Laberinto enviroment using OpenAI gym

Ignacio Noguera Martinez[a], Brenda Cruz Noguera[a]

[a]*Alumnos de Inteligencia Artificial de la Facultad de Ingeniería, Universidad Nacional de Asunción*

PALABRAS CLAVE

Laberinto
Q- learning.
Reinforcement learning.

RESUMEN

El Reinforcement learning es un área del aprendizaje donde la máquina aprende por sí sola el comportamiento a seguir en base a recompensas y penalizaciones. OpenAI Gym es un conjunto de entornos de aprendizaje por refuerzo. El Q-learning permite resolver problemas de decisión secuencial en los cuales la utilidad de una acción depende de una secuencia de decisiones.

1. Introducción

Conociendo los conceptos básicos de RL formamos el entorno *Laberinto* al cual le damos una interfaz fija y definida, para poder entonces reutilizar el mismo entorno en todos nuestros problemas. Al adoptar una interfaz común, podemos colocar este entorno en cualquier sistema existente que también implemente la misma interfaz. Todo lo que tenemos que hacer es decidir qué interfaz debemos usar. Afortunadamente para nosotros, esto ya se ha hecho y se llama interfaz OpenAI Gym. La interfaz para todos los entornos de OpenAI Gym se puede dividir en 3 partes:

1. Inicialización: Crear e inicializar el entorno.(ver Figura 6)
2. Ejecución: Realizar acciones repetidas en el entorno. En cada paso, el entorno proporciona información para describir su nuevo estado y la recompensa recibida como consecuencia de realizar la acción especificada. Esto continúa hasta que el entorno señala que el episodio está completo. (ver Figura 8)
3. Terminación: Limpiar y destruir el ambiente. (ver Figura 9)

2. Objetivos

Crear un ambiente para entrenar un robot para que encuentre la salida en un laberinto, pudiendo éste empezar desde cualquier posición del mundo de cuadrícula simple, agregando componentes, como paredes y trampas, para aumentar las complejidades de los desafíos.

3. Principio de Funcionamiento

Hay una meta designada en el mundo de la cuadrícula. Cuando comienza el episodio, el robot se pone en marcha, en un cuadro al azar. El robot se mueve por el trayecto más rápido hasta la meta G(reen), pero además hay trampas por el camino R(ed). Una vez que llega, el episodio termina. Las acciones posibles son:

- ’ 0 ’ : moverse abajo.
- ’ 1 ’ : moverse arriba.
- ’ 2 ’ : moverse derecha.
- ’ 3 ’ : moverse izquierda.

4. Diseño propuesto

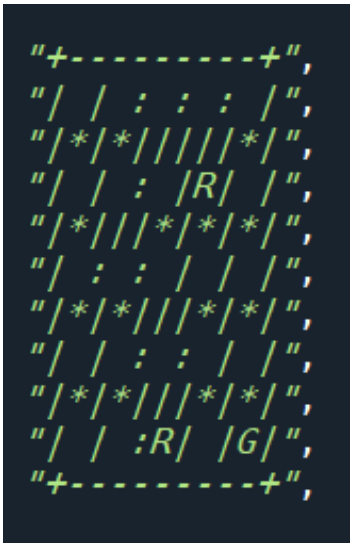

Fig. 1. Laberinto propuesto.

El ambiente consta de:

- ’ / ’ : representa una pared.
- ’ : ’ : representa un espacio donde puede continuar horizontalmente.
- ’ * ’ : espacio donde puede continuar verticalmente.

Con esta configuracion se obtienen un total de 25 estados discretos, ya que contamos con 5 filas, 5 columnas y una meta ’G’. Tambien contamos con 2 trampas ’R’.

5. Entrenamiento

Q-learning permite al agente utilizar las recompensas del entorno para aprender, con el tiempo, la mejor acción a realizar en un estado determinado. Un valor Q para una combinación particular de estado y acción es representativo de la çalidad"de una acción tomada desde ese estado. Mejores valores Q implican mejores posibilidades de obtener mayores recompensas. Los valores Q se inicializan en un valor arbitrario y, a medida que el agente se expone al entorno y recibe diferentes recompensas al ejecutar diferentes acciones, los valores Q se actualizan mediante la ecuación:

Q(estado,acción)←(1−alpha)*Q(estado,acción)+alpha*(recompensa + gamma*maxaQ(siguiente estado,todas las acciones))

- α (alfa) es la tasa de aprendizaje (0 ¡$\alpha \leq 1$): al igual que en los entornos de aprendizaje supervisado,α es la medida en que nuestros valores Q se actualizan en cada iteración.
- γ (gamma) es el factor de descuento ($0 \leq \gamma \leq 1$) determina cuánta importancia queremos dar a las recompensas futuras. Un valor alto para el factor de descuento (cerca de 1) captura la recompensa efectiva a largo plazo, mientras que un factor de descuento de 0 hace que nuestro agente considere solo una recompensa inmediata, lo que lo vuelve codicioso.
- ϵ (epsilon) es el criterio pra el uso de lo valores aleatorios.

Estamos actualizando el valor Q del estado y la acción actuales del agente tomando primero un peso (1−gamma) del valor Q anterior y luego agregando el valor aprendido. El valor aprendido es una combinación de la recompensa por realizar la acción actual en el estado actual y la recompensa máxima descontada del próximo estado en el que estaremos una vez que realicemos la acción actual. maxaQ(siguiente estado, todas las acciones), significa que el valor Q del paso actual se basa en el valor Q del paso futuro. Esto significa que inicializamos los valores Q para St y St+1 en algunos valores aleatorios al principio. En la primera iteración de entrenamiento, actualizamos Q-Value en el estado St en función de la recompensa y en esos valores aleatorios de Q-Value en el estado St+1. Dado que la recompensa sigue guiando nuestro sistema, eventualmente convergerá al mejor resultado. estamos aprendiendo la acción adecuada para tomar en el estado actual al observar la recompensa para el combo de estado/acción actual y las recompensas máximas para el siguiente estado. Todos estos valores Q se almacenan dentro de la tabla Q, que es solo la matriz con las filas para los estados y las columnas para las acciones.

6. Resultados de simulación y discusión

Creamos una pagina donde podemos modificar los valores de α , γ para visualizar con facilidad los parametros y curva recompesa/epocas la cual representa la calidad del entrenamientos . Podemos apreciar los siguientes resultados:

Tabla 1. Tabla de Epocas = 100 y gamma= 0.1 .

Alpha	LLego
0.1	24
0.5	50
0.9	65

Fig. 2. Mejor Tabla 1.

Tabla 2. Tabla de Epocas = 100 y alpha= 0.5 .

Gamma	LLego
0.1	52
0.5	55
0.9	48

Fig. 3. Mejor Tabla 2.

Tabla 3. Tabla de Epocas = 1000 y gamma= 0.1 .

Alpha	LLego
0.1	66, 8
0.5	87, 5
0.9	90,7

Fig. 4. Mejor Tabla 3.

Tabla 4. Tabla de Epocas = 1000 y alpha= 0,5.

Gamma	LLego
0.1	86, 4
0.5	88,7
0.9	89,5

Fig. 5. Mejor Tabla 4.

7. Anexos

```
def __init__(self):
    self.desc = np.asarray(MAP, dtype="c")
    self.locs= locs=[(1,3),(4,2),(4,4)]
    num_states = 25
    num_rows = 5
    num_columns = 5
    max_row = num_rows - 1
    max_col = num_columns - 1
    self.initial_state_distrib = np.zeros(num_states)
    num_actions = 4 #arriba, abajo, izquierda, derecha
    self.P = {state: {action: [] for action in range(num_actions)}
              for state in range(num_states)}
    for row in range(num_rows):
        for col in range(num_columns):
            state = self.encode(row, col)
            self.initial_state_distrib[state] += 1
            for action in range(num_actions):
                robot_loc = (row, col)
                new_row, new_col = row, col
                done = False
                reward = (-1)
                if action == 0 and self.desc[2*row+2 , 2*col+1]== b"*": #desc es e
                    new_row = min(row+1, max_row)
                elif action == 1 and self.desc[2*row , 2*col+1]== b"*":
                    new_row = max(row-1, 0)

                if action == 2 and self.desc[1 + row*2, 2 * col + 2] == b":":
                    new_col = min(col + 1, max_col)
                elif action == 3 and self.desc[1 + row*2, 2 * col] == b":":
                    new_col = max(col - 1, 0)

                if robot_loc == locs[2]:
                   reward = 20
                   done = True
                elif robot_loc == locs[0] or robot_loc == locs[1]:
                   reward = -10
                   done = True
                new_state = self.encode(new_row, new_col)
                self.P[state][action].append((1.0, new_state, reward, done))
    self.initial_state_distrib /= self.initial_state_distrib.sum()
    self.action_space = spaces.Discrete(num_actions)
    self.observation_space = spaces.Discrete(num_states)
```

Fig. 6. Instancia inicializadora.

```
alpha=0.5
gamma=0.9
epsilon=0.001
q_table=np.zeros([entorno.observation_space.n,entorno.action_space.n])
estado_anterior=None
mostrar_entrenamiento=False
r_mean=[]
steps_por_episodios=[]
nro_de_episodios=100
listarecompnesa=[]
for episodio in tqdm(range(0,nro_de_episodios)):
  #Resetear el entorno
  estado = entorno.reset()
  estado_anterior= estado
  #Initialize variables
  recompensa=0
  r_sum=0
  r_sums=[]
  steps=0
  terminado=False
  while not terminado:
    #tomar el camino aprendido o explorar nuevas acciones basadas en  eps
    if random.uniform(0,1)<epsilon:
      accion=entorno.action_space.sample()
    else:
      accion=np.argmax(q_table[estado])
    #tomar la accion
    siguiente_estado, recompensa, terminado, info = entorno.step(accion)
    #Recalcular
    q_valor=q_table[estado,accion]
    max_valor=np.max(q_table[siguiente_estado])
    nuevo_q_valor=(1-alpha)*q_valor + alpha*(recompensa + gamma*max_valor)
    #Guardar estado anterior
    estado_anterior=estado
    r_sum +=recompensa
    steps += 1
    #Actualizar Q-table
    q_table[estado,accion]=nuevo_q_valor
    estado= siguiente_estado
  listarecompnesa.append(r_sum)
```

Fig. 7. Entrenamiento.

```
def step(self, a):
    transitions = self.P[self.s][a]
    i = categorical_sample([t[0] for t in transitions], self.np_random)
    p, s, r, d = transitions[i]
    self.s = s
    self.lastaction = a
    return (int(s), r, d, {"prob": p})
```

Fig. 8. Pasos.

```
def reset(self,*,seed: Optional[int] = None,return_info: bool = False,
    options: Optional[dict] = None,):

    super().reset(seed=seed)
    self.s = categorical_sample(self.initial_state_distrib, self.np_random)
    self.lastaction = None
    if not return_info:
        return int(self.s)
    else:
        return int(self.s), {"prob": 1}
```

Fig. 9. Reset.

8. Conclusión

Después de varios entrenamientos, variando los valores de gamma,alpha,epsilon y la epocas se llego a que el mejor resultado obtenido se debe a que valores de gamma=0.1 alpha=0.9 epsilon= 0.001 ,a partir de estos valores no se notan cambios signficativos con la variacion de las epocas y los resultados son muy óptimos

9. Referencias

- OpenAI GYM
- Kaggle
- www.aprendemachinelearning.com