qui peut être assimilé à la critique, et enfin, respectivement, un réseau cible pour chacun des deux.
En résumé, nous avons :

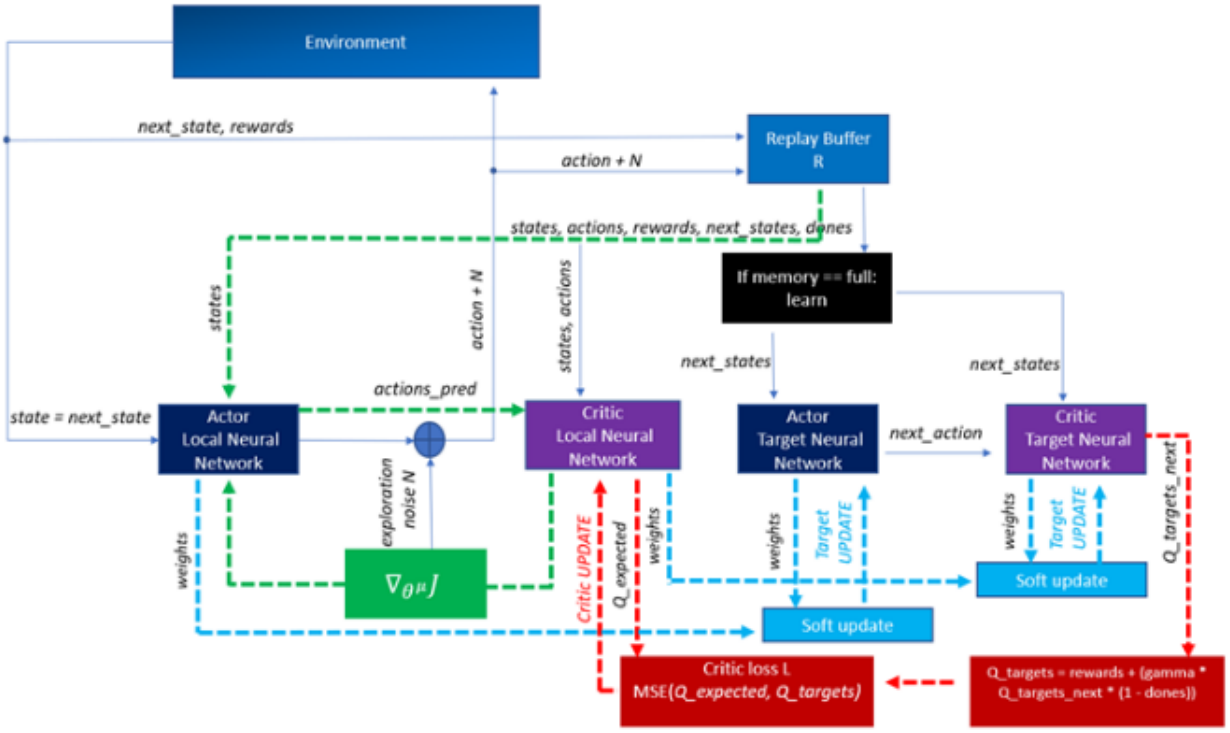
— le réseau Q ;

— la politique ;

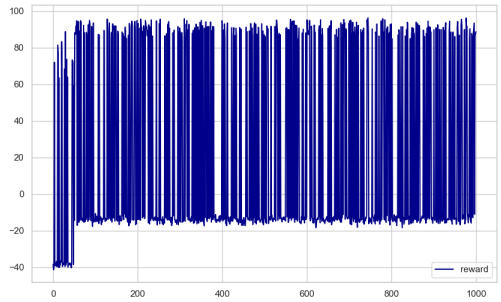
— le réseau cible Q et

— la politique cible.

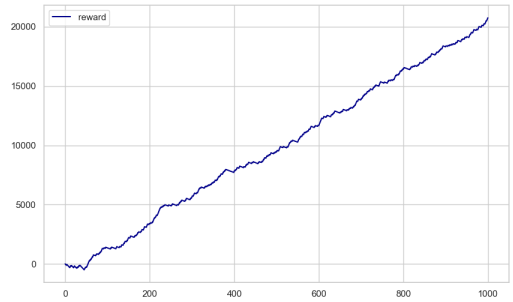
Le schéma ci-dessous représente assez bien l'algorithme *DDPG*.



Deep Deterministic Policy Gradients Algorithm



Reward on MountainCarContinuous-v0



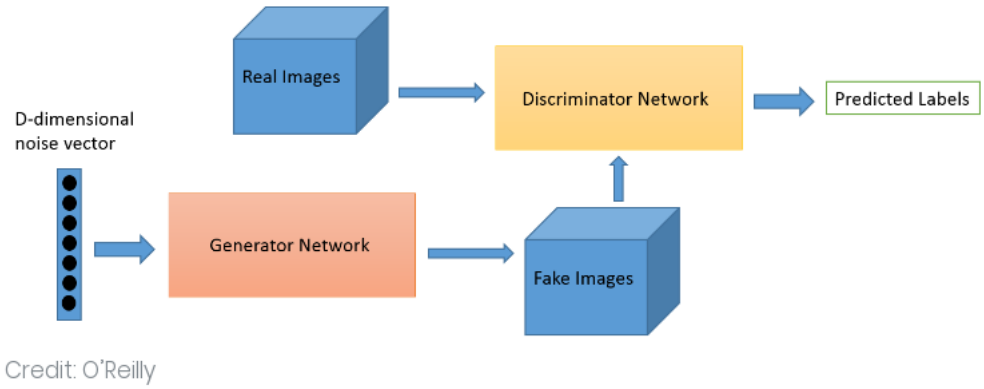
CumulatedReward MountainCarContinuous

Nous avons réalisé les expériences sur l'environnement du *MountainCarContinuous-v0* et observons les résultats ci-dessus. Le *DDPG* arrive à résoudre le problème même si quelque fois, il prend plus de temps pour y arriver. Les résultats nous paraissent stables.

GAN

Les *Generative Adversarial Networks* sont très certainement l'une des plus grosses avancées dans le domaine de la vision par ordinateur depuis les dix dernières années. Les domaines d'applications sont très nombreux, augmentation de bases de données pour l'apprentissage ou encore anonymisation de données, génération de nouvelles images dans la mode, des motifs de T-shirt, etc... La super-résolution pour augmenter la qualité des images d'une base de données, l'impainting. Oui les secteurs d'applications sont nombreux et les résultats sont d'ores et déjà inspirants.

Le principe général est simple. Nous avons un générateur, chargé de générer les images (fausses images) et un discriminateur, chargé de détecter si l'image générer est réelle ou fausse. L'objectif étant donc de générer des images (fausses) mais qui semblent si réelles que le discriminateur ne sache pas faire la différence. Autrement dit, avoir un générateur qui trompe le discriminateur.



Generative Adversarial Networks architecture

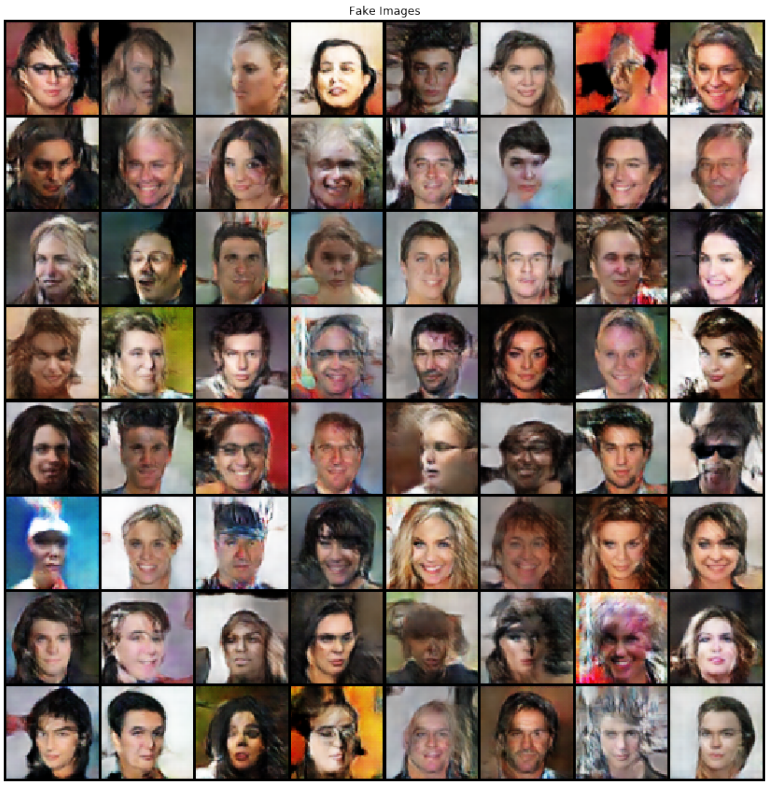
ARCHITECTURE DU PAPIER ORIGNAL DCGAN

Nous avons implémenté l'architecture du TME ainsi que l'architecture du papier origianl DCGAN et avons respectivement obtenu les résultats suivants.

Nous observons que les images générées par les deux architectures semblent toutes réalistes, à première vue. Mais après une observation plus critique, nous voyons toute suite, qu'il y a plusieurs images dont les visages sont complètement déformés. Mais au vu du peu de ressources à notre disposition et des résultats obtenus après quelques 8000 itérations, nous sommes plutôt satisfaits.

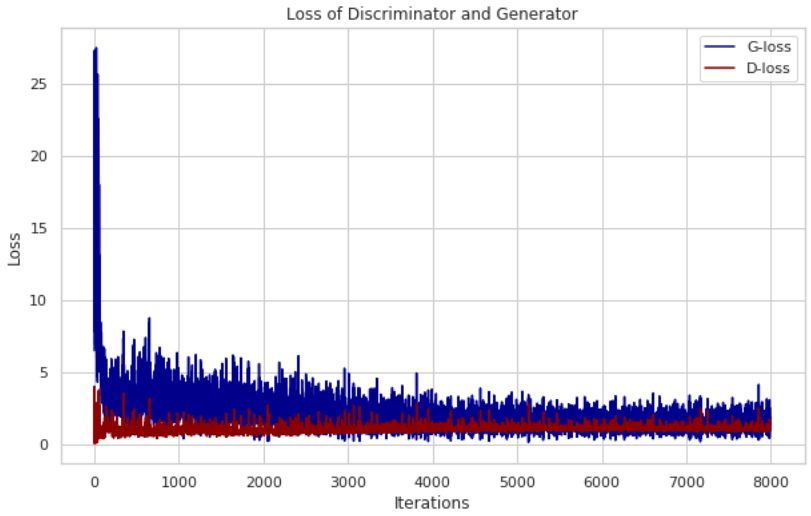
Notons cependant que les résultats obtenus avec l'architecture du papier original semblent être plus performants.

Les courbes d'erreurs du générateur et du discriminateur convergent. Nous n'observons alors pas de problèmes du générateur plus fort que le discriminateur ou inversement, ce qui aurait occasionné une divergence de l'une ou l'autre des courbes d'erreurs.



Génération d'images de Célébrités avec l'architecture simple du Générator

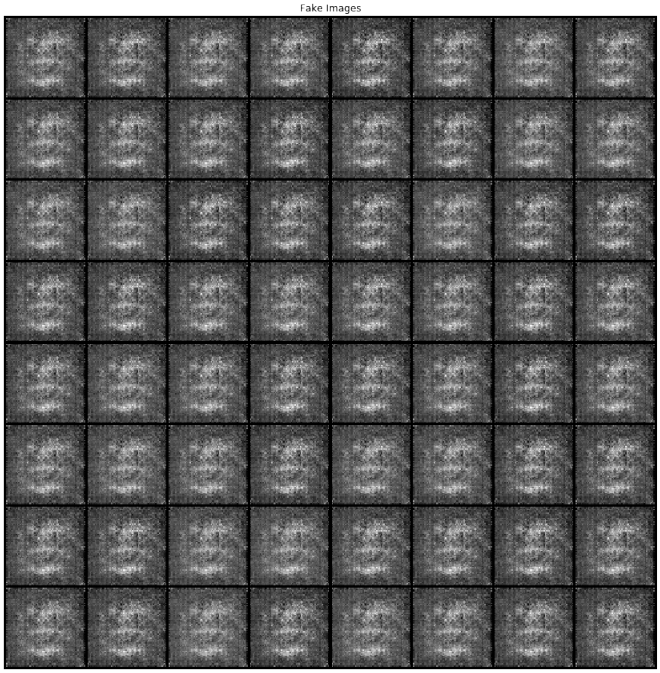
Génération d'images de célébrités avec l'architecture originale DCGAN



DCGAN Losses

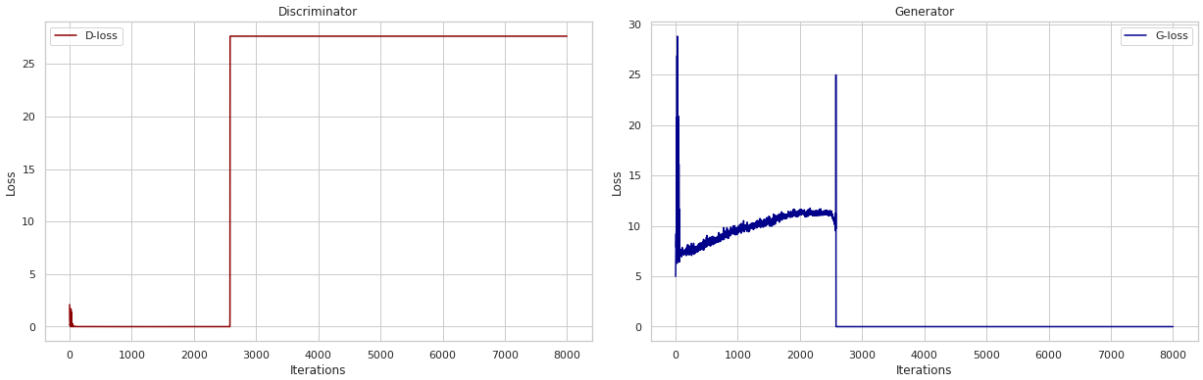
BONUS : TEST SUR MNIST

Après application du DCGAN sur le mnist, nous retrouvons des résultats peu convaincants. Nous avons l'impression de n'avoir que des 5 ou des 3 et les images sont d'une visibilité très faible.



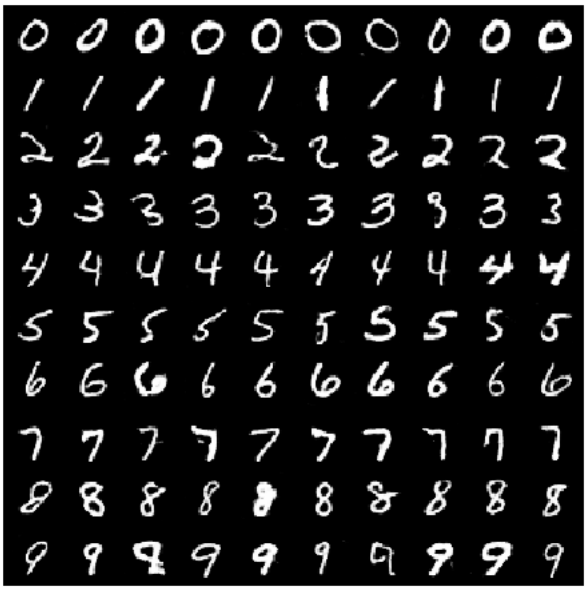
Géneration de chiffres MNIST avec l'architecture simple du Générator

Les courbes d'erreurs sont assez étranges. Nous nous attendions à ce que le discriminateur converge très rapidement vu la faible qualité des images produites, il n'est normalement pas censé avoir du mal à détecter si une image est dans ce cas-ci fausse ou réelle mais après un certain moment il diverge à notre grande surprise, tandis que le générateur converge.



cDCGAN

Pour palier au problème que nous avons énoncé plus haut, notamment le fait que nous ayons pratiquement les mêmes chiffres générés, nous avons essayé le conditional DCGAN. Le principe étant maintenant de conditionner la fonction objective par le label ou la classe. Ce pourrait être un label "vieux" ou "jeune", "femme" ou "homme", ou encore en l'occurrence des chiffres de 0 à 9.



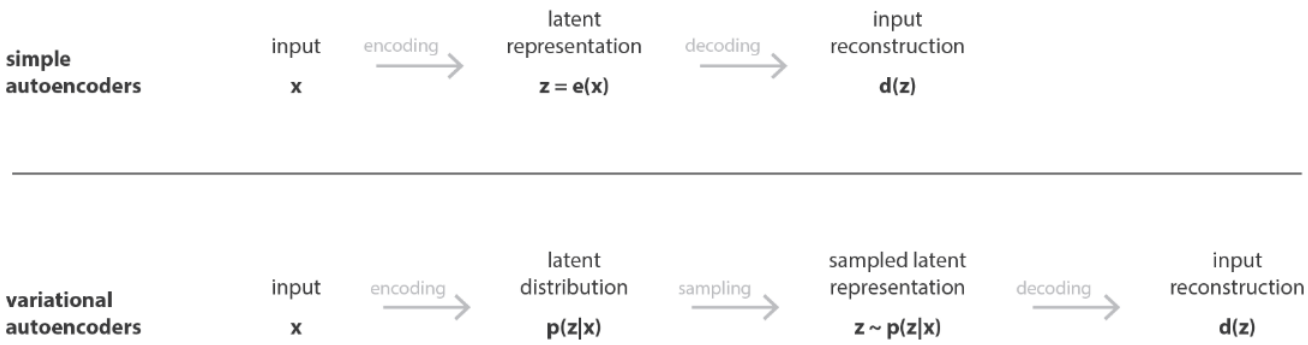
Génération du chiffre avec le cDCGAN

Nous obtenons des résultats assez performants ou chaque ligne est conditionné par un label qui est son indice de 0 à 9.

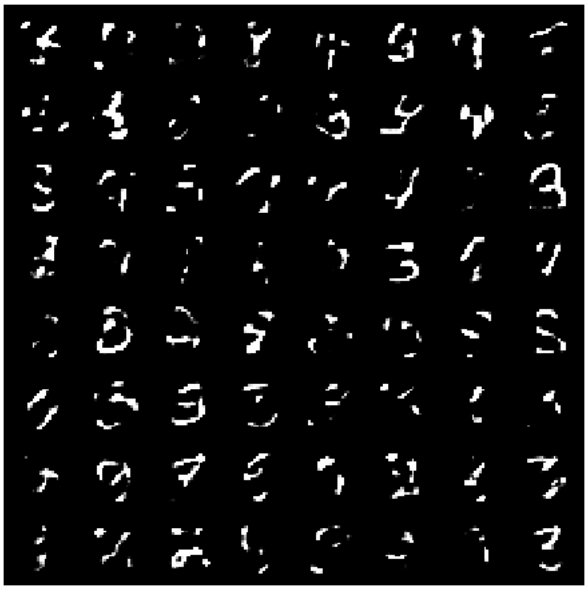
VAE

Les *auto-encodeurs* par définition, encode des données en de simples vecteurs, des représentations compressées des entrées. Ils sont souvent combinés à des décodeurs qui nous permettent de reconstruire les

données en entrée à partir de leur représentation cachée. Les auto-encodeurs variationnels que l'on étudie ici sont tout simplement des auto-encodeurs dont les représentations cachées sont normalisées. Ils sont alors capables de compresser des données tels les *AE* et d'en générer de nouvelles tels les *GAN*. Quand bien même, soulignons que les images générées par les auto-encodeurs variationnels sont un peu plus floues.



Différence entre Auto-Encodeurs (déterministe) et Auto-Encodeurs Variationnels (probabiliste)



Génération de chiffres avec VAE

Notre expérience sur le dataset *MNIST* nous a fourni des résultats moyens. Les résultats ne sont clairement pas transcendants et sont assez bruités à l'exception de quelques chiffres qui se démarquent.

CONCLUSION

Pour conclure, nous avons pu implémenter et réaliser quelques expériences avec quelques algorihtmes notamment, *Value Iteration*, *Policy Iteration*, *Q-Learning*, *DQN*, *A2C* et *DDPG* pour la partie ReInforcement Learning. Mais également les *GANs* et *VAE*. Les résultats sont pertinents pour les uns et beaucoup moins pour les autres. Nous aurions également aimé réaliser des expériences sur les *PPO* et *MADDPG*, malheureusement nous y sommes pas parvenus car notre implémentation comportait des erreurs.