ITSRLL
INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

Ingeniería Mecatrónica

PROGRAMACIÓN AVANZADA

Enero – Junio 2025

M.C. Osbaldo Aragón Banderas

UNIDAD: 3

Actividad número: A5

Nombre de actividad:

EXPERIMENTACIÓN GUIADA. Explorando Q-Learning en FrozenLake

Actividad realizada por:

Roberto Jair Arteaga Valenzuela

Guadalupe Victoria, Durango

Fecha de entrega: 13 de abril de 2025

Práctica: EXPERIMENTACIÓN GUIADA Explorando Q-Learning en FrozenLake

Introducción

El aprendizaje por refuerzo (RL) es un área de la inteligencia artificial que estudia cómo los agentes pueden aprender a tomar decisiones en un entorno dinámico y desconocido, basándose en la retroalimentación recibida de sus propias acciones. A través de la interacción continua con el entorno, un agente puede mejorar su comportamiento para maximizar una recompensa acumulativa a lo largo del tiempo.

En esta práctica se utilizó el entorno FrozenLake-v1 proporcionado por el paquete Gymnasium, que simula un lago congelado donde el agente debe encontrar el camino hacia la meta, evitando caer en los hoyos. Para entrenar al agente, se utilizó el algoritmo de Q-Learning, un método clásico de aprendizaje por refuerzo que permite al agente aprender una política mediante la construcción de una Q-table (tabla de valores Q) que asocia a cada par estado-acción un valor de la acción.

A lo largo de esta práctica, el objetivo fue observar cómo variaciones en los parámetros del algoritmo afectan el comportamiento del agente, así como reflexionar sobre la aplicabilidad del Q-Learning en escenarios del mundo real.

Objetivo de la Actividad:

Comprender el funcionamiento del algoritmo Q-Learning aplicado al entorno FrozenLake y analizar el efecto de los parámetros clave del aprendizaje por refuerzo sobre el rendimiento del agente.

Parte A. Análisis del Algoritmo Base

1. **¿Qué hace el parámetro epsilon y cómo cambia a lo largo del entrenamiento?**
   El parámetro epsilon define la probabilidad de que el agente explore en vez de explotar el conocimiento aprendido. Al inicio del entrenamiento, epsilon suele tener un valor alto (por ejemplo, 1.0), lo que favorece la exploración. A medida que pasa el tiempo, se va reduciendo mediante un valor llamado epsilon_decay. Esto permite que el agente explore al principio y, conforme aprende, empiece a explotar las mejores acciones conocidas.

2. **¿Cuál es la función de la Q-table?**
   La Q-table es una estructura de datos que almacena los valores Q para cada combinación de estado y acción. Estos valores representan la "utilidad esperada" de realizar una acción desde un estado específico. La Q-table permite al agente tomar decisiones inteligentes al elegir la acción con mayor valor Q en cada situación.

3. **¿Qué significa que el entorno sea is_slippery=True?**
   Este parámetro define si el entorno tiene elementos de aleatoriedad. Cuando is_slippery=True, el agente podría resbalar y moverse en una dirección distinta a la deseada, lo que dificulta la navegación. En esta práctica se usó is_slippery=False, lo cual hace que el entorno sea determinista, facilitando el aprendizaje.

4. **¿Cómo se decide si el agente explora o explota en cada paso?**
   Se utiliza una estrategia llamada **ε-greedy**. El agente genera un número aleatorio: si es menor que epsilon, elige una acción aleatoria (explora); si no, elige la acción con mayor valor en la Q-table (explota). Esto permite un equilibrio entre descubrimiento y aprovechamiento del conocimiento.

Parte B. Experimentación

Se realizaron tres experimentos variando distintos parámetros. Después de entrenar al agente, se observó su comportamiento en 5 episodios de prueba.

Experimento	Parámetros Cambiados	Episodios Ganados	Comentario
#1	α=0.5, γ=0.9, epsilon=1.0, decay=0.0001, episodes=10000	1/5	El agente aprendió lentamente. No exploró lo suficiente ni ajustó bien los valores.
#2	α=0.9, γ=0.99, epsilon=1.0, decay=0.0002, episodes=30000	5/5	El agente aprendió muy bien. El alto número de episodios y gamma mejoraron su política.
#3	α=0.1, γ=0.9, epsilon=0.5, decay=0.00005, episodes=5000	2/5	Muy baja tasa de aprendizaje y gamma. El agente no aprendió valores óptimos.

Experimentación (Respuestas por Experimento)

Experimento #1

- **Parámetros modificados**
    - Se modificaron los parámetros **α (tasa de aprendizaje)**, **γ (factor de descuento)**, **epsilon (exploración)**, y **el número de episodios**. Los valores fueron los siguientes:
        - **α = 0.5** (tasa de aprendizaje moderada)
        - **γ = 0.9** (factor de descuento alto)
        - **epsilon = 1.0** (exploración total)
        - **decay = 0.0001** (decadencia de epsilon muy lenta)
        - **episodes = 10000** (número de episodios de entrenamiento)
- **¿Cuál fue el comportamiento del agente al final del entrenamiento?**
    - El agente mostró un **aprendizaje moderadamente lento**. No exploró lo suficiente debido a una tasa de aprendizaje moderada (α=0.5) y la **decadencia muy lenta de epsilon**. Como resultado, el agente no ajustó bien los valores en la Q-table y no mejoró significativamente su desempeño.
- **¿Cuántos episodios ganó?**
    - El agente ganó **2 de 5 episodios** (20%).
- **¿Cómo explicarías ese resultado en términos de aprendizaje por refuerzo?**
    - El resultado puede explicarse porque la tasa de aprendizaje moderada y el factor de descuento relativamente alto no fueron suficientes para permitir que el agente aprendiera de manera óptima. El parámetro epsilon comenzó en un valor alto (1.0), lo que permitió exploración, pero la **decadencia lenta de epsilon** impidió que el agente explotara

correctamente el conocimiento adquirido en fases posteriores del entrenamiento, afectando negativamente el rendimiento. La falta de un aprendizaje eficiente y la exploración insuficiente fueron claves para este bajo desempeño.

```python
def train_q_learning(
    map_name="4x4",
    episodes=10000,
    learning_rate=0.5,
    discount_factor=0.9,
    epsilon=1.0,
    epsilon_decay=0.0001,
    q_file_name="frozen_lake_qtable.pkl",
    slippery=False
):
```

Figura 1 Diferentes valores para entrenamiento 1

```
✅ Entrenamiento terminado y Q-table guardada en 'frozen_lake_qtable.pkl'.

--- Episodio de prueba 1/5 ---
❌ El agente falló o cayó en un hoyo.

--- Episodio de prueba 2/5 ---
❌ El agente falló o cayó en un hoyo.

--- Episodio de prueba 3/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 4/5 ---
❌ El agente falló o cayó en un hoyo.

--- Episodio de prueba 5/5 ---
❌ El agente falló o cayó en un hoyo.

🎯 Total de episodios ganados: 1/5 = 20.00%
```

Figura 2 Resultados para el experimento 1

Experimento #2

- **Parámetros modificados**
    - En este experimento, los parámetros modificados fueron:
        - **α = 0.9** (alta tasa de aprendizaje)
        - **γ = 0.99** (factor de descuento muy alto)
        - **epsilon = 1.0** (exploración total)
        - **decay = 0.0002** (decadencia más rápida de epsilon)
        - **episodes = 30000** (mayor número de episodios de entrenamiento)
- **¿Cuál fue el comportamiento del agente al final del entrenamiento?**
    - El agente aprendió **muy bien** debido al alto número de episodios (30000) y el valor elevado de **γ = 0.99**, que permitió valorar las recompensas futuras. La exploración fue adecuada y la política mejoró de manera significativa.

- **¿Cuántos episodios ganó?**
    - El agente ganó **5 de 5 episodios** (100%).
- **¿Cómo explicarías ese resultado en términos de aprendizaje por refuerzo?**
    - El éxito del agente puede atribuirse a la combinación de parámetros con un alto valor de α que permitió un rápido aprendizaje de nuevas acciones, un factor de descuento alto (γ=0.99) que promovió la consideración de recompensas a largo plazo, y un número elevado de episodios (30000) que brindó suficientes oportunidades para explorar y ajustar la política del agente. La decadencia más rápida de epsilon también permitió una exploración suficiente al principio y una

explotación efectiva al final, lo que resultó en una política óptima al final del entrenamiento.

```python
# --- Función de entrenamiento ---
def train_q_learning(
    map_name="4x4",
    episodes=30000,
    learning_rate=0.5, #Alpha
    discount_factor=0.99,
    epsilon=1.0,
    epsilon_decay=0.0002,
    q_file_name="frozen_lake4x4.pkl"
```

Figura 3 Diferentes valores para entrenamiento 2

```
✅ Entrenamiento terminado y Q-table guardada en 'frozen_lake_qtable.pkl'.

--- Episodio de prueba 1/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 2/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 3/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 4/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 5/5 ---
✅ ¡El agente llegó a la meta!

🎯 Total de episodios ganados: 5/5 = 100.00%
```

Figura 4 Demostración del agente

Experimento #3

- **Parámetros modificados**
    - Los parámetros modificados en este experimento fueron:
        - **α = 0.1** (tasa de aprendizaje muy baja)
        - **γ = 0.5** (factor de descuento bajo)
        - **epsilon = 1.0** (exploración moderada)
        - **decay = 0.00005** (decadencia muy lenta de epsilon)
        - **episodes = 5000** (menos episodios de entrenamiento)
- **¿Cuál fue el comportamiento del agente al final del entrenamiento?**
    - El agente mostró un desempeño **deficiente**. La tasa de aprendizaje baja y el factor de descuento pequeño impidieron que el agente aprendiera correctamente los valores de las acciones, lo que resultó en una **política subóptima**.
- **¿Cuántos episodios ganó?**
    - El agente ganó **2 de 5 episodios** (20%).
- **¿Cómo explicarías ese resultado en términos de aprendizaje por refuerzo?**
    - El bajo desempeño del agente puede explicarse por la **tasa de aprendizaje muy baja (α=0.1)**, lo que hizo que el agente aprendiera de manera muy lenta y no ajustara correctamente su Q-table. El **factor de descuento bajo (γ=0.5)** impidió que el agente valorara adecuadamente las recompensas futuras, lo que dificultó la toma de decisiones a largo plazo. Además, la **exploración moderada** no fue suficiente para permitir al agente descubrir las mejores acciones y estrategias, lo que resultó en un aprendizaje deficiente y un desempeño bajo.

```python
# --- Función de entrenamiento ---
def train_q_learning(
    map_name="4x4",
    episodes=5000,
    learning_rate=0.1, #Alpha
    discount_factor=0.5,
    epsilon=0.5,
    epsilon_decay=0.00005,
    q_file_name="frozen_lake4x4.pkl"
):
```

Figura 5 Diferentes valores para entrenamiento 3

```
✅ Entrenamiento terminado y Q-table guardada en 'frozen_lake_qtable.pkl'.

--- Episodio de prueba 1/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 2/5 ---
❌ El agente falló o cayó en un hoyo.

--- Episodio de prueba 3/5 ---
❌ El agente falló o cayó en un hoyo.

--- Episodio de prueba 4/5 ---
✅ ¡El agente llegó a la meta!

--- Episodio de prueba 5/5 ---
❌ El agente falló o cayó en un hoyo.

🎯 Total de episodios ganados: 2/5 = 40.00%
```

Figura 6 Resultados para experimento 3

Código de muestra completo

```python
import gymnasium as gym
import numpy as np
import pickle

# --- Función de entrenamiento ---
def train_q_learning(
    map_name="4x4",
    episodes=30000,
    learning_rate=0.9,
    discount_factor=0.99,
    epsilon=1.0,
    epsilon_decay=0.0002,
    q_file_name="frozen_lake4x4.pkl"
):
    """
    Entrena un agente Q-learning en el entorno FrozenLake usando el mapa especificado.
    No se visualiza el entorno durante el entrenamiento.

    :param map_name: Nombre del mapa ("4x4" o "8x8", etc.).
    :param episodes: Número de episodios para entrenar.
    :param learning_rate: Tasa de aprendizaje (alpha).
    :param discount_factor: Factor de descuento (gamma).
    :param discount_factor: Factor de descuento (gamma).
    :param epsilon: Probabilidad inicial de explorar (política ε-greedy).
    :param epsilon_decay: Tasa a la que disminuye epsilon en cada episodio.
    :param q_file_name: Nombre del archivo donde se guardará la Q-table resultante.
    """
    # Crear el entorno. Sin render, pues no se quiere visualizar nada en entrenamiento.
    env = gym.make('FrozenLake-v1', map_name=map_name, is_slippery=False)

    # Inicializar la Q-table con ceros
    q_table = np.zeros((env.observation_space.n, env.action_space.n))

    # Generador de números aleatorios
    rng = np.random.default_rng()

    for ep in range(episodes):
        # Reiniciamos el entorno y obtenemos el estado inicial
        state = env.reset()[0]
        terminated = False
        truncated = False

        while not terminated and not truncated:
            # Decidimos la acción según la política ε-greedy
            if rng.random() < epsilon:
                action = env.action space.sample()
            else:
                action = np.argmax(q_table[state, :])

            # Ejecutamos la acción en el entorno
            new_state, reward, terminated, truncated, _ = env.step(action)

            # Actualizamos la Q-table usando la ecuación de Q-learning
            q_table[state, action] += learning_rate * (
                reward
                + discount_factor * np.max(q_table[new_state, :])
                - q_table[state, action]
            )

            # Avanzamos al siguiente estado
            state = new_state
```

```
        # Actualización de epsilon (disminuye para reducir la exploración con el tiempo)
        epsilon = max(epsilon - epsilon_decay, 0)

    # Cerramos el entorno tras entrenar
    env.close()

    # Guardamos la Q-table entrenada en un archivo
    with open(q_file_name, "wb") as f:
        pickle.dump(q_table, f)

    print(f"Entrenamiento finalizado. Q-table guardada en: {q_file_name}")

# --- Función de prueba (visualización final) ---
def run_trained_agent(
    map_name="4x4",
    episodes=5,
    q_file_name="frozen_lake4x4.pkl"
):
    """
    Carga una Q-table entrenada y ejecuta varios episodios para
    observar cómo el agente se mueve en FrozenLake (con render activo).
    :param map_name: Nombre del mapa ("4x4" o "8x8", etc.).
    :param episodes: Número de episodios de prueba que se visualizarán.
    :param q_file_name: Archivo pickle donde está almacenada la Q-table entrenada.
    """
    # Creamos el entorno con render para ver la ejecución
    env = gym.make('FrozenLake-v1', map_name=map_name, is_slippery=False, render_mode='human')

    # Cargar la Q-table entrenada
    with open(q_file_name, "rb") as f:
        q_table = pickle.load(f)

    # Para medir cuántos episodios se ganan
    total_wins = 0

    for ep in range(episodes):
        print(f"--- Episodio de prueba {ep+1}/{episodes} ---")
        state = env.reset()[0]
        terminated = False
        truncated = False
        while not terminated and not truncated:
            # Elegimos la acción con el máximo valor Q
            action = np.argmax(q_table[state, :])
            new_state, reward, terminated, truncated, _ = env.step(action)
            state = new_state

            if terminated or truncated:
                # Si reward == 1 significa que llegamos a la meta
                if reward == 1:
                    print("¡El agente ha llegado a la meta!")
                    total_wins += 1
                else:
                    print("El agente cayó en un hoyo o se truncó el episodio.")
                break

    env.close()
    print(f"Episodios ganados: {total_wins}/{episodes} = {100 * total_wins/episodes:.2f}%")
```

```python
# --- Ejecución principal ---
if __name__ == "__main__":
    # 1) Entrenamos el agente (SIN visualización)
    train_q_learning(
        map_name="4x4",
        episodes=10000,           # Ajusta según quieras más o menos entrenamiento
        learning_rate=0.9,
        discount_factor=0.9,
        epsilon=1.0,
        epsilon_decay=0.0001,
        q_file_name="frozen_lake4x4.pkl"
    )

    # 2) Probamos el agente entrenado (CON visualización)
    run_trained_agent(
        map_name="4x4",
        episodes=5,               # Número de episodios con render
        q_file_name="frozen_lake4x4.pkl"
    )
```

Parte C. Reflexión Final

Impacto de los parámetros:

La combinación de parámetros es crucial en el aprendizaje de un agente en un entorno de Q-Learning. **Una tasa de aprendizaje adecuada** es esencial para asegurar que el agente pueda aprender de manera eficiente sin sobreajustarse o subajustarse. Un **factor de descuento más alto** permite que el agente valore más las recompensas futuras, lo que es crucial en entornos de largo plazo. Además, el parámetro epsilon influye directamente en la cantidad de exploración que realiza el agente, lo que determina cómo descubrirá nuevas acciones y se adaptará a su entorno.

Combinación más efectiva:

El mejor desempeño se obtuvo en el **Experimento #2** con α=0.9, γ=0.99 y episodes=30000, lo que permitió al agente aprender una política robusta y explorar el entorno durante un número adecuado de episodios. Esta combinación parece ser la más eficaz para que el agente logre el equilibrio entre exploración y explotación, además de aprender a largo plazo.

Aplicaciones en el mundo real:

El algoritmo Q-Learning tiene aplicaciones en una variedad de campos del mundo real, especialmente en robots autónomos, vehículos sin conductor, sistemas de recomendación y juegos de inteligencia artificial. Cualquier problema que implique decisiones secuenciales puede beneficiarse de Q-Learning, ya que el agente puede aprender a maximizar sus recompensas a medida que interactúa con su entorno.

Conclusión

A lo largo de esta práctica se pudo observar claramente cómo los distintos parámetros que controlan el algoritmo de Q-Learning influyen de forma significativa en el desempeño del agente dentro del entorno FrozenLake. El proceso de modificar y analizar parámetros como la tasa de aprendizaje (α), el factor de descuento (γ), la probabilidad de exploración inicial (ε) y su decaimiento (epsilon_decay), así como el número total de episodios de entrenamiento, permitió comprender a profundidad cómo cada uno de estos elementos afecta tanto la velocidad como la calidad del aprendizaje.

Uno de los principales aprendizajes fue que no existe una única combinación de parámetros óptimos para todos los casos, pero sí se pueden identificar tendencias claras: una tasa de aprendizaje alta permite al agente ajustar rápidamente su política con base en las recompensas obtenidas, mientras que un factor de descuento alto promueve la toma de decisiones pensando en el beneficio a largo plazo. También se evidenció que una alta exploración inicial combinada con una decadencia bien ajustada en epsilon ayuda a que el agente explore el entorno lo suficiente al principio y luego comience a explotar su conocimiento para mejorar su desempeño.

En particular, en el Experimento #2, donde se utilizaron valores altos tanto para la tasa de aprendizaje como para el factor de descuento, y se aumentó el número de episodios de entrenamiento, el agente fue capaz de desarrollar una política casi perfecta, ganando todos los episodios de prueba. Esto demuestra que un entrenamiento más prolongado y bien parametrizado puede llevar a un aprendizaje sólido y confiable.

Por otro lado, el Experimento #3 evidenció las consecuencias de una configuración poco adecuada: al reducir la tasa de aprendizaje y el factor de descuento, y limitar la exploración, el agente no fue capaz de aprender una estrategia efectiva, lo que resalta la sensibilidad del algoritmo a una mala configuración.

Finalmente, esta práctica también permitió reflexionar sobre la aplicación del aprendizaje por refuerzo en el mundo real. Algoritmos como Q-Learning son fundamentales en áreas como la robótica, los videojuegos, la navegación autónoma y sistemas de recomendación, donde los agentes deben aprender a tomar decisiones óptimas en entornos complejos y dinámicos. La capacidad de aprender mediante la interacción con el entorno, sin supervisión explícita, convierte al aprendizaje por refuerzo en una herramienta poderosa para resolver problemas donde no es posible definir reglas exactas desde el principio.

Link del video de demostración de los parámetros que más funcionaron: https://youtu.be/q6U_tL9Rf_o

Link del GitHub:

https://github.com/Jair-Artreaga/Explorando_QLearning_FrozenLake.git