Assessment 3: Reinforcement learning

Machine Learning II

Máster en Big Data. Tecnología y Analítica Avanzada

Pablo Fernández Pita

Índice

1. Introducción

1.1.Objetivos

El objetivo principal de esta práctica es utilizar un algoritmo de reinforcement learning para solucionar el problema planteado por el entorno de Gym elegido.

En este caso, el entorno elegido es Mountain Car. El objetivo que alcanzar en este entorno es manejar un pequeño coche para que alcance una bandera que se sitúa en lo alto de una montaña.

Si bien este entorno posee, dos versiones, una con un número de acciones continuas y otro con acciones discretas, se optó por la versión sencilla para reducir el tiempo de entrenamiento necesario.

1.2. Descripción del entorno

El entorno escogido tiene las siguientes características:

- Espacio de observaciones:
  - Tenemos un espacio continuo con dos elementos: la posición del coche en el eje x y su velocidad. Estas dos variables son continuas y varían entre -1.2 y 0.6 en el caso de la posición, y entre -0.07 y 0.07 en el caso de la velocidad
- Espacio de acciones: como hemos comentado antes, el espacio es discreto y solo tiene 3 acciones deterministas:
  - Acelerar hacia la izquierda
  - Acelerar hacia la derecha
  - No acelerar

Dado una acción, el coche sigue las siguientes dinámicas de transición:

$$velocidad_{t+1} = velocidad_t + (acción - 1) * fuerza - cos(3 * posiciónt) * gravedad$$

$$posición_{t+1} = posición_t + velocidad_{t+1}$$

donde la fuerza = 0.001 y la gravedad = 0.0025. Las colisiones en ambos extremos son inelásticas con la velocidad establecida en 0 al colisionar con la pared.

La posición inicial del coche se asigna un valor aleatorio uniforme en el intervalo [-0.6, -0.4]. La velocidad inicial del coche siempre se asigna a 0.

1.3.Final del episodio

El episodio termina cuando el coche llega a la bandera objetivo o se llega al episodio 200 sin alcanzar dicha bandera.

1.4.Métodos de RL a aplicar

En este estudio, se ha optado por emplear el método Q-learning para abordar el problema. Esta decisión se fundamenta en la reputación de Q-learning como uno de los algoritmos más sencillos de implementar en el ámbito del aprendizaje por refuerzo. A diferencia de otros enfoques, como Reinforce, DQN o A2C, que, si bien son potentes y pueden resolver una amplia gama de problemas, también requieren mayores recursos computacionales. Por el contrario, Q-learning es más liviano en términos de demanda de recursos, lo que lo hace ideal para este proyecto. Con Q-learning, se confía en que se pueden lograr resultados significativos sin la carga adicional de complejidad computacional, lo que permite centrarse en la esencia del desafío y obtener resultados prometedores de manera más eficiente.

2. Implementación del algoritmo

En este apartado se explorará la implementación realizada para el algoritmo de Q-learning desde Python, las librerías utilizadas y el funcionamiento del script.

2.1.Transformaciones realizadas al entorno

Si bien como hemos mencionado con anterioridad el espacio de entrada es continuo y por tanto potencialmente infinito, se puede discretizar sencillamente para que nuestra matriz Q sea de un tamaño comedido.

Esto se realiza con la función linspace de numpy. Con esto reducimos las dimensiones de nuestra matriz Q a 40x40x3. Cuarenta fue el número elegido para dividir la velocidad en trozos y también para la velocidad. Elegimos dividir tanto la posición como la velocidad en 40 partes iguales para mantener un equilibrio entre la granularidad de la representación y la eficiencia computacional.

2.2. Hyperparámetros utilizados

Para entrenar nuestro algoritmo de Q-learning se han utilizado los siguientes valores:

- Learning rate: 0.9
- Discount factor = 0.9
- Epsilon = 1
- Epsilon decay = 2/num_episodes

2.3. Explicación algoritmo programado

El algoritmo de Q-learning se implementa dentro de un bucle principal que itera a lo largo de un número dado de episodios. En cada episodio, el agente comienza en un estado inicial y toma acciones basadas en una política epsilon-greedy. Si el agente está en modo de entrenamiento, la política puede ser más exploratoria con una probabilidad epsilon, de lo contrario, elige la acción óptima según la matriz Q actualizada.

Después de cada acción, se actualiza la matriz Q utilizando la regla de actualización de Q-learning, que tiene en cuenta la recompensa recibida y el valor máximo esperado de la próxima acción. Esta actualización se realiza con la tasa de aprendizaje especificada.

El proceso continúa hasta que se alcanza un estado terminal o se alcanza el límite de episodios definido. Durante el entrenamiento, se imprime el estado actual del proceso cada 100 episodios para monitorear el progreso.

Finalmente, la matriz Q resultante se guarda en un archivo para su posterior uso. Además, se calcula y traza la recompensa media por episodio a lo largo del tiempo para evaluar el rendimiento del algoritmo.

2.4.Formas de correr el programa

El programa “mountain_car.py” está pensado para entrenar y testear el modelo al mismo tiempo. Por eso toma como parámetro la acción que quieres que tome. Si le añades el parámetro *train* el algorimto entrenará durante 5000 episodios usando el algoritmo explicado anteriormente. Nos mostrará por consola algunas métricas cada 100 episodios y el tiempo de entrenamiento total.

Si, por el contrario, ponemos el modo *test* el programa renderizará 10 iteraciones del algoritmo jugando en el entorno y logrando su objetivo, siempre y cuando haya sido entrenado correctamente.

Además, durante ambos modos hace un diagrama de las recompensas obtenidas de media por cada episodio para observar la evolución del algoritmo y detectar posibles problemas.

3. Resultados obtenidos

Con el modo test implementado y explicado anteriormente, podemos comprobar que en cada una de las 10 iteraciones del algoritmo que vemos, llega a su objetivo.

Con esto, el objetivo base del proyecto está cumplido, pero no solo queremos quedarnos ahí, sino también observar cual ha sido la evolución del agente durante el entrenamiento, y observar cuanto ha tardado en ejecutarse.

El algoritmo, con los hiperparámetros mencionados anteriormente, tarde únicamente 103 segundos en entrenarse. Este tiempo es verdaderamente bueno y demuestra que Q-learning a pesar de ser una técnica básica y más antigua, funciona muy bien cuando el problema se ajusta a sus limitaciones que normalmente vienen por el tamaño de la tabla Q.

Métricas de rendimiento

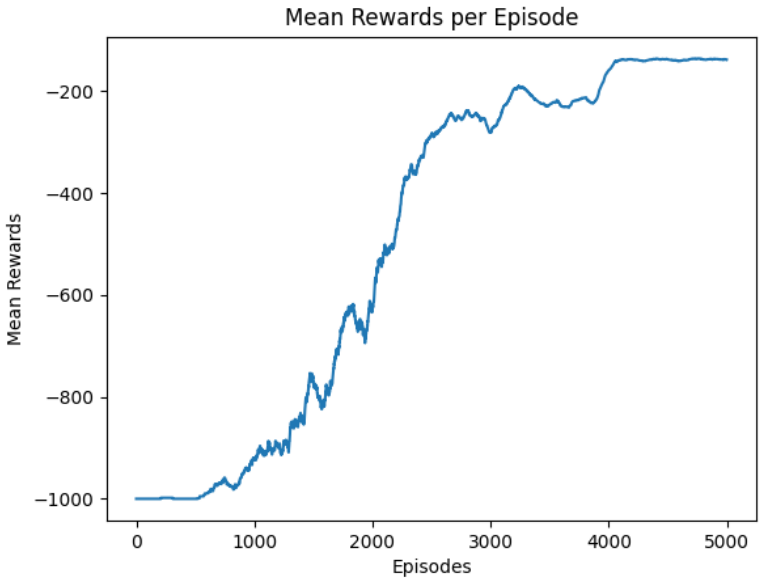

Figura 1. Recompensa media por episodio

Como podemos observar en el gráfico de entrenamiento, durante los primeros 1000 episodios, la mejora es muy lenta y el agente tiene un desempeño malo. Sin embargo, experimenta una mejoría muy notable entre los episodios 1000 y 3000. Si hubiéramos parado el algoritmo ahí, ya sería bastante bueno. Sin embargo, le queda por mejorar aún, como se aprecia, hasta las 4200 iteraciones, cuando realmente se vuelve perfectamente estable y obteniendo las mejores recompensas.

Como el objetivo es llegar a la parte de arriba de la montaña lo antes posible, el agente acaba con un reward negativo debido a como está configurado el entorno. Ya que este es penalizado con -1 por cada paso hasta que llega a la bandera. Por tanto, es imposible obtener una recompensa netamente positiva y esto no debe preocuparnos.

4. Conclusiones

Para validar los resultados, se realizó una prueba adicional en la que se comparó el desempeño del agente Q-learning con un agente aleatorio en el mismo entorno. El agente aleatorio no mostró mejora alguna en su desempeño a lo largo de los episodios como si hizo nuestro agente de Q-learning.

Aprovechando la forma de Q-learning para aprender una política óptima a través de un proceso de ensayo y error, sin necesidad de conocer el modelo subyacente del entorno he logrado un resultado óptimo en un tiempo mínimo, que era uno de los objetivos principales de esta práctica. Esto lo hace especialmente útil en situaciones donde el modelo es desconocido o demasiado complejo para ser modelado.

Sin embargo, si que he encontrado una serie de limitaciones con el algoritmo a la hora de probar entornos más complejos como los de Atari. Debido al espacio de observaciones de la consola Atari, crear y ajustar la matriz Q en un espacio continuo con 3 colores y 2 dimensiones me resulto imposible. Discretizar dos dimensiones como la posición en el eje x y la velocidad fue sencillo, pero hacer eso con una pantalla RGB no fue tarea tan sencilla. Las limitaciones por el tamaño de la matriz Q son evidentes en casos más complejos como este, pero considero que conocer bien la forma del algoritmo que sirve de base para modelos más complejos como DQN o doble DQN era más importante que entrenar un modelo más potente y divertido durante horas sin entender bien lo que está haciendo por detrás.

En conclusión, el algoritmo Q-learning demostró ser una excelente elección para resolver el problema planteado en el entorno Mountain Car y me ha ayudado a comprender aspectos tan importantes en el reinforcement learning como el espacio de observaciones y de acciones.

5. Anexo

Adjuntos junto con este reporte se encuentra una demostración del modo de test del algoritmo ya entrenado jugando durante 10 iteraciones en este entorno.