Aplicación de conceptos generales de RL a varios entornos.

Proyecto Final

Ulises Díez Santaolalla
Jorge Enebral Alonso
Ignacio Felices Vera

Aprendizaje Por Refuerzo
4º Grado en Ingeniería Matemática e Inteligencia Artificial

Tabla de Referencias

1. Navegación con tile coding

1.1 Objetivo:

El objetivo es que el agente llegue al área objetivo sin chocarse o con paredes o con las estanterías. Se busca que aprenda una política óptima que maximice la recompensa en cada episodio.

1.2 Agente:

El agente usado para este apartado es SARSA. Es un agente lineal, on-policy y online. Actualiza la Q usando la acción elegida en el estado que está, y balancea exploración con explotación mediante un épsilon-greedy, que inicializa en 0.5 y va decreciendo con los episodios de entrenamiento.

1.3 Representación del estado:

El entorno es continuo, y como se usa un agente lineal se necesita discretizar las variables de estado. Para ello hacemos uso de Tile Coding, que generan múltiples particiones solapadas para obtener una codificación dispersa y generalización. De esta forma estados cercanos comparten características. El vector de estado final es binario, con pocas tiles activas por cada observación.

1.4 Sistema de recompensas:

- Recompensa por llegar al objetivo: +1.

- Penalización por avanzar: 0.

- Penalización por colisión: -1.

1.5 Resultados:

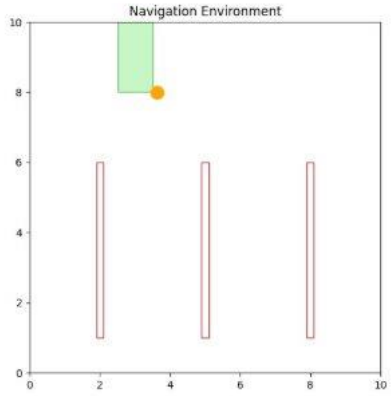

Figura 1. Instantánea de cómo el agente llega al objetivo al final del episodio.

2. Operación en almacén

2.1 Objetivo:

El objetivo es que el agente depende del entorno, pero debe cumplirlo sin chocarse o con paredes o con las estanterías. Se suponen estanterías fijas y área objetivo fija.

	Objetos Fijos	Recoger Objeto	Entregar Objeto
Entorno 1	SI	SI	NO
Entorno 2	SI	SI	SI
Entorno 3	NO	SI	SI

Figura 2. Tipos de entornos y sus objetivos

2.2 Elección del agente:

Para afrontar el problema podemos elegir un agente que maneje el estado linealmente o no linealmente.

- Si elegimos un agente que maneje el estado linealmente como un SARSA necesitamos hacer una discretización del estado para que el agente pueda aproximar el estado. Para ello necesitamos usar técnicas como tile coding, coarse coding, etc...
- En cambio, si elegimos un agente que maneje el estado no linealmente como una DQN no necesitamos discretizar el estado porque la red neuronal ya aproxima el estado no linealmente, por lo que no hace falta tile coding u otras técnicas.

Para la elección final hemos valorado dos aspectos: qué tan fácil es implementarlos y cómo de potente es el agente. En ambos aspectos gana la DQN, ya que se puede importar de stable-baselines3 y prácticamente solo hay que modificar hiperparámetros, no hay que discretizar estado, y además una red neuronal es más capaz de generalizar y aprender.

2.3 Hiperparámetros:

Para la selección de hiperparámetros se fue iterando por ellos para diferentes valores, haciendo seguimiento de la evolución del entrenamiento para cada uno de ellos, y quedándonos con los siguientes, los cuales resultaron más óptimos:

- El diseño de la MLP es de 2 capas ocultas de 128 neuronas cada una.
- Para el factor de exploración hemos usado que comience en 1 y termine en 0.01, decayendo casi linealmente hasta la mitad de los timesteps seleccionados.

- El resto se seleccionaron por prueba y error: un tau bajo para que sea estable. Actualizaciones de la red de estimaciones a la objetivo frecuentemente, cada 5 timesteps. Un gamma de 0.95, normal. Un learning rate de 5e-4, normal. Un buffer size grande para que haya muestras variadas y un batch size de 128.

2.4 Ingeniería de Variables

Tras entrenar el agente con las observaciones iniciales obtuvimos poca tasa de éxito, por lo que decidimos añadir a la observación también las estanterías (su punto central x,y y la altura y ancho), junto con la distancia al objeto más cercano, la distancia al área objetivo, y la dirección normalizada del agente al objetivo actual (que puede ser el objeto más cercano o el área objetivo, dependiendo de si el agente tiene un objeto o no).

2.5 Sistema de recompensas:

Inicialmente entrenamos el agente con el código base y poniendo ciertos valores a las recompensas. Nos entrenaba, pero la tasa de éxito era muy baja.

Es por ello que tuvimos que añadir otras recompensas, empezando por recompensar al agente por acercarse al objetivo (si no tiene objeto al acercarse al objeto más cercano y si tiene objeto al acercase al área de entrega). Con esto ya pegó un salto en el entrenamiento, pero vimos que varias veces la recompensa era muy alta. Esto era porque la recompensa de acercarse era mucho mayor que la de alejarse + penalización por avanzar, por lo que se quedaba en bucle ganando recompensas. Esto lo solucionamos en vez de sumar o restar X cantidad fija si se acerca o aleja escalando la diferencia entre la distancia antigua menos la nueva. Si es positivo -> recompensa positiva, sino negativa. Con esto alcanzamos un 70% de éxitos en el apartado 2, pero muchas veces se quedaba aun así en bucle de arriba-abajo, izquierda-derecha, pick-drop.

Para evitar los bucles creamos varias listas de memoria al agente. Si las últimas 4 acciones son de arriba-abajo, izquierda-derecha, pick-drop se da una penalización grande. Y si en las últimas 8 acciones la cantidad de estados en los que ha estado es 3 también se penaliza, porque ha encontrado un bucle repetitivo más complejo.

Además, también ayudó no permitir soltar el agente fuera del área objetivo, solo dando recompensa negativa.

Como resultado objetemos:

- Recompensas:
  - Coger objeto: +30
  - Entregar objeto: +100
  - Acercarse objetivo: +5
- Penalizaciones:

- Por cada paso: -0.05
- Por colisión: -10
- Por acción inútil: -1.5 (aquí entra coger un objeto en un radio donde no hay objeto o cuando ya tiene un objeto, soltar un objeto cuando no tiene objeto o en un área que no es el objetivo).
- Penalización por bucle: -8
- Penalización por quedarse pillado: -4

2.6 Tracking del entrenamiento del Agente

Para poder evaluar la evolución del entrenamiento del agente, se registraron logs de manera continua a lo largo de su entrenamiento, para de esta manera poder monitorizar el progreso del mismo. De esta manera, y como se puede ver a continuación con los logs del modelo final, se puede visualizar como al inicio debido a la mayor propensión a exploración, los rewards del modelo son bajas y la longitud de sus episodios largas, mientras que según entrena, los rewards consiguen remontar y ascender, mientras que la longitud de los episodios disminuye. También, y de manera directamente relacionada, en la gráfica superior de la esquina derecha se puede ver cómo el número de episodios exitosos asciende, y en la esquina inferior se puede monitorizar la disminución de la tasa de exploración. Cabe destacar que estas gráficas están suavizadas, por lo que sus valores no corresponden a los valores puramente reales.

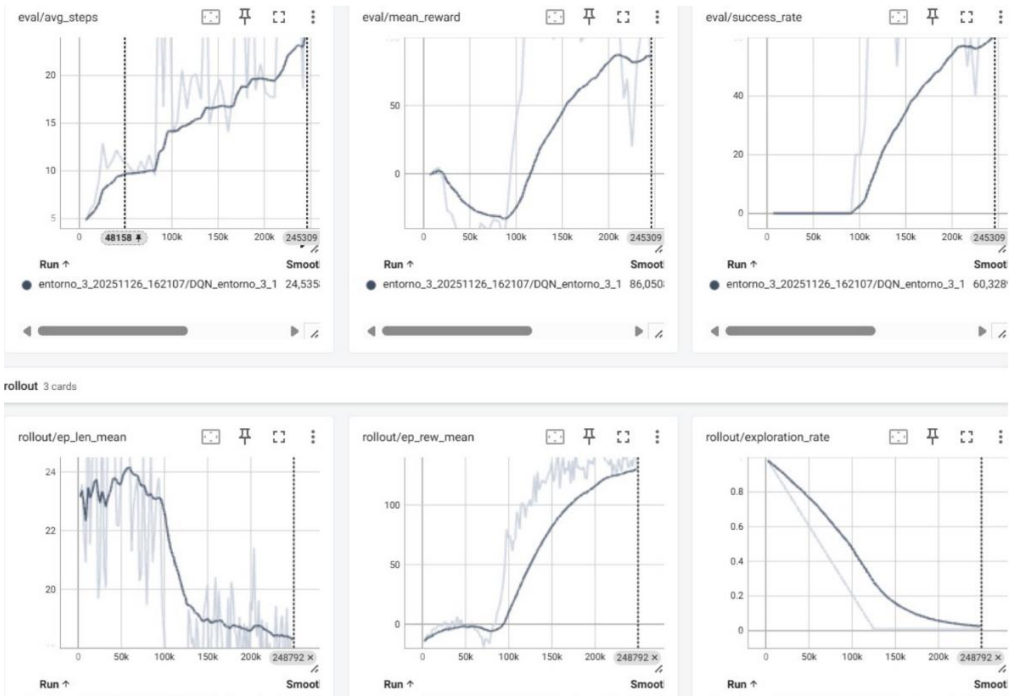

Figura 3. Seguimiento por TensorBoard

Gracias a este seguimiento con TensorBoard, se pudo hacer una selección más fina y precisa de hiperparámetros, analizando la evolución de los entrenamientos de los modelos. Adicionalmente, se monitorizaba la Loss del modelo para poder estudiar cómo esta disminuía a lo largo del entrenamiento del agente, clara señal de que está aprendiendo correctamente.

2.7 Resultados:

2.7.1 Resultados del entorno 1:

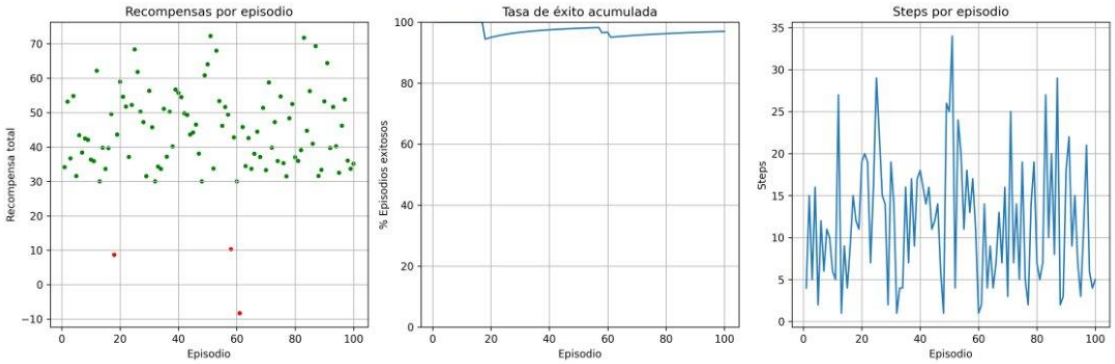

Figura 4. Gráficas resultados entorno 1: recompensas medias por episodio; tasa de éxito acumulada; steps por episodio.

2.7.2 Resultados del entorno 2:

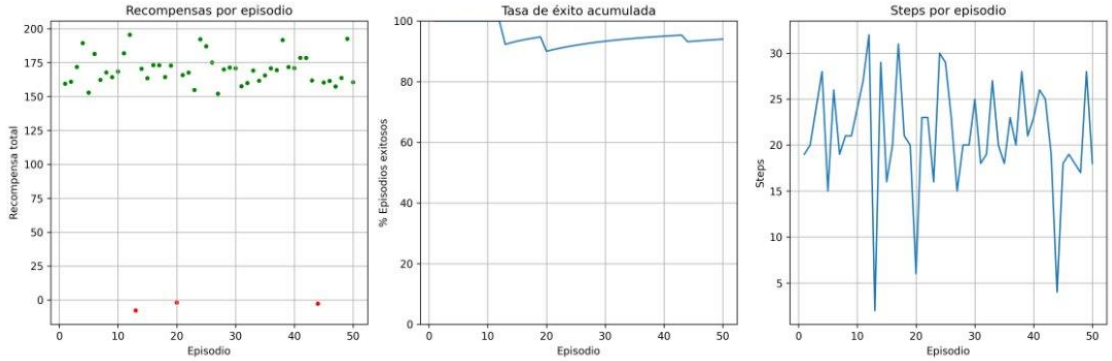

Figura 5. Gráficas resultados entorno 2: recompensas medias por episodio; tasa de éxito acumulada; steps por episodio.

2.7.3 Resultados del entorno 3:

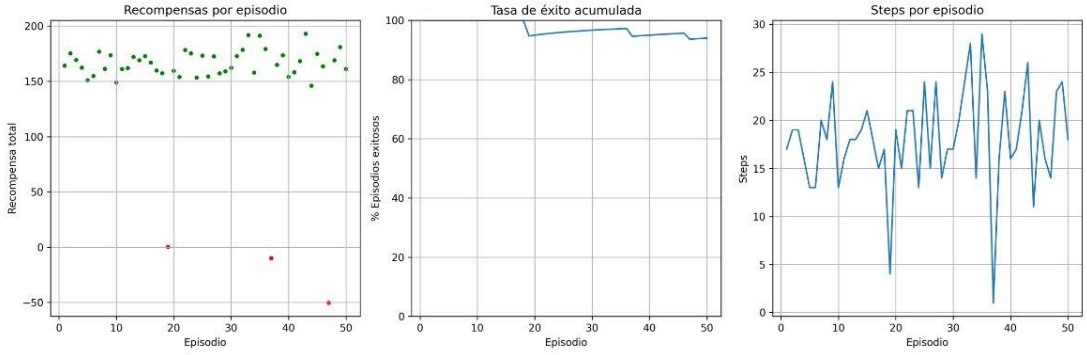

Figura 6. Gráficas resultados entorno 3: recompensas medias por episodio; tasa de éxito acumulada; steps por episodio.