Aprendizaje en entornos complejos

Autores

9 de marzo de 2025

Resumen

Este documento presenta un trabajo final sobre el aprendizaje por refuerzo en entornos complejos, abordando problemas de toma de decisiones secuenciales en presencia de incertidumbre. Se introduce el dilema exploración-explotación y se menciona el uso de bandidos contextuales y Procesos de Decisión de Markov (MDP). La solución a estos problemas, cuando la dinámica del entorno es desconocida, radica en el aprendizaje por refuerzo, donde un agente interactúa con el entorno observando estados, tomando acciones, recibiendo recompensas y actualizando su política para maximizar la recompensa acumulada.

El documento describe diversas categorías de algoritmos de aprendizaje por refuerzo, incluyendo métodos tabulares como Monte Carlo y Diferencias Temporales (TD) (SARSA y Q-Learning), los cuales son efectivos en entornos con espacios de estados y acciones reducidos. Para problemas con espacios más grandes, se exploran técnicas de aproximación como SARSA semi-gradiente y se menciona Deep Q-Learning. Se destaca el papel fundamental de las políticas de selección de acción, como la política ε-greedy, para equilibrar la exploración y la explotación.

Finalmente, se detalla la evaluación experimental de estos algoritmos en diferentes entornos de Gymnasium, tanto discretos (FrozenLake-v1, Taxi-v3) como continuos (Acrobot-v1, CartPole-v1, MountainCar). Los resultados muestran que los métodos basados en diferencias temporales (SARSA y Q-learning) tienden a ser más eficientes que Monte Carlo en entornos discretos, mientras que SARSA semi-gradiente demuestra efectividad en entornos continuos. Se subraya la importancia de la correcta elección de hiperparámetros para el desempeño de los algoritmos.

1. Introducción

El problema del bandido de k-brazos permite estudiar el dilema fundamental de la exploración y la explotación en entornos estacionarios. Sin embargo, en la mayoría de los casos, la toma de decisiones se desarrolla dentro de un contexto más complejo, en el cual el entorno puede ser parcialmente observable o estar regido por una dinámica subyacente desconocida. Una de las estrategias más comunes para abordar esta problemática es el uso de los llamados bandidos contextuales, mientras que una aproximación más generalizada asume que las decisiones siguen un Proceso de Decisión de Markov [2] con recompensas, potencialmente estocásticas.

Para resolver este tipo de problemas, la formulación matemática adecuada se basa en las ecuaciones de Bellman, que pueden resolverse mediante cálculo matricial o técnicas iterativas, siempre que se disponga de información completa sobre la dinámica del entorno, la matriz de transiciones y las recompensas asociadas. No obstante, en la práctica, este modelo es desconocido: no se dispone de la función de transición ni de un conocimiento previo sobre las recompensas o las mejores decisiones a tomar en cada estado.

Dado este escenario, la solución radica en aprovechar la experiencia del agente al interactuar con el entorno. De esta manera, emergen diversas técnicas de aprendizaje por refuerzo que siguen una secuencia de pasos estructurada:

1. El agente observa el estado actual del entorno.
2. Basándose en este estado, el agente toma una acción de acuerdo con una política si no se encuentra en un estado terminal.
3. Como consecuencia de la acción tomada, el entorno transita a un nuevo estado.
4. El agente recibe una recompensa, la cual se utilizará para evaluar la decisión tomada.
5. Se repite el proceso a partir del nuevo estado.

En este contexto, los algoritmos de aprendizaje por refuerzo pueden dividirse en varias categorías. Las técnicas de Monte Carlo [2] esperan a que el agente alcance un estado terminal antes de actualizar la política, optimizando las decisiones para maximizar la recompensa acumulativa futura. Por otro lado, los métodos basados en Diferencias Temporales (TD) [2] (TD) actualizan la política de manera incremental conforme el agente toma decisiones, sin necesidad de esperar al final del episodio. Entre ambos enfoques existen variantes híbridas que consideran solo un subconjunto de recompensas dentro del episodio para actualizar los valores de estado o acción.

El enfoque clásico para implementar estos métodos se basa en una representación tabular de las funciones de evaluación de estados y acciones. Sin embargo, en problemas con grandes espacios de estados y acciones, la representación tabular se vuelve inviable, dando lugar a técnicas de aproximación basadas en descenso del gradiente, tales como el uso de redes neuronales. Alternativamente, otro conjunto de técnicas optimiza directamente la política sin requerir la evaluación explícita de estados y acciones, lo que da lugar a los métodos Actor-Crítico [2].

Las tareas a realizar en esta parte de la práctica incluyen:

- Familiarización con el entorno Gymnasium.
- Estudio de algunos algoritmos básicos de aprendizaje por refuerzo.

A lo largo de esta práctica, se analizarán y compararán diver-

sas estrategias para abordar problemas de toma de decisiones secuenciales, evaluando su desempeño en diferentes escenarios de simulación.

2. Desarrollo

Como ya hemos comentado anteriormente, el problema abordado en esta práctica se enmarca dentro del aprendizaje por refuerzo, donde un agente toma decisiones secuenciales en un entorno incierto con el objetivo de maximizar su recompensa acumulada. En particular, se estudian técnicas tabulares y de aproximación para la resolución de problemas de decisión de Markov (MDP), utilizando el entorno Gymnasium para la experimentación.

El aprendizaje por refuerzo se ha abordado mediante múltiples enfoques. En el caso de problemas con espacios de estados y acciones reducidos, los métodos tabulares como Monte Carlo [2] y Diferencias Temporales (TD) [2] (TD) han demostrado ser efectivos. Por otro lado, para escenarios más complejos, se utilizan técnicas basadas en aproximaciones de funciones, como SARSA semi-gradiente y Deep Q-Learning, que permiten extender las capacidades del aprendizaje por refuerzo a dominios de alta dimensionalidad.

El desarrollo de esta práctica se fundamenta en tres pilares clave:

2.1. Gymnasium

Gymnasium [3] es una plataforma ampliamente utilizada en la comunidad de aprendizaje por refuerzo para la simulación de entornos estandarizados. Permite probar y comparar algoritmos de aprendizaje por refuerzo en escenarios controlados y reproducibles. Esta herramienta facilita la implementación de agentes de aprendizaje mediante interfaces homogéneas, proporcionando un banco de pruebas ideal para evaluar distintas estrategias de exploración y explotación.

2.2. Agentes de Aprendizaje por Refuerzo

Un agente de aprendizaje por refuerzo es una entidad que interactúa con un entorno mediante acciones, observando el estado actual y recibiendo recompensas en función de su comportamiento. Su objetivo es aprender una estrategia óptima de toma de decisiones que maximice la recompensa acumulada a lo largo del tiempo[2]. Existen distintos enfoques para implementar agentes de aprendizaje por refuerzo pero nos centraremos en:

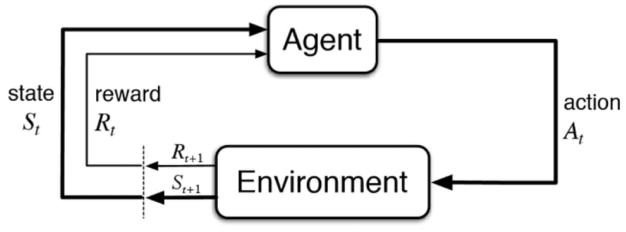

Figura 1: Interacción entre el agente y el entorno en un proceso de aprendizaje por refuerzo.

- **Métodos Tabulares**[2]:
  - **Monte Carlo [2]**: Métodos que estiman valores de acción basándose en episodios completos. En su variante *on-policy*, el agente aprende utilizando la misma política con la que genera sus acciones, mientras que en la versión *off-policy*, el aprendizaje se realiza a partir de experiencias generadas por una política diferente. Estos métodos son efectivos en entornos donde los episodios son bien definidos y terminan en un número finito de pasos.
  - **Diferencias Temporales (TD) [2] (TD)**: Métodos como SARSA y Q-Learning, que actualizan los valores de estado-acción en función de estimaciones parciales de la recompensa futura. SARSA es un método *on-policy*, donde la política seguida para generar los datos es la misma que la utilizada para actualizar los valores. En cambio, Q-Learning es un método *off-policy*, que busca optimizar la mejor política posible independientemente de la que sigue el agente durante la exploración. Estos algoritmos son ampliamente utilizados debido a su capacidad para aprender en tiempo real sin necesidad de esperar a que un episodio termine.
- **Control con Aproximaciones**[2]:
  - **SARSA semi-gradiente**: Una variante del algoritmo SARSA que emplea funciones de aproximación en lugar de tablas, permitiendo su aplicación en espacios de estados continuos. Mediante el uso de técnicas de regresión, como redes neuronales o funciones de base radial, el agente puede generalizar conocimiento a partir de un subconjunto de experiencias, en lugar de depender completamente de valores tabulares.
  - **Deep Q-Learning**: Un enfoque basado en redes neuronales profundas para aproximar la función de valores de acción. A través del uso de una *red Q profunda* (DQN), el agente es capaz de aprender representaciones abstractas de los estados, lo que le permite escalar a problemas con grandes espacios de estados, como los videojuegos o la robótica. Una característica fundamental de DQN es la utilización de un buffer de experiencia para almacenar transiciones previas y realizar actualizaciones de los valores de manera más estable.

2.3. Políticas en Aprendizaje por Refuerzo

[2] Las políticas juegan un papel fundamental en la convergencia de los agentes hacia soluciones óptimas dentro de un entorno dado. Una política define la estrategia de selección de acciones del agente en función del estado observado. Dependiendo de su formulación, pueden ser deterministas (asignan siempre la misma acción a un estado) o estocásticas (asignan probabilidades a diferentes acciones en un estado dado).

La calidad de una política determina la eficiencia del aprendizaje y la capacidad del agente para encontrar soluciones óptimas. Estrategias como el trade-off entre exploración y explotación, mediante métodos como ε-*greedy*, son cruciales para asegurar una exploración efectiva del espacio de estados y evitar que el agente quede atrapado en óptimos locales. En este enfoque, el agente selecciona la acción óptima con probabilidad $1-\varepsilon$, mientras que con probabilidad ε selecciona una acción aleatoria para garantizar la exploración del entorno. Este método permite equilibrar la explotación de la informa-

ción obtenida con la necesidad de descubrir nuevas estrategias más eficientes.

En métodos más avanzados, las políticas pueden ser optimizadas directamente mediante métodos de *Gradiente de Política*, los cuales ajustan los parámetros de la política de manera continua en función de la recompensa recibida.

3. Algoritmos

3.1. Estructura de Clases

Para la implementación de los algoritmos de aprendizaje por refuerzo en esta práctica, se ha utilizado una estructura modular basada en clases, lo que permite flexibilidad en la experimentación y facilita la mantenibilidad del código. La arquitectura del sistema está diseñada para ser extensible, permitiendo agregar nuevos métodos o variantes sin necesidad de modificar el núcleo del sistema.

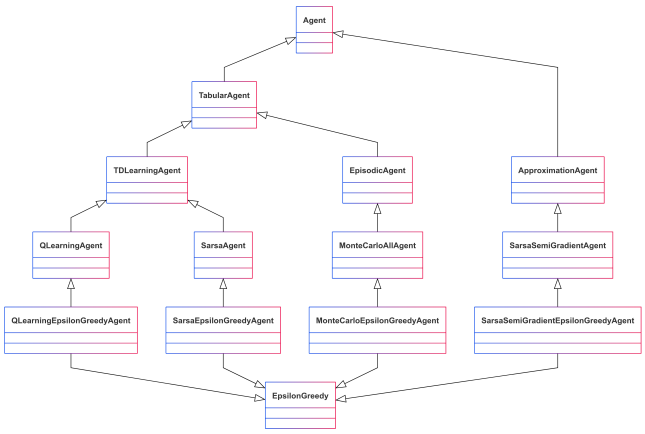

Figura 2: Estructura de clases utilizada en la implementación.

La implementación sigue un esquema jerárquico:

- **Agent**: Clase base abstracta que define la estructura común de todos los agentes de aprendizaje por refuerzo. Contiene métodos abstractos que deben ser implementados por las subclases, como `get_action()`, `update()` y `pi_star()`.
- **TabularAgent**: Clase intermedia que extiende a `Agent` y proporciona una implementación base para métodos tabulares. Maneja la tabla Q y el factor de descuento.
- **EpisodicAgent**: Hereda de `TabularAgent` y está diseñada para agentes que requieren la evaluación completa de episodios antes de actualizar su política, como el método Monte Carlo [2].
- **TDLearningAgent**: También hereda de `TabularAgent` y proporciona la base para algoritmos basados en Diferencias Temporales (TD) [2], como SARSA y Q-Learning.
- **MonteCarloAllAgent**: Implementa el método Monte Carlo [2] de todas las visitas, actualizando la tabla de valores Q al final de cada episodio.
- **MonteCarloEpsilonGreedyAgent**: Variante del método Monte Carlo [2] que incorpora una estrategia ε-greedy para equilibrar la exploración y la explotación.
- **QLearningAgent**: Implementa el algoritmo Q-Learning, permitiendo aprendizaje off-policy mediante actualizaciones iterativas de la tabla Q.
- **QLearningEpsilonGreedyAgent**: Extiende Q-Learning incorporando una estrategia ε-greedy para mejorar la exploración en entornos desconocidos.
- **SarsaAgent**: Implementa el algoritmo SARSA, ajustando la política mientras aprende la función de valor de acción.
- **SarsaEpsilonGreedyAgent**: Variante de SARSA que incorpora una estrategia ε-greedy para mejorar la exploración.
- **SarsaSemiGradientAgent**: Implementa el algoritmo SARSA Semi-Gradiente, utilizando funciones de aproximación en lugar de tablas de valores.
- **SarsaSemiGradientEpsilonGreedyAgent**: Extensión de SARSA Semi-Gradiente con exploración ε-greedy.
- **ApproximationAgent**: Clase base para métodos de aproximación, proporcionando estructura para agentes como SARSA Semi-Gradiente y Deep Q-Learning.
- **TileCodingFeatureExtractor**: Módulo auxiliar para la codificación de características en espacios continuos mediante tile coding.
- **EpsilonGreedy**: Mixin que proporciona funcionalidad ε-greedy a cualquier agente que lo necesite, permitiendo la selección de acciones con exploración decreciente.

Ademas de esas clases cabe destacar que se ha implementado un modulo, *Plotting*, para la visualización de las distintas gráficas que vamos a ver durante este estudio.

La elección de esta estructura modular permite la reutilización de código, facilita la implementación de nuevas variantes de algoritmos y mantiene una clara separación entre la lógica de aprendizaje y las estrategias de exploración. Además, al definir clases base como `Agent` y `TabularAgent`, se facilita la extensión de la funcionalidad a nuevos métodos de aprendizaje sin alterar el diseño fundamental del sistema. Si se desea se puede consultar en [1]. Ahora pasaremos a hacer una revisión del pseudocódigo de los algoritmos implementados y posteriormente los entornos donde han sido evaluados.

Un ejemplo de algoritmo es el **Algortihm 1**. No es necesario traducirlo al español.

3.2. Pseudocodigo e Implementación

En esta sección se presentan los pasos clave de los algoritmos implementados en la práctica. Se incluyen los métodos Monte Carlo [2], SARSA, Q-Learning y SARSA Semi-Gradiente, destacando sus diferencias en cuanto a exploración, actualización de valores y dependencia de la política. También se hará una revisión de las políticas implementadas asi como de los escenarios de Gymnasium escogidos.

3.3. Pseudocódigo de los Algoritmos

A continuación, se presentan los algoritmos utilizados en esta práctica en forma de pseudocódigo. Cada algoritmo se introduce con una breve descripción de su funcionamiento y sus aspectos clave.

Algorithm 1 Calculate $y = x^n$

Require: $n \geq 0 \vee x \neq 0$
Ensure: $y = x^n$
$y \leftarrow 1$
if $n < 0$ **then**
  $X \leftarrow 1/x$
  $N \leftarrow -n$
else
  $X \leftarrow x$
  $N \leftarrow n$
end if
while $N \neq 0$ **do**
  if N is even **then**
    $X \leftarrow X \times X$
    $N \leftarrow N/2$
  else[N is odd]
    $y \leftarrow y \times X$
    $N \leftarrow N - 1$
  end if
end while

3.3.1. Monte Carlo [2] (Todas las Visitas)

El método Monte Carlo [2] de todas las visitas actualiza los valores de acción-estado $Q(s, a)$ después de completar cada episodio, utilizando el retorno acumulado. Se promedia el valor de $Q(s, a)$ con todas las visitas registradas para cada par estado-acción.

Algorithm 2 Monte Carlo [2] - All Visits

Initialize $Q(s, a)$ arbitrarily
Initialize returns count $N(s, a) \leftarrow 0$
for each episode **do**
  Generate an episode: $(s_0, a_0, r_1, s_1, a_1, ..., s_T)$ following policy (ε-greedy in our case)
  Initialize $G \leftarrow 0$
  for each step t in reverse order **do**
    $G \leftarrow r_t + \gamma G$
    $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \frac{1}{N(s_t, a_t)}(G - Q(s_t, a_t))$
  end for
end for

3.3.2. SARSA

SARSA es un algoritmo basado en diferencias temporales (TD) que sigue una estrategia on-policy, es decir, la misma política utilizada para seleccionar acciones es la que se actualiza. A diferencia de Q-Learning, que usa la mejor acción en el siguiente estado, SARSA usa la acción efectivamente seleccionada por la política actual.

3.3.3. Q-Learning

Q-Learning es un algoritmo de diferencias temporales (TD) basado en una estrategia off-policy. A diferencia de SARSA, que sigue la política actual, Q-Learning actualiza la función de valor de acción considerando siempre la mejor acción posible en el siguiente estado, lo que lo hace más eficiente pero potencialmente menos estable.

Algorithm 3 SARSA Algorithm

Initialize $Q(s, a)$ arbitrarily
for each episode **do**
  Initialize state s
  Choose action a using policy (ε-greedy in our case)
  while episode not terminated **do**
    Take action a, observe reward r and next state s'
    Choose next action a' using policy (ε-greedy in our case)
    Update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$, $a \leftarrow a'$
  end while
end for

Algorithm 4 Q-Learning Algorithm

Initialize $Q(s, a)$ arbitrarily
for each episode **do**
  Initialize state s
  while episode not terminated **do**
    Choose action a using policy (ε-greedy in our case)
    Take action a, observe reward r and next state s'
    Update $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$
  end while
end for

3.3.4. SARSA Semi-Gradiente

SARSA Semi-Gradiente es una extensión de SARSA que usa funciones de aproximación en lugar de tablas de valores discretas. Este método permite manejar entornos con espacios de estado continuos utilizando técnicas como **tile coding** para representar estados mediante características.

Algorithm 5 SARSA Semi-Gradient Algorithm

Initialize weight vector w arbitrarily
for each episode **do**
  Initialize state s
  Compute feature vector $x(s, a)$
  Choose action a using ε-greedy policy
  while episode not terminated **do**
    Take action a, observe reward r and next state s'
    Choose next action a' using ε-greedy policy
    Compute feature vector $x(s', a')$
    Compute TD error: $\delta \leftarrow r + \gamma w^T x(s', a') - w^T x(s, a)$
    Update weights: $w \leftarrow w + \alpha \delta x(s, a)$
    $s \leftarrow s'$, $a \leftarrow a'$
  end while
end for

3.3.5. DeepQLearning

Para finalizar esta sección cabe destacar que se realizo una aproximación al algoritmo de DeepQLearning pero a día de la entrega nos fue imposible hacerlo funcionar correctamente pero la implementación seguida puede consultarse en el

repositorio.

3.4. Políticas de Selección de Acción

En aprendizaje por refuerzo, una política define la estrategia utilizada por un agente para seleccionar acciones en función del estado actual. A continuación, se describe la política ε-*greedy* que es la que ha sido usada a lo largo de la practica.

3.4.1. Política ε-Greedy

La política ε-greedy selecciona la mejor acción con probabilidad $1-\varepsilon$, pero permite la exploración eligiendo una acción aleatoria con probabilidad ε. Esto evita que el agente quede atrapado en soluciones subóptimas.

Algorithm 6 Epsilon-Greedy Policy

Input: State s, Action-value function $Q(s,a)$, Action space size nA
Choose a random number $p \sim U(0,1)$
if $p < \varepsilon$ **then**
    Select a random action $a \sim U(0, nA-1)$
else
    Select greedy action $a \leftarrow \arg\max_a Q(s,a)$
end if
Return action a

3.5. Entornos

3.5.1. Entornos Discretos para Métodos Tabulares

Estos entornos presentan un espacio de estados y acciones finito, lo que permite evaluar algoritmos tabulares como SARSA, Q-Learning y Monte Carlo [2].

FrozenLake-v1 (4x4 y 8x8):

Este entorno consiste en una cuadrícula de tamaño $N \times N$ donde el agente debe desplazarse desde una posición inicial hasta un objetivo, evitando caer en celdas peligrosas que representan agua. La dinámica del entorno es estocástica, ya que los movimientos pueden no ejecutarse exactamente como se planearon si se indica que sea un entorno 'resbaladizo'.

Espacio de estados: El espacio de estados S está definido por todas las celdas en la cuadrícula, es decir, $S = \{0, 1, 2, ..., N^2 - 1\}$.

Espacio de acciones: El agente puede moverse en cuatro direcciones: izquierda, derecha, arriba y abajo, representadas por el conjunto de acciones $A = \{\text{left}, \text{right}, \text{up}, \text{down}\}$.

Modelo de transición: La probabilidad de transición $P(s'|s,a)$ depende de la dinámica estocástica del entorno, donde la acción deseada tiene una probabilidad de ejecutarse correctamente, pero el agente también puede moverse en direcciones no intencionadas con cierta probabilidad.

Recompensa: El agente recibe una recompensa de $+1$ al alcanzar el objetivo y 0 en cualquier otro estado.

Taxi-v3:

En este entorno, el agente representa un taxi que debe recoger pasajeros en una cuadrícula y llevarlos a su destino. El problema es un caso de control secuencial con múltiples objetivos intermedios.

Espacio de estados: El estado se representa como una tupla (x, y, p, d) donde (x, y) es la ubicación del taxi en la cuadrícula, p es la posición del pasajero y d es la ubicación de destino.

Espacio de acciones: El conjunto de acciones es $A = \{\text{move up}, \text{move down}, \text{move left}, \text{move right}, \text{pick up}, \text{drop off}\}$.

Modelo de transición: El taxi se mueve determinísticamente en la dirección seleccionada, a menos que choque contra un obstáculo (bordes del mapa).

Recompensa: El taxi recibe una penalización de -1 por cada movimiento para incentivar soluciones eficientes. Intentar recoger o dejar pasajeros en lugares incorrectos da una penalización de -10. Llevar al pasajero al destino correcto otorga una recompensa de $+20$.

3.5.2. Entornos Continuos para Métodos de Aproximación

Estos entornos presentan un espacio de estados continuo, lo que requiere métodos basados en aproximación en lugar de representaciones tabulares.

Acrobot-v1:

Este entorno simula un sistema de dos segmentos conectados en serie (similar a un brazo robótico sin actuación en el primer segmento) donde el agente debe aprender a balancear el sistema para alcanzar una determinada altura.

Espacio de estados: El estado está representado por una tupla continua $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2)$ donde θ_1 y θ_2 son los ángulos de los segmentos respecto a la vertical y $\dot{\theta}_1, \dot{\theta}_2$ son sus respectivas velocidades angulares.

Espacio de acciones: El agente puede aplicar un torque discreto en el segundo segmento con tres opciones: $A = \{-1, 0, 1\}$.

Modelo de transición: La dinámica del sistema está gobernada por ecuaciones diferenciales del movimiento del doble péndulo, con integración numérica en cada paso de simulación.

Recompensa: Se otorga una recompensa de -1 en cada paso hasta que el agente logre levantar el segundo segmento por encima de una determinada altura, terminando el episodio.

CartPole-v1:

En este entorno, el agente debe equilibrar un poste montado sobre un carrito moviéndolo lateralmente para evitar que caiga. Es un problema de control clásico que requiere estrategias de aprendizaje para mantener la estabilidad del sistema.

Espacio de estados: El estado está representado por una tupla $(x, \dot{x}, \theta, \dot{\theta})$, donde x es la posición del carrito, $\dot{x}$ es su velocidad, θ es el ángulo del poste y $\dot{\theta}$ es la velocidad angular del poste.

Espacio de acciones: El agente puede aplicar una fuerza hacia la izquierda o la derecha: $A = \{\text{left}, \text{right}\}$.

Modelo de transición: La dinámica del sistema sigue ecuaciones de movimiento del péndulo invertido, resueltas mediante integración numérica en cada paso de simulación.

Recompensa: Se otorga una recompensa de +1 en cada paso donde el poste permanezca dentro de un rango de ángulos permitido, y el episodio termina si el poste cae o si el carrito se sale de los límites del entorno.

4. Evaluación/Experimentos

En esta sección se detallan los experimentos realizados para evaluar el desempeño de los algoritmos implementados en distintos escenarios. Se han llevado a cabo dos estudios diferenciados:

- **Estudio 1**: Evaluación de métodos tabulares (SARSA, Q-Learning y Monte Carlo [2]) en entornos discretos.
- **Estudio 2**: Evaluación de métodos de aproximación (SARSA Semi-Gradiente) en entornos con espacios de estado continuos.

Antes de comenzar con las configuraciones de los experimentos del estudio vamos a hacer una parada en lo que serian las evaluaciones que se han escogido para los experimentos.

4.1. Métricas de Evaluación

Para comparar el desempeño de los algoritmos en los diferentes entornos, se han seleccionado las siguientes métricas:

- **Recompensa por Episodio**: Esta métrica mide la suma total de recompensas obtenidas por el agente en cada episodio. Permite visualizar el proceso de aprendizaje y evaluar cómo el desempeño del agente mejora con el tiempo. Un aumento en la recompensa por episodio indica que el agente está aprendiendo políticas más eficientes. Además, ayuda a analizar las cotas de rendimiento alcanzadas y si el agente converge a una solución óptima o subóptima.
- **Longitud del Episodio**: Esta métrica mide la cantidad de pasos que el agente necesita para completar un episodio. En algunos problemas, esta información es clave para entender en qué fase del aprendizaje se encuentra el agente, ya que una reducción en la longitud de los episodios puede indicar que el agente ha encontrado estrategias más eficientes en problemas de caminos mas cortos. En otros casos, una mayor longitud puede sugerir que el agente está explorando antes de explotar estrategias óptimas. En resumen la longitud del episodio ofrece información complementaria que permite interpretar la estrategia del agente y su eficiencia en la tarea.

Además aunque no se ha empleado en el estudio, existe la posibilidad de mostrar las acciones tomadas por el agente de forma que un espectador tome la decisión de si converge a una solución o si no lo hace.

Ahora se presentan las configuraciones experimentales y combinaciones evaluadas en cada estudio.

4.2. Configuraciones Evaluadas en Métodos Tabulares

Para los métodos tabulares, se han considerado distintas combinaciones de entornos y algoritmos:

- **MonteCarloEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = \{0{,}3, 0{,}7\}, decay\_rate = 1$
- **SarsaEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = \{1, 0{,}7\}, \alpha = \{1, 0{,}7\}, decay\_rate = 0{,}9995, min\_value = 0{,}001$
- **QLearningEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = \{1, 0{,}7\}, \alpha = \{1, 0{,}7\}, decay\_rate = 0{,}9995, min\_value = 0{,}001$

Cabe destacar que se ha hecho un ultimo experimento comparando las mejores evaluaciones para cada uno de los algoritmos anteriores. Estas son:

-
- **MonteCarloEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = 0{,}3, decay\_rate = 1$
- **SarsaEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = 0{,}7, \alpha = 0{,}7, decay\_rate = 0{,}9995, min\_value = 0{,}001$
- **QLearningEpsilonGreedyAgent** para los entornos *FrozenLake-v1 4x4* ,*Taxi-v3* con los hiperparámetros $\varepsilon = 0{,}7, \alpha = 0{,}7, decay\_rate = 0{,}9995, min\_value = 0{,}001$

Además se ha establecido en todos los experimentos el $n\_episodes = 25000$.

4.3. Configuraciones Evaluadas en Métodos Aproximados

4.4. Resultados

A continuación se presentaran las gráficas de los diferentes resultados obtenidos por los algoritmos.

4.4.1. MonteCarlo

Se pueden observar en la Figura 3

4.4.2. SARSA

Se pueden observar en la Figura 4

4.4.3. QLearning

Se pueden observar en la Figura 5

4.4.4. Comparación entre modelos tabulares

Se pueden observar en la Figura 6

4.4.5. SARSA SemiGradiente

Se pueden observar en la Figura 7

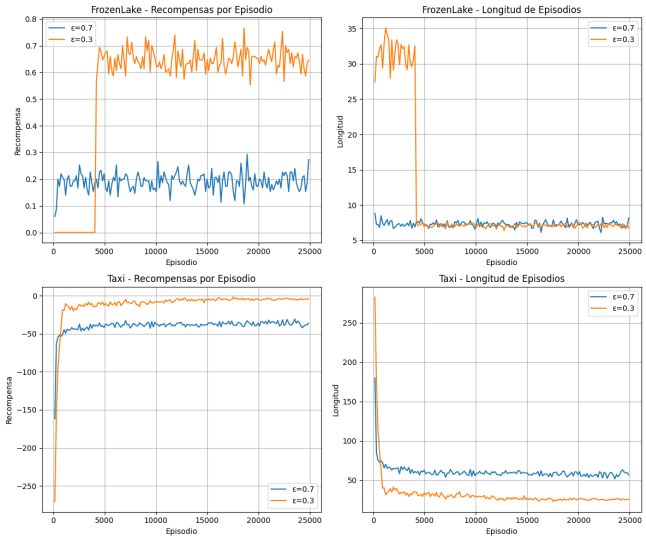

Figura 3: Comparativa de MonteCarlo con distintos parámetros.

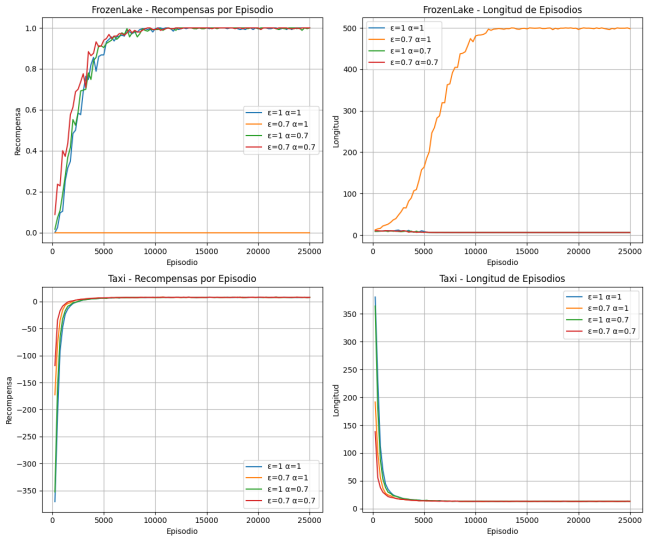

Figura 4: Comparativa de Sarsa con distintos parámetros.

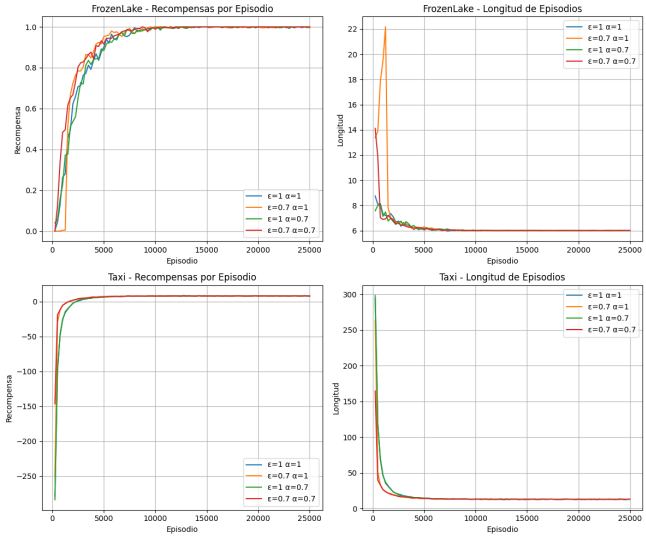

Figura 5: Comparativa de QLearning con distintos parámetros.

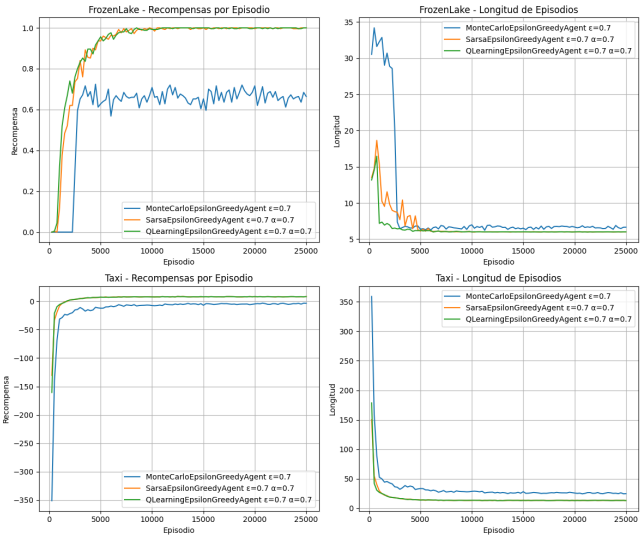

Figura 6: Comparativa de los distintos métodos tabulares.

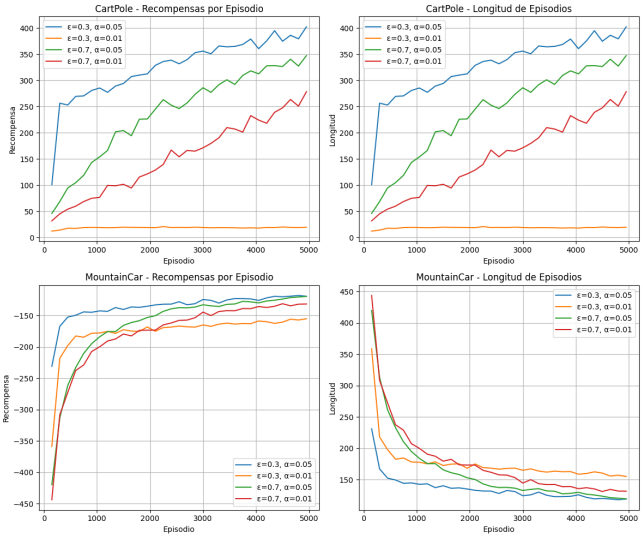

Figura 7: Comparativa de SARSA SemiGradiente con distintos parámetros

4.5. Análisis de los resultados

4.5.1. Monte Carlo

En la Figura 3 se presentan los resultados obtenidos al evaluar el método Monte Carlo en los entornos *FrozenLake* y *Taxi*, con valores de ε de 0,3 y 0,7.

En el entorno *FrozenLake*, se observa claramente que un valor de $\varepsilon = 0{,}3$ proporciona mejores recompensas promedio y mayor estabilidad a lo largo del tiempo. Inicialmente, el método presenta una rápida adaptación después de unos pocos episodios, manteniendo posteriormente un desempeño relativamente constante y superior al obtenido con $\varepsilon = 0{,}7$. Sin embargo, al considerar conjuntamente las recompensas y la longitud del episodio, podemos obtener información adicional que nos permite entender un poco mas el problema. Un episodio corto en *FrozenLake* no necesariamente indica éxito; podría significar que el agente ha caído rápidamente en un lago. Al analizar conjuntamente con la recompensa, es evidente que el agente con $\varepsilon = 0{,}3$ logra alcanzar consistentemente el

final, mientras que con $\varepsilon = 0{,}7$ cae frecuentemente en lagos, resultando en episodios cortos pero con bajas recompensas.

En el entorno *Taxi*, se repite una tendencia similar. La recompensa promedio obtenida con $\varepsilon = 0{,}3$ es significativamente mayor, mostrando una capacidad efectiva del agente para aprender una política que optimiza sus decisiones. Asimismo, la longitud promedio de los episodios es un indicador clave del desempeño, y en este caso, también refleja claramente un mejor funcionamiento con $\varepsilon = 0{,}3$, ya que se observa una considerable disminución en el número de pasos necesarios para completar la tarea. Esto implica que el agente es más eficiente en aprender y ejecutar una estrategia óptima, en comparación con el valor más exploratorio de $\varepsilon = 0{,}7$.

Estos resultados resaltan la importancia del balance adecuado entre exploración y explotación, siendo evidente que un menor valor de ε favorece una explotación más rápida de las estrategias aprendidas, mejorando así tanto la efectividad como la eficiencia general del agente.

4.5.2. SARSA

En la Figura 4 se presentan los resultados obtenidos al evaluar el método SARSA en los entornos *FrozenLake* y *Taxi*, utilizando distintas combinaciones de ε y α descritas anteriormente..

En el entorno *FrozenLake*, casi todas las combinaciones convergen rápidamente hacia recompensas cercanas al máximo posible.Caben destacar la combinaciones $\varepsilon = 0{,}7$ y $\alpha = 1$ por no converger ni conseguir finalizar el entorno y la combinación $\varepsilon = 0{,}7$ y $\alpha = 0{,}7$ por tener la mejor convergencia en ambos entornos.

En el entorno *Taxi*, la situación es más sencilla y transparente. La recompensa promedio obtenida converge rápidamente hacia el valor óptimo para todas las combinaciones de parámetros evaluadas. En este entorno, la longitud de los episodios es directamente indicativa del rendimiento del agente, dado que cada movimiento es penalizado con una recompensa negativa (-1). Por ello, reducir la longitud del episodio implica minimizar penalizaciones acumuladas y optimizar el desempeño. Como se observa claramente en la figura, todas las configuraciones muestran una rápida disminución en la longitud de los episodios hasta estabilizarse en valores mínimos, indicando así un aprendizaje eficiente del agente SARSA.

4.5.3. Método Q-learning

En la Figura 5 se presentan los resultados obtenidos al evaluar el método Q-learning en los entornos *FrozenLake* y *Taxi*, usando combinaciones de ε y α descritas anteriormente.

En el entorno *FrozenLake*, se observa que todas las configuraciones convergen rápidamente hacia recompensas cercanas al valor óptimo, indicando que Q-learning es eficiente y robusto ante diferentes configuraciones. En cuanto a la longitud de episodios, se puede apreciar cómo rápidamente disminuyen hasta estabilizarse en valores bajos, indicando que el agente aprende estrategias eficaces rápidamente y evita caer en bucles repetitivos. Aunque cabe destacar que la combinación $\varepsilon = 0{,}7$ y $\alpha = 1$ tiene problemas al inicio para converger respecto a las demás.

En el entorno *Taxi*, los resultados son igualmente positivos. El método Q-learning consigue rápidamente alcanzar recompensas óptimas en todas las combinaciones probadas. Además, la longitud de los episodios disminuye significativamente, indicando una rápida y efectiva convergencia a una política óptima.

4.5.4. Comparación entre modelos tabulares

En la Figura 6 se presenta la comparación de los métodos Monte Carlo, SARSA y Q-learning, aplicados a los entornos *FrozenLake* y *Taxi*, utilizando parámetros consistentes ($\varepsilon = 0{,}7$ y $\alpha = 0{,}7$ para SARSA y Q-learning).

En el entorno *FrozenLake*, se observa una clara diferencia en desempeño entre el método Monte Carlo y los métodos basados en diferencias temporales (SARSA y Q-learning). Monte Carlo muestra recompensas considerablemente más bajas y una mayor longitud promedio de los episodios, indicando dificultades significativas para converger hacia una política óptima. Esto podría ser resultado del aprendizaje que depende exclusivamente de la recompensa obtenida al final del episodio, siendo menos efectivo cuando los episodios concluyen frecuentemente con fallos tempranos (caídas en lagos). Por el contrario, tanto SARSA como Q-learning convergen rápidamente hacia recompensas óptimas y estabilizan la longitud de los episodios en valores mínimos, reflejando políticas aprendidas eficaces.

En el entorno *Taxi*, los tres métodos muestran un desempeño relativamente similar en términos de recompensas promedio, acercándose rápidamente al óptimo tras pocos episodios. No obstante, al analizar la longitud promedio de los episodios, se identifica que nuevamente los métodos basados en diferencias temporales, especialmente Q-learning, logran una convergencia más rápida y eficiente. Monte Carlo, aunque obtiene recompensas comparables, presenta una longitud de episodios algo mayor, indicando una eficiencia ligeramente menor al requerir más acciones para completar la tarea.

Estos resultados destacan la ventaja de los métodos basados en diferencias temporales, particularmente Q-learning, debido a su capacidad para actualizar el conocimiento del agente continuamente tras cada paso, facilitando una convergencia más rápida y efectiva hacia políticas óptimas en ambos entornos estudiados.

4.5.5. Sarsa gradiente descendiente

4.5.6. Método SARSA Semigradiente

En la Figura 7 se muestran los resultados obtenidos utilizando el método SARSA Semigradiente en los entornos *CartPole* y *MountainCar*, probando varias combinaciones de parámetros ε y tasas de aprendizaje (α).

En el entorno *CartPole*, se observa claramente que configuraciones con menor valor de ε (0.3) obtienen recompensas significativamente superiores en comparación con valores mayores de exploración ($\varepsilon = 0{,}7$). Además, dentro de estas configuraciones, un valor mayor de α (0.05) muestra una curva de

aprendizaje más rápida y consistente en términos de recompensa y longitud de los episodios. La longitud de los episodios aumenta junto con la recompensa, reflejando que el agente logra mantenerse más tiempo en equilibrio, cumpliendo así exitosamente con el objetivo del entorno.

En contraste, en el entorno *MountainCar*, todas las configuraciones presentan una mejora progresiva, aunque nuevamente se destacan los valores menores de ε y mayores de α, alcanzando mejores recompensas y episodios más cortos. En este entorno, reducir la longitud del episodio implica una mejora clara, ya que refleja la capacidad del agente para resolver la tarea en menos pasos. Es destacable cómo la combinación de menor exploración ($\varepsilon = 0{,}3$) y una tasa de aprendizaje relativamente alta ($\alpha = 0{,}05$) resulta especialmente efectiva para acelerar la convergencia del aprendizaje y mejorar la eficiencia del agente.

5. Conclusión

A lo largo de este trabajo se ha evaluado y comparado el desempeño de distintos algoritmos de aprendizaje por refuerzo, incluyendo métodos tabulares (Monte Carlo, SARSA y Q-learning) y métodos basados en aproximaciones (SARSA Semigradiente), en varios entornos de simulación.

De manera general, se ha podido observar que los métodos basados en diferencias temporales (SARSA y Q-learning) muestran una clara ventaja en eficiencia y rapidez de aprendizaje respecto al método Monte Carlo. Este último método, aunque sencillo y efectivo en contextos con episodios bien definidos, presenta limitaciones significativas cuando los episodios pueden finalizar prematuramente, ya que su aprendizaje depende exclusivamente del resultado final del episodio.

Dentro de los métodos tabulares, Q-learning mostró un desempeño particularmente robusto y eficiente, logrando una convergencia más rápida hacia políticas óptimas en comparación con SARSA, especialmente en entornos que requieren equilibrio entre exploración y explotación.

Por otra parte, al evaluar entornos más complejos con espacios de estados continuos, como CartPole y MountainCar, el método SARSA Semigradiente ha demostrado ser altamente efectivo. La capacidad de este método para generalizar mediante funciones de aproximación permitió que el agente se adaptase rápidamente a tareas complejas, destacando especialmente la importancia crítica del balance entre exploración (ε) y la tasa de aprendizaje (α).

Es importante mencionar que los métodos tabulares no son aplicables en entornos como CartPole y MountainCar debido a que estos poseen espacios de estado continuos, lo que imposibilita el uso de tablas para almacenar valores de cada estado-acción. Esto se debe a que el número de combinaciones de estados se vuelve infinitamente grande, haciendo impracticable mantener una representación explícita y completa en memoria, por lo que es necesario recurrir a métodos basados en funciones de aproximación.

Las limitaciones del estudio incluyen la dependencia del desempeño de los algoritmos en la correcta elección de hiperparámetros. Futuros trabajos podrían avanzar en la implementación de estrategias adaptativas para la selección automática de estos parámetros en funcion del entorno que se encuentre, así como la evaluación en entornos más desafiantes o la implementación de algoritmos avanzados como Deep Q-Learning.

En resumen, este trabajo subraya la importancia de seleccionar adecuadamente tanto el método como los hiperparámetros en función del tipo de entorno y del problema específico abordado.

Referencias

[1] *Repositorio de código: Aprendizaje en entornos complejos*. GitHub. Recuperado de https://github.com/pedrojosefernandez1/RL_FCPSSL

[2] Richard S. Sutton and Andrew G. Barto. (2018). *Reinforcement Learning: An Introduction* (2ª ed.). MIT Press.

[3] *Gymnasium: A toolkit for developing and comparing reinforcement learning algorithms*. Recuperado de https://www.farama.org/