ENSEIRB-MATMECA

Rapport du projet IF222

Auteur :
Tarek LAMMOUCHI

Responsable du projet :
Guillaume LAGARDE

31 mai 2020

1 CartPole

Dans cette première partie du projet, nous allons attaquer le problème du Cart Pole, donc pour commencer j'ai pris le code du secend TP et j'ai essayé de l'adapter à notre situation actuelle, il n'y avait qu'un seul problème qui ressortait de le début :

1.1 Discrétisation de l'état

NB : Cette solution est inspirée du livre Reinforcement Learning : An Introduction de Richard S. Sutton et Andrew G. Barto

Au début, j'ai choisi de représenter chaque état par un entier (182 états au total), chaque entier est mappé sur un tuple spécifique (pos, pos_v, ang, ang_v), chaque variable renvoyée par l'environnement est d'abord discrétisée selon les intervalles suivants, ensuite, ils sont fusionnés en utilisant l'exponentiation pour obtenir un entier :
Cart Position (pos) : [-2.4,-0.8[, [-0.8,0.8[, [0.8,2.4]
Cart Velocity (pos_v) :]-Inf,-0.5[, [-0.5,0.5[, [0.5,+Inf[
Pole Angle (ang) : [-41.8°,-6°[, [-6°,-1°[, [-1°,0°[, [0°,1°[, [1°,6°[, [6°,41.8°]
Pole Velocity At Tip (ang_v) :]-Inf,-50°[, [-50°,50°[, [50°,+Inf[

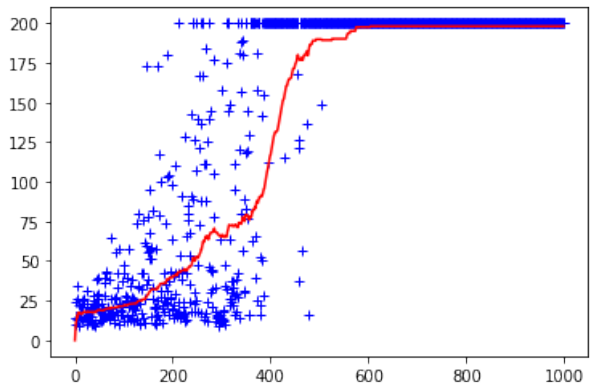
Comme nous pouvons le voir, les intervalles sont petits pour les valeurs autour du milieu et plus grands aux extrémités, par exemple pour l'angle du pôle notre objectif est de stabiliser le pôle, donc les intervalles autour de 0 sont petits car ici nous devons faire des réglages fins
Les valeurs exactes sont inspirées des ressources en ligne, notamment le livre mentionné ci-dessus

Si je m'arrête ici, tout fonctionnera comme prévu, mais j'ai choisi d'aller plus loin et d'essayer de réduire le nombre d'états (actuellement 182) car il faut un certain temps pour trouver la meilleure politique.
Donc, en regardant l'animation de ma solution, j'ai remarqué que le chariot ne se déplaçait pas vraiment souvent, et je n'avais besoin d'équilibrer le poteau que pendant 200 pas de temps, ce qui n'était pas si long. aussi, le chariot ne dérivait généralement pas aussi loin en équilibrant le poteau.
C'est pourquoi j'ai essayé de supprimer les "pos" et "pos_v" de la discrétisation et de ne laisser que les "ang" et "ang_v", et avec cela, j'ai remarqué une réduction significative du nombre d'états (seulement 18 états) et par conséquence le temps de training.

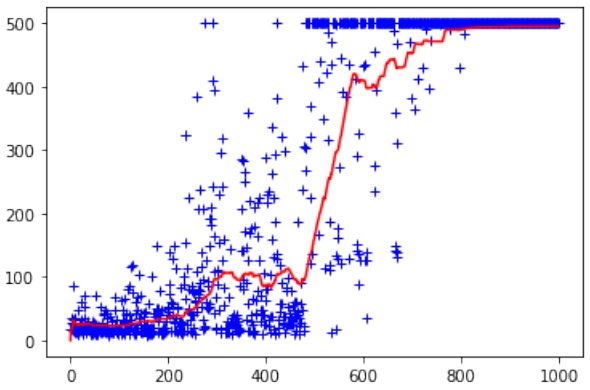
1.2 Résultat

Le training n'a pas été si longue en raison du nombre réduit d'états, et comme vous pouvez le voir sur le graphique, je me retrouve avec une moyenne de 200 pour la v0 et 500 pour la v1 dans les 100 dernières itérations, cela n'a pris que l'agent entre 400 et 600 itérations pour converger vers la meilleure politique.

NP : En raison de l'extrême réduction du nombre d'états, les résultats de V1 ne sont pas cohérents car parfois je me retrouve avec des résultats pires (converge à 300 au lieu de 500)



(a) V0



(b) V1

FIGURE 1 – Résultat Ex1

2 MountainCar

Dans cette seconde partie du projet nous allons attaquer le problème de la voiture de montagne, donc pour commencer j'ai pris le code du premier TP et j'ai essayé de l'adapter à notre situation actuelle, deux problèmes qui étaient évidents dès le départ :

2.1 Discrétisation de l'état

Pour la discrétisation, j'ai choisi de représenter chaque état par un couple composé de deux entiers qui représentent respectivement l'indice de l'intervalle de la position et l'indice de l'intervalle de la vitesse :
(p, v) *avec* $0 <= p < pos\_size$ *et* $0 <= v < vel\_size$
les variables pos_size et vel_size représentent le nombre d'intervalles utilisés pour discrétiser respectivement la position et la vitesse.

Pour déterminer ces variables, j'ai essayé diverses configurations jusqu'à ce que je me retrouve avec une configuration qui semble raisonnable, pour pos_size :
$pos\_size = 9$: [-1.2, -1. , -0.8, -0.6, -0.4, -0.2, 0. , 0.2, 0.4, 0.6]
chaque paire représente ici les deux bornes de l'intervalle de la position : [-1.2,-1.0]]-1.0,-0.8]]-0.8,-0.6] ...
et pareil pour vel_size :
$vel\_size = 14$: [-0.07, -0.06, -0.05, -0.04, -0.03, -0.02, -0.01, 0. , 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07]

Donc par exemple l'état $(1, 2)$ représente $-1.0 <= position <= -0.8$ *et* $-0.05 < vitesse <= -0.04$

2.2 Déterminer les transitions et les récompenses

Commençons d'abord par celui qui a été le plus facile à définir :
Les récompenses sont toutes initialisées à la valeur 0 et les récompenses pour les transitions menant à une position gagnante (position d'état d'arrivée supérieure à 0,5) sont initialisées à 1.

Pour les transitions, j'ai choisi une approche probabiliste, donc, d'abord la table de transition est initialisée à 0, puis j'ai généré pour chaque état un nombre paramétrable d'échantillons, et pour chaque échantillon, je calcule l'effet de l'action pour déterminer l'état suivant, puis j'ajoute (1/nombre_d'échantillons) à la ligne de la table de transition associée à tuple (état,action,état_suivant)

2.3 Résultat

La fonction value_iteration converge rapidement (en 350 itérations) car j'ai un petit nombre d'états, et après avoir calculé la politique optimale et exécuté les tests, je me retrouve avec une moyenne de -137 sur 100 itérations.

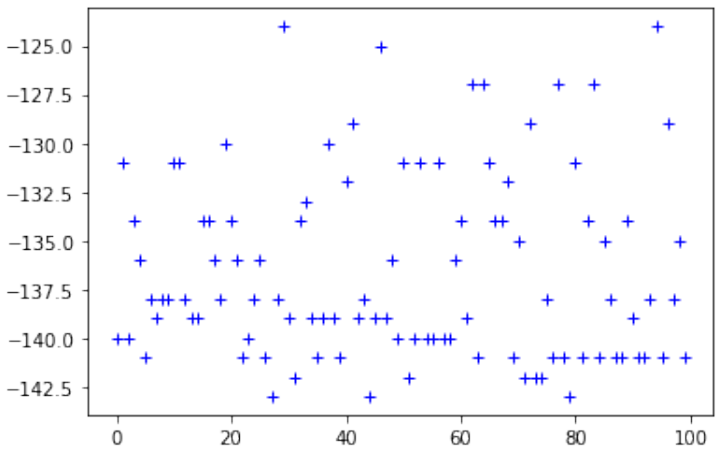


FIGURE 2 – `Résultat Ex 2`