Universidad Carlos III de Madrid

Aprendizaje Automático

Grado en Ingeniería Informática

Grupo 83

Tutorial 4: Introduccíon al aprendizaje por refuerzo

Autores:
Daniel Medina García
alejandro rodríguez Salamanca

15 de abril de 2016

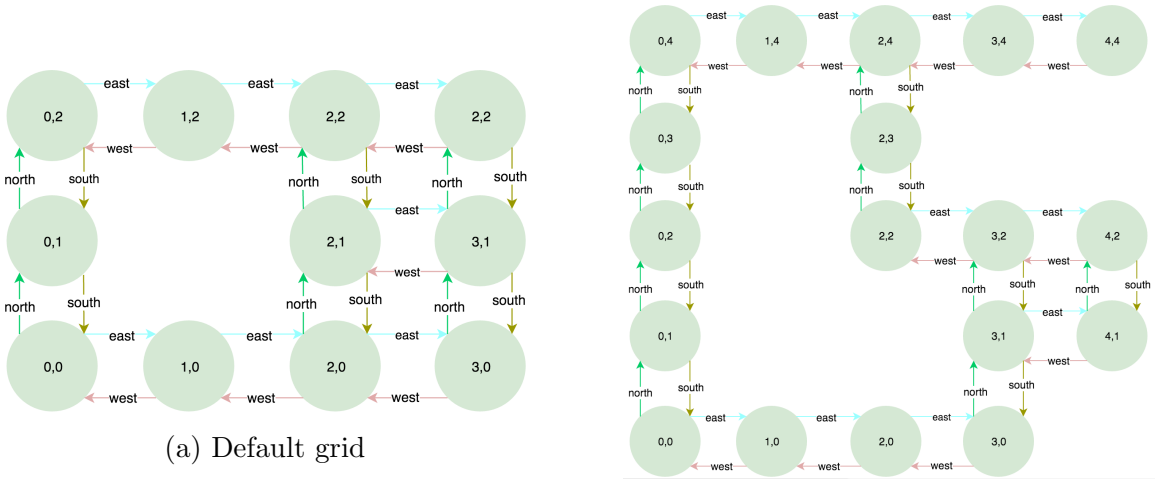
Índice

Introducción

En este tutorial resolveremos una serie de ejercicios con técnicas de aprendizaje no supervisado haciendo uso del método de *Q-learning*. Para ello, implementaremos un agente en Python que recorra un mapa, tratando de obtener las políticas óptimas. Dicho agente realizará movimentos tanto deterministas como estocásticos, para estudiar sus diferencias.

1. Ejercicio 1

En este primer ejercicio tomamos contacto con el programa ejecutando tanto el agente manual como el estándar. Por cada movimiento ejecutado, en la salida por consola aparecen el estado inicial, la acción tomada, el estado al que se llega y la recompensa obtenida. El agente por defecto es *random*, que toma acciones aleatorias entre las permitidas. La Figura 1 muestra los MDP deterministas requeridos en el enunciado del tutorial:



(a) Default grid

(b) AAGrid

Figura 1: Deterministic MDPs

Estos laberintos, al igual que los demás incluidos en el código proporcionado, se almacenan en forma de arrays bidimensionales. Cada posición de la matriz representa una posición del laberinto, con los siguientes símbolos:

- **Un espacio** representa una casilla común, sin recomensa ni coste algunos, por la que el agente puede pasar.
- **Un valor numérico** representa una casilla con una recompensa o un precio. Si el agente se mueve a esta casilla cambiará su puntuación añadiéndole el valor contenido en dicha casilla.
- **Una almohadilla (#)** es considerada un muro, o casilla inalcanzable por el agente. Ninguna acción legal puede llevar al agente a estas posiciones.
- Por último, la posición de la que parte el agente se guarda como **una 'S'**.

De esta forma, podemos crear un nuevo laberinto siguiendo las instrucciones mencionadas. La siguiente figura muestra un laberinto creado por nosotros, en la que el agente deberá buscar una ruta sorteando los muros si quiere conseguir la mayor recompensa:

```
[[' ',' ',' ',' ',]
 ['8','#','3',' ',]
 ['#',' ',' ','#',]
 [' ',' ',' ',' ',]
 [' ','S','#',' ',]]
```

Figura 2: New grid

La elección de la política para guiar a un agente es clave para su éxito, y para ello se busca elegir la política óptima. Sin embargo, esta política no tiene por qué ser única y se caracteriza por compartir el mayor *valor Q* de todas las políticas posibles. Así, para el laberinto por defecto tenemos dos políticas óptimas, las cuales muestra la figura a continuación.

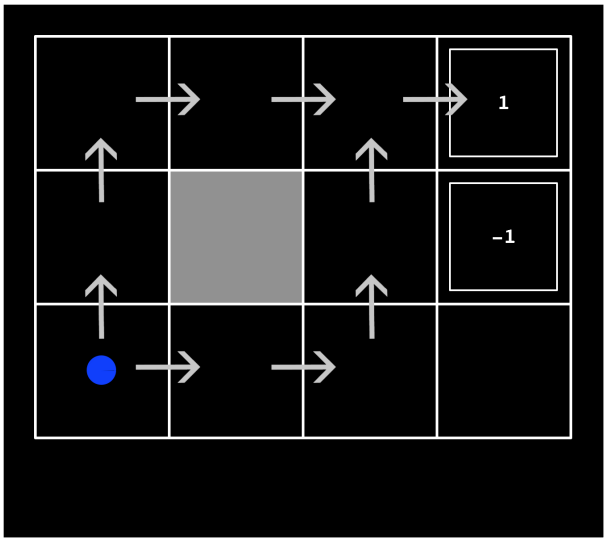


Figura 3: Políticas óptimas para el laberinto por defecto

2. Ejercicio 2

En este ejercicio, implementamos el código necesario para que un agente automático fuese capaz de basar su política de acción en una *tabla Q* dada a través de un fichero. Para ello:

- El constructor de la clase inicializa los *valores Q* a través de `readQtable`.
- `readQtable` guarda en un array `q_table` los *valores Q* de cada acción para cada uno de las acciones posibles desde el archivo donde se almacena dicha tabla.
- `writeQtable` actualiza el fichero con la actualizada *tabla Q* tras la ejecución del juego.
- `computeActionFromQValues` devuelve, dado un estado, la acción con mayor *valor Q* según la *tabla Q* actual. De esta forma, devuelve la acción elegida por la política más avara de todas.
- `computeQValueFromQValues` devuelve, dado un estado, una acción y la *tabla Q*, el *valor Q* actual para dicha acción. Se tuvo que refactorizar el código para que el nombre de esta función fuese el mismo, dado que por defecto se denominaba `getQValue`.

Inicialmente, la *tabla Q* tiene todos sus valores iguales a cero.
La política obtenida podemos observarla en la siguiente imagen:

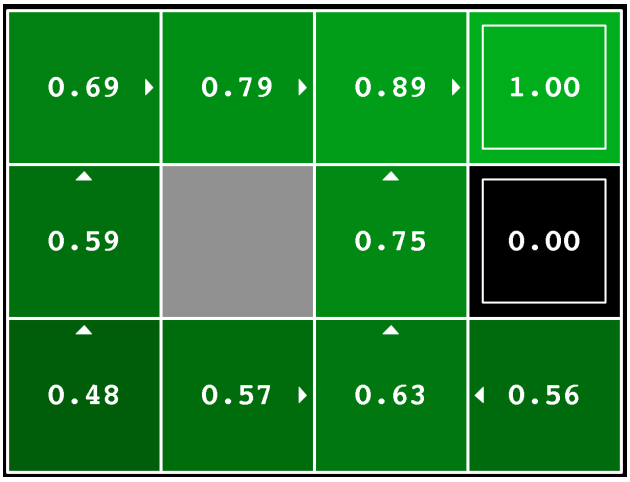


Figura 4: Política obtenida

Como podemos ver, nuestro agente ha sido capaz de obtener las dos políticas óptimas que llevan desde el estado inicial al estado final en el que la recompensa es mayor dentro del conjunto de estados finales. Ambas políticas óptimas son las que se habían predicho en el ejercicio 1 de este tutorial.

3. Ejercicio 3

En este ejercicio generamos una nueva *tabla Q*, esta vez con un MDP estocástico. Esto es debido al ruido inducido del 0.3, causando que la probabilidad de que la acción elegida sea la ejecutada se reduzca a un 70 %. La figura a continuación muestra el MDP resultante de este ruido:

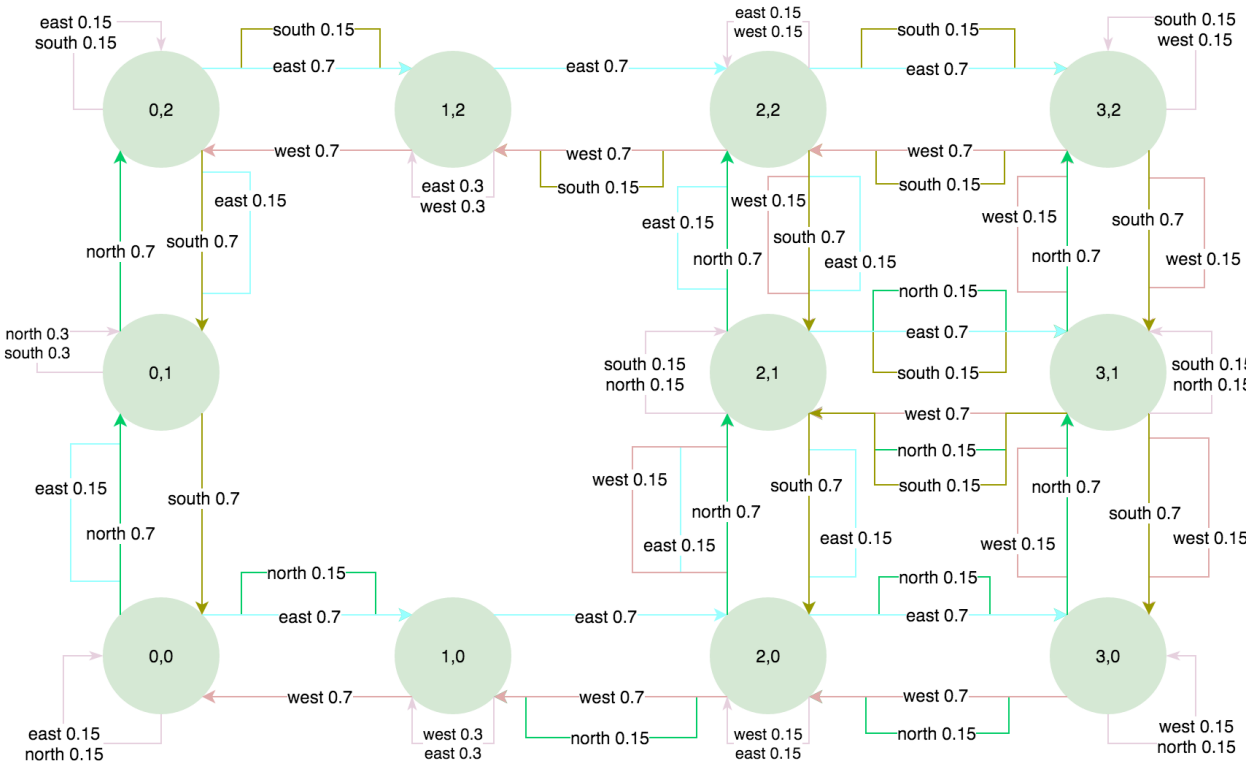
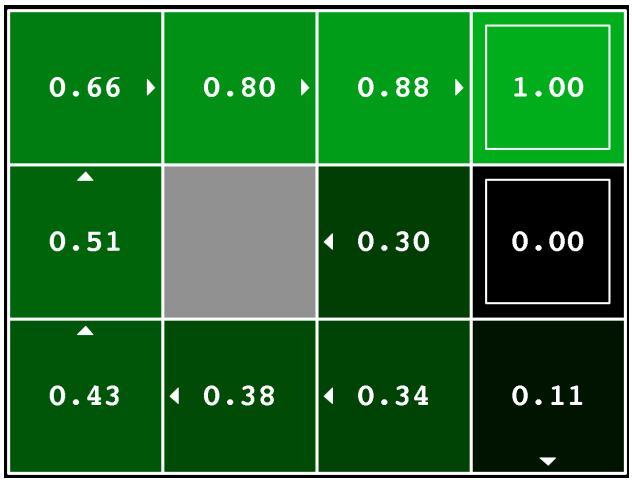


Figura 5: MDP estocástico

Para elaborar dicho MDP, se ha tenido en cuenta varias consideraciones

- Dada una acción, nunca se realizará la acción opuesta: eligiendo Norte nunca se tomará Sur, eligiendo Este nunca se tomará Oeste, y viceversa en ambos casos.
- La probabilidad de tomar cada una de las dos acciones adyacentes en caso de no tomar la acción elegida será la misma. Así, si tanto Norte como Este como u Oeste se encuentran entre las acciones legales de un estado y se elige Norte, las probabilidades de tomar cada una de las acciones serán 0.1, 0.45 y 0.45, respectivamente.
- Si una acción alternativa (adyacente) a la elegida no se encuentra entre las acciones legales para un estado determinado, el agente se queda parado.

Dado un MDP indeterminista, la política conseguida tras ejecutar el algoritmo de *QLearning* tras jugar un número bastante elevado de partidas es la que se muestra en la siguiente imagen:



Como podemos observar, nuestro agente ha conseguido descubrir una de las dos posibles políticas óptimas tras ser ejecutado más de cinco mil veces, pese al ruido que tienen los movimientos.
A diferencia del agente del ejercicio 2, al ser estocástico y no realizar siempre la acción que le indica la política, el número de partidas jugadas ha tenido que aumentar para poder obtener unos resultados similares.
Además, quisimos probar también qué ocurría en caso de que el ruido del agente fuese mucho mayor, llegando al caso extremo de probar el agente con `n = 0.9`.

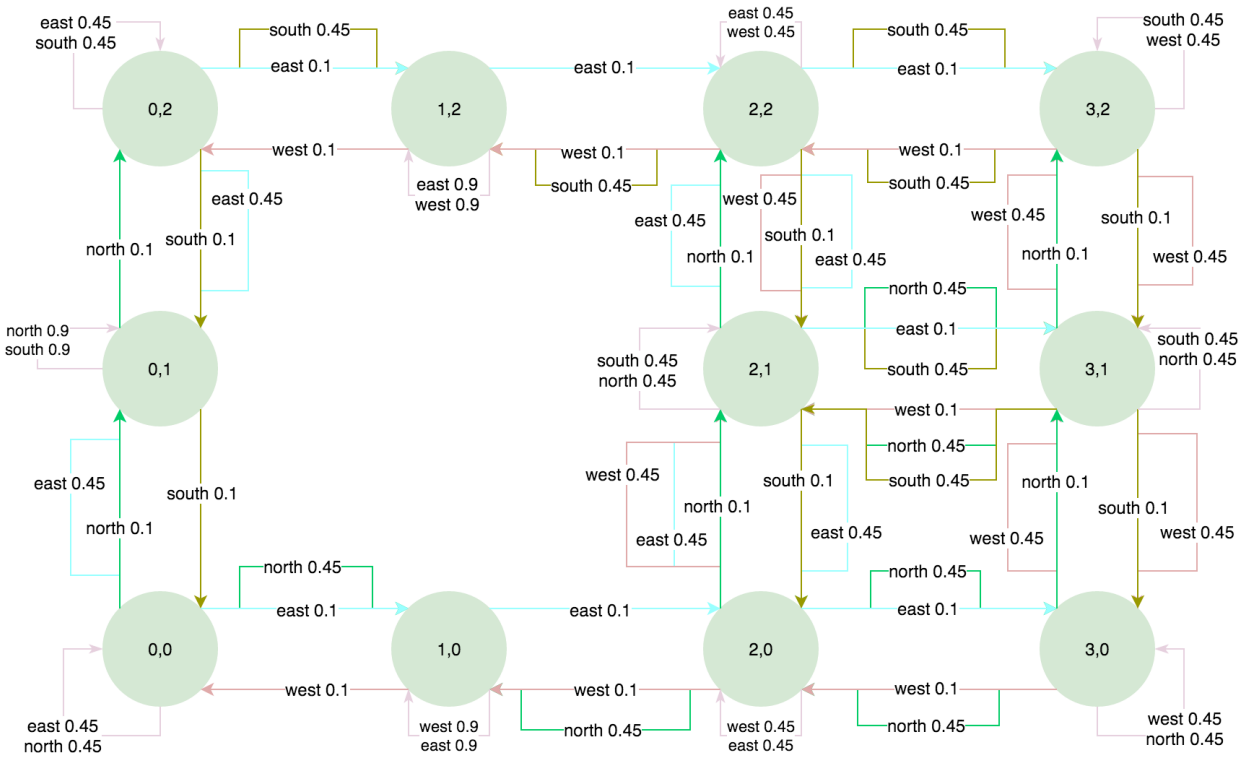


Figura 6: MDP estocástico

Y la política que obtenemos tras ejecutar cinco mil partidas con el agente.

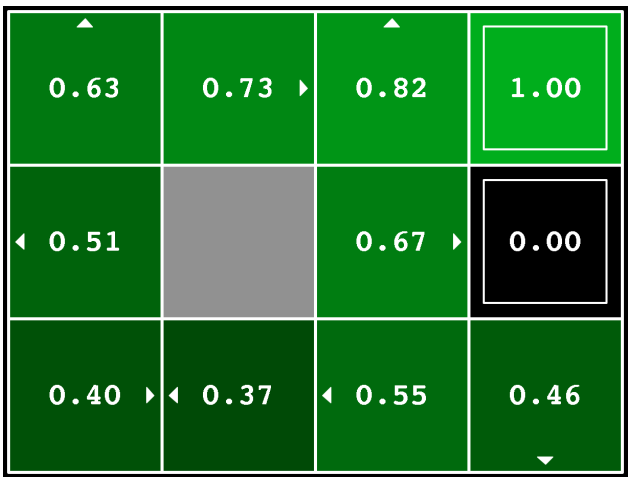


Figura 7: Política obtenida con el MDP estocástico

Para entender esta política aparentemente incorrecta, hace falta un análisis más en detalle. Como puede observarse, el agente trata de buscar la acción alternativa frente a la principal, ya que ha descubierto que es realizada con más frecuencia. Así, si quiere ir al Este elegirá la acción Norte o Sur, ya que sabe que de esta forma se moverá hacia la dirección que quiere con una probabilidad mucho mayor.
Tras explicar este enfoque, podríamos considerar la política del agente como óptima. Aún es más, si se observa la ejecución del agente se aprecia claramente cómo éste se desenvuelve con bastante soltura por el mapa y tiene unos resultados considerablemente satisfactorios, probando el razonamiento anterior.
Se puede entender que estas condiciones de aprendizaje son muy difíciles para el algoritmo, por lo que éste necesitará muchas más iteraciones para conseguir la política óptima que en los ejercicios anteriores.

4. Conclusiones

Este tutorial nos ha resultado bastante interesante para descubrir otros métodos de aprendizaje no supervisado más allá del clustering. Además, hemos podido observar cómo se comporta un agente automático y su proceso de aprendizaje.
El mayor problema que nos hemos encontrado a la hora de llevarlo a cabo ha sido un guión muy ambiguo y con errores, que nos hizo realizar trabajo que no era necesario. Además, por este motivo, en lugar de realizar el cálculo de las tablas Q a mano, como se supone que debía haberse hecho, hicimos una implementación en Python del agente, y aunque esto nos ha beneficiado más que perjudicado porque también ha servido de adelanto para el trabajo que se debe realizar en la práctica 3, sabemos que no era el objetivo de este tutorial.