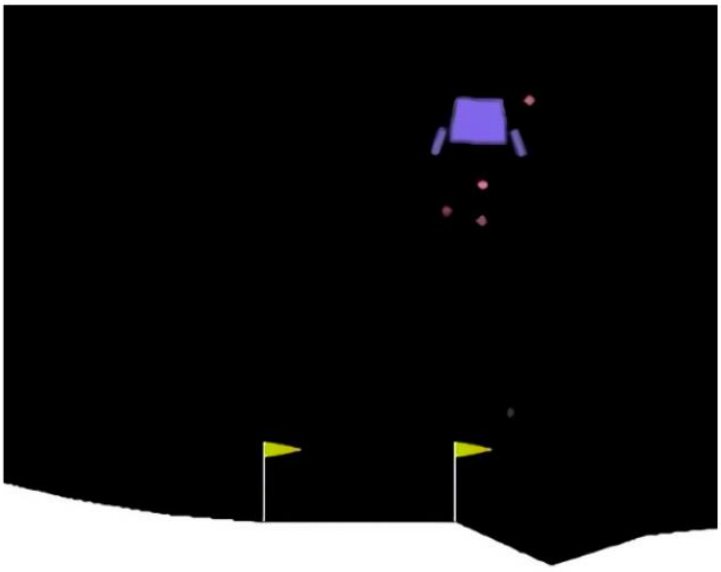
APRENDIZAJE REFORZADO

IMPLEMENTACIÓN DE ALGORITMO DQN PARA EL APRENDIZAJE REFORZADO EN EL ENTORNO LUNARLANDER-V2



VNIVERSITAT DE VALÈNCIA

Trabajo realizado por: Luis Enrique Palma | 2023

Índice

Índice de Ilustraciones

1. Descripción del problema

El problema que me he propuesto resolver corresponde a "*Lunar-Lander-v2*", este entorno es parte de los entornos Box2D y a continuación se explica en que consiste el juego.
El objetivo del juego es optimizar la trayectoria de un cohete espacial, de acuerdo con el principio de Ponrtyagin, es óptimo encender el motor a toda velocidad o apagarlo, razón por la cual el entorno tiene acciones discretas: motor encendido o apagado.
La plataforma de aterrizaje siempre tiene coordenadas (0,0). Las coordenadas son los primeros dos números en el vector de estado. Aterrizar fuera de la plataforma de aterrizaje es posible. El combustible es infinito, por lo que el agente puede aprender a volar y aterrizar correctamente después de su primer intento de aterrizaje.

2. Espacio de Acción (Action space)

Hay cuatro opciones discretas:

0	Hacer nada
1	Accionar motores de propulsión izquierda
2	Accionar el motor principal
3	Accionar motor de propulsión derecha

En la ilustración 1 se marcan las posibles de acción.

3. Espacio de observaciones (Observation space)

El estado es un vector de 8 dimensiones: las coordenadas del cohete en x e y, sus velocidades lineales en x e y, su ángulo, velocidad angular, y dos booleanos que representan si cada pierna está en contacto con el suelo o no.

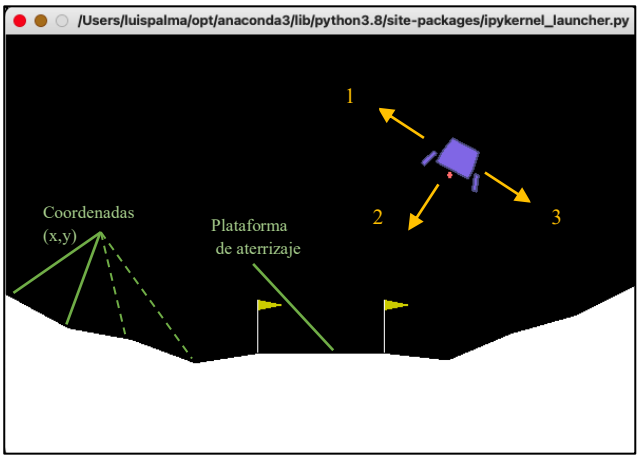


Ilustración 1. Entorno 'Lunar-Lander-v2'

4. Recompensas (Rewards)

Después de cada acción se otorga una recompensa. La recompensa total de un episodio es la suma de las recompensas de todas las acciones dentro de ese episodio.

Por cada acción, la recompensa:

- Aumenta/disminuye cuanto más cerca/más lejos está el cohete de la plataforma de aterrizaje.
- Aumenta/disminuye cuanto más lento/rápido se mueve el cohete.
- Disminuye cuanto más se inclina el cohete (ángulo no horizontal).
- Se incrementa en 10 puntos por cada pierna que esté en contacto con el suelo.
- Se reduce en 0,03 puntos cada fotograma que dispara un motor lateral.
- Se reduce en 0,3 puntos cada fotograma que dispara el motor principal.

5. Terminación de episodios

El episodio termina si se da una de las siguientes situaciones:

- El cohete se estrella (el cuerpo del cohete tiene contacto con la superficie).
- El cohete se sale de la pantalla.
- El cohete deja de moverse.

6. Explicación del código utilizado

En el notebook está comentado todo lo que se ha realizado línea por línea. Sin embargo, como resumen, se ha utilizado el algoritmo *DQN* (*Deep Q-Network*) para entrenar un agente. Se incluye la configuración del entorno, se han definido unos hiperparámetros y estructura de datos (replay buffer). Luego, se ha creado la arquitectura de red neuronal con una capa de entrada con el tamaño del estado, tres capas ocultas de 256 neuronas y una capa de cuatro (tamaño de acciones) salidas. El motivo por el que se escogió esa arquitectura de red es porque fue la que ha dado el mejor rendimiento, se probó con capas ocultas de tamaño “128,64,32,16”, “128,128”, “,150,120”.
Se ha creado un bucle de 1000 episodios en que el algoritmo inicia explorando el entorno hasta que se almacena en la memoria al menos el tamaño del lote de la red. Luego, utiliza la política *épsilon-greedy para* decidir si el algoritmo explora o explota.

Se ha creado un bucle de entrenamiento del agente de 1000 episodios, en cada episodio se reinicia el entorno y se actualizan los estados del entorno. Se registra el tiempo ejecutado, la recompensa total y el valor de épsilon. Durante el entrenamiento almacena las transiciones de experiencia en la memoria, cada 10 episodios se actualiza el modelo principal para que sea más eficiente que si se actualizara por cada episodio.

Observación: Se ha realizado el código con *Pytorch*, al principio estaba escrito con *Tensorflow*, pero la compilación con este último era demasiado lenta, pudiendo hacer 50

episodios en un poco más de un par de horas. Por otro lado, con la utilización de *Pytorch* se podían hacer 1000 episodios en 25 a 30 minutos.

7. Resultados del entrenamiento

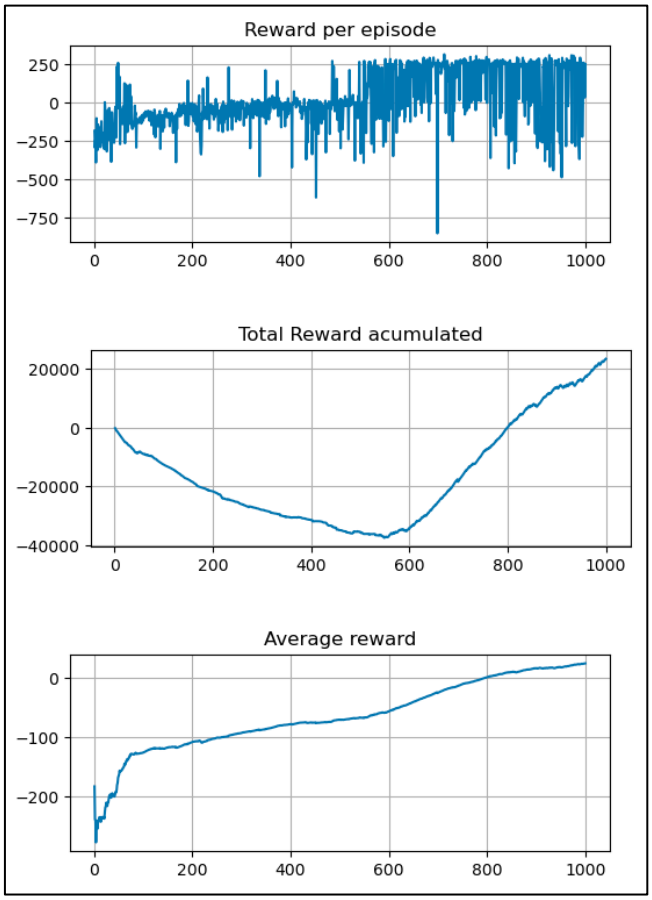


Ilustración 2. Gráficos de desempeño de recolección de recompensas

En cada episodio del entrenamiento se ha guardado la recompensa obtenida por las acciones escogidas (primer gráfico). Se observa que aproximadamente, a partir del episodio 570, el agente empieza a obtener recompensas positivas no aleatorias. Puede comprobarse en el segundo gráfico de Recompensa total acumulada en la que se puede ver que, a partir del episodio mencionado, cambia la tendencia y empieza a crecer. En el episodio 800 se puede observar que el agente ha equilibrado las recompensas negativas recibidas por la exploración y porque no se ha optimizado la red neuronal.

Por otra parte, aunque sé que el agente puede aprender mucho más con un número mayor de episodios de entrenamiento, puede decirse que, hasta este punto, el objetivo de que el agente aprenda ya está cumplido. Por lo tanto, todavía, no podría concluirse sobre que recompensa seria la esperada, sin embargo, con este entrenamiento podría decirse que será positiva, porque el modelo aún no ha convergido. Podemos observar en el segundo gráfico que entre el episodio 800 y 1000 se han acumulado unos 22.000 puntos de recompensa, si se calcula un valor promedio seria 22000 puntos/200 episodios ~ 110 puntos/episodio. Notar que para este razonamiento no se utiliza el ultimo valor del tercer

gráfico, porque este corresponde al promedio de todos los episodios y a este nivel, la recompensa esperada será mayor.

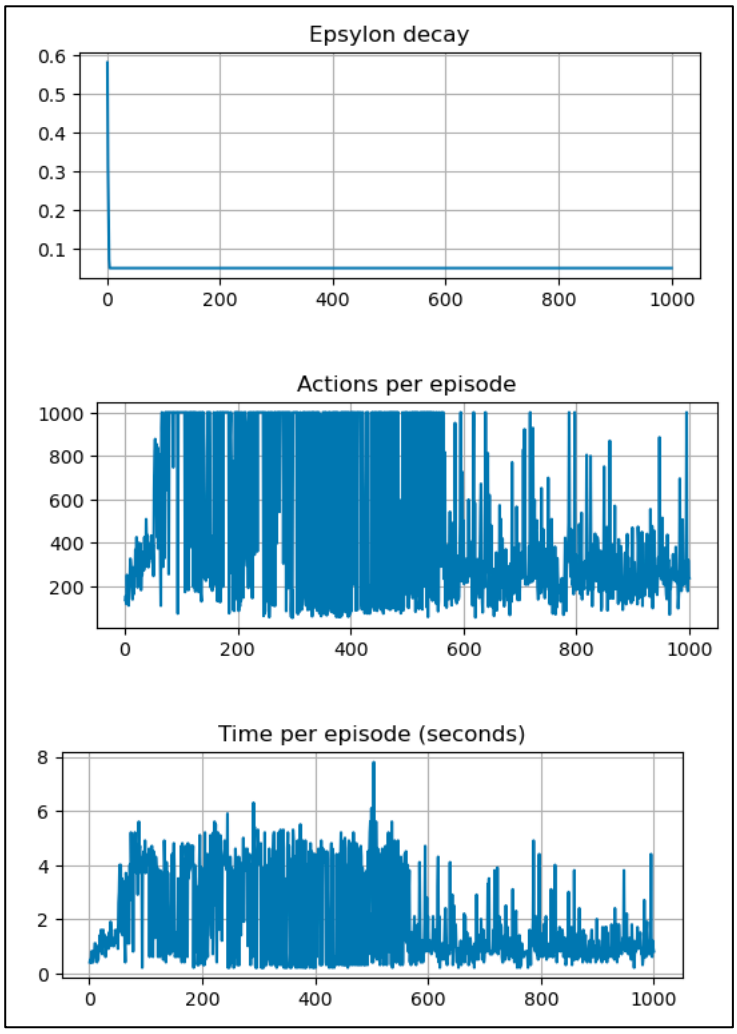


Ilustración 3. Gráficos de desempeño del rendimiento del entrenamiento

Al estudiar los resultados de los gráficos del decaimiento de épsilon, acciones y tiempo por episodios.
Puede observarse que en los primeros episodios el algoritmo consigue alcanzar el valor mínimo de épsilon. Respecto a las acciones y tiempo por episodio puede verse un comportamiento bastante parecido entre ellos, los episodios que estan antes de aproximadamente 570 se comportan de manera bastante errática, por otra parte, cuando se cruza este umbral, se puede ver que se logra estabilizar en un rango más estrecho, con algunos resultados aleatorios, pero notablemente menores en cantidad y rango.

Sitios consultados

Los sitios consultados han servido para recopilar algunas soluciones propuestas por otros usuarios. Y para tener una mejor comprensión del algoritmo DQN.

1. https://gymnasium.farama.org/environments/box2d/lunar_lander/
2. https://github.com/openai/gym/blob/master/gym/envs/box2d/lunar_lander.py
3. https://www.reddit.com/r/reinforcementlearning/comments/g6h7x6/observation_space_of_openai_gym_continuous_lunar/
4. https://shiva-verma.medium.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197
5. https://shiva-verma.medium.com/solving-reinforcement-learning-classic-control-problems-openaigym-1b50413265dd
6. chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
7. https://github.com/yuchen071/DQN-for-LunarLander-v2
8. https://goodboychan.github.io/python/reinforcement_learning/pytorch/udacity/2021/05/07/DQN-LunarLander.html