Ejercicio 1

Intro problema

El problema planteado al equipo fue realizar un sistema de estabilizacion de una nave espacial completamente automatica y no tripulada con destino al planeta Marte. Para la resolución de dicho problema se decidio utilizar la tecnica Q-Learning. Está tecnica se basa en el aprendizaje reforzado basado en el cambio de estados y retroalimentacion. Hay 3 factores a tener en cuenta diseñando :

1. Estados
2. Acciones
3. Recompensas

Abordaje

Interaccion con el simulador

La interacción con el simulador fue llevada a cabo dentro de un ambiente gym del tipo cartpole v1 (version limita cantidad max de iteraciones).

Cada paso nos daba como resultado una percepción con 4 datos, los cuales en un principio no utilizamos en su totalidad, y terminamos utilizando para mejorar la performance de nuestro agente.

Params utilizados

El ambiente nos proveía 4 parámetros por observación:

Parámetro	Min	Max
Posición del carro	-4.8	4.8
Velocidad del carro	-inf	inf
Ángulo del palo	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
Velocidad angular del palo	-inf	inf

Como se puede ver, todos los valores obtenidos pertenecen a espacios continuos, por lo cual tuvimos que discretizar las percepciones para poder llevar a un modelo de q learning. Para llevar a cabo esto, utilizamos dos funciones de la librería `numpy` : `linspace` y `digitize` para mapear los espacios continuos a un espacio discreto de bins/buckets y colocar las percepciones obtenidas dentro de los mismos.

A la hora de armar los buckets/bins, tuvimos en cuenta varias cosas que observamos y leimos de la documentación.

La primera es que los valores de posicion del carro y ángulo del palo tenían un rango mayor al que se iban a encontrar la mayoría de las lecturas. Por ejemplo, en el caso de la posición, los valores posibles eran entre -4.8 y 4.8, pero si el carro salía del rango (-2.4,2.4) el juego se terminaba, por lo cual a la hora de armar los buckets los hicimos con los rangos en los que iban a estar la mayoria de valores.

Cuando arrancamos a ejecutar pruebas y ver los Q resultantes de estos buckets, nos dimos cuenta que muchos de los mismos quedaban vacíos, es decir, nunca se actualizaba su valor, lo cual es una perdida para nosotros dado que no mejoraba el modelo.

Fue por eso que tomamos la decisión de realizar las pruebas que se van a encontrar en el archivo `graphs.ipynb`, para determinar de manera correcta como utilizar los buckets para obtener una distribucion al menos similar a normal. No buscamos una distribucion uniforme porque claramente los episodios tendian a distribuirse de otra forma, y el valor que iban a agregar en esos valores borde estaria mejor utilizado aprovechando a actualizar mas los valores mas utilizados.

Fue por eso que finalmente acotamos los espacios lineales de la siguiente manera: . Posicion -0.2 a 0.2 . Velocidad -1 a 1 . Angulo -0.25 a 0.25 . Velocidad Angular -1.5 a 1.5

Haciendo esto y trabajando la cantidad de buckets utilizados para cada param, logramos obtener una matriz Q mucho mas densa y mejor calculada que nuestros primeros intentos.

Tiempo de ejecucion

Se realizaron varias iteraciones de entrenamiento y simulacion con el agente entrenado, en general, las de entrenamiento fueron de entre 1 y 2 hs, con entre 1 y 10 millones de episodios (distinto hardware).

A la hora de probar el agente entrenado, variaba bastante según la calidad del aprendizaje, es decir, cuando aprendía mejor, los episodios eran más largos y por lo tanto duraban más las ejecuciones, pero las pruebas que realizamos con 10 mil iteraciones para promediar el valor obtenido, no pasaban de los cinco minutos.

Para minimizar y agilizar las pruebas se realizo un mecanismo para persistir los resultados de los entrenamientos anteriores.

Resultados obtenidos

Primero que nada utilizamos como benchmark el valor esperado promediando las recompensas obtenidas en 10 mil iteraciones, con el algoritmo aleatorio. Obtuvimos que el valor esperado de este algoritmo es de 22, por lo cual teniamos que obtener valores iniciales por arriba de esto para estar tranquilos de que estaba aprendiendo el agente.

Primero probamos solo con posicion, logrando valores de 15, lo cual era completamente inaceptable. Luego con posicion y velocidad, obteniendo valores por arriba de 22 con consistencia. Al utilizar todos los valores obtenidos por las percepciones, arrancamos a ver valores por arriba de 150 con consistencia, indicando que ibamos por el camino correcto y solo restaba afinar buckets y entrenamiento.

Finalmente, al arrancar a entrenar con más de 2 millones de episodios, arrancamos a ver resultados ampliamente satisfactorios, obteniendo varias veces el valor máximo posible y promediando en el entorno de 350.

La evidencia de estas ejecuciones, se encuentra en los archivos third_approachJP y

Performance y Resultados

Resultados pruebas con distinta cantidad de buckets. Resultados pruebas evaluados utilizando una funcion que descartaba resultados de los bordes

PROBAR CON MAS BUCKETS PARA POSICION/VELOCIDAD (4 CADA UNO) PROBAR ACHICNDO EL LINSPACE DE VELOCIDAD ANGULAR (-1.5, 1.5) Vos quedas con model evaluation function, serializacion y deserializacion del array q, yo quedo con arrancar a investigar lo del 2048

Ejercicio 2

Intro problema

El problema planteado al equipo fue realizar un sistema que jugase solo al 2048 y ganase con una frecuencia razonable. Para resolver este problema se podia utilizar MiniMax o ExpectiMax.

Usamos minimax o expectimax? creo que utilizamos minimax al final.

Explicar nuestra implementacion de pruning.

Interaccion con el simulador

Las interacciones con el simulador fueron bastante menos interesantes en este caso,

Funcion de evaluacion

Pruebas iniciales

Cantidad espacios vacios Cantidad merges posibles

Version final