Laboratorio 6 Grupal "Aprendizaje por Refuerzo".

Descripción del Trabajo.

Nombres:

- Aramayo Valdez Joaquin.

- Piza Nava Vladimir.

- Viza Hoyos Maria Belen.

- Mendoza Ovando Carlos Saul.

- Solorzano Diego.

Link GitHub:
https://github.com/Joaco15045F/InteligenciaArtificial/tree/main/Laboratorio6%20grupal%20AxR

1. **Importación de Librerías**

```python
import numpy as np
import gymnasium as gym
import matplotlib.pyplot as plt
from collections import defaultdict
```

Primero, se importan las librerías necesarias:

- numpy se usa para manejar cálculos numéricos, como la manipulación de arrays.
- gymnasium proporciona el entorno BipedalWalker, que es un simulador de un robot bípedo que debe aprender a caminar.
- matplotlib.pyplot se utiliza para generar gráficos que muestran el progreso del aprendizaje del agente.
- collections.defaultdict permite crear un diccionario especializado que manejará la tabla Q, asignando valores por defecto si una clave no existe.

2. **Inicializamos el Entorno**

```python
# Inicializar el entorno
env = gym.make("BipedalWalker-v3")
```

Se inicializa el entorno del BipedalWalker utilizando la función gym.make("BipedalWalker-v3"). Esto crea una instancia del entorno de

simulación del robot bípedo, que proporciona un espacio donde el agente aprenderá a caminar. El entorno incluye la definición de estados y acciones posibles, además de la dinámica física que determinará cómo se mueve el agente en respuesta a las acciones seleccionadas. A partir de aquí, se podrán realizar simulaciones para entrenar y evaluar el rendimiento del agente.

3. **Definimos los parámetros de Q-Learning.**

```
# Parámetros de Q-learning
alpha = 0.1       # Tasa de aprendizaje
gamma = 0.99      # Factor de descuento
epsilon = 0.1     # Tasa de exploración
n_bins = 5        # Número de divisiones para discretización en
cada dimensión
```

En esta sección se definen los parámetros esenciales para el algoritmo de Q-learning:

- **alpha (Tasa de aprendizaje)**: Controla cuánto impacto tiene la nueva información en la actualización de la Tabla Q. Un valor bajo hace que los cambios sean más lentos, mientras que un valor alto los hace más rápidos.
- **gamma (Factor de descuento)**: Determina la importancia de las recompensas futuras. Un valor cercano a 1 prioriza recompensas a largo plazo, mientras que valores bajos priorizan recompensas inmediatas.
- **epsilon (Tasa de exploración)**: Controla la probabilidad de explorar nuevas acciones en lugar de explotar el conocimiento actual. Un valor más alto fomenta la exploración, mientras que un valor bajo promueve la explotación.
- **n_bins**: Establece la cantidad de divisiones para discretizar el espacio de observación del entorno, simplificando el estado continuo a una representación discreta que la Tabla Q puede manejar.

Estos parámetros son clave para equilibrar el aprendizaje y la eficiencia del agente durante el entrenamiento.

4. **Discretización de los estados del entorno y de las acciones.**

```
# Discretizar el espacio de observación
def discretize_obs(obs):
    bins = [np.linspace(-5, 5, n_bins) for _ in range(len(obs))]
    discretized = tuple(np.digitize(o, bins[i]) for i, o in
enumerate(obs))
    return discretized
```

```
# Crear un conjunto de acciones discretizadas para cada dimensión
del espacio de acción
action_space_dim = env.action_space.shape[0]
action_bins = np.linspace(-1, 1, n_bins)  # Rango de acción para
cada articulación

# Inicializar Q-table como un diccionario
q_table = defaultdict(lambda: np.zeros((n_bins,) *
action_space_dim))
```

En esta sección, se establece la forma en que se discretizan tanto los estados del entorno como las acciones que puede realizar el agente:

1. **discretize_obs**: Es una función que toma un estado continuo (obs) del entorno y lo convierte en un estado discreto utilizando bins. Los bins son divisiones del rango de valores posibles para cada dimensión del estado, y se crean con np.linspace(-5, 5, n_bins). Luego, np.digitize asigna cada valor a un bin, lo que genera un estado discreto representado como una tupla.
2. **Discretización del espacio de acción**: La variable action_space_dim se determina a partir del número de articulaciones controlables en el entorno. Los posibles valores de acción para cada articulación se dividen en n_bins usando np.linspace(-1, 1, n_bins), cubriendo el rango completo de movimiento permitido.
3. **Inicialización de la Tabla Q (q_table)**: Se utiliza un defaultdict que crea una tabla Q donde cada estado es una clave, y los valores corresponden a una matriz de ceros con dimensiones (n_bins,) * action_space_dim. Esta matriz representa los Q-valores asociados a las acciones posibles en ese estado discreto.

5. **Definir la Política de acción para el Agente.**

```
# Definir la política de acción (ε-greedy)
def choose_action(state, training=True):
    # Discretizar el estado para usarlo como clave
    state = discretize_obs(state)

    if training and np.random.rand() < epsilon:
        # Acción aleatoria en cada dimensión
        return np.random.choice(action_bins, size=action_space_dim)
    else:
        # Seleccionar la mejor acción para cada dimensión
```

```
        best_action_idx =
np.unravel_index(np.argmax(q_table[state]), (n_bins,) *
action_space_dim)
        return np.array([action_bins[i] for i in best_action_idx])
```

Aquí definimos la política de acción para el agente utilizando una estrategia **ε-greedy**. La política ε-greedy balancea la exploración y la explotación al tomar decisiones:

1. **Discretización del estado**: El estado continuo se convierte en un estado discreto mediante la función discretize_obs. Esto permite utilizarlo como clave en la Tabla Q (q_table).
2. **Exploración vs. Explotación**:
    - Si training es True y un número aleatorio generado por np.random.rand() es menor que epsilon (parámetro de exploración), el agente elige una acción al azar. La acción se selecciona generando valores aleatorios a partir de action_bins, lo que permite explorar nuevas acciones.
    - Si el agente decide explotar (lo cual sucede si no se cumple la condición anterior), selecciona la mejor acción posible en base a la Tabla Q. Esto se logra usando np.argmax para identificar el índice de la mejor acción, y luego convertir este índice en los valores correspondientes de action_bins.

Esta función le permite al agente encontrar un equilibrio entre explorar nuevas estrategias y explotar el conocimiento aprendido para optimizar su rendimiento en el entorno.

6. **Entrenamiento y Gráfica de recompensas.**

```
# Entrenar el agente con Q-learning
rewards_per_episode = []

for episode in range(5000):
    state, _ = env.reset()
    done = False
    total_reward = 0

    while not done:
        action = choose_action(state)
        next_state, reward, terminated, truncated, _ =
env.step(action)
        done = terminated or truncated
```

```
        # Discretizar el estado y la acción
        state_discrete = discretize_obs(state)
        next_state_discrete = discretize_obs(next_state)
        action_idx = tuple(np.digitize(a, action_bins) - 1 for a in
action)

        # Actualizar Q-table usando la ecuación de Q-learning
        if not done:
            next_action_idx =
np.unravel_index(np.argmax(q_table[next_state_discrete]), (n_bins,)
* action_space_dim)
            best_next_q_value =
q_table[next_state_discrete][next_action_idx]
            td_target = reward + gamma * best_next_q_value
        else:
            td_target = reward

        td_error = td_target - q_table[state_discrete][action_idx]
        q_table[state_discrete][action_idx] += alpha * td_error

        # Acumular la recompensa
        total_reward += reward
        state = next_state

    # Guardar la recompensa total del episodio
    rewards_per_episode.append(total_reward)

    # Mostrar progreso
    if episode % 100 == 0:
        print(f"Episodio {episode}, Recompensa total:
{total_reward}")

# Graficar las recompensas por episodio
window_size = 100
smoothed_rewards = np.convolve(rewards_per_episode,
np.ones(window_size) / window_size, mode='valid')
plt.plot(smoothed_rewards)
plt.xlabel('Episodios')
plt.ylabel('Recompensa Promedio (Promedio móvil)')
plt.title('Progreso del Aprendizaje del Agente en BipedalWalker-
v3')
plt.show()
```

En esta parte del código se lleva a cabo el **entrenamiento del agente** utilizando el algoritmo de **Q-learning** durante 5000 episodios. El proceso se divide en varias etapas clave:

1. **Inicialización del Episodio**:

- Se reinicia el entorno al comienzo de cada episodio con env.reset().
- Se inicializan variables como total_reward para llevar un seguimiento de las recompensas obtenidas en cada episodio.

2. **Bucle de Entrenamiento**:
   - El agente elige una acción usando la política definida en choose_action.
   - Se ejecuta la acción en el entorno con env.step(action), obteniendo la nueva observación (next_state) y la recompensa asociada.
   - El estado y la acción se **discretizan** para facilitar la actualización de la **Q-table**.
3. **Actualización de la Q-table**:
   - Si el episodio no ha terminado, se calcula el **objetivo temporal** (td_target) usando la recompensa recibida y el valor Q estimado para el siguiente estado.
   - Si el episodio ha terminado, el td_target es simplemente la recompensa obtenida.
   - Se calcula el **error temporal** (td_error) y se actualiza la Q-table en base a la fórmula de Q-learning.
4. **Seguimiento de Recompensas**:
   - Se acumula la recompensa total del episodio y se almacena en rewards_per_episode.
   - Cada 100 episodios, se muestra en pantalla la recompensa total obtenida para evaluar el progreso del agente.
5. **Visualización del Rendimiento**:
   - Al finalizar el entrenamiento, se genera una gráfica que muestra la recompensa promedio utilizando un **promedio móvil** para suavizar las fluctuaciones y evaluar el progreso del agente en el entorno BipedalWalker-v3.

7. **Visualizar la Tabla Q.**

```
# Mostrar una parte de la Tabla Q
print("\n--- Muestra de la Tabla Q ---")
sample_states = list(q_table.keys())[:5]  # Muestra de los primeros
5 estados
for state in sample_states:
    print(f"Estado {state}: Q-valores {q_table[state]}")
```

Este código muestra una parte de la Tabla Q, permitiendo observar algunos de los valores Q aprendidos durante el entrenamiento del agente. Se seleccionan los primeros cinco estados registrados en la tabla y se imprimen junto con sus valores Q correspondientes, que representan la calidad esperada de las acciones en esos estados. Esto ayuda a entender cómo el agente ha aprendido a valorar distintas decisiones en situaciones específicas.

8. **Prueba del Agente en modo Explotación (épsilon = 0)**

```
# Prueba del agente en modo explotación (sin exploración)
def test_agent(num_episodes=10):
    epsilon = 0  # Desactivar exploración
    total_rewards = []

    # Crear el entorno con render_mode="human" solo para la fase de
prueba
    env = gym.make("BipedalWalker-v3", render_mode="human")

    for episode in range(num_episodes):
        state, _ = env.reset()
        done = False
        total_reward = 0
        while not done:
            action = choose_action(state, training=False)
            next_state, reward, terminated, truncated, _ =
env.step(action)
            total_reward += reward
            state = next_state
            done = terminated or truncated
            env.render()  # Mostrar el render del entorno en cada
paso (solo en explotación)
        total_rewards.append(total_reward)
        print(f"Recompensa total en episodio de prueba {episode +
1}: {total_reward}")
    print(f"\nRecompensa promedio en modo explotación:
{np.mean(total_rewards)}")

# Ejecutar la prueba del agente
test_agent()
```

Este código realiza una prueba del agente en modo explotación, es decir, sin exploración, para evaluar su rendimiento. En este modo, el agente siempre selecciona las mejores acciones basadas en lo aprendido durante el entrenamiento, utilizando la política ε-greedy sin exploración (epsilon = 0). El

entorno se inicializa con la opción render_mode="human", lo que permite visualizar el entorno y la interacción del agente en cada paso. El código ejecuta múltiples episodios, mostrando las recompensas totales obtenidas en cada uno, y calcula la recompensa promedio de todas las pruebas para evaluar su desempeño.

Imagen de la evaluación con render.