APRENDIZAJE AUTOMÁTICO AVANZADO

INFORME TÉCNICO UNIDAD IV – APRENDIZAJE POR REFORZAMIENTO

PRESENTADO POR:

Edgar Leandro Jiménez Jaimes

Santiago Echeverri Calderón

DOCENTE:

José Lisandro Aguilar Castro

UNIVERSIDAD EAFIT

MEDELLÍN

MAESTRÍA EN CIENCIAS DE LOS DATOS Y ANALÍTICA

MARZO DE 2020

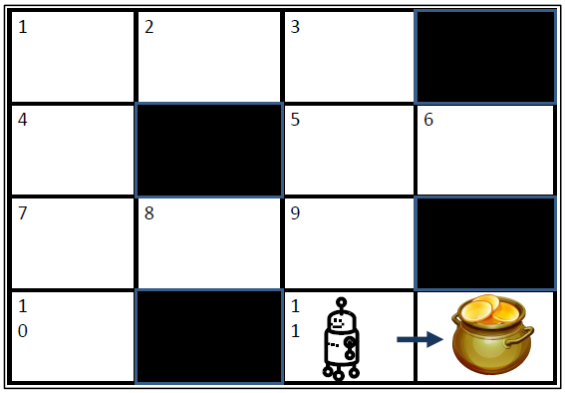
1. Objetivo de la iteración

Comprender e implementar un modelo de Aprendizaje por Reforzamiento y aplicarlo en la solución de un problema.

2. Contextualización del problema

Para la presente iteración se seleccionó el algoritmo *Q-Learning*, un algoritmo que proporciona a los agentes la capacidad de aprender a actuar de manera óptima en dominios de Markov al experimentar las consecuencias de las acciones, sin necesidad de que construyan mapas de los dominios[1].

Como era la primera vez que usábamos un algoritmo de este tipo, con el fin de comprender mejor el mismo, decidimos hacer la implementación sin el uso de librerías sobre un caso sencillo de un Proceso de Decisiones de Markov (MDP). Para esto usamos el ejemplo de las diapositivas de clase, el cual consiste en un entorno MDP con 16 posibles estados, un agente, una recompensa y diferentes obstáculos[2]:



Si bien usamos este entorno, diseñamos el algoritmo de manera que pudieran configurarse diferentes escenarios iniciales en los que el agente, la recompensa y los obstáculos puedan posicionarse en el cualquiera de los estados.

[1] Watkins, C. J., & Dayan, P. (1992). Q-learning. Machine learning, 8(3-4), 279-292.
[2] Aguilar, J 2020, *CM0891: Aprendizaje Automático Avanzado: Aprendizaje Reforzado*, Notas de Clase, Universidad Eafit, 29 Feb 2020.

3. Diseño del modelo

3.1. Creación del entorno

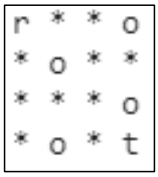
El código se basó en la implementación de Venelin Valkov (https://github.com/curiousily), la cual estaba desarrollada para un entorno de sólo 4 estados y no calculaba la solución de un entorno completo, sino únicamente la acción a que maximiza Q para un estado dado. En nuestra adaptación se creció el entorno a 16 estados y se creó un bucle para calcular todas las acciones que selecciona el agente hasta llegar a la recompensa.

Se comenzó representando el entorno en una lista, en la cual se usaron 4 variables: el robot (agente), los obstáculos, el tesoro (recompensa) y los estados o casillas vacías.

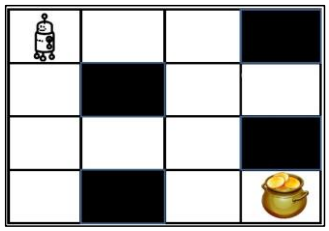
```
# Defición de las variables

OBSTACULO = "o"
ROBOT = "r"
TESORO = "t"
VACIO = "*"

# Definición del entorno o cuadricula
grid = [
    [ROBOT, VACIO, VACIO, OBSTACULO],
    [VACIO, OBSTACULO,VACIO, VACIO],
    [VACIO,VACIO,VACIO,OBSTACULO],
    [VACIO,OBSTACULO,VACIO,TESORO]
]
```

Ingreso de variables para crear entorno de exploración

Para el estado inicial se posicionó al robot en la casilla opuesta al tesoro:



Representación del entorno en el código



Entorno inicial planteado

El entorno se asumió estacionario y para identificar los estados o casillas en el código se usó un sistema de coordenadas de fila y columna con índice basado en cero. Es decir, que en el entorno planteado el robot comienza en la casilla (0,0) y el tesoro en la casilla (3,3).

También se definieron 4 posibles acciones del agente en el entorno, estas son: desplazarse una casilla arriba (UP), desplazarse una casilla abajo (DOWN), desplazarse una casilla a la izquierda (LEFT) o desplazarse una casilla a la derecha (RIGHT), asumimos en este enfoque que solo puede dar un movimiento a la vez.

```
# Definición del las acciones y asignaciones variables

UP = 0
DOWN = 1
LEFT = 2
RIGHT = 3

ACTIONS = [UP, DOWN, LEFT, RIGHT]
```

Posibles movimientos del robot

Después se creó una función para ejecutar una acción en un estado dado y devuelva el nuevo estado, la recompensa y si con la acción ejecutado el episodio concluye o no. Para esto se usaron los siguientes valores para cada variable que puede tomar una casilla, en caso de que el robot pare en ella:

Estado	Recompensa	Concluye el episodio
Vacía	-1	No
Obstáculo	-100	**Sí**
Tesoro	1000	**Sí**
Robot	-1	No

Como se puede observar si el agente llega a una casilla vacía o regresa a la casilla del robot, es penalizado con una recompensa negativa y el episodio no finaliza. En cambio, si llega a una casilla de obstáculo, recibe una penalización mayor y finaliza el episodio. Finalmente, si llega a la casilla del tesoro, recibe una recompensa positiva de 1000 y finaliza el episodio de manera satisfactoria.

3.2. Entrenamiento del modelo

En el algoritmo Q-Learing, el entrenamiento consiste en la exploración del entorno. A continuación, se crearon los parámetros para la exploración y una función para almacenar las políticas obtenidas de la exploración en la tabla Q:

```
# Semilla para aleatorios
random.seed(42)

# Definición de pasos a seguir
N_STATES = 16           # Posibles estados del entorno
N_EPISODES = 100        # Ajustamos los episodios de exploracion

MAX_EPISODE_STEPS = 100  # Número máximo de pasos por episodio

MIN_ALPHA = 0.02

alphas = np.linspace(1.0, MIN_ALPHA, N_EPISODES)
gamma = 1.0
eps = 0.2

q_table = dict()
```

Como la exploración es aleatoria se usó una semilla para poder reproducir los resultados. Y se definieron unas variables, algunos pueden considerarse hiperparámetros iniciales que se evaluarán posteriormente en la fase de análisis de resultados.

- **N_STATES**: corresponde a la cantidad de estados o cuadriculas del MDP.
- **N_EPISODES**: establece el número de episodios de exploración que hará el algoritmo.
- **MIN_ALPHA**: valor mínimo al que llegará la tasa de aprendizaje alfa. Esta tasa se disminuye en cada episodio cada episodio, de esta forma a medida que el agente explora más, "creerá" que no queda mucho por aprender.
- **alphas**: es un array que contiene el valor alfa a usar en cada episodio, el cual va disminuyendo hasta llegar al valor MIN_ALPHA.
- **eps**: épsilon, establece el balance entre exploración y explotación del entorno. La exploración es el acto de explorar el entorno para encontrar información al respecto y la explotación es el acto de explotar la información que ya se conoce para maximizar el rendimiento.

Después de realizar la exploración se obtiene el valor de recompensa de cada uno de los episodios de la exploración:

```
Episode 1: total reward -> -113
Episode 2: total reward -> -104
Episode 3: total reward -> -102
Episode 4: total reward -> -108
Episode 5: total reward -> -101
Episode 6: total reward -> -107
Episode 7: total reward -> -107
Episode 8: total reward -> -101
Episode 9: total reward -> -111
Episode 10: total reward -> -104
Episode 11: total reward -> -106
Episode 12: total reward -> -105
```

3.3. Prueba básica del modelo

Para probar el modelo se parte del estado inicial y mediante la función '*q'* se consultan los valores '*q'* para cada acción en ese estado:

```
r = q(start_state)
print(f"up={r[UP]}, down={r[DOWN]}, left={r[LEFT]}, right={r[RIGHT]}")
```

```
up=792.9452455350621, down=211.5794085743094, left=968.1105568278055, right=994.9999999820002
```

Se observa que la acción de mayor valor *'q'* es 'RIGHT', así esta debería ser la acción a ejecutar pues es la que maximiza la recompensa.

Se ejecuta entonces acción 'RIGHT' con la función *'act'*, que además proporciona la posición en donde quedó el robot, cuál fue la recompensa de la acción y si ya finalizó la tarea:

```
new_state, reward, done = act(start_state, RIGHT)

print("Nueva cuadricula del entorno: ")

for row in new_state.grid:
    print(' '.join(row))

print('Posición del robot: ', new_state.robot_pos)
print('Recompensa de la acción: ', reward)
print('¿Tarea terminada?: ', done)
```

```
Nueva cuadricula del entorno:
* r * o
* o * *
* * * o
* o * t
Posición del robot:  [0, 1]
Recompensa de la acción:  -1
¿Tarea terminada?:  False
```

En la cuadricula se puede observar que el robot se desplazó a la derecha, quedando en la posición (0,1). Como llegó a una cuadricula vacía su recompensa fue -1. Y como no ha llegado al tesoro, ni se chocó con un obstáculo la tarea aún no termina.

Se comprobó entonces que el modelo tiene información del entorno y sus estados, y es capaz de realizar las acciones definidas sobre este. El siguiente paso consiste en crear un bucle que automáticamente ejecute las mejores acciones para cada estado según el valor *'q'*.

3.4. Automatización del modelo

Se creó entonces un bucle que, con un estado inicial de entrada, busque maximizar la recompensa, es decir, llegar lo más cerca posible al tesoro. Al bucle se le incluyeron 2 reglas de parada, la primera fue un límite de iteraciones y la segunda una interrupción si el estado de la última acción ejecutada indica que ya se había finalizado la tarea, lo cual ocurriría si se llegaba al tesoro o se chocaba con un obstáculo.

Finalmente, el bucle daría como salida la posición en la que había parado el robot, el número de iteraciones o pasos para llegar a ese estado y la cuadricula del entorno final.

```
while  (i != max_iter or finished == True):
  # Escoger la accion segun el estado
  action = choose_action(actual_state)
  # Ejecutar la accion definda y calcular la recompensa
  new_state, reward, done = act(actual_state, action)
  # print(i, action, done)
  # Definir el nuevo estado y si ya llegó el al objetivo
  actual_state = new_state
  finished = done
  i += 1
  #total_reward = []
  #total_reward += reward
  # Si se llegó al objetivo imprime
  if grid[new_state.robot_pos[0]][new_state.robot_pos[1]] == 't':
    print("Lo lograste!, encontraste el tesoro!:",new_state.robot_pos)
    print("Iteraciones realizadas : ",i)
    print("Cuadricula del entorno final : ")
    for row in new_state.grid:
      print(' '.join(row))
    break
  elif grid[new_state.robot_pos[0]][new_state.robot_pos[1]] == 'o':
    print("El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo")
    print("Quedaste en la posición: ",new_state.robot_pos)
    print("Iteraciones realizadas : ",i)
    print("Cuadricula del entorno final : ")
    for row in new_state.grid:
      print(' '.join(row))
    break
  elif i== max_iter:
   print('El algoritmo aun no converge, prueba mas iteraciones o mas episodios')
```

Comentado [ELJJ1]: Santi porfa cambia esta imagen al bucle actualizado, es que mi Colab esta negro y se vería maluco el cambio de imágenes, incluso si lo pongo en github los colores de los comentarios no son los mismos.

4. Análisis de Resultados

Resultados dejando todos los parámetros iguales y aumentando el número de episodios:

Episodios	Obtuvo la recompensa	Máximo iteraciones	Iteraciones realizadas	Resultado	Posición final
N_EPISODES = 20	NO	10	10	<El algoritmo no converge, prueba más iteraciones o más episodios>	NA
	NO	30	6	<El robot quedó en el obstáculo, entrénalo mejor e intenta de nuevo>	[0,3]
	NO	50	22	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[3,1]
	NO	100	15	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[3,1]
N_EPISODES = 50	SI	10	10	<Lo lograste!, encontraste el tesoro!>	[3,3]
	SI	30	6	<Lo lograste!, encontraste el tesoro!>	[3,3]
	NO	50	4	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[3,1]
	NO	100	4	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[1,1]
N_EPISODES = 100	SI	10	6	<Lo lograste!, encontraste el tesoro!>	[3,3]
	NO	30	5	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[2,3]
	NO	50	8	<El robot quedó en el obstaculo, entrenalo mejor e intenta de nuevo>	[3,1]
	SI	100	6	<Lo lograste!, encontraste el tesoro!>	[3,3]

Conclusión

Se logró construir un algoritmo Q-Learning desde cero y probarlo en un entorno sencillo de Proceso de Decisiones de Markov. Se realizaron pruebas para diferentes cambiando el numero de episodios, que podrían ser interpretados como el nivel de entrenamiento del algoritmo, y efectivamente se comprueba que a mayor nivel de entrenamiento el robot posee un mayor conocimiento del entorno y toma decisiones que lo benefician en términos de la recompensa logrando esquivar los obstáculos y llegar al tesoro. También se puede observar que a mayor nivel de entrenamiento el robot es más eficiente en el camino que escoge, esto se puede evidenciar en que logra resolver el problema en un numero menor de iteraciones cuando tiene mayor conocimiento del entorno.

5. Cuaderno Jupyter

El Notebook con la implementación puede encontrarse en:

https://github.com/santiagoec/CM0891-Aprendizaje-Automatico/blob/master/04_Aprendizaje_Reforzado_Q_Learning.ipynb