Práctica final – Aplicación de conceptos generales de RL a varios entornos.
Sesión primera. Navegación con tile coding

1. Introducción

En esta sesión comenzamos a trabajar de cara a la presentación final de la asignatura. Para facilitar la comprensión del paso de aprendizaje tabular a función de aproximación, planteamos una práctica guiada en la que aprenderemos a modelar espacios continuos con una discretización que permite generalizar de forma razonadamente controlada la experiencia obtenida desde una posición a un determinado entorno de ésta: el *tile coding*.

El funcionamiento de los métodos que permiten a un agente aprender con función de aproximación puede complicarse mucho. Por ello, con la idea de que puedas comprender con suficiente profundidad este proceso, lo vamos a simplificar notablemente en esta primera sesión. Por un lado, vamos a usar un entorno sencillo, en el que solo tenemos que llegar a una zona objetivo desde una posición de partida aleatoria evitando chocarnos con los obstáculos (paredes). En la *Figura 1* se observa un posible estado del entorno. El agente se representa en color naranja y la zona objetivo en verde.

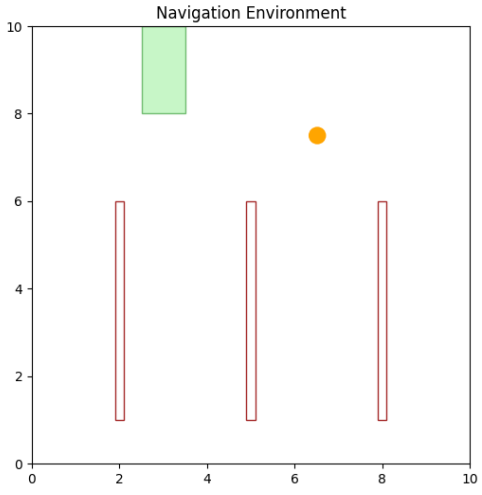

Figura 1: El entorno de navegación

Por otro lado, en parte gracias a la sencillez del entorno, vamos a usar aproximación lineal de la función de valor de acción con *tile coding*.

Detalles del entorno

Planteamos una interacción episódica que termina tanto si el agente llega a la zona objetivo (éxito) como si choca (fracaso) con alguna de las paredes del perímetro o de los obstáculos (en rojo). Por lo tanto, podremos modelar esta tarea como un MDP episódico.

Como puede observarse en la *Figura 1*, el entorno tiene forma cuadrada con 10 metros de lado. El área objetivo es un rectángulo cuyo vértice inferior izquierdo se encuentra situado en (x = 2.5, y = 8), con dimensiones (l_x = 1, l_y = 2). Por otro lado, hay tres obstáculos con dimensiones (l_x = 1, l_y = 5) y cuyos vértices inferiores izquierdos se encuentran situados respectivamente en (x = 1.5, y = 1), (x = 4.5, y = 1) y (x = 7.5, y = 1).

El entorno recibe una acción del agente que, para todos los estados, puede ser $\mathcal{A}(s) = \{arriba, abajo, izquierda, derecha\}$. Entonces, el agente se desplaza 0.25 metros en la dirección elegida. Cuando termina el desplazamiento, el script del entorno devuelve únicamente la nueva posición del agente y dos variables binarias que representan si ha habido colisión o si se ha llegado a la zona objetivo. En ambos casos, el episodio termina (variable *terminated* a True). Es importante notar que estas señales no son la observación que llega al agente. La señal tiene que ser generada procesando posición, colisión y llegada. Para ello, usaremos otro script que construye la realimentación de la observación que llega al agente.

Detalles del agente

Lo definiremos en más profundidad en las secciones de descripción del trabajo. Por ahora, recibe esta señal de realimentación, la procesa y actualiza sus estimadores de la función de valor de acción en la fase de entrenamiento.

2. Trabajo previo

- Lee la sección 9.5.4 de Sutton & Barto: Introduction to reinforcement learning, donde se presenta el concepto de *tile coding*.
- Visita: http://incompleteideas.net/tiles/tiles3.html, la web donde R. Sutton comparte un código para hacer *tile coding* de forma eficiente. Lee la documentación. La biblioteca se proporciona entre los ficheros de la práctica.
- Abre el fichero entorno_navegacion.py. Analiza la clase **Navegacion**. En concreto, observa la numeración de las acciones y la forma en la que el entorno devuelve su observación "cruda" del estado.
- Ahora ve al final del script. Observa que hay un código que instancia el entorno y toma varias acciones aleatorias. Ejecuta el fichero. Verás que aparece una representación del entorno parecida a la de la *Figura 1* pero en la que se observa el área en la que influye el agente desde una determinada posición. Realiza la siguiente prueba:
    - Prueba varios valores de número de tiles por dimensión (x, y) y número de tilings y analiza las diferencias en el área de influencia del agente en función del número de tiles y tilings.
      Recuerda que para visualizar la figura debes tener abierta la aplicación de MobaXterm.

- Analiza el pseudocódigo del algoritmo de semi-gradient SARSA (Sutton&Barto, pg. 244).
- Antes de comenzar la sesión habrá un test evaluable con preguntas sobre este trabajo previo y, en general, sobre la aproximación lineal de función de valor.

3. Ingeniería de variables y diseño de la recompensa

Abre el script *representacion.py*. En el mismo, está descrita la clase **FeedbackConstruction**, que permite construir una pasarela entre el entorno y el agente que se encarga de la ingeniería de variables (representación del estado para obtener una representación que sea una señal de Markov). Observa que en el método de inicialización hay una serie de variables que configuran algunos parámetros para el *tiling*, como el escalado para que funcione la implementación del *tile coding* de Sutton (tiles unitarios dentro de sus funciones), la *hash table* y sus dimensiones máximas en función del número de *tiles* y *tilings*. **No toques esas líneas**.

Recuperación de los tiles activos

En el método *_get_active_tiles(.)*, añade un código que detecte los tiles activos. Para evitar la presencia de artefactos en el área de influencia del agente, desplaza los tilings con offset 3 en el eje x y offset 1 en el eje y.

Cálculo de la recompensa

Para seguir el estándar de *gymnasium*, el cálculo de la recompensa lo vamos a implementar dentro de la función step(.) del entorno de navegación. Puedes integrarlo directamente en la función step o extraerlo en una función del cálculo de la recompensa. Lo que prefieras.

4. Entrenamiento del agente

Abre el script *agente.py*. Vamos a trabajar con la clase **SarsaAgent**. En el método de inicialización verás una serie de atributos que son necesarios, y cuyos valores vamos a controlar cuando instanciemos un objeto agente. No toques esas líneas. Después, tenemos los pesos del modelo inicializados a 0, lo que es razonable para este problema. No obstante, siéntete libre de modificar esto si te pareciera oportuno.

Es importante que notes que hemos reservado un juego de pesos para cada acción. Esto es: el agente reserva un vector de pesos con las dimensiones de la observación que llega de la pasarela de realimentación para cada una de las acciones.

Por último, dejamos un espacio para añadir atributos que te sirvan para monitorizar el proceso de aprendizaje.

Analiza el método *get_action*. Está planteado para muestrear una política ε-greedy, permitiendo hacer que ε sea el valor con el que se inicializa al instanciar al agente, y también que podamos controlarlo dinámicamente durante el proceso de

entrenamiento, algo que es útil en general. Veremos a continuación cómo usarlo. En todo caso, este método ya está desarrollado, no hay que tocarlo.

A continuación, tenemos el método para la actualización de pesos (*update(.)*). Éste, se apoya en el método de cálculo de los valores de q, que también debes resolver tú, en la función *get_q_values*(.). En *update*(.), te damos calculado el error TD. Ahora, tú debes añadir las líneas de código para actualizar los pesos de la función de aproximación de q. Recuerda que los pesos son un atributo del agente. Usa la fórmula de actualización con aproximación lineal.

Después verás el método *train(.)*. Este método está ya desarrollado, pero tienes 3 hiperparámetros con los que jugar, que básicamente modulan la exploración. Se ofrece ya programada una estrategia en la que se explora con ε inicial hasta un determinado porcentaje del número total de episodios del entrenamiento y a partir de ahí se reduce exponencialmente ε hasta un valor mínimo. **NOTA**: Los valores de los tres hiperparámetros son deliberadamente malos, por lo que tendrás que rediseñarlos. Asegúrate de que el agente está suficiente tiempo explorando con un ε bajo al final del entrenamiento para que la política cuyos valores aproxima el agente se parezca a una política determinista, que es como vamos a evaluarla.

También puedes cambiar la frecuencia con la que muestras resultados parciales del entrenamiento. Además, en este método tendrás que actualizar todos los atributos que hayas definido para monitorizar el entrenamiento (número de veces que se ha alcanzado un par estado-acción, longitud de las trayectorias, valores de algunos estados prefijados, etc.).

Finalmente, la clase presenta el método *evaluate(.)*, en el que se simulan episodios moviendo al agente por el entorno con su estimación de q.

5. Evaluación de los agentes

Como habrás observado en el trabajo previo, el diseño del *tile coding* puede impactar en el aprendizaje del agente. Para que practicando comprendas mejor los equilibrios entre particularización y generalización de la experiencia, vamos a medir el rendimiento de dos tipos de agente: 1) un agente entrenado con sólo 10000 episodios de experiencia y 2) un agente entrenado con toda la experiencia que consideres oportuna.
En ambos casos, el banco de ensayos constará de 1000 episodios con punto de partida pseudoaleatorio (por lo tanto, el mismo para todos los agentes).
La evaluación la haremos los profesores (no conocerás esas posiciones iniciales) una sola vez, con los dos agentes que nos envíes.

6. Opcional

Modifica la clase SarsaAgent para convertirla QLearningAgent, de manera que un agente que se apoye en ella aprenda usando Q-learning. Usa también de momento aproximación lineal.

7. Memoria de trabajo

Aunque este trabajo lo vamos a evaluar en la presentación final, debes recoger en un cuaderno de trabajo todo lo que vayas haciendo. Eventualmente podemos pedírtelo para comprobar alguno de los resultados/desarrollos tras la presentación final.