Universidad de Buenos Aires
Facultad de Ingeniería

Especialización en Inteligencia Artificial

Aprendizaje por Refuerzo I 2025
Desafío Práctico

Agosto 2025

Autores:
- Agustin Lopez Fredes

Asignatura: Aprendizaje por Refuerzo I
Tema: Desafío Práctico
URL Repositorio:
https://github.com/Agustinlopezf/CEIA-AprendizajeRefuerzoI

Contenido

1. Introducción al Problema

El desafío práctico consiste en desarrollar un script en Python que resuelva un problema simple utilizando técnicas de Aprendizaje por Refuerzo (RL) vistas en el cursado, específicamente Monte Carlo, Diferencia Temporal, o Deep Q-Network (DQN). Se seleccionó el entorno FrozenLake-v1 de la biblioteca Gymnasium, un problema clásico de navegación en una grilla donde un agente debe cruzar un lago congelado evitando agujeros para llegar al objetivo.

Descripción del Entorno

- Grilla: 4x4 (por defecto), con posiciones:
  - 'S': Inicio (Start).
  - 'F': Casillas congeladas seguras.
  - 'H': Agujeros (Holes), estados terminales negativos.
  - 'G': Objetivo (Goal), estado terminal positivo.

- Acciones: 4 posibles (arriba, abajo, izquierda, derecha).
- Recompensas Originales:
  - +1 al llegar a 'G'.
  - 0 al caer en 'H' o en pasos intermedios.

- Características: Deshabilitamos el resbalón (is_slippery=False) para enfocarnos en aprendizaje determinístico, aunque el entorno es estocástico en su versión original.
- Objetivo: Aprender una política óptima que maximice la recompensa acumulada, minimizando pasos y evitando fallos.

Este problema modela escenarios de toma de decisiones secuenciales con incertidumbre, aplicable a robótica, juegos, etc. Se eligió Q-Learning (Diferencia Temporal off-policy) por su simplicidad y efectividad en entornos discretos.
El entorno se crea usando Gymnasium (no Gym, obsoleto desde 2022, como indica la consigna), evitando problemas de dependencias.

2. Enfoque de Solución

2.1. Algoritmo Seleccionado: Q-Learning

Q-Learning es un algoritmo de RL basado en Diferencia Temporal que aprende una función Q(s, a) estimando el valor esperado de recompensa futura para cada par estado-acción. Es off-policy, permitiendo aprender de trayectorias generadas por una política diferente (ε-greedy para exploración).

Fórmula de actualización:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma * \max_{a'} Q(s',a') - Q(s,a)]$$

- α: Tasa de aprendizaje (con decaimiento).
- γ: Factor de descuento (0.99).
- r: Recompensa inmediata.
- s': Próximo estado.

Política: ε-greedy con decaimiento (ε de 1.0 a 0.01) para balancear exploración (acciones aleatorias) y explotación (mejor acción conocida).
Entrenamiento: 200,000 episodios, suficiente para convergencia en este entorno pequeño.

2.2. Modificaciones al Entorno

Se implementó un wrapper (WrapperRecompensaNegativa) para enriquecer las recompensas:

- Penalización por paso: -0.01 (incentiva caminos cortos).
- Penalización por agujero: -0.3 (penaliza fallos más que el 0 original).

Esto mejora el aprendizaje al proporcionar gradientes de recompensa más informativos, evitando que el agente "vague" indefinidamente.

3. Implementación del Código

El repositorio se ubica en :
https://github.com/Agustinlopezf/CEIA-AprendizajeRefuerzoI

Los puntos más importantes son:

Wrapper para modificar recompensas:

```
class WrapperRecompensaNegativa(gym.Wrapper):
    """
    Clase wrapper para modificar las recompensas en el entorno.
    Agrega penalizaciones por pasos y por caer en agujeros.
    """
    def __init__(self, env, step_penalty=-0.005, hole_penalty=-0.3):
        super().__init__(env)
        self.step_penalty = step_penalty  # Penalización por cada paso
        self.hole_penalty = hole_penalty  # Penalización por caer en un agujero

    def step(self, action):
        """
        Sobrescribe el método step para aplicar las penalizaciones personalizadas.
        """
        next_state, reward, terminated, truncated, info = self.env.step(action)
        if terminated and reward == 0:  # Cayó en un agujero
            reward = self.hole_penalty
        elif not terminated and not truncated:  # Paso sin terminación
            reward = self.step_penalty
        return next_state, reward, terminated, truncated, info

# Crear entorno con el wrapper
env = gym.make("FrozenLake-v1", is_slippery=False)  # Entorno FrozenLake sin resbalones
env = WrapperRecompensaNegativa(env, step_penalty=-0.01, hole_penalty=-0.3)  # Aplicar wrapper con
penalizaciones
```

Elección de acción balanceando exploración con ε-greedy:

```
def choose_action(state, epsilon):
    """
    Elige una acción usando ε-greedy: explora con probabilidad epsilon, explota otherwise.
    """
    if random.random() < epsilon:
        return random.randint(0, env.action_space.n - 1)  # Acción aleatoria
    return np.argmax(Q[state])  # Mejor acción conocida
```

Actualización off-policy en base a mejor acción:

```
# Actualizar valor Q usando la fórmula de Q-learning
    Q[state][action] += alpha * (
       reward + gamma * Q[next_state][best_next_action] - Q[state][action]
    )
```

3.1. Parámetros y Métricas

- Episodios: 200,000.
- α: Decae de 0.2 a 0.01.
- ε: Decae de 1.0 a 0.01.
- Trackeo: Recompensas, éxitos, caídas, longitudes de episodios.
- Evaluación: Cada 10k episodios, tasa de éxito en 100 pruebas greedy.

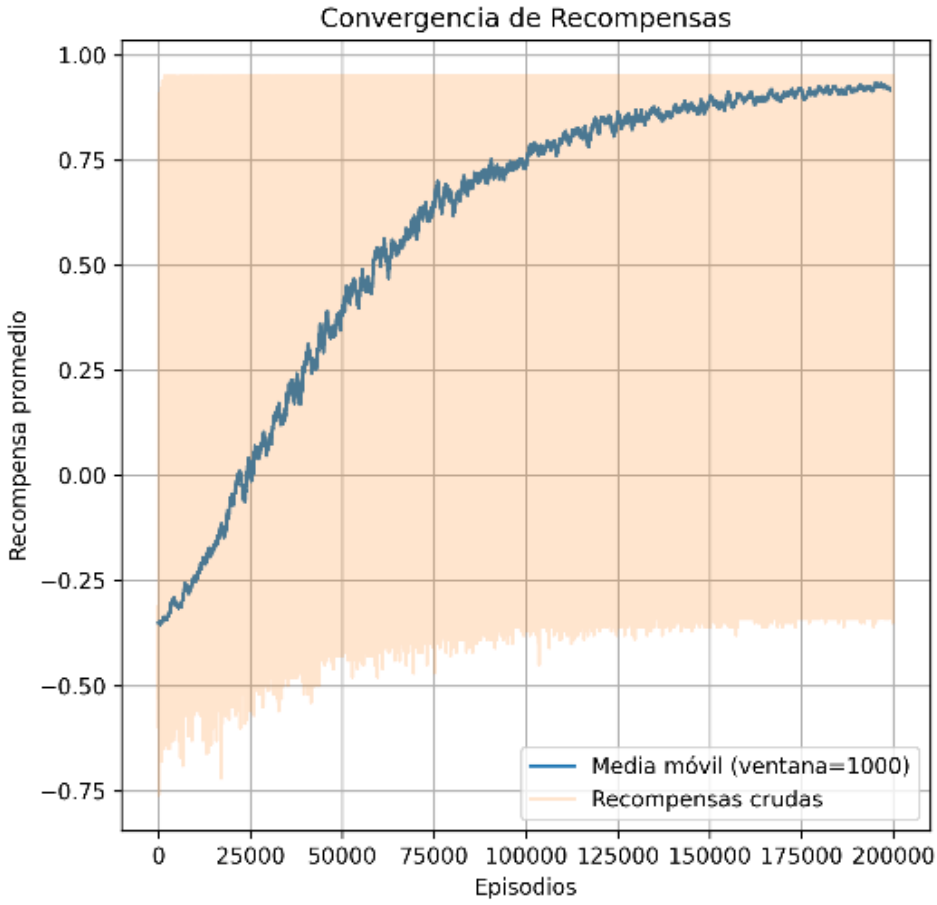
4. Inconvenientes Encontrados y Soluciones

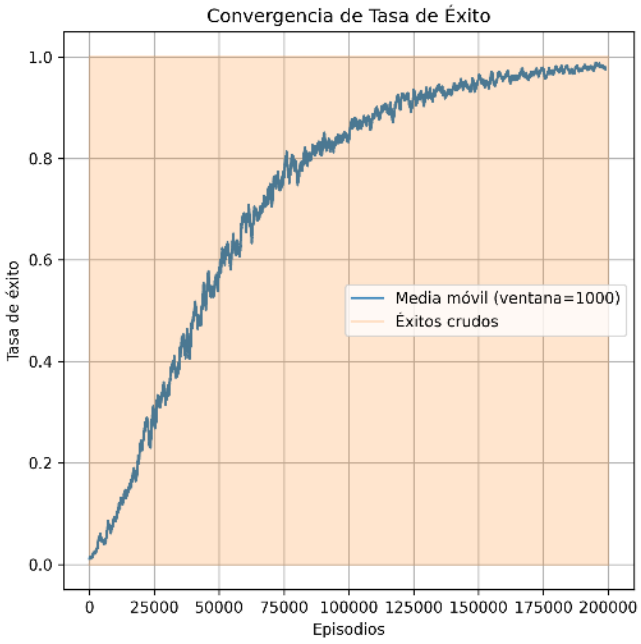
- Convergencia Lenta Inicial: Alta exploración (ε=1.0) causa muchas caídas aleatorias. Solución: Decaimiento exponencial de ε y α para transitar rápidamente a explotación.
- Recompensas Ruidosas: Variaciones altas en recompensas crudas dificultan visualización. Solución: Media móvil con ventana de 1,000 para suavizar curvas en gráficos.
- Detección de Éxito Imprecisa: Inicialmente basada en recompensa total >0, pero penalizaciones por pasos podían confundir. Solución: Variable booleana reached_goal activada solo al recibir +1.

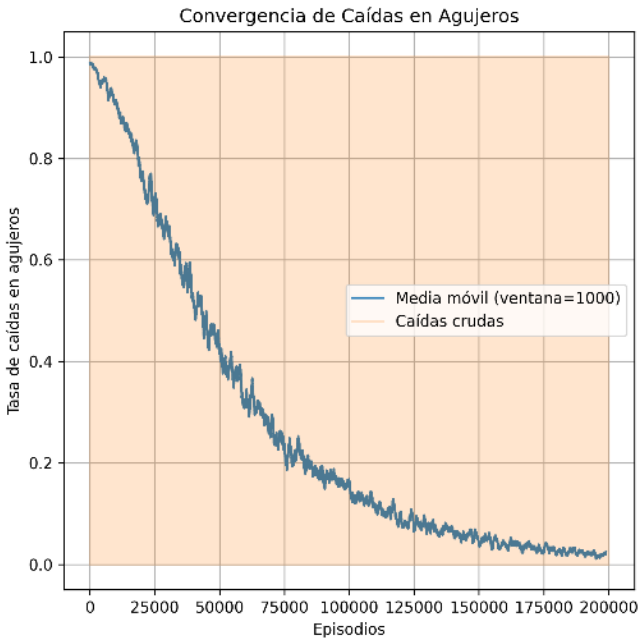
5. Resultados y análisis

5.1. Gráficos de Convergencia

Se incluye un gráfico de recompensas vs. Episodios, así como la tasa de éxitos y fracasos (caída en agujeros) cada 1000 episodios. Usamos media móvil para suavizar.


Convergencia de Recompensas
Recompensa promedio
Episodios
Media móvil (ventana=1000)
Recompensas crudas


Convergencia de Tasa de Éxito
Tasa de éxito
Episodios
Media móvil (ventana=1000)
Éxitos crudos


Convergencia de Caídas en Agujeros
Tasa de caídas en agujeros
Episodios
Media móvil (ventana=1000)
Caídas crudas

5.2. Análisis:

- Recompensas: Aumentan de negativas a aproximadamente 0.94 (1 menos penalizaciones por 6 pasos óptimos). Convergencia alrededor de 50k episodios.
- Tasa de Éxito: Asciende a aproximadamente 0.99, indicando política óptima.
- Caídas: Descienden a aproximadamente 0.01, mostrando aprendizaje de evitación.

Estos gráficos demuestran convergencia efectiva del algoritmo.

5.3. Métricas Finales

Ejemplo de salida (de una ejecución):

- Éxitos: 198,500/200,000 (99.25%).
- Tasa de éxito: 99.25%.
- Caídas: 1,200/200,000 (0.60%).
- Longitud promedio: 6.2 pasos (óptimo 6 en grilla 4x4).

Tasa de éxito en pruebas finales: 100%.

5.4. Ejecución de política aprendida

A continuación, se muestra la ejecución de la política aprendida por el agente, en formato ansi:

Leyenda:

- **S**: Inicio (start)
- **F**: Superficie congelada (frozen)
- **H**: Agujero (hole)
- **G**: Objetivo (goal)
- **X**: Posición del agente
- **Acciones**: 0 = Izquierda, 1 = Abajo, 2 = Derecha, 3 = Arriba

Configuración:

- Penalización por paso: -0.01
- Penalización por caer en un agujero: -0.3
- Recompensa por alcanzar el objetivo: 1.0

Secuencia de pasos:

Paso	Estado Actual	Acción	Recompensa	Próximo Estado	Tablero (Posición del Agente)
0	0	-	-	-	**S**FFF FHFH FFFH HFFG
1	0	Abajo (1)	-0.01	4	SFFF **X**HFH FFFH HFFG
2	4	Abajo (1)	-0.01	8	SFFF FHFH **X**FFH HFFG
3	8	Derecha (2)	-0.01	9	SFFF FHFH F**X**FH HFFG
4	9	Derecha (2)	-0.01	10	SFFF FHFH FF**X**H HFFG
5	10	Abajo (1)	-0.01	14	SFFF FHFH FFFH HF**X**G
6	14	Derecha (2)	1.0	15	SFFF FHFH FFFH HFF**X**

Se observa que llega correctamente al objetivo en una secuencia mínima de pasos.

6. Conclusiones

Se resolvió exitosamente el problema de FrozenLake usando Q-Learning, logrando alta tasa de éxito y convergencia rápida. Las modificaciones al entorno y decaimientos mejoraron el aprendizaje. También se observa que el agente aprendió una política óptima de pasos.
Este trabajo demuestra aplicación práctica de RL en entornos discretos.

Referencias

- Gymnasium Documentation. Disponible en: https://gymnasium.farama.org/
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction. MIT Press.
- Material del cursado: Aprendizaje por Refuerzo I, FIUBA.