Estudio en aprendizaje en entornos complejos

Esteban Becerra, Carlos Cruzado, Anastasiya Ruzhytska
estebanbecerraf@um.es, carlos.cruzadoe1@um.es, anastasiya.r.r@um.es

9 de marzo de 2025

Resumen

En este trabajo se explora el aprendizaje por refuerzo en entornos complejos mediante el uso de Gymnasium, enfocándose en la comparación de distintos algoritmos. Se investigan métodos tabulares como Monte Carlo, SARSA y Q-Learning, así como técnicas más avanzadas basadas en aproximaciones de funciones, como SARSA semigradiente y Deep Q-Learning.

Dado que en muchos casos el modelo del entorno es desconocido, los agentes aprenden a tomar decisiones a través de la experiencia, siguiendo un proceso iterativo basado en las ecuaciones de Bellman. Se implementan y analizan distintos enfoques para evaluar su desempeño en entornos de distinta complejidad, destacando la importancia de los métodos avanzados cuando el espacio de estados es muy grande. Finalmente, se comparan los resultados obtenidos a través de visualizaciones, permitiendo identificar las fortalezas y limitaciones de cada técnica.

1. Introducción

El aprendizaje por refuerzo (RL) se ha consolidado como una de las áreas más prometedoras dentro de la inteligencia artificial, permitiendo a los agentes aprender a tomar decisiones óptimas mediante la interacción con entornos complejos y, a menudo, inciertos. Mientras que en problemas estacionarios –como el clásico problema del bandido de k-brazos– el entorno permanece inmutable, en escenarios reales la situación se complica significativamente debido a la existencia de estados dinámicos y a la presencia de transiciones no determinísticas. Esto requiere la adopción de estrategias que vayan más allá de los simples esquemas de exploración-explotación, abordándose mediante modelos de procesos de decisión de Markov (MDP) en los que ni la política óptima ni las recompensas se conocen a priori.

En este contexto, se han desarrollado diversas técnicas que permiten abordar la problemática desde distintos enfoques:

- Métodos basados en episodios: Las técnicas de Monte Carlo, tanto en su variante on-policy como off-policy, actualizan la política una vez finalizado el episodio, aprovechando la totalidad de las recompensas acumuladas para retroalimentar la acción tomada.
- Métodos basados en diferencias temporales: Algoritmos como SARSA y Q-Learning realizan actualizaciones de manera incremental en cada paso, sin necesidad de esperar a la finalización del episodio.
- Métodos con función aproximadora: Se ha optado por el uso de redes neuronales (por ejemplo, mediante Deep Q-Learning) y Sarsa Semigradiente que permiten generalizar el aprendizaje a partir de muestras limitadas.

La práctica que se presenta en este informe se centra en la implementación y análisis comparativo de estos algoritmos en entornos complejos, utilizando la plataforma Gymnasium para simular escenarios realistas. Los notebooks desarrollados para este estudio (véase, por ejemplo, los contenidos de notebook2, estudio_Taxi y estudio_FrozenLake) han permitido la experimentación con distintos algoritmos, evidenciando cómo la evolución de métricas tales como la recompensa acumulada por episodio y la evolucion de la longitud de los episodios puede interpretarse para evaluar la efectividad de cada método. Asimismo, se ha prestado especial atención a la reproducibilidad de los experimentos, implementando estrategias para fijar semillas de aleatoriedad y garantizar resultados consistentes, tal y como se recomienda en las directrices del proyecto.

La motivación principal de este trabajo radica en la necesidad de comprender y analizar la evolución de los algoritmos de aprendizaje en entornos donde el modelo subyacente es desconocido y las recompensas pueden variar de forma estocástica. Al estudiar tanto métodos tradicionales basados en técnicas tabulares como aquellos que utilizan aproximadores de funciones mediante redes neuronales, se pretende no solo evaluar el rendimiento en términos cuantitativos, sino también obtener una perspectiva cualitativa sobre cómo cada técnica se adapta a la complejidad del entorno.

Objetivos del informe:

- Realizar un análisis detallado de diferentes técnicas de aprendizaje por refuerzo aplicadas a entornos complejos.
- Implementar y comparar algoritmos basados en métodos Monte Carlo, diferencias temporales y aproximaciones con redes neuronales, utilizando como entorno de prueba Gymnasium.
- Evaluar la evolución del aprendizaje mediante la comparación de métricas clave y establecer qué métodos resultan más adecuados según la naturaleza del entorno y la configuración de los experimentos.

2. Fundamentos teóricos

2.1. Definición Matemática

2.1.1. Monte Carlo (On-Policy y Off-Policy)

- **On-Policy (Todas las visitas)**
  - Se actualiza el valor de cada estado s en cada visita dentro de un episodio.

- La actualización se realiza como:

$$V(s) \leftarrow V(s) + \frac{1}{N(s)} \left(G_t - V(s)\right)$$

- G_t representa el retorno desde el instante t hasta el final del episodio.
- $N(s)$ es el número total de veces que el estado s ha sido visitado en todos los episodios.
- Con suficientes repeticiones ($N(s) \to \infty$), la estimación converge a:

$$v_\pi(s) = E_\pi[G_t \mid S_t = s]$$

- **Off-Policy**
  - Se usa ponderación por importancia para evaluar una política objetivo π con datos de una política de comportamiento b.
  - La estimación del valor de un estado se calcula como:

$$V(s) = \frac{\sum_{t \in T(s)} \rho_{t:T(t)-1} G_t}{\sum_{t \in \mathcal{T}(s)} \rho_{t:T(t)-1}}$$

  - $\rho_{t:T(t)-1}$ es la razón de probabilidades entre la política objetivo y la de comportamiento.
  - Permite aprender una política óptima π mientras se sigue una política diferente b.

2.1.2. Diferencias Temporales (SARSA y Q-Learning)

- **SARSA (On-Policy TD)**
  El algoritmo SARSA (State-Action-Reward-State-Action) es un método de control basado en Diferencias Temporales (TD) que aprende la función valor-acción $Q(s, a)$ siguiendo directamente una política específica π. La actualización toma la forma:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \quad (1)$$

  donde:
  - α es la tasa de aprendizaje ($0 < \alpha \leq 1$).
  - γ es el factor de descuento, que determina la importancia de recompensas futuras.
  - $Q(S_t, A_t)$ es el valor estimado del par estado-acción actual.
  - $Q(S_{t+1}, A_{t+1})$ es el valor estimado del próximo par estado-acción, siguiendo la política actual π.
- **Q-Learning (Off-Policy)**
  Q-Learning es un método Off-Policy de aprendizaje por refuerzo basado en Diferencias Temporales, que aprende la política óptima independientemente de la política utilizada para generar los datos. Su actualización sigue la ecuación:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \quad (2)$$

  - $\max_a Q(S_{t+1}, a)$ es el máximo valor estimado para el siguiente estado considerando todas las acciones posibles.

  Este método busca optimizar directamente la política óptima, sin depender de la política que se sigue durante el aprendizaje.

2.1.3. SARSA Semi-Gradiente

Se utiliza cuando se hace una aproximación con funciones parametrizadas $q(s, a, w) \approx q_\pi(s, a)$, actualizando parámetros mediante gradiente semi-estocástico:

$$w_{t+1} = w_t + \alpha \Big[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}, w_t) - q(S_t, A_t, w_t)\Big] \nabla q(S_t, A_t, w_t) \quad (3)$$

2.1.4. Deep Q-Learning (DQN)

Deep Q-Learning extiende Q-Learning utilizando redes neuronales profundas para aproximar la función $q(s, a)$:

- **Función Objetivo (Target):**

$$y = r + \gamma \max_{a'} q(s', a'; w_{target}) \quad (4)$$

- **Función de Pérdida (Loss):**

$$L(w) = \mathbb{E}\left[(y - q(s, a; w))^2\right] \quad (5)$$

- **Actualización por Descenso de Gradiente:**

$$w_{t+1} = w_t + \alpha \left[y - q(S_t, A_t, w_t)\right] \nabla q(S_t, A_t, w_t) \quad (6)$$

2.2. Explicacion de los entornos de Gymnasium

2.2.1. Taxi

El objetivo principal de este estudio es desarrollar y evaluar agentes de aprendizaje por refuerzo en el entorno Taxi-v3 de Gymnasium. En este entorno, el agente (representado por un taxi) debe aprender a:

- Navegar por una cuadrícula discretizada.
- Recoger a un pasajero en una ubicación designada.
- Transportar y dejar al pasajero en el destino correcto.

Para lograrlo, se implementan y comparan diferentes estrategias y algoritmos, que veremos a continuación. La meta es que, a través de la interacción con el entorno, el agente aprenda una política óptima que maximice la recompensa acumulada en cada episodio.

Además, en este entorno la función de recompensa está diseñada para guiar al agente hacia un comportamiento eficiente y seguro. Las recompensas se asignan de la siguiente manera:

- Recompensa Positiva: Se otorga un bonus (por ejemplo, +20) cuando el taxi deposita al pasajero en el destino correcto. Este incentivo refuerza la meta principal del agente.
- Coste por Movimiento: Cada acción (por ejemplo, moverse de una casilla a otra) genera un coste (por ejemplo, -1). Este coste penaliza la ineficiencia y motiva al agente a encontrar la ruta más corta.
- Penalizaciones por Acciones Incorrectas: e aplican penalizaciones (por ejemplo, -10) cuando se realizan acciones no permitidas o erróneas (Intentar dejar un pasajero en un destino no autorizado)

2.2.2. FrozenLake

En este entorno, el agente (representado por un personaje que debe cruzar un lago helado) debe aprender a:

- Navegar de manera segura por una cuadrícula de casillas resbaladizas.
- Evitar caer en agujeros que terminan el episodio prematuramente.
- Alcanzar el objetivo final (casilla de llegada) de manera eficiente.

El objetivo final es que, a través de la interacción con el entorno, el agente aprenda una política óptima que le permita maximizar la recompensa acumulada en cada episodio y llegar a la meta de forma segura y eficiente.

En este entorno, la función de recompensa está diseñada para fomentar un comportamiento eficiente y evitar caídas en los agujeros. Las recompensas se asignan de la siguiente manera:

- Recompensa Positiva: Se otorga una recompensa de $+1$ al llegar a la casilla objetivo con éxito.
- Coste por Movimiento: Todos los movimientos intermedios tienen recompensa 0, lo que implica que el agente debe minimizar los pasos innecesarios para maximizar la probabilidad de éxito.
- Penalización Indirecta: Si el agente cae en un agujero, el episodio termina sin recibir recompensa, lo que refuerza la necesidad de evitar estas casillas.

Dado que el entorno es estocástico (cuando es slippery=True, lo que simula el hielo resbaladizo), el agente debe aprender a manejar la incertidumbre y elegir estrategias que equilibren la exploración con la explotación para alcanzar la meta de manera consistente.

3. Algoritmos

En esta sección se exponen los algoritmos empleados en el estudio, los cuales se han implementado para abordar tanto la estimación de funciones de valor (predicción) como la optimización de políticas (control) en entornos donde el modelo del proceso de decisión de Markov es desconocido. Los métodos se dividen en tres grandes bloques:

1. Métodos basados en Diferencias Temporales (TD),
2. Métodos Monte Carlo (MC),
3. Control con Aproximación Funcional y Deep Q-Learning.

Cada uno de estos enfoques ha sido seleccionado en función de sus características inherentes a la actualización incremental y a la capacidad de generalización en espacios de estados de alta dimensión.

3.1. Métodos de Diferencias Temporales

Los métodos TD aprovechan la técnica de *bootstrapping* para actualizar las estimaciones de la función de valor de manera incremental, sin esperar la finalización del episodio. Esto resulta especialmente útil en entornos en línea y con procesos continuos.

TD(0) – Predicción Tabular

El algoritmo TD(0) actualiza la estimación $V(s)$ usando la siguiente regla:

$$V(s_t) \leftarrow V(s_t) + \alpha\,[R_{t+1} + \gamma\, V(s_{t+1}) - V(s_t)]$$

A continuación se presenta el pseudocódigo en el formato solicitado:

Algorithm 1 TD(0) Tabular

Inicializar $V(s)$ arbitrariamente para todo $s \in S$.
for cada episodio **do**
  Inicializar el estado S.
  while el episodio no termine **do**
    Seleccionar una acción A según la política actual.
    Ejecutar A, observar la recompensa R y el siguiente estado S'.
    Actualizar:

$$V(S) \leftarrow V(S) + \alpha\Big[R + \gamma\, V(S') - V(S)\Big]$$

    Establecer $S \leftarrow S'$.
  end while
end for

Este algoritmo es la base para extender los métodos a control, como se observa en los algoritmos SARSA y Q-Learning (véase).

SARSA – Control On-Policy

SARSA es un método on-policy que actualiza la función de acción $Q(s,a)$ utilizando la acción realmente tomada en el siguiente estado. La regla de actualización es:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\,[R_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

Algorithm 2 SARSA

Inicializar $Q(s,a)$ arbitrariamente para todos los pares (s,a).
for cada episodio **do**
  Inicializar S y seleccionar A (usando, por ejemplo, una política ε-greedy).
  while S no sea terminal **do**
    Ejecutar A, observar R y S'.
    Seleccionar A' de S' según la política.
    Actualizar:

$$Q(S,A) \leftarrow Q(S,A) + \alpha\Big[R + \gamma\, Q(S', A') - Q(S,A)\Big]$$

    Establecer $S \leftarrow S'$ y $A \leftarrow A'$.
  end while
end for

Q-Learning – Control Off-Policy

Q-Learning utiliza la máxima estimación de la función de acción en el siguiente estado, permitiendo la actualización off-policy:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[R_{t+1} + \gamma\, \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)\right]$$

Algorithm 3 Q-Learning

Inicializar $Q(s, a)$ arbitrariamente.
for cada episodio **do**
  Inicializar S.
  while S no sea terminal **do**
    Seleccionar A (usualmente explorando, por ejemplo con ε-greedy).
    Ejecutar A, observar R y S'.
    Actualizar:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \Big[R + \gamma \max_{a'} Q(S', a') - Q(S, A) \Big]$$

    Establecer $S \leftarrow S'$.
  end while
end for

3.2. Métodos Monte Carlo

Los métodos Monte Carlo estiman la función de valor mediante el promedio de los retornos obtenidos al finalizar episodios completos, sin utilizar bootstrapping. Se distinguen principalmente dos variantes: de primera visita y de todas las visitas. Aunque la que hemos usado es todas las vistas

Predicción Monte Carlo – Todas las Visitas

Para métodos on-policy, se estima $v_\pi(s)$ como el promedio de todos los retornos G_t asociados a cada visita al estado s en todos los episodios:

Algorithm 4 Predicción MC Todas Visitas (On-Policy)

Inicializar $V(s) = 0$, $N(s) = 0$ para todo $s \in S$.
for cada episodio **do**
  Generar un episodio: $S_0, A_0, R_1, \ldots, S_T$.
  Calcular retornos acumulados $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}$.
  for cada paso $t = 0, 1, \ldots, T-1$ **do**
    $s \leftarrow S_t$
    $N(s) \leftarrow N(s) + 1$
    Actualizar:

$$V(s) \leftarrow V(s) + \frac{1}{N(s)} \left(G_t - V(s) \right)$$

  end for
end for

Para métodos **off-policy**, se utiliza muestreo por importancia ponderado. Sea π la política objetivo y b la política de comportamiento. El valor $V(s)$ se estima como:

$$V(s) = \frac{\sum_{i=1}^{n} \rho_{t_i:T-1}^{(i)} G_t^{(i)}}{\sum_{i=1}^{n} \rho_{t_i:T-1}^{(i)}}$$

donde $\rho_{t:T-1} = \prod_{k=t}^{T-1} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$.

3.3. Control con Aproximación Funcional y Deep Q-Learning

Cuando el espacio de estados es muy grande o continuo, los métodos tabulares resultan ineficientes. En estos casos se recurre a la aproximación funcional para estimar la función de valor-acción $\hat{q}(s, a; w)$ mediante un conjunto de parámetros w.

Algorithm 5 Predicción MC Todas Visitas (Off-Policy)

Inicializar $V(s) = 0$, $C(s) = 0$ para todo $s \in S$.
for cada episodio **do**
  Generar episodio con b: $S_0, A_0, R_1, \ldots, S_T$.
  Calcular $\rho_{t:T-1}$ y $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}$.
  $W = 1$
  for cada $t = T-1, T-2, \ldots, 0$ **do**
    $s \leftarrow S_t$
    $G \leftarrow \gamma G + R_{t+1}$
    $C(s) \leftarrow C(s) + W$
    Actualizar:

$$V(s) \leftarrow V(s) + \frac{W}{C(s)} \left(G - V(s) \right)$$

    Actualizar peso:

$$W \leftarrow W \cdot \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

    if $W = 0$ **then**
      break
    end if
  end for
end for

Sarsa Semi-Gradiente (One-Step)

Este método utiliza una actualización basada en el gradiente estocástico para minimizar el error entre el objetivo y la estimación actual:

$$\delta = R_{t+1} + \gamma \, \hat{q}(S_{t+1}, A_{t+1}; w) - \hat{q}(S_t, A_t; w)$$

La actualización de los parámetros se realiza según:

$$w \leftarrow w + \alpha \, \delta \, \nabla_w \hat{q}(S_t, A_t; w)$$

Algorithm 6 Sarsa Semi-Gradiente

Inicializar el vector de pesos w de forma arbitraria.
for cada episodio **do**
  Inicializar el estado S y seleccionar A usando una política ε-greedy derivada de $\hat{q}(s, a; w)$.
  while el episodio no termine **do**
    Ejecutar A, observar R y S'.
    Seleccionar A' de S' según la política.
    Calcular el error TD:

$$\delta = R + \gamma \, \hat{q}(S', A'; w) - \hat{q}(S, A; w)$$

    Actualizar los pesos:

$$w \leftarrow w + \alpha \, \delta \, \nabla_w \hat{q}(S, A; w)$$

    Establecer $S \leftarrow S'$ y $A \leftarrow A'$.
  end while
end for

Deep Q-Learning (DQN)

DQN extiende Q-Learning al utilizar una red neuronal profunda para aproximar $\hat{q}(s, a; w)$. Entre las características clave del método se destacan:

- **Target Network:** Se utiliza una red neuronal separada (con parámetros w_{target}) para generar el objetivo:

$$y = R_{t+1} + \gamma \max_{a'} \hat{q}(S_{t+1}, a'; w_{\text{target}})$$

- **Experience Replay:** Las transiciones (s, a, R, s') se almacenan en un buffer y se muestrean de forma aleatoria para romper la correlación entre datos consecutivos y estabilizar el entrenamiento.

Algorithm 7 Deep Q-Learning

Inicializar la red Q con parámetros w y la red objetivo con $w_{\text{target}} = w$.
Inicializar un buffer de experiencia.
for cada episodio **do**
  Inicializar el estado S.
  while el episodio no termine **do**
    Seleccionar una acción A utilizando una política ε-greedy basada en $\hat{q}(S, a; w)$.
    Ejecutar A, observar R y S'.
    Almacenar la transición (S, A, R, S') en el buffer.
    Muestrear un minibatch de transiciones del buffer.
    for cada transición en el minibatch **do**
      Calcular el objetivo:

$$y = R + \gamma \max_{a'} \hat{q}(S', a'; w_{\text{target}})$$

    end for
    Actualizar los parámetros w minimizando el error cuadrático medio entre y y $\hat{q}(S, A; w)$.
    Cada cierto número de pasos, actualizar $w_{\text{target}} \leftarrow w$.
  end while
end for

La elección de estos métodos se fundamenta en la necesidad de disponer de algoritmos que no solo aprendan de forma incremental y en línea (como es el caso de los métodos TD y Monte Carlo) sino que, además, sean escalables a problemas de gran dimensionalidad. La combinación de Sarsa/Q-Learning semigradiente y DQN permite explorar diferentes aspectos del problema: desde la eficiencia en entornos con espacios discretos y moderadamente grandes, hasta la capacidad de generalización en dominios continuos o de alta complejidad .

4. Evaluación/Experimentos

4.1. Entornos de Gymnasium escogidos para el estudio

4.1.1. Taxi-v3 y FrozenLake-v1

- El entorno *Taxi* de Gymnasium es un clásico escenario de aprendizaje por refuerzo en el que un agente (taxi) debe aprender a recoger un pasajero en diferentes posiciones de una cuadrícula y transportarlo a su destino de la manera más eficiente posible. El agente recibe recompensas por completar correctamente la tarea y penalizaciones por movimientos incorrectos o por tardar demasiado tiempo en alcanzar el objetivo.
  El entorno está definido en una cuadrcula de 5×5 con cuatro ubicaciones especficas de recogida y destino. El agente dispone de un conjunto de seis acciones posibles:
  - ⋆ Moverse hacia arriba.
  - ⋆ Moverse hacia abajo.
  - ⋆ Moverse hacia la izquierda.
  - ⋆ Moverse hacia la derecha.
  - ⋆ Recoger un pasajero.
  - ⋆ Dejar al pasajero en su destino.

  El estado del entorno se representa mediante una codificación discreta que considera la posición del taxi, la ubicación del pasajero y el destino objetivo, lo que genera un espacio de estados finito y limitado.

- El entorno *FrozenLake* de Gymnasium representa un escenario basado en una cuadrícula que simula un lago congelado, donde un agente debe desplazarse desde un punto inicial hasta un objetivo evitando caer en agujeros ocultos.Existen dos versiones principales del entorno:
  - **Determinista (no resbaladiza)**: cada movimiento del agente ocurre exactamente según lo planeado.
  - **Estocástica (resbaladiza)**: la superficie del lago es resbaladiza, lo que introduce un comportamiento aleatorio en el desplazamiento del agente, generando incertidumbre en el aprendizaje.

  El agente recibe una recompensa únicamente al alcanzar la meta, lo que implica un desafío en el equilibrio entre *exploración* y *explotación*, dado que no existen recompensas intermedias a lo largo del trayecto.

Finalmente, para ambos estudios en ambos entornos se han utilizado las mismas métricas de evaluación:

- **Recompensa acumulada por episodio**: Mide la efectividad del agente en la tarea.
- **Longitud del episodio**: Cantidad de pasos requeridos para completar un episodio.

4.2. Métodos Tabulares y Control de aproximaciones en el entorno Taxi -v3

Para realizar una comparativa entre los diferentes métodos, se han usado los entornos introducidos anteriormente (Taxi y FrozenLake). Cabe destacar que en todos los estudios se ha aplicado la politica e- greedy y se ha ido variando el valor de epsilón y alpha para afinar los resultados del comportamiento.
Hiperparámetros utilizados para el estudio On Policy para el entorno Taxi-V3: Hiperparámetros utilizados

Tabla 1: Hiperparámetros utilizados en política On-Policy

Hiperparámetro	Valor
Número de episodios (N)	20,000
Factor de descuento (γ)	0.99
Probabilidad inicial de exploración (ϵ)	1.0
Tasa de decaimiento de ϵ	0.9999
Mínimo valor de ϵ	0.01

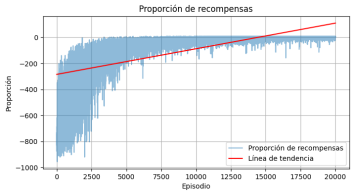

Figura 1: MonteCarlo On Policy - Proporción de recompensas

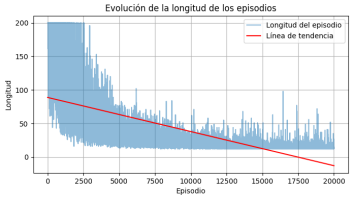

Figura 2: MonteCarlo On Policy - Longitud de episodios

para el estudio Off Policy para el entorno Taxi- V3:

Tabla 2: Hiperparámetros utilizados en política Off-Policy

Hiperparámetro	**Valor**
Número de episodios (N)	20,000
Factor de descuento (γ)	0.99
Probabilidad inicial de exploración (ϵ)	1.0
Tasa de decaimiento de ϵ	0.999
Mínimo valor de ϵ	0.002

4.2.1. Comparación: Métodos tabulares (MonteCarlo OnPolicy y OffPolicy) y Diferencias Temporales (SARSA y Q Learning)

Monte Carlo on Policy: este método evalúa y mejora la política que se utiliza para generar episodios.A medida que el agente ejecuta las acciones en el entorno, se registra cada acción y se acumulan los frames renderizados hasta alcanzar un estado terminal.

Observando la siguiente gráfica [Figura 1] vemos que mientras avanza el entrenamiento, la línea de tendencia (en rojo) muestra que las recompensas promedio van mejorando (se vuelven menos negativas), lo cual indica que el agente va aprendiendo a completar la tarea con menor coste y menos errores:

En comparación con la gráfica [Figura 2] de evolución de episodios podemos afirmar que la longitud de los episodios disminuye progresivamente, reflejando que el agente aprende rutas más eficientes para completar la tarea, mientras que la mejora en el **éxito promedio** de -531 a -97 confirma cuantitativamente su progreso a lo largo de las 20,000 iteraciones.

Monte Carlo off Policy: permite mejorar una política objetivo utilizando episodios generados por una política distinta, corrigiendo la diferencia entre ambas mediante muestreo por importancia. Esto facilita el aprendizaje sin depender exclusivamente de la política objetivo, optimizando la exploración en entornos donde seguirla directamente sería ineficiente.

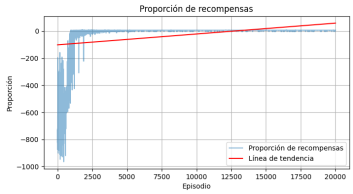

Figura 3: MonteCarlo Off Policy- Proporción de recompensas

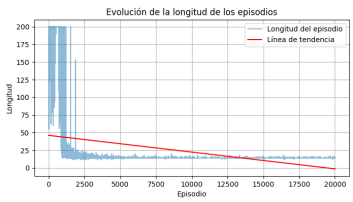

Figura 4: MonteCarlo Off Policy - Longitud de Episodios

A continuación se presentán las graficas de proporción de recompensas y longitud de episodios:

La gráfica [Figura 3] muestra una mejora progresiva en la proporción de recompensas, pasando de valores altamente negativos (˜1000) a estabilizarse cerca de 0, reflejando cómo el agente reduce penalizaciones y optimiza su desempeño, con un éxito promedio que mejora de -270 a -24.

La gráfica [Figura 4] evidencia una disminución significativa en la longitud de los episodios, reduciéndose de más de 200 a aproximadamente 20-30 pasos, lo que indica una optimización progresiva de la política del agente y una mejora en la eficiencia de su toma de decisiones a lo largo del entrenamiento.

Finalmente visualizaremos la comparativa de ambas gráficas:

La gráfica [Figura 5] de recompensas muestra que ambos enfoques comienzan con valores negativos y convergen progresivamente hacia 0, aunque Off-Policy presenta una curva de aprendizaje más rápida y estable.

La gráfica [Figura 6] muestra que Off-Policy reduce más rápido la longitud de los episodios, logrando rutas óptimas con mayor estabilidad, mientras que On-Policy presenta mayor variabilidad y una convergencia más lenta. Esto sugiere que Off-Policy ofrece un aprendizaje más eficiente en Taxi-v3.

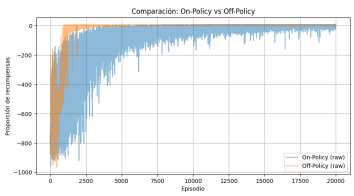

Figura 5: Comparativa de Monte Carlo en OnPolicy vs Off-Policy - Proporción de episodios

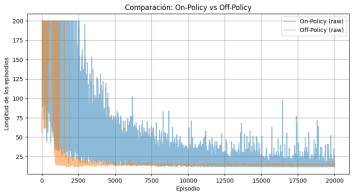

Figura 6: Comparación de MonteCarlo en OnPolicy y OffPolicy- Longitud de episodios

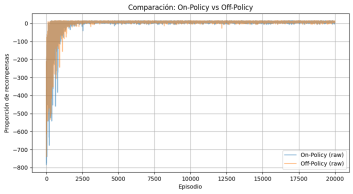

Figura 7: Comparación de proporción de recompensas en SARSA(onPolicy)Y Q-Learning(OffPolicy)

SARSA: es un método **On-Policy** basado en diferencias temporales que actualiza los valores Q(s,a)Q(s, a)Q(s,a) considerando la acción realmente tomada en el siguiente paso. Su enfoque conservador permite equilibrar **exploración y explotación**, adaptándose mejor a la política en uso y evitando estrategias demasiado arriesgadas.

Q-Learning:es un método **Off-Policy** que aprende la política óptima sin depender de la política usada para recopilar datos. En cada actualización, selecciona la mejor acción posible en el siguiente estado, lo que permite un aprendizaje más agresivo, aunque con mayor riesgo si la exploración no está bien regulada. **Comparación SARSA- Q Learning**

Q-Learning converge más rápido que **SARSA** debido a su enfoque **Off-Policy**, que actualiza valores basándose en la acción óptima estimada, mientras que **SARSA**, al ser **On-Policy**, progresa de manera más estable pero con una convergencia más lenta. Los métodos de **Diferencias Temporales (TD)**, como SARSA y Q-Learning, **aprenden más rápido que Monte Carlo**, ya que actualizan valores en cada paso mediante **bootstrapping**, mientras que **Monte Carlo requiere completar episodios**, lo que ralentiza la optimización de la política.[Figura 7 y Figura 8]

4.2.2. Comparación: Control de aproximaciones (SARSA semi gradiente y Deep Q Learning)

SARSA semi gradiente: método OnPolicy. SARSA Semi-Gradiente extiende SARSA incorporando la aproximación de funciones para estimar Q(s,a)Q(s,a) mediante modelos parametrizados, actualizando los parámetros mediante diferencias temporales sin propagar el gradiente al valor objetivo, lo que mejora la estabilidad y eficiencia en espacios de estado grandes o continuos.

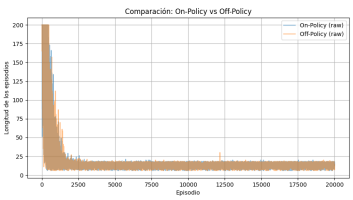

Figura 8: Comparación de longitud de episodios en SARSA(onPolicy)Y Q-Learning(OffPolicy

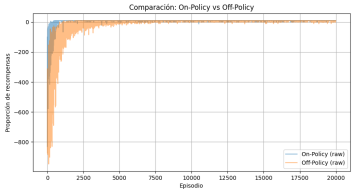

Figura 9: Comparación de proporción de recompensas en SARSA semi gradiente(onPolicy) y Deep Q learning(offPolicy)

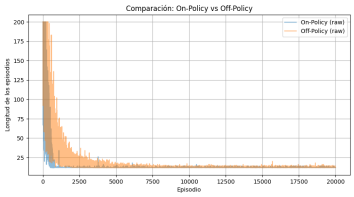

Figura 10: Comparación de longitud de episodios en Sarsa semigradiente (OnPolicy) y Deep Q learning (offPolicy)

Deep Q Learning: es un método **Off-Policy** que extiende **Q-Learning** utilizando redes neuronales profundas para aproximar Q(s,a)Q(s,a), permitiendo el aprendizaje en espacios de estado de alta dimensión. Para mejorar la estabilidad y la convergencia, emplea técnicas como **experience replay**, que reduce la correlación entre muestras, y **target networks**, que estabiliza la actualización de parámetros.

Las gráficas [Figura 9] muestran que **Deep Q-Learning (DQN) y SARSA Semigradiente** convergen hacia recompensas cercanas a 0, indicando una reducción progresiva de penalizaciones y una mejora en la estrategia del agente. Sin embargo, **DQN exhibe mayor volatilidad en las primeras etapas**, alcanzando su convergencia de forma más rápida, mientras que **SARSA Semigradiente mantiene una evolución más estable**. En cuanto a la **longitud de los episodios**, **DQN comienza con episodios más extensos y dispersos**, pero los reduce drásticamente conforme aprende, mientras que **SARSA Semigradiente sigue un patrón más controlado**, evitando variaciones extremas. A pesar de estas diferencias, **ambos métodos logran minimizar el número de pasos necesarios para completar la tarea**, evidenciando el aprendizaje de rutas más eficientes. [Figura 10]

4.3. Métodos Tabulares y Control de aproximaciones en el entorno FrozenLake v1

Hiperparámetros utilizados para el estudio On Policy para el entorno FrozenLake-V1:

Hiperparámetros utilizados para el estudio Off Policy para el entorno FrozenLake- V1:

Tabla 3: Hiperparámetros utilizados en On-Policy

Hiperparámetro	Valor
Número de episodios (N)	20,000
Factor de descuento (γ)	0.99
Probabilidad inicial de exploración (ϵ)	1.0
Tasa de decaimiento de ϵ	0.9999
Mínimo valor de ϵ	0.01

Tabla 4: Hiperparámetros utilizados en Monte Carlo Off-Policy

Hiperparámetro	Valor
Número de episodios (N)	20,000
Factor de descuento (γ)	0.99
Probabilidad inicial de exploración (ϵ)	1.0
Tasa de decaimiento de ϵ	0.9999
Mínimo valor de ϵ	0.002

4.3.1. Comparación: Métodos tabulares (MonteCarlo OnPolicy y OffPolicy) y Diferencias Temporales (SARSA y Q Learning)

Las gráficas de **proporción de recompensas** muestran que **Monte Carlo On-Policy y Off-Policy** convergen de manera similar en **FrozenLake-v1**, alcanzando una tasa de éxito cercana a **1**. A partir de los **5000 episodios**, ambas curvas se estabilizan y se superponen, indicando que el agente ha aprendido una estrategia óptima de forma consistente. La ausencia de diferencias significativas en las gráficas sugiere que, en este entorno, la elección entre **On-Policy y Off-Policy** no afecta significativamente el rendimiento del aprendizaje. Las gráficas de **longitud de episodios** [Figura 12] muestran que **Off-Policy** presenta episodios más largos y mayor variabilidad, sugiriendo un aprendizaje menos estable y eficiente que **On-Policy**. Aunque ambos métodos mejoran con el entrenamiento, **Off-Policy sigue explorando rutas más largas**, mientras que **On-Policy converge de manera más eficiente** en la optimización de la tarea.

A diferencía de los resultados obtenidos en los estudios realizados en SARSA y QLearning.

En las gráficas [Figura 13] podemos observar como se compara el rendimiento de **SARSA (On-Policy)** y **Q-Learning (Off-Policy)** en términos de la **proporción de recompensas por episodio**.

- **SARSA** sigue una estrategia más *conservadora y estable*, evitando cambios bruscos en la política, pero sin alcanzar siempre la máxima recompensa.

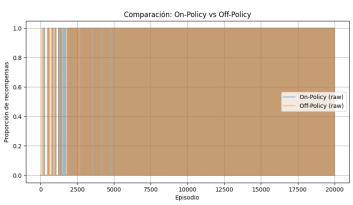

Figura 11: Comparacación de recompensas entre Monte Carlo On policy y Off Policy en el entorno FrozenLake v1

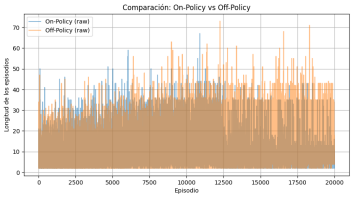

Figura 12: Comparación de longitud de episodios de Monte-Carlo on y off policy en el entorno FrozenLake v1

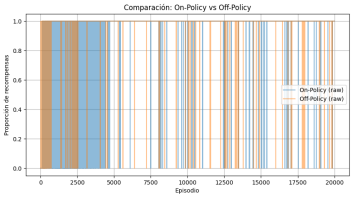

Figura 13: Comparación de proporción de recompensas por episodio de SARSA (onpolicy) y QLearning (offPolicy) en el entorno frozenLake

- **Q-Learning** adopta un enfoque más *exploratorio e intensivo*, encontrando soluciones más rápido, aunque con mayor variabilidad al inicio.

La Figura [Figura 14] la evolución de la longitud de los episodios a lo largo del entrenamiento. Se observa que **Q-Learning** tiende a generar episodios más cortos en promedio en comparación con **SARSA**, lo que indica una mayor eficiencia en la toma de decisiones.

Sin embargo, la gráfica también evidencia que **ambos métodos presentan dificultades para converger de manera estable**, reflejando un aprendizaje errático. La variabilidad en la longitud de los episodios sugiere que el agente no ha logrado identificar una *política óptima*, lo que podría requerir ajustes en los hiperparámetros o una mayor cantidad de entrenamiento.

En conclusión, ambos métodos presentan periodos de **alta variabilidad**, pero **Q-Learning logra recompensas de manera más consistente con el tiempo**. No obstante, los **métodos de Monte Carlo han mostrado un mejor desempeño general** en este experimento.

4.3.2. Comparación: Control de aproximaciones (SARSA semi gradiente y Deep Q Learning)

La Figura [Figura 15] compara el desempeño de **SARSA Semigradiente (On-Policy)** y **Deep Q-Learning (DQN,**

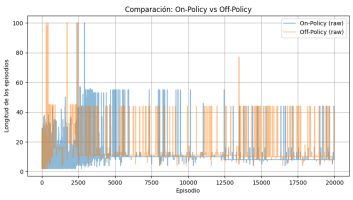

Figura 14: Comparación de longitud de episodios en Sarsa (onPolicy) y Q Learning (offPolicy)en el entorno Frozen Lake v1

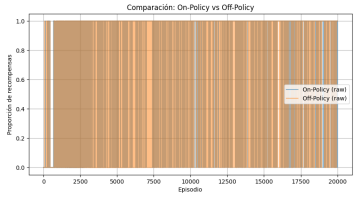

Figura 15: Comparación de proporción de recompensas entre SARSA semigradiente y DeepQLearning en el entorno frozen-Lake v1

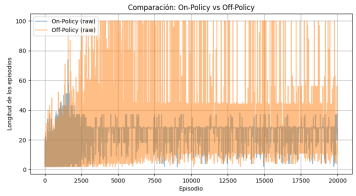

Figura 16: Comparación de Longitud de episodios en SARSA semigradiente (onpolicy) y DeepQ Learning (off Policy) en el entorno frozen Lake v1

Off-Policy) en términos de la **proporción de recompensas obtenidas** a lo largo del entrenamiento en el entorno `FrozenLake-v1`.

- **Superposición de curvas**: Las líneas de SARSA Semigradiente y DQN están mayormente superpuestas, lo que indica que ambos métodos logran resolver la tarea con una alta tasa de éxito.
- **Aprendizaje rápido**: Desde los primeros episodios ($\sim$2500), ambos algoritmos alcanzan una proporción de recompensas cercana a 1, lo que sugiere que aprenden rápidamente la política óptima y la mantienen estable durante la mayor parte del entrenamiento.
- **Diferencias mínimas**: En este entorno, las diferencias entre ambos métodos son poco significativas; sin embargo, en problemas más complejos, **DQN podría mostrar ventajas en términos de eficiencia y generalización**.

Ambos métodos logran un desempeño similar en este experimento. No obstante, en entornos **más dinámicos o con ruido**, SARSA Semigradiente podría ofrecer una mayor estabilidad, mientras que **DQN sería más beneficioso en escenarios con un espacio de estados más grande y con mayor cantidad de datos disponibles**.

En la Figura [Figura 16] muestra la comparación de la duración de los episodios entre **SARSA Semigradiente** y **Deep Q-Learning (DQN)**. Se observa que SARSA Semigradiente mantiene una duración promedio de episodios significativamente menor en comparación con DQN.

Aunque ambos métodos logran resolver la tarea con un desempeño similar en términos de éxito, SARSA Semigradiente aprende una **política más eficiente**, permitiéndole completar los episodios en menos pasos y optimizando su desempeño en el entorno.

5. Conclusiones

Para concluir analizemos la tabla comparativa [Tabla 5] y se ha de destacar los siguientes puntos:

- **Monte Carlo Off-Policy** es el método más eficiente en **FrozenLake-v1**, convergiendo rápidamente y con menos pasos.
- **En Taxi-v3, los métodos Off-Policy (Q-Learning y Monte Carlo Off)** destacan por reducir rápidamente la longitud de los episodios.
- **SARSA y Monte Carlo On-Policy** muestran un aprendizaje más progresivo, con SARSA siendo más conservador.
- **SARSA Semigradiente ofrece un punto intermedio** entre Q-Learning y SARSA, logrando mejoras tempranas.
- **Deep Q-Learning es potente pero innecesario en estos entornos**, debido a su alto costo computacional y la falta de recompensas intermedias en **FrozenLake-v1**.

Tabla 5: Comparación de métodos de aprendizaje por refuerzo en Taxi-v3 y FrozenLake-v1

Método	**Taxi-v3**	**FrozenLake-v1**
Monte Carlo On-Policy	Convergencia progresiva con mejoras estables, pero más lenta. Se adapta bien a entornos discretos con recompensas continuas.	Más eficiente que en Taxi-v3, con rápida convergencia gracias a la naturaleza discreta de las recompensas en FrozenLake.
Monte Carlo Off-Policy	Uno de los métodos más eficientes, con reducción rápida en la longitud de los episodios.	Método más eficiente en este entorno, logrando la política óptima con menos pasos y convergiendo rápidamente.
SARSA (On-Policy)	Mejora más lenta que los métodos off-policy, con convergencia estable pero conservadora.	Menos eficiente que Monte Carlo, mostrando dificultades para encontrar una política óptima.
Q-Learning (Off-Policy)	Excelente reducción en la longitud de los episodios, optimizando la política rápidamente.	Tiene dificultades debido a la falta de recompensas intermedias, sin superar a Monte Carlo en eficiencia.
SARSA Semigradiente	Punto intermedio entre SARSA y Q-Learning, logrando mejoras tempranas sin convergencia agresiva.	Buen desempeño, pero en entornos sin feedback intermedio, Monte Carlo sigue siendo superior.
Deep Q-Learning	Prometedor en problemas grandes, pero innecesario en Taxi-v3 debido a su alto costo computacional.	Dificultades para encontrar una política estable por la falta de recompensas intermedias, sin superar a Monte Carlo.