Rapport des TME

Reinforcement learning and Advanced Deep Learning

Miguel Alejandro MARTINEZ HERRERA, M2A

Ce rapport resume les différents algorithmes vus au long du semestre et il présente certaines courbes et résultats montrant la convergence pour certains des environnements et données.

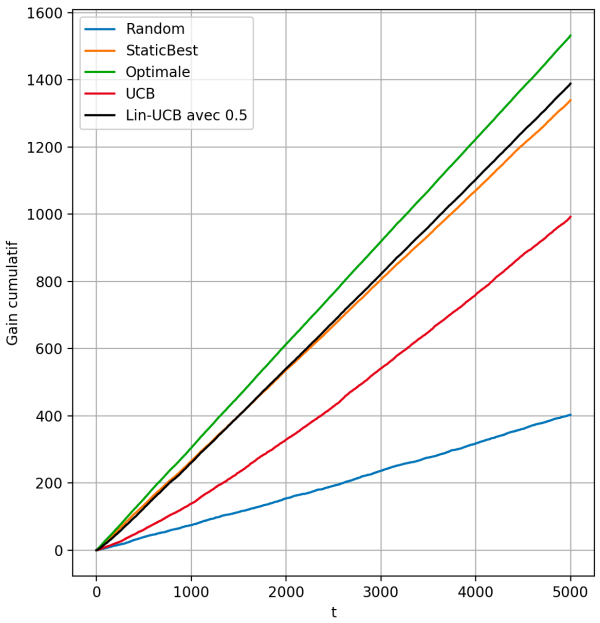
TME 1 : Problèmes de Bandits.

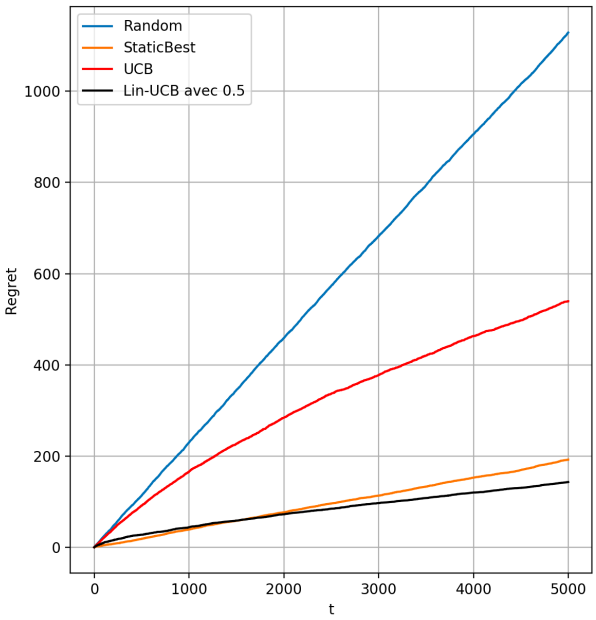
Le premier TME consistait à étudier deux types d'algorithmes utilisés d'habitude dans les problèmes de bandits. Dans ce genre de situations, les actions n'ont pas d'impact sur l'état du monde. En générale, il est possible de supposer que l'on ne connaît pas le système de rewards et alors il est important de réaliser une partie d'exploration de l'espace afin de trouver le bandit optimal.

On a considéré deux algorithmes différents:

- **UCB** : Algorithme qui consiste à choisir le bandit dont la borne supérieure de confiance est la plus grande, borne qui rétrécit plus un bandit à été joué. Ceci permet de donner de la chance à des bandits que l'on a pas beaucoup explorés et donc éviter se trouver dans des situations sous-optimales, gardant une connexion avec la moyenne des gains rapportés par bandits.
- **Lin-UCB** : Variation de l'algorithme précédent où l'on tient compte d'un contexte fourni, dépendant de chaque étape et permettant éventuellement d'obtenir des informations supplémentaires sur l'espace.

En utilisant trois baselines différentes qui permettent de comparer les résultats:



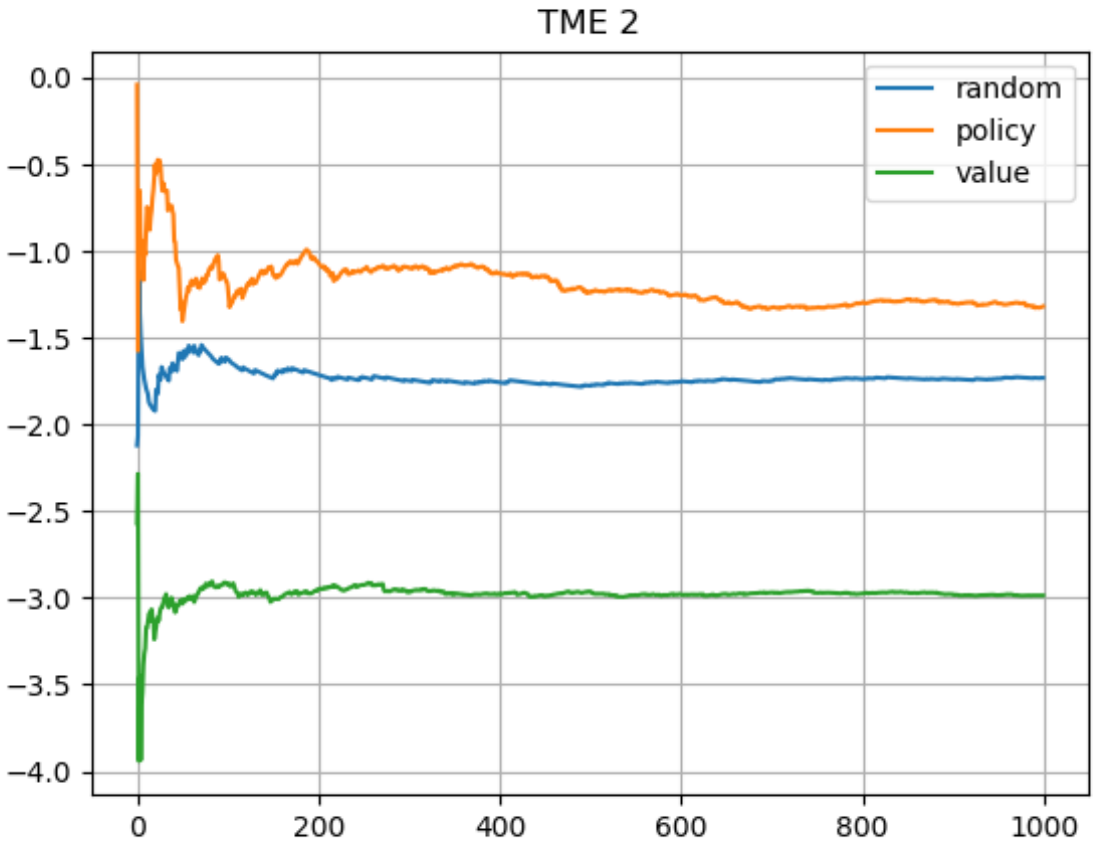


On peut observer que bien que la méthode UCB permet d'obtenir des résultats corrects, la méthode de Lin-UCB permet d'approcher le mieux le résultat optimal, obtenant le plus petit regret de toutes les méthodes observées.

TME 2 : Programmation dynamique

À partir de ce TME on a laissé derrière le cas où les actions n'affectent pas le monde, pour considérer un monde dynamique. Ce TME a consisté à appliquer deux méthodes qui présupposent la connaissance du mode de fonctionnement de l'univers (les mdp des mondes), donnant une connaissance assez complète du monde. Les méthodes sont :

- **Policy iteration** : algorithme qui consiste à essayer d'estimer une valeur qui permettrait de comparer les différentes politiques considérées. Ainsi, cet algorithme estime une valeur V pour une politique considérée et ensuite ajuste la politique selon les valeurs obtenus. Ceci est réalisé jusqu'a obtenir une convergence de la politique.
- **Value iteration** : algorithme ressemblant au précédent, mais évaluant une seule fois une valeur V, dependant uniquement de l'espace (recompenses, probabilités) qui sera utilisé pour évaluer en une seule fois la politique finale. Cette méthode permet d'alléger le calcul d'une valeur qui, pour l'algorithme précédent se fait pour chaque nouvelle politique évaluée.



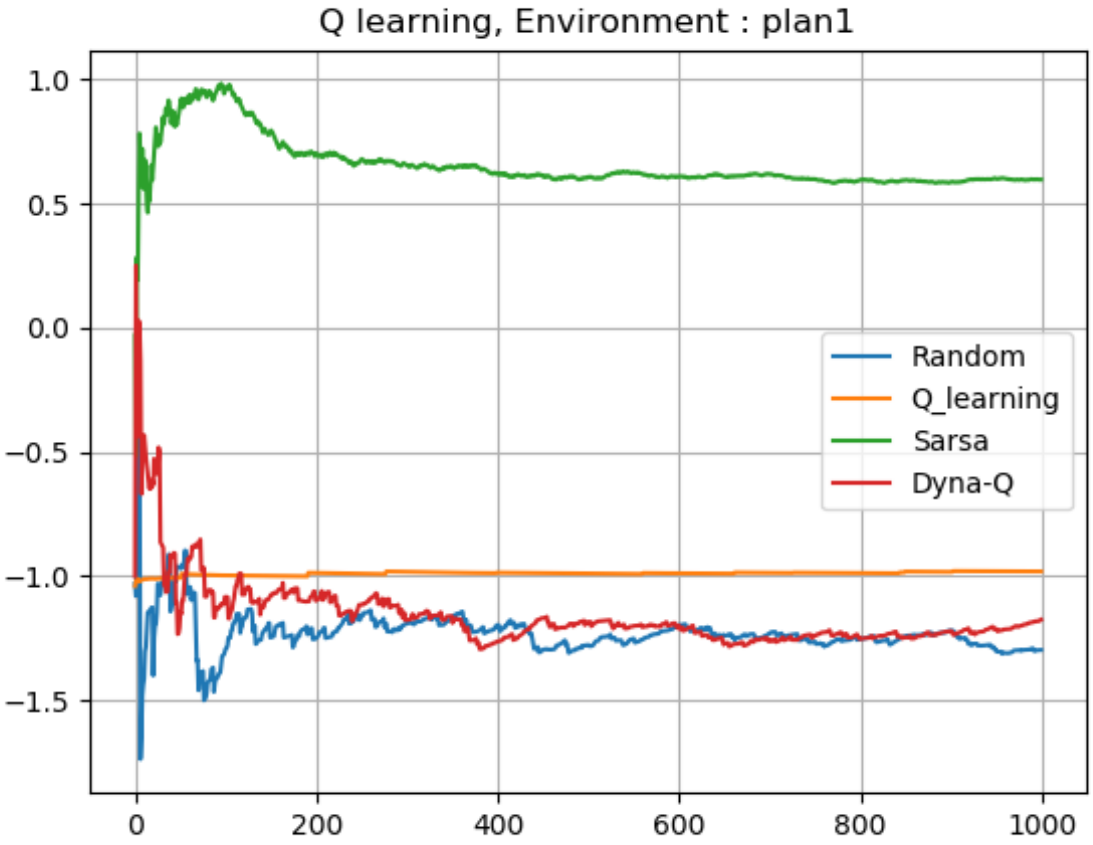
On observe ici un example où l'une des méthodes permet de faire mieux qu'un agent qui prend des actions au hasard. Précisons que les méthodes parfois ont tendance à pas trouver des solutions optimales, dû parfois à une mauvaise estimation des valeurs de V, dépendant aussi parfois des valeurs hasardeuses de l'initialisation et des transitions.

TME 3 : Q-Learning

À partir de ce TME, il est supposé que l'on a pas vraiment des données sur le monde, concernant les récompenses, les transitions et le fonctionnement en général (commande mdp interdite). Il est important alors maintenant de réaliser de l'exploration de l'espace. Cette fois, on utilise une fonction Q qui permet d'évaluer

l'espérance des gains selon l'état et l'action, au lieu de V qui l'évalue selon l'état uniquement. Les algorithmes sont :

- **Q-Learning** : De façon générale, cet algorithme mets à jour la valeur de Q pour l'état actuel et l'action choisie, en fonction de l'état d'arrivée et la récompense observée, permet d'estimer la valeur moyenne espérée pour les gains du couple.
- **Version SARSA** : Une adaptation de cette méthode consiste à considérer l'exploration dans le processus. En effet, la version précédente tient en compte uniquement la valeur des Q pour faire un choix, tandis qu'ici on considère des choix selon par exemple des stratégies ϵ-greedy.
- **Dyna-Q** : Une méthode hybride utilisant la fonction Q comme evaluation, et essayant d'estimer les paramètres P et R du monde, afin de les utiliser en tant que mdp et copier ainsi une méthode de value-iteration.

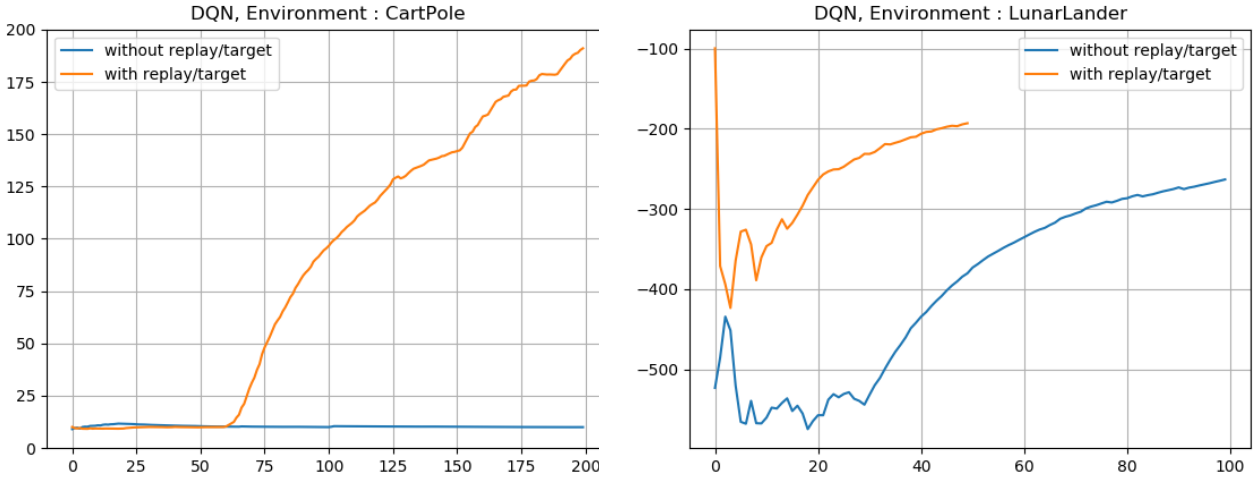


Ici on a un example de la performance de ces trois algorithmes sur le plan1 de l'environnement GridWorld. On peut voir que la version SARSA arrive à avoir la meilleur performance, sachant que l'on à deux cases de +1 et une de -1, elle arrive bien à éviter la case -1 (sinon en moyenne elle ferait moins de 0). Ceci n'est pas le cas pour les deux autres. On voit que Q-Learning arrive à dépasser l'agent randomisé, néanmoins on à une performance plus basse du Dyna-Q. On peut supposer que l'on a pas encore réussi à le faire converger vu le peu nombre d'étapes, car les tâches à apprendre sont plus complexes.
Ici on peut voir grâce au SARSA (ayant les mêmes paramètres que Q-Learning) l'avantage d'avoir un algorithme qui tient en compte non pas seulement l'optimisation mais aussi l'exploration.

TME 4 : DQN

Ce TME a introduit l'utilisation de vecteurs de caractéristiques permettant de décrire l'espace et ne pas se limiter à la connaissance d'un état, permettant ainsi de relier éventuellement des états qui se ressemblent, diminuant ainsi la taille des matrices considérées. On a introduit aussi l'utilisation de réseaux propres de l'apprentissage profond.

- **DQN** : Modèle utilisant des valeurs Q pour l'évaluation des couples état-action ainsi que pour le choix des actions. On utilise des réseaux profonds, ici des couches Linear, et des méthodes de descente de gradient pour minimiser des pertes, mettant ainsi à jour les paramètres du réseau.
- **Experience Replay** : Un ajout optionnel aux différentes méthodes permettant de garder en mémoire les observations passées. À chaque mise à jour du réseau, des sous-ensembles de la mémoire sont cherchés et utilisés, permettant de maintenir une stabilité pour l'apprentissage et éviter l'oubli de certaines expériences.
- **Target Network** : Deuxième ajout possible consistant à séparer le réseau d'apprentissage en deux: un réseau cible, permettant de choisir les actions et de servir comme objectif pour le calcul des pertes; et un réseau qui sera mis à jour constamment dans la boucle d'apprentissage. Ceci permet de stabiliser le calcul des objectifs. La mise à jour de la cible se fait après un certain nombre d'itérations.



On peut regarder pour les courbes d'apprentissage, où les deux versions, sans les deux ajouts et avec, pour deux algorithmes avec les mêmes paramètres, la performance du DQN + Replay + Target dépasse largement celle du DQN classique. Il est important de remarquer que leur performance au début, pour des initialisations similaires, sont semblables, mais on voit qu'au bout d'un moment le DQN + Replay + Target apprend de plus en plus vite, montrant à la fois qu'il s'améliore grâce à ses expériences passées, mais aussi il faut attendre la mise à jour des réseaux target aussi. Remarquons aussi qu'un désavantage du DQN "amélioré" et le temps qu'il met à mettre à jour ses poids, expliqué surtout par la présence du Experience replay.

Ensuite on a encore une comparaison de résultats, cette fois sur LunarLander. Remarquons que le réseau avec Replay et Target a été arrêté avant car le temps devient très long entre chaque episode, mais on peut déjà voir la différence de performance.

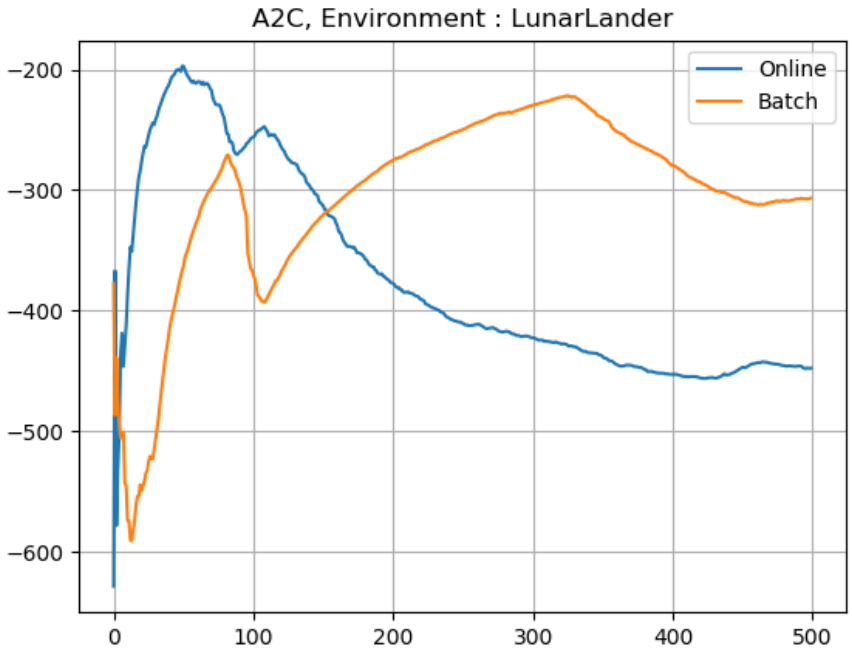
TME 5 : Policy gradients

Un autre type d'algorithme considéré est celui des Actor-Critic. Ces algorithmes considère deux réseaux d'apprentissage différents. Le premier, appelé « Actor », est chargé de faire les choix (Policy-based) tandis que le deuxième, « Critic », se focalise à évaluer le gain espéré pour un état. On a étudié deux méthodes :

- **Online Actor-Critic** : Cette variation fais une mise à jour des poids des réseaux de façon constante à chaque observation réalisée. On met à jour le Critic pour ensuite évaluer une « Avantage » (pouvant

avoir des expressions différentes, s'inspirant par exemple des méthodes TD(λ)) pour mettre à jour ensuite l' Actor.
- **Batch Actor-Critic** : Variation qui consiste cette fois à faire des mises à jour pour une trajectoire entièrement récupéré. Les mises à jour se font alors à la fin de chaque lancement de l'environnement (avoir atteint une fin de trajectoire).

On peut observer dans le graphique ci-dessus la performance des deux versions, à nouveau avec les mêmes paramètres. On voit que Batch semble avoir la meilleure performance, mais il est très probable que de meilleurs paramètres puissent être utilisés pour une meilleure comparaison. Néanmoins, on peut voir l'apparent problème de stabilité car il semble que les algorithmes apprennent à certains moments et ensuite la performance diminue.

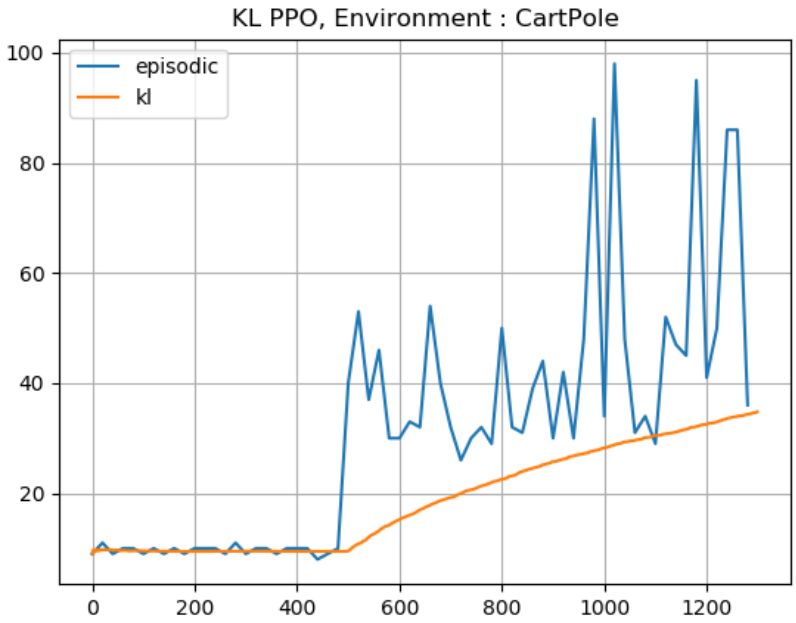


TME 6 : Advanced policy gradients

Dans ce TME, il était question d'implémenter deux version de l'algorithme de Proximal Policy Optimisation, faisant partie des algorithmes d'optimisation basées sur les méthodes de Policy gradients, essayant de résoudre certains de ses problèmes comme stabilité ou robustesse.

- **PPO Adaptative KL** : Cette première version rajoute au coût une partie de divergence de Kullback-Leibler avec un paramètre adaptatif, permettant de considérer les changement brutaux entre deux tours de mise à jour
- **PPO with Clipped Objective** : Version qui s'adapte selon le calcul de l'avantage. Globalement, la perte considérée est adapté selon le rapport entre la nouvelle politique et l'ancienne, en limitant les grandes variations, « clippant » ou conservant seulement les valeurs dans la loss dans un intervalle ou zone de confort.

Pour ce TME, le seul algorithme que j'ai réussi à faire converger est la variante KL, où l'on peut voir ici le graphique correspondant à la fois à la moyenne cumulée (orange) et aux recompenses tous les 20 episodes. Pour ce cas, l'algorithme se met à jour tous les 250 épisodes, ce qui explique le saut à 500, et qui explique aussi que la moyenne cumulée reste assez basse bien que les récompenses deviennent significativement plus importantes.



TME 7 : Continuous actions

Ce TME introduit un nouveau type d'environnements, cette fois considérant des actions qui sont dans un espace continu et non pas discret comme auparavant. Pour ce type de cas on utilise par exemple un réseau pour générer des paramètres pour une distribution gaullienne qui nous permet ensuite d'obtenir les actions à réaliser. La méthode étudiée était la suivante:

- **DDPG** : Cette méthode est une melange d'une méthode DPG et de l'ajout de Memory Replay et Target Network (TME 4). La méthode DPG consiste non pas à générer les paramètres d'une gaussienne mais de générer directement un vecteur d'actions (qui sera bruité par la suite pour l'exploration) à partir d'un réseau profond. Comme pour les TME précédents, on utilise une méthode Actor-Critic avec les deux réseaux qui lui correspondent.

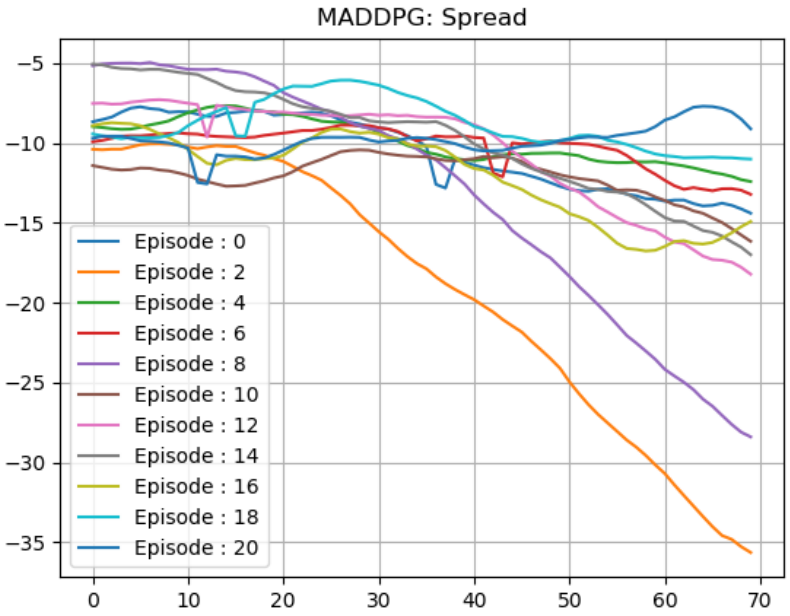
Je n'ai pas réussi à trouver les bons paramètres pour faire converger le DDPG, il est parfois observable qu'à des certains moments l'agent paraît apprendre sur LunarLanderContinuous mais il semble devoir être lancé un très grand nombre de fois.

TME 8 : Multi-Agents RL

Le dernier TME qui traite de la partie Renforcement du cours. Cette fois on considère des environnement à plusieurs agents qu'il faut apprendre, que ce soit à coopérer, à s'opposer ou tout mélange de ces tâches.

- **MADDPG** : C'est une méthode utilisant la méthode DDPG, en créant des réseaux pour chacun des agents à entraîner. Il est important de préciser que plusieurs approches peuvent être considérées telles que le partage d'un Critic pour les agents ayant pour but d'apprendre à réaliser le même type de tâche, un Critic différent pour chacun, et différentes façons de partager l'information.

Encore une fois, ce programme a été difficile à régler correctement afin de le faire converger dans un temps raisonnable. Ci-dessus on peut trouver un example lors de l'apprentissage pour un groupe d'agents pour l'environnement de cooperation. On peut observer, bien que assez peu significatif, que vers la fin, les agents semblaient enfin commencer à comprendre le fonctionnement à avoir.



TME 9 : Generative Adversarial Networks.

Ce TME est consacré à l'étude d'un paradigme dans l'apprentissage profond qui utilise la modélisation de distributions des données. Ce paradigme consiste à l'application d'un réseau GAN à la génération de visages.

- GAN : Type de structure concentré sur des tâches de génération qui utilise deux types de réseaux. Le premier réseau, appelé Discriminateur, cherche à trouver la difference entre des données acquises et des données générées. Le deuxième réseau, appelé Générateur, cherche à son tour à générer de fausses données essayant de tromper le Discriminateur. L'entraînement consiste alors à améliorer le deux réseaux, on parle alors de méthode adversarial car les tâches sont opposées l'une à l'autre (Discriminer et Tromper), permettant d'obtenir de meilleurs résultats de vraisemblance lors de la génération.

On peut observer dans les images, l'évolution de l'algorithme correspondant au générateur du GAN pour la génération de visages. Bien que loin de générer des images parfaites, on peut voir que au bout de 4 époques, l'algorithme à commencer à fournir plus de details au niveau du visage, et à avoir moins de « bruit ».

TME 10 : Variational Auto-Encoder

Le dernier TME, également axé autour de l'apprentissage profond et la modélisation des distributions, était focalisé sur une tâche de reconstruction des données MNIST à travers d'un auto-encodeur.

- VAE : C'est une structure similaire à celle d'un auto-encodeur traditionnel, où l'on trouve deux réseaux: un encodeur et un décodeur. Néanmoins, ce type d'auto-encodeur va essayer d'encoder l'information et obtenir des paramètres d'une loi de distribution et le décodeur est censé reconstruire les données à partir d'une génération selon la distribution respective.

On peu observer dans les images suivantes les performances pour deux VAE différents avec en première ligne l'image orignal et ensuite les images produites par les algorithmes toutes les deux époques. Tous les deux ont presque les mêmes paramètres, différant seulement au niveau de la dimension de l'espace latent; pour celui de gauche on a un espace latent de 2 tandis que pour celui de droite on a un espace latent de dimension 10.

On peut voir l'impact alors de cette dimension car, le premier algorithme semble reconstruire très pauvrement les images, tandis que le deuxième au bout de cinq itérations semble reconstruire presque toutes les images, à part, par example ici, un huit qui paraît être un cas exceptionnel pour l'image source.