Búsqueda del camino óptimo

Daniel, Carraso Merino
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
dancarmer@alum.us.es

José Antonio, Vidal Alcón
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
josvidalc@alum.us.es

***Resumen*— El objetivo principal de nuestro trabajo era implementar un algoritmo en Python, capaz de moverse por un tablero de un punto 'A', a un punto 'B', siguiendo el mejor camino posible. A este algoritmo posteriormente se le deberían realizar unas mejoras que se acomodasen al enunciado propuesto. Tales como casillas prohibidas, casillas a priorizar y una actualización de nuestro algoritmo principal.**

Tras haber realizado el proyecto, hemos llegado a las siguientes conclusiones. El algoritmo es capaz de recorrer el camino de forma correcta si ha sido entrenando lo suficiente y debería ser capaz de priorizar caminos cortos hasta el destino. Al ser en parte aleatorio, cada vez que se ejecutaba, la solución podía cambiar. En la última parte, indicando casillas mejores o peores (con valores negativos o positivos), el programa era capaz de priorizar si evitar o conducir hacía una casilla, siempre que ello no le perjudicase mucho en llegar al destino. Por último, que en matrices mayores a 3x3, el resultado depende, en parte, de los parámetros de entrada (Gamma, Alpha, Épsilon)

Palabras Clave—Inteligencia Artificial, episodio(iteración), matriz, casilla, fila, columna, adyacente.

I. INTRODUCCIÓN

El ejercicio consiste en recorrer un mapa de números no repetidos. Se comienza en un punto origen y se termina al llegar a un punto destino. Se puede avanzar por las casillas adyacentes, horizontal, vertical y diagonalmente. El objetivo es recorrer el menor número de casillas desde el origen hasta llegar al destino. Un ejemplo de mapa sería una matriz dada 3x3 con la siguiente estructura:

8	7	6
3	2	5
0	1	4

- En la variante más simple del problema, solo se decide el origen y el destino del mapa que nos piden.
- En la variante un poco más compleja, se incluye la opción de poner casillas intermedias más o menos favorables. Además, el mapa será generado aleatoriamente a partir de unas medidas.

Para la realización del programa hemos utilizado como referencia un ejercicio resuelto [1] , que nos ayudó a comprender el funcionamiento del algoritmo y nos fue muy útil para su implementación.

II. PRELIMINARES

En esta sección se hace una breve introducción de las técnicas empleadas y también trabajos relacionados, si los hay.

A. Métodos empleados

La técnica empleada para la resolución del ejercicio de búsqueda del camino en un tablero ha sido el aprendizaje por refuerzo. A diferencia del aprendizaje supervisado, el aprendizaje por refuerzo no recibe ejemplos de un supervisor externo, ni tampoco las acciones que debe tomar, sino que debe descubrir cuales son las acciones desde un estado que más recompensa le va a dar. Esto lo hará probándolas [2].

Dentro del aprendizaje por refuerzo, existen varios algoritmos. En nuestro caso, nos centramos en el algoritmo Q-Learning.

El algoritmo Q-Learning, se basa en asignar una recompensa a los pares (Estado, Acción). Para ello, crea una tabla inicial de recompensas (R), basadas en el problema inicial, que usará para ir poblando otra tabla de recompensas (Q) a partir de una fórmula repetida durante n episodios. Una vez poblada Q (entrenada), el valor más alto de un estado a sus acciones será la acción que tomará por correcta [3].

B. Trabajo Relacionado

A partir de una página que nos recomendaban en la propuesta, hemos encontrado el juego de las Torres de Hanoi, que está realizado con aprendiza por refuerzo. [4]

III. METODOLOGÍA

El trabajo se dividía en tres fases. Cada una de ellas era una implementación a lo realizado en la/s anterior/es. También se ha añadido un cuarto punto para la interfaz de usuario.

1) Fase 1

Para la implementación del problema en la fase 1, hemos decidido crear una serie de funciones. No ha sido necesario crear ninguna clase ya que hemos trabajado directamente con los tipos que ofrece Python. La secuencia completa de la fase 1 sería

a) Generar Matriz R (generar_r)

Lo primero que hace el programa es generar la matriz R de recompensas a partir de la matriz inicial (mapa). La matriz R sirve para saber las recompensas por coger una determinada acción en un estado. En este caso:

- -1 si no son adyacentes
- 0 si son adyacentes, pero no son el destino
- 100 si son adyacentes y además son el destino

Para generarla, lo primero que necesitamos es hacer una matriz de n filas x n columnas, siendo n el número máximo del mapa. Sabiendo que no se repiten números y que van en orden sin saltarse ninguno, n se calcula como:

n = (filas del mapa x columnas del mapa)

Ahora se rellena esta matriz con -1 en todas las posiciones, porque primero tomaremos como que ninguna acción es adyacente y después cambiaremos los valores que sí los son. Para ello, iremos recorriendo nuestra matriz original casilla a casilla y las posiciones que rodean a cada casilla (adyacentes). El número de la casilla central será el estado en R (filas), y las que la rodean serán las acciones (columnas). El valor de R(estado,acción) será 0 o 100 (si además de adyacente, la acción es el destino)

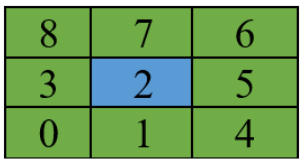


Ilustración 1 Casillas adyacentes a casillas céntricas.

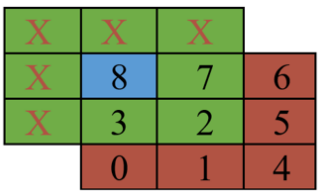


Ilustración 2 Casillas adyacentes a casillas de esquinas

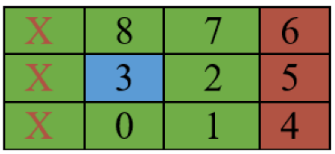


Ilustración 3 Casillas adyacentes a casillas laterales.

Para evitar errores de índex en las casillas que no existen (las verdes marcadas con una cruz) se ponen en una estructura de control de excepciones.

b) Generar Matriz Q inicial (generar_q)

Esta función recibe la matriz inicial (mapa), y con el número de filas y columnas de ésta, genera una matriz cuadrada de tamaño n*m repleta de ceros, ya que la inicial está vacía.

Esta matriz se usará para ir poblándola de números y será la matriz que se recorrerá para obtener nuestro resultado final.

c) Algoritmo Q-Learning (q_learning_fase1)

Una vez llega a este punto, el algoritmo ya tiene el mapa en una matriz y sabe crear la matriz R y la matriz Q vacía (a cero). Esta función se encarga de llamar a las funciones anteriores (generar_r y generar_q) y ejecutar el algoritmo de Q-learning. Este va a realizar n iteraciones poblando la matriz Q antes generada.

Algoritmo Q-learning [4 Propuesta de trabajo]
Entrada:
- Una matriz M
- Una matriz R
- Un entero casilla_final
- Un valor[0,1) gamma
- Un entero iteraciones

Salidas:
- Una matriz Q poblada

Algoritmo:
1. Generar Q vacía
2. Para cada iteración
   a. Mientras no haya llegado al destino
      i. Seleccionar estado aleatorio
      ii. Escoger acción posible aleatoria de ese estado
      iii. Obtener mayor valor de Q de la acción escogida y sus posibles nuevas acciones
      iv. Hacer Q(estado, acción) = R(estado, acción) + Gamma * Max[Q(acción, acciones posibles de acción]
      v. La acción pasa a ser el nuevo estado
3. Devolver Q

Pseudocódigo 1 Algoritmo q-learning fase 1

	0	1	2	3	4	5	6	7	8
0	0	100	60	60	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	60	100	0	60	60	60	36	36	36
3	60	100	60	0	0	0	0	36	36
4	0	100	60	0	0	60	0	0	0
5	0	100	60	0	60	0	36	36	0
6	0	0	60	0	0	60	0	36	0
7	0	0	60	60	0	60	36	0	36
8	0	0	60	60	0	0	0	36	0

Ilustración 4 Matriz Q después de 300 iteraciones (Camino de la posición 7 a la 1)

d) Función solución (generar_solucion)

Teniendo ya calculada nuestra matriz Q, tras haber realizado las iteraciones indicadas, obtendremos una matriz final poblada de números que usaremos para sacar una solución de los números por los que debe ir pasando para llegar del origen al destino. Para ello, recibiremos por parámetros la posición inicial y la posición final, ambas indicadas por el propio usuario. Comenzaremos recorriendo las filas de mi matriz Q, empezando por la fila del número origen, de esta, sacamos el valor máximo, lo guardaremos en una lista que acumula los pasos y la nueva fila será la fila del valor máximo antes obtenido. Se repetirá hasta llegar al destino.

	0	1	2	3	4	5	6	7	8
0	0	100	60	60	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	60	100	0	60	60	60	36	36	36
3	60	100	60	0	0	0	0	36	36
4	0	100	60	0	0	60	0	0	0
5	0	100	60	0	60	0	36	36	0
6	0	0	60	0	0	60	0	36	0
7	0	0	60	60	0	60	36	0	36
8	0	0	60	60	0	0	0	36	0

5 -> Destino
3
4
1
2

Ilustración 5 Pasos en rojo de generar solución con Q apartado anterior

2) Fase 2

Para la fase 2 no era necesario cambiar todas las funciones, tan solo la función del algoritmo Q-learning, las demás se reutilizan de la fase 1.

a) Algoritmo Q-Learning (q_learning_fase2)

Para el algoritmo Q-Learning de la fase 2, se añaden 2 nuevos parámetros, Epsilon y Alpha. Estos se usarán para la elección de las acciones posibles de un estado.

Tal y como en la fase 1, llamará primero a las funciones generar_q y generar_r, para pasar a ejecutar el algoritmo de Q-learning con los nuevos cambios.

Algoritmo Q-learning 2
Entrada:
- Una matriz M
- Una matriz R
- Un entero casilla_final
- Un valor[0,1) gamma
- Un valor[0,1) epsilon
- Un valor[0,1) alpha
- Un entero iteraciones

Salidas:
- Una matriz Q poblada

Algoritmo:
4. Generar Q vacía
5. Para cada iteración
   a. Mientras no haya llegado al destino
      i. Seleccionar estado aleatorio
      ii. Se genera un número aleatorio de 0 a 1
      iii. Si este es mayor que épsilon
         1. Escoger acción posible aleatoria de ese estado
      iv. De lo contrario
         1. Escoger acción posible que más valor en la matriz Q tenga
      v. Obtener mayor valor de Q de la acción escogida y sus posibles nuevas acciones
      vi. Hacer Q(estado, acción) = R(estado, acción) + Gamma * Max[Q(acción, acciones posibles de acción]
      vii. La acción pasa a ser el nuevo estado
   b. Épsilon se multiplica por alpha
6. Devolver Q

Pseudocódigo 2 Algoritmo q-learning fase 2

	0	1	2	3	4	5	6	7	8
0	0	100	16	40	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0
2	40	0	0	40	40	0	0	0	0
3	40	100	0	0	0	0	0	0	0
4	0	100	0	0	0	0	0	0	0
5	0	100	0	0	40	0	0	0	0
6	0	0	16	0	0	40	0	0	0
7	0	0	16	0	0	0	0	0	0
8	0	0	16	0	0	0	0	0	0

Ilustración 6 Matriz Q despues de 300 iteraciones (Camino de la posición 8 a la 4)

3) Fase 3

En la fase 3 se añaden 2 nuevos grandes cambios.

- La matriz ya no será una establecida, sino que se generará aleatoriamente a partir de las filas y columnas que se le pasen como parámetros
- Se pueden elegir que casillas intermedias sean más o menos propensas a pasar por ellas o evitarlas.

Para esta fase usaremos algunas funciones usadas en las anteriores fases. Las funciones q_learning se mantendrán iguales. En este caso lo que habrá que modificar es la generación de la matriz R y crear una nueva que cree la matriz aleatoria a partir de n filas y m columnas.

a) Generar matriz aleatoria (generar_matriz_aleatoria)

Esta función recibe como parámetros n filas y m columnas para crear una matriz. Dentro de ésta, generaremos números únicos (sin repeticiones) entre un rango de 0 a (n*m)-1. (Como empieza en 0 al último valor hay que restarle 1).

Para hacerlo, usamos la librería Random [5] y Numpy [6] de Python a partir de una expresión lambda [7]

b) Generar matriz R (generar_r_fase3)

Empezaremos generando una matriz de n filas x n columnas (siendo n el número máximo del mapa) con valores a -100 en todas las posiciones. Hemos usado -100 para que las casillas que se quieran evitar usan un valor negativo, pero no tan grande como ese. Cómo antes, se ha hecho esto con la intención de ahora cambiar los valores que sí son adyacentes por 0, o por 100 en caso de ser una casilla de destino.

Para el cambio por 0 o 100, se realiza el mismo proceso que en la función generación_r.

El cambio más significativo viene cuando una vez generada esta matriz se le indiquen las posiciones a evitar o aquellas que queramos que intente pasar. Este proceso se hará sustituyen los valores de toda una columna (excepto si hay un -100) de R, correspondiente a una casilla, por el valor que se le haya dado. Si el valor es negativo [-25,0), estará indicando que no debe pasar por esa casilla, cuanto más cerca a -25, más intentará evitarla. Esto mismo se aplica a positivo (0, 25], cuanto más cerca de 25, más intentará coger por esa casilla.

	0	1	2	3	4	5	6	7	8
0	-100	-100	0	-100	0	-100	-100	0	-100
1	-100	-100	-100	-100	0	0	0	0	100
2	0	-100	-100	0	0	-100	0	0	-100
3	-100	-100	0	-100	0	-100	0	-100	-100
4	0	0	0	0	-100	0	0	0	100
5	-100	0	-100	-100	0	-100	0	-100	-100
6	-100	0	0	0	0	0	-100	-100	-100
7	0	0	0	-100	0	-100	-100	-100	100
8	-100	0	-100	-100	0	-100	-100	0	-100

Ilustración 7 Matriz R SIN RESTRICCIONES

	0	1	2	3	4	5	6	7	8
0	-100	-100	0	-100	15	-100	-100	0	-100
1	-100	-100	-100	-100	15	0	-15	0	100
2	0	-100	-100	0	15	-100	-15	0	-100
3	-100	-100	0	-100	15	-100	-15	-100	-100
4	0	0	0	0	-100	0	-15	0	100
5	-100	0	-100	-100	15	-100	-15	-100	-100
6	-100	0	0	0	15	0	-100	-100	-100
7	0	0	0	-100	15	-100	-100	-100	100
8	-100	0	-100	-100	15	-100	-100	0	-100

Ilustración 8 Matriz R CON RESTRICCIONES

Como se ve en la Ilustración 8, se han creado 2 restricciones: Recompensa de 15 si pasan por la casilla 4, recompensa de -15 si pasan por la casilla 6.

4) Interfaces de usuario

Para las interfaces de usuario hemos usado la librería de Python Tkinter [8].

La interfaz servirá para cambiar los parámetros que sean editables.

- En la fase 1: Iteraciones, origen, destino, gamma
- En la fase 2: Iteraciones, origen, destino, gamma, epsilon, alpha

- En la fase 3: A cada fase se la añaden n y m para generar la matriz aleatoria y un cuadro de texto para poner las restricciones.

Esos valores que se piden serán guardados en una variable para luego ser usados al llamar a las respectivas funciones.

En la fase 3, la matriz será guardada en una variable global debido a que primero hay que generarla y después calcular sobre ella. Para ello luego de esto pediremos el resto de los datos, que serán introducidos.

Serán las llamadas a la función calcula la que hará las llamadas a las funciones de la fase en la que nos encontremos.

IV. Resultados

En esta sección se detallará tanto los experimentos realizados como los resultados conseguidos:

Para mostrar e interpretar los resultados hemos experimentado con varios valores en todas las fases y de cada una hemos ido recopilando tanto el tiempo de ejecución del algoritmo Q-learning en esa fase, como la progresión del rendimiento en un número determinado de iteraciones. El rendimiento se calcula como:

Rendimiento = (SUM(Q) / MAX(Q)) x 100

1) Tiempo

Para la medida del tiempo ejecutamos 5 veces cada algoritmo y después sacamos una media.

FASE 1				
Origen	Destino	Gamma	Instrucciones	Tiempo(s)
0	6	0.5	100	0.116 s
0	6	0.5	100	0.118 s
0	6	0.5	100	0.128 s
0	6	0.5	100	0.129 s
0	6	0.5	100	0.116 s
Media				0.121 s

FASE 2						
Origen	Destino	Gamma	Insts	Eps	Alpha	Tiempo(s)
0	6	0.5	100	0.8	0.5	0.149 s
0	6	0.5	100	0.8	0.5	0.119 s
0	6	0.5	100	0.8	0.5	0.106 s
0	6	0.5	100	0.8	0.5	0.108 s
0	6	0.5	100	0.8	0.5	0.120 s
Media						0.120 s

Para la fase 3 omitimos el origen y destino ya que los mapas son aleatorios. Como referencia, hemos cogido de origen y destino esquinas opuestas. No hemos puesto tampoco ninguna restricción, porque solo se está midiendo el tiempo del Q-learning y no de generar_r

Fase 3 - Q learning 1			
Tamaño nxm	Gamma	Insts	Tiempo(s)
4x4	0.9	500	1.15 s
5x5	0.9	500	1.56 s
6x6	0.9	500	2.93 s
7x7	0.9	500	2.98 s
8x8	0.9	500	9.8 s

Fase 3 - Q learning 2					
Tamaño nxm	Gamma	Eps	Alph	Insts	Tiempo(s)
4x4	0.9	0.1	0.1	500	0.35 s
5x5	0.9	0.1	0.1	500	0.51 s
6x6	0.9	0.1	0.1	500	1.3 s
7x7	0.9	0.1	0.1	500	2.93 s
8x8	0.9	0.1	0.1	500	4.15 s

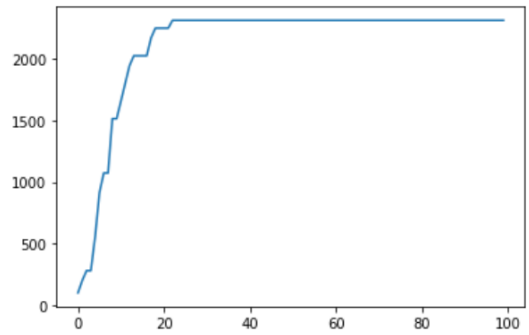
Observando los resultados se pueden observar varias cosas. Cuando la matriz es pequeña, los resultados entre el algoritmo q learning y el algoritmo q learning modificado con alpha y epsilon son insignificantes. A medida que se va aumentando el tamaño de la matriz, se va notando cada vez más que el segundo es mucho más rápido, llegando a ser hasta 3 veces aproximadamente más rápido en una matriz un poco más grande y 2 veces más rápido en una matriz mucho más grande (8x8).

Para llegar a una conclusión, tenemos que sacar los rendimientos.

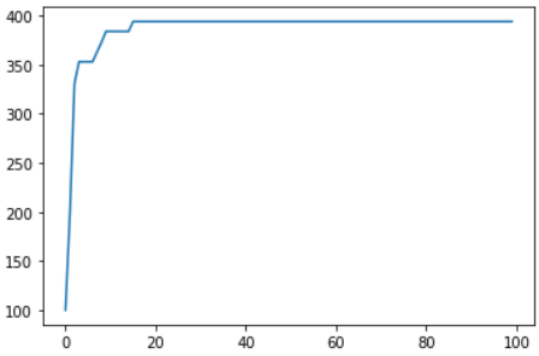
2) Rendimientos

Para los rendimientos, en vez de usar tablas con los valores, vamos a representarlos mediante gráficas para que sea más fácil observar los datos. Los valores de la tabla en 'x' son el número de instrucciones y en 'y' el rendimiento.

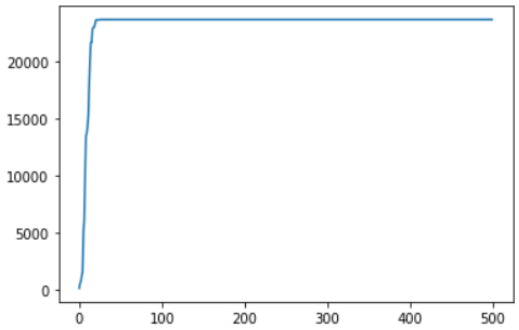
a) En la fase 1, con 100 instrucciones y gamma 0.8



b) En la fase 2, con 100 instrucciones y gamma 0.8

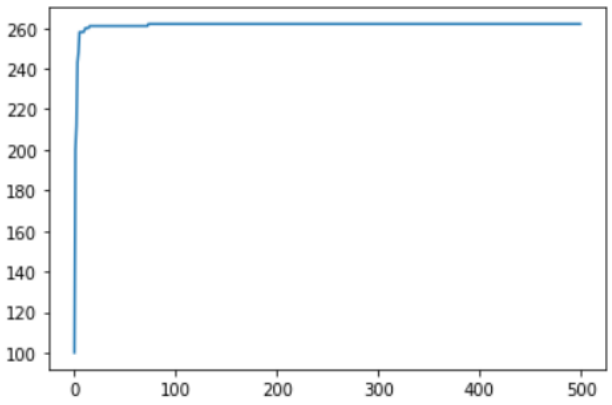


c) En la fase 3, Q-learning 1, con 500 instrucciones, gamma 0.8 y matriz de 8x8



d) En la fase 3, Q-learning 2, con 500 instrucciones, gamma 0.8, epsilon 0.1, alpha 0,1 y matriz de 8x8

,



Por los resultados de los rendimientos, se puede observar que q-learning de la fase 1 pobla mucho más la matriz Q a igual número de instrucciones.

V. Conclusiones

Después de haber realizado el proyecto hemos aprendido que para resolver el problema dado podíamos usar dos algoritmos basados en el aprendizaje por refuerzo. El q learning y una modificación de este. El primero se trata de recorrer caminos de forma aleatoria siempre, por lo que es más probable que cree una matriz Q más completa, logrando dar siempre, si gamma es adecuado, el camino óptimo. La parte negativa es que como hemos podido ver en los resultados, los estudios de rendimiento y los de tiempo, esto le lleva un mayor tiempo que a su alternativa.

La alternativa, sin embargo, va intentando alejarse de la aleatoriedad a medida que más veces se va ejecutando, e intenta ir guiando siempre por los caminos que ya sabe que es más probable que sean óptimos. Esto le lleva a perderse a veces otra alternativa que en realidad serían más correctas, y no siempre va a dar el mejor camino. Y, al contrario que el otro, gracias a llegar antes al destino al elegir las posibles acciones, se ahorra operaciones, lo que se traduce en tiempo de ejecución menor.

Para acabar, explicar la importancia de gamma. Cuanto mayor es gamma, mayor es la recompensa de futuros caminos, mientras que cuanto más se acerque a 0, mayor será la recompensa de las acciones inmediatas.

VI. Anexo. Marco Teórico

1) Aprendizaje por refuerzo. Q learning [9]

Nuestro proyecto se basa en la idea del q-learning, que se trata de un modelo de aprendizaje por refuerzo que consiste en extraer acciones de los estados disponibles, maximizando la recompensa. Estos modelos trabajan mediante lo que llamaremos políticas, que nos indica dependiendo del estado en el que nos encontremos a que acciones podremos acceder.

Los algoritmos irán experimentando los distintos caminos que puedan, es decir, no le diremos qué camino tomar, sino que con la práctica, el algoritmo debería hacerlo por si solo siguiendo las acciones que lo lleven a tomar una mayor recompensa. Cuanto más practica o entrenamiento tengan estos algoritmos más rápidos y eficaces se volverán. Podría decirse que se hacen "más inteligentes". Este tipo de modelo es muy utilizado en el mundo de los video juegos y se pueden llegar a ver inteligencias artificiales capaces de vencer a personas, cuando se tratan de inteligencias muy adiestradas.

Una inteligencia artificial que aprenda mediante un aprendizaje por refuerzo podría aprender a jugar al ajedrez en tan solo 13 horas jugando contra sí misma. Un ejemplo es la IA AlphaZero de DeepMind, que en lugar de analizar los movimientos posibles se centra en aquellos que les ha dado mejores resultados. Esta ya ha conseguido vencer a varios de los mejores jugadores de ajedrez del mundo.

En este caso nos centraremos en uno de los modelos de aprendizaje por refuerzo, el q learning. El q learning, contiene una lista de todas las posibles recompensas. Se le indica un estado final, el cual es el objetivo, y se irán introduciendo diferentes recompensas en otra lista, lo que hará a nuestro algoritmo decidirse por una acción antes que por otra.

Tendremos también que tener en cuenta el factor $\gamma \in [0,1]$ que nos indica la importancia de nuestras recompensas tanto positivas en el caso de priorizar acciones, como negativas, en el caso de que queramos prohibir a nuestro algoritmo tomar distintas acciones. El algoritmo no tendrá en cuenta la política(que acción tomar en cada estado) sino que con la práctica y la experimentación esta se irá generando, tomando como recompensa futura la óptima, por lo que los caminos más lejanos a nuestro objetivo irán tomando valores más atenuados y se invitará seguir estos caminos.

La fórmula de Q-learning es:

Q(estado, acción) = R(estado, acción) + Gamma * Max[Q(siguiente estado, todas las acciones)]

a) Aplicaciones reales [10]

Algunas de las aplicaciones que se usan hoy en día, o se han hecho pruebas con aprendizaje por refuerzo son:

En semáforos [11]: Para intentar crear un diseño de controlador de semáforos que redujera la congestión del tráfico. El sistema, que se recreó en un sistema simulado, consistía en un estado que correspondía a la disposición de las luces, y las acciones son la siguiente disposición de las luces, como recompensa, la reducción del retardo con respecto al paso anterior.

En sistemas de publicidad: En los sistemas de recomendación de publicidad, son frecuente el uso del aprendizaje por refuerzo, que almacenan y monitorizan aquellas cosas que más sueles visitar y qué más podrían gustarte. Por ende, te recomienda productos que se asimilan a tus gustos. Este proceso podría realizarse asignando una mayor recompensa a las cosas que más se asimilen a tus gustos.

En robótica: En este campo, el aprendizaje por refuerzo tiene muchas aplicaciones. Es un método muy común para enseñar a un robot a realizar acciones. Este, va aprendiendo y mediante la repetición acaba cumpliendo un objetivo designado. Como, por ejemplo, coger algo de un lugar. [12]

Por lo que hemos podido observar este tipo de ejercicio es muy similar a los ejercicios de planificación bajo incertidumbre, así como con los procedimientos de decisiones de Markov. Como podemos ver se cumple que siguen las siguientes características.

Un conjunto finito de acciones y estados, el sistema solo cambia por medio de las acciones, el objetivo pertenece al conjunto de estados, ejecución instantánea de acciones, el problema mientras se planifica y efectos deterministas de las acciones. Como se vio en dicho tema tenemos acciones, estados y podemos tanto penalizar como recompensar acciones.

2) SARSA [13]

Otro modelo de aprendizaje por refuerzo es el denominado SARSA, que como q-learning intenta aprender de los valores óptimos de los pares generados (estado-acción). SARSA, a diferencia de q-learning, es un algoritmo 'on policy'. Esto quiere decir que aprende el valor basado en la diferencia de su valor actual y el del siguiente estado.

El algoritmo es que al contrario del q learning que solo aprendía los caminos óptimos este también aprende los distintos caminos que no deberá seguir. SARSA necesita más episodios que q learning porque recorre más caminos, ya que recorre también caminos negativos. SARSA aprende a evitar los caminos negativos, a diferencia de q learning que a veces, puede llevar a caminos equivocados. Esto lo hace más fiable, a cambio de hacer más iteraciones.

El algoritmo es el siguiente:

1- Inicializar cada entrada $Q(s, a)$ arbitrariamente
2- DO (por cada episodio)
3- Determinar estado inicial s
4- WHILE (no finalice el episodio)
5- Seleccionar una acción a desde el estado s usando una política derivada de Q (por ejemplo $\epsilon-$greedy)
6- WHILE (no se llegue a un estado terminal)
7- Ejecutar la acción a. Observar recompensa r y nuevo estado s'
8- Seleccionar acción a' desde s' usando una política derivada de Q (por ejemplo $\epsilon-$greedy)
9- $Q(s, a) = Q(s, a) + \alpha.[r + \gamma.Q(s', a') - Q(s, a)]$
10- $s = s'$
11- $a = a'$

3) *DYNA*

El algoritmo de Dyna sigue la siguiente estructura:

Tabular Dyna-Q

Initialize $Q(s, a)$ and $Model(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop forever:
(a) $S \leftarrow$ current (nonterminal) state
(b) $A \leftarrow \varepsilon$-greedy(S, Q)
(c) Take action A; observe resultant reward, R, and state, S'
(d) $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
(e) $Model(S, A) \leftarrow R, S'$ (assuming deterministic environment)
(f) Loop repeat n times:
$S \leftarrow$ random previously observed state
$A \leftarrow$ random action previously taken in S
$R, S' \leftarrow Model(S, A)$
$Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$

REFERENCIAS

[1] http://mnemstudio.org/path-finding-q-learning-tutorial.htm
[2] https://web.archive.org/web/20090806064734/http://www.cs.ualberta.ca/~sutton/book/ebook/node7.html
[3] http://www.cs.us.es/~fsancho/?e=109
[4] http://mnemstudio.org/path-finding-q-learning-tutorial.htm
[5] https://docs.python.org/3/library/random.html
[6] https://numpy.org/doc/stable/
[7] https://www.quora.com/How-can-I-generate-a-grid-or-matrix-of-unique-random-numbers-in-Python
[8] https://wiki.python.org/moin/TkInter
[9] https://www.xataka.com/robotica-e-ia/alphazero-ia-capaz-aprender-ella-a-jugar-al-ajedrez-ganar-a-todas-a-ias-adiestradas-humanos
[10] https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12
[11] http://web.eecs.utk.edu/~ielhanan/Papers/IET_ITS_2010.pdf
[12] Tema Planificación bajo incertidumbre (Asignatura IA - ETSII – US)
[13] https://www.quora.com/What-is-the-difference-between-Q-learning-and-SARSA-learning
http://sedici.unlp.edu.ar/handle/10915/23038