Aprendizaje por refuerzo en entornos complejos con *gymnasium*

Ana Gil Molina
José María García Ortiz
Levi Malest Villarreal

9 de marzo de 2025

Resumen

En este documento se presentará un estudio de desempeño para ciertos algoritmos avanzados del aprendizaje por refuerzo, sobre diferentes entornos (tanto discretos como continuos) de la librería *gymnasium*. Para el estudio hemos elegido *FrozenLake*, *CliffWalking* y *MountainCar* como entornos, y nuestro objetivo es conseguir resolver todos los problemas de los entornos con los algoritmos pertinentes. Finalmente, incorporamos un análisis de los experimentos y añadimos unas conclusiones que están respaldadas por los estudios (replicables) que pueden encontrar en [1].

1. Introducción

1.1. Descripción general del problema

Por lo general, contamos con un problema definido sobre un entorno que simula situaciones del mundo real o situaciones ficticias sujetas a ciertas leyes de la física, o multitud de eventos que podemos modelar. En cada entorno existe un sistema de recompensas mediante el cual nuestro agente debe aprender y mejorar su comportamiento de cara a conseguir el objetivo que se plantea en cada problema. Por ejemplo, escapar de un laberinto, subir una montaña u optimizar una ruta de un punto A a un punto B con obstáculos.

1.2. Motivación

El aprendizaje por refuerzo (RL) es un paradigma de aprendizaje automático en el que un agente aprende a tomar decisiones mediante la interacción con un entorno, optimizando una política basada en la retroalimentación recibida en forma de recompensas. Este enfoque ha demostrado ser altamente efectivo en una amplia gama de aplicaciones, incluyendo robótica, videojuegos, sistemas de recomendación y control de procesos industriales.

Para desarrollar, entrenar y evaluar agentes de RL de manera eficiente, es crucial contar con entornos estandarizados y fácilmente accesibles. En este contexto, la librería *Gymnasium* proporciona una colección de entornos de simulación diseñados para facilitar la experimentación y comparación de algoritmos de RL.

La elección de *Gymnasium* como plataforma de experimentación se debe a varias razones fundamentales:

- **Variedad de entornos:** Incluye problemas de control clásico, tareas de robótica, videojuegos y escenarios personalizados, lo que permite evaluar algoritmos en contextos diversos.
- **Estandarización de la API:** Los entornos siguen una interfaz homogénea, simplificando la implementación y prueba de diferentes algoritmos de RL.
- **Compatibilidad con bibliotecas de aprendizaje profundo:** Se integra fácilmente con herramientas como *PyTorch* y *TensorFlow*, lo que facilita la implementación de arquitecturas avanzadas de RL.
- **Flexibilidad y personalización:** Permite la creación de entornos personalizados y la modificación de los existentes, adaptándose a necesidades específicas de investigación.

El uso de *Gymnasium* permite establecer una metodología experimental rigurosa para analizar el comportamiento de distintos algoritmos de RL en escenarios desafiantes. En este trabajo, exploraremos diversas estrategias de exploración-explotación y optimización de políticas en múltiples entornos de *Gymnasium*, con el objetivo de mejorar el rendimiento de los agentes en tareas de toma de decisiones secuenciales.

1.3. Objetivos del trabajo

El objetivo de nuestro trabajo es familiarizarnos con el uso de gymnasium y resolver problemas más elaborados mediante el uso de agentes de decisión más sofisticados, como Monte Carlo, SARSA y Q-learning.

1.4. Estructura del documento

En la sección 2 se encuentra una introducción general a los métodos de aprendizaje por refuerzo basados en políticas, a lo que le sigue la implementación detallada de los algoritmos empleados en la sección tercera. En la cuarta sección encontraremos un análisis del desempeño y dificultades de cada agente en según qué entorno, y para terminar añadimos un conclusión sobre el trabajo realizado.

2. Fundamentos del Aprendizaje por Refuerzo Basado en Políticas

El *aprendizaje por refuerzo* (RL, por sus siglas en inglés) es un paradigma del aprendizaje automático en el que un agente aprende a tomar decisiones a lo largo del tiempo mediante la interacción con un entorno. Este proceso se modela formalmente como un *proceso de decisión de Markov* (MDP), definido por la tupla:

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma), \tag{1}$$

donde:

- $\mathcal{S}$ es el conjunto de estados posibles del entorno,
- $\mathcal{A}$ es el conjunto de acciones disponibles para el agente,
- $P(s' \mid s, a)$ es la función de transición de estados, que define la probabilidad de alcanzar el estado s' al tomar la acción a en el estado s,
- $R(s, a)$ es la función de recompensa, que determina la retroalimentación recibida al ejecutar la acción a en el estado s,
- $\gamma \in [0, 1]$ es el factor de descuento, que pondera la importancia de las recompensas futuras.

El objetivo del agente es encontrar una política $\pi : \mathcal{S} \to \mathcal{A}$ que maximice la recompensa acumulada esperada a lo largo del tiempo, definida por el retorno:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2}$$

Para evaluar la calidad de una política π, se definen dos funciones fundamentales:

- La función de valor del estado bajo la política π:

$$V^\pi(s) = \mathbb{E}^\pi \left[G_t \mid S_t = s \right]. \tag{3}$$

- La función de valor de acción bajo la política π:

$$Q^\pi(s, a) = \mathbb{E}^\pi \left[G_t \mid S_t = s, A_t = a \right]. \tag{4}$$

Dado que la política π determina la probabilidad de seleccionar cada acción en un estado determinado, podemos definir dos tipos de aprendizaje basado en políticas:

2.1. Métodos *On-Policy* y *Off-Policy*

El aprendizaje por refuerzo basado en políticas puede clasificarse en dos enfoques principales:

- **Métodos On-Policy**: Aprenden la política óptima evaluando y mejorando directamente la política que se está utilizando para la exploración. Un ejemplo es el algoritmo **SARSA**, que actualiza la función de acción-valor mediante:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]. \tag{5}$$

- **Métodos Off-Policy**: Aprenden la política óptima utilizando datos generados por una política distinta. Un ejemplo es el algoritmo **Q-learning**, que sigue la ecuación de actualización:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]. \tag{6}$$

2.2. Métodos Basados en Monte Carlo y Diferencias Temporales

Los métodos de aprendizaje basados en políticas también pueden diferenciarse según la forma en que estiman la función de valor:

- **Métodos de Monte Carlo**: Aprenden a partir de episodios completos, actualizando las estimaciones de $Q^\pi(s, a)$ basándose en la recompensa total obtenida en cada episodio. Se dividen en:
  - **Monte Carlo On-Policy**: Se actualiza la política evaluada.
  - **Monte Carlo Off-Policy**: Utiliza importancia ponderada para aprender de episodios generados por otra política.
- **Métodos de Diferencias Temporales (TD)**: Actualizan las estimaciones de $Q^\pi(s, a)$ en cada paso del episodio, combinando Monte Carlo y programación dinámica. Ejemplo de esto son los algoritmos SARSA y Q-learning antes mencionados.

3. Algoritmos

3.1. Monte Carlo on-policy

El algoritmo **Monte Carlo On-Policy con política ε-soft** es un método basado en la generación de episodios completos para aprender la función de acción-valor $Q(s, a)$. A diferencia del enfoque off-policy, este método actualiza la política π directamente en función de la experiencia obtenida, garantizando una exploración mínima mediante una política ε-soft.

Cada episodio es generado siguiendo la política actual π, y se actualizan las estimaciones de los valores de acción utilizando el retorno promedio de las visitas a cada par (s, a). La actualización de $Q(s, a)$ se realiza como:

$$Q(S_t, A_t) \leftarrow \frac{1}{|\text{Returns}(S_t, A_t)|} \sum G$$

donde:

- G es la suma acumulada de recompensas con descuento.
- $\text{Returns}(s, a)$ es la lista de retornos obtenidos para el par (s, a).

Una vez actualizados los valores de $Q(s, a)$, se ajusta la política π para favorecer la acción óptima $A^* = \arg\max_a Q(s, a)$ mientras se mantiene una probabilidad de exploración ε:

$$\pi(a|S_t) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|A(S_t)|}, & \text{si } a = A^* \\ \frac{\varepsilon}{|A(S_t)|}, & \text{si } a \neq A^* \end{cases}$$

Los pasos del algoritmo son:

1. **Inicialización:**
   - Se define una política ε-soft arbitraria π.
   - Se inicializan $Q(s,a)$ y las listas de retornos como vacías.
2. **Iteración por episodios:**
   - Se genera un episodio siguiendo la política π.
   - Se retropropagan las recompensas acumuladas para actualizar $Q(s,a)$.
3. **Actualización de la política:**
   - Se recalcula la acción óptima A^* basada en $Q(s,a)$.
   - Se ajusta la política π manteniendo la exploración con ε.

Este método garantiza la convergencia a la política óptima en el límite, siempre que todas las acciones sean visitadas con probabilidad no nula, lo cual es asegurado por la política ε-soft.

A continuación mostramos el pseudocódigo que lleva a cabo estos procesos:

Algorithm 1 Monte Carlo On-Policy all visits

Inicialización:
$\pi \leftarrow$ política ε-soft arbitraria
$Q(s,a)$ arbitrario
Returns$(s,a) \leftarrow$ lista vacía
while no terminar el proceso **do**
  Generar un episodio siguiendo π: $(S_0, A_0, R_1, ..., S_{T-1}, A_{T-1}, R_T)$
  $G \leftarrow 0$
  for $t = T-1$ **to** 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$
    Agregar G a Returns(S_t, A_t)
    $Q(S_t, A_t) \leftarrow$ promedio de Returns(S_t, A_t)
    $A^* \leftarrow \arg\max_a Q(S_t, a)$
    for cada $a \in A(S_t)$ **do**
      if $a = A^*$ **then**
        $\pi(a|S_t) \leftarrow 1 - \varepsilon + \frac{\varepsilon}{|A(S_t)|}$
      else
        $\pi(a|S_t) \leftarrow \frac{\varepsilon}{|A(S_t)|}$
      end if
    end for
  end for
end while

3.2. Monte Carlo off-policy

El algoritmo **Monte Carlo Off-Policy con ponderación por importancia** permite aprender una política óptima π a partir de episodios generados por una política distinta b (política exploratoria). Se basa en la estimación de la función $Q(s,a)$ mediante la acumulación de recompensas a lo largo de episodios completos.

A diferencia de los métodos on-policy, este enfoque ajusta $Q(s,a)$ usando ponderación por importancia para corregir la diferencia entre las distribuciones de las políticas β y π. La actualización se realiza como:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} \left(G - Q(S_t, A_t)\right)$$

donde:

- G es la suma acumulada de recompensas con descuento.
- $C(s,a)$ es el acumulador de pesos de importancia.
- W es el factor de ponderación por importancia definido como:

$$W \leftarrow W \times \frac{1}{\beta(A_t|S_t)}$$

- La política óptima se actualiza como:

$$\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$$

El algoritmo sigue los siguientes pasos:

1. **Inicialización:**
   - Se inicializa $Q(s,a)$ arbitrariamente y $C(s,a) = 0$.
   - La política óptima se define como $\pi(s) = \arg\max Q(s,a)$.
2. **Iteración por episodios:**
   - Se genera un episodio siguiendo la política exploratoria β.
   - Se retropropagan las recompensas y se actualizan $Q(s,a)$ y $\pi(s)$.
   - Si la acción tomada no coincide con la acción óptima, se detiene la actualización.
3. **Ponderación por importancia:**
   - Se ajustan las actualizaciones utilizando el factor de ponderación W.
   - Se evita la actualización descontrolada deteniendo el proceso si la acción tomada no es la óptima.

Este método garantiza la convergencia a la política óptima bajo ciertas condiciones, corrigiendo el sesgo introducido por la política exploratoria mediante la ponderación por importancia.

Y en pseudocódigo:

Algorithm 2 Monte Carlo Off-Policy all visits

Initialize:
$\pi(s) \leftarrow \arg\max Q(s,a)$
$Q(s,a)$ arbitrary
$C(s,a) = 0$
repeat
  $b \leftarrow$ any soft policy
  Generate an episode following b: $(S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T)$
  $G \leftarrow 0$
  $W \leftarrow 1$
  for $t = T-1$ to 0 **do**
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)}(G - Q(S_t, A_t))$
    $\pi(S_t) \leftarrow \arg\max Q(S_t, a)$
    if $A_t \neq \pi(S_t)$ **then**
      break (proceed to next episode)
    end if
    $W \leftarrow W \times \frac{1}{b(A_t|S_t)}$
  end for
until forever (for each episode)

3.3. SARSA

El algoritmo **SARSA (State-Action-Reward-State-Action)** es un método de aprendizaje por refuerzo basado en la actualización temporal de la función de acción-valor $Q(s,a)$. Se clasifica como un método *on-policy*, ya que aprende de la política que sigue durante la exploración.

SARSA actualiza $Q(s,a)$ utilizando la ecuación:

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

donde:

- S, A son el estado y la acción actuales.
- R es la recompensa obtenida tras tomar la acción A.
- S' es el nuevo estado alcanzado.
- A' es la acción seleccionada en S' según la política actual.
- α es la tasa de aprendizaje.
- γ es el factor de descuento que pondera futuras recompensas.

El algoritmo sigue los siguientes pasos:

1. **Inicialización:**
   - Se inicializa la función de acción-valor $Q(s,a)$ arbitrariamente, excepto para estados terminales donde $Q(\text{terminal}, \cdot) = 0$.
2. **Iteración por episodios:**
   - Se inicializa el estado S.
   - Se elige una acción A según la política derivada de Q (ej. ε-greedy).
   - Se interactúa con el entorno hasta alcanzar un estado terminal.
3. **Actualización de valores:**
   - Se observa la recompensa R y el nuevo estado S'.
   - Se elige una nueva acción A' en S' siguiendo la política actual.
   - Se actualiza $Q(S,A)$ usando la ecuación de actualización temporal.
   - Se avanza al nuevo estado y acción (S', A').

A continuación se muestra su pseudocódigo:

Algorithm 3 SARSA

Parámetros: tasa de aprendizaje $\alpha \in (0,1]$, pequeña $\varepsilon > 0$
Inicialización:
  Inicializar $Q(s,a)$ arbitrariamente para todos $s \in S^+$, $a \in A(s)$, excepto que $Q(\text{terminal}, \cdot) = 0$
while para cada episodio **do**
  Inicializar S
  Elegir A desde S usando la política derivada de Q (ej. ε-greedy)
  while para cada paso del episodio **do**
    Tomar acción A, observar recompensa R y nuevo estado S'
    Elegir A' desde S' usando la política derivada de Q (ej. ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma Q(S',A') - Q(S,A)\right]$
    $S \leftarrow S'$
    $A \leftarrow A'$
  end while
end while

3.4. Q-Learning

El algoritmo Q-learning es un método de aprendizaje por refuerzo que permite a un agente aprender a tomar decisiones óptimas en un entorno basado en la retroalimentación que recibe del mismo. A continuación se describe el proceso que sigue este algoritmo.

- **Entrada:** El algoritmo toma dos parámetros principales:
  - **Paso de aprendizaje** $\alpha \in (0,1]$, que controla la rapidez con la que el agente actualiza su conocimiento sobre el valor de las acciones.
  - **Epsilon** $\epsilon > 0$, que determina la probabilidad con la que el agente explorará acciones aleatorias, lo que favorece la exploración en lugar de la explotación.
- **Inicialización:** El valor de la función de acción-valor $Q(s,a)$ es inicializado para todos los estados $s \in S^+$ y todas las acciones $a \in A(S)$. En general, los valores se inicializan arbitrariamente, aunque el valor de la función $Q(\text{terminal}, .)$ se establece en 0 para los estados terminales.
- **Ciclo de episodios:**
  - En cada episodio, el algoritmo comienza por inicializar un estado S en el entorno.
  - Durante cada paso del episodio, el algoritmo realiza las siguientes acciones:
    - **Selección de acción:** El agente elige una acción A a partir del estado actual S siguiendo una política derivada de los valores actuales de la función Q. Una

de las estrategias más comunes es la *epsilon-greedy*, donde el agente selecciona la acción con el valor máximo de $Q(s, a)$ con probabilidad $1 - \epsilon$, y elige una acción aleatoria con probabilidad ϵ.

- ⋆ **Actualización de Q:** El agente actualiza el valor de la función Q en el estado actual S y la acción A de acuerdo con la siguiente fórmula:

$$Q(S, A) \leftarrow Q(S, A) + \\ + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$$

donde:

- · R es la recompensa recibida por el agente al tomar la acción A desde el estado S.
- · γ es el factor de descuento que refleja la importancia de las recompensas futuras.
- · S' es el nuevo estado en el que el agente se encuentra después de realizar la acción A.

Esta actualización permite que el agente ajuste gradualmente sus estimaciones de la función de valor para cada acción en cada estado.

- ⋆ **Transición al siguiente estado:** El agente pasa al siguiente estado S', que se convierte en el nuevo estado actual S.

- **Condición de terminación:** El ciclo del episodio se repite hasta que el agente alcanza un estado terminal.

Este algoritmo permite que el agente aprenda una política óptima mediante la actualización continua de sus estimaciones de la función Q, maximizando la recompensa total a largo plazo. Con el tiempo, el agente mejora su capacidad para elegir las mejores acciones en cada estado, lo que le permite tomar decisiones más informadas y eficientes en entornos dinámicos.

Aquí tenemos su pseudocódigo:

Algorithm 4 Q-Learning

Input: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize: $Q(s, a)$, for all $s \in S^+$, $a \in A(S)$, arbitrarily, except that $Q(\text{terminal}, .) = 0$
for each episode **do**
  Initialize S
  for each step of the episode **do**
    Choose A from S using policy derived from Q (e.g., epsilon-greedy: $P(A' = \arg\max Q(s', a)) = 1 - \epsilon$)
    $Q(S, A) \leftarrow Q(S, A) + \\ + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$
    $S \leftarrow S'$
  end for
end for

3.5. SARSA semi-gradiente

El algoritmo SARSA semigradiente es una variante de SARSA en la que la función de acción-valor se parametriza utilizando un vector de pesos w. La idea central de SARSA semigradiente es actualizar los pesos de la función de acción-valor de manera semigradiente en cada paso, usando la retroalimentación proporcionada por la recompensa observada y la acción seleccionada. Este enfoque permite mejorar el rendimiento del agente en entornos con espacios de estados y acciones grandes, donde la aproximación por tableros no es factible.

La intuición detrás de SARSA semigradiente es que, a diferencia de los métodos basados en tablas, los valores de acción se aproximan mediante un modelo diferenciable de $q(S, A, w)$. Los parámetros w se ajustan gradualmente para reducir el error de predicción. En cada paso, el algoritmo actualiza los pesos en función de la recompensa observada, la acción tomada, y el valor estimado de las acciones futuras.

Los pasos que sigue el algoritmo son:

1. **Inicialización:**
   - Establecer una parametrización diferenciable de la función de acción-valor $\hat{q}(S, A, w)$.
   - Inicializar los pesos de la función de valor $w \in \mathbb{R}^d$ arbitrariamente (por ejemplo, $w = 0$).
2. **Selección de acción inicial (S_0, A_0):** El agente selecciona el estado inicial S_0 y la acción inicial A_0 siguiendo una política derivada de la función de acción-valor (por ejemplo, ϵ-greedy).
3. **Iteración de cada paso del episodio:** En cada paso, el algoritmo realiza las siguientes operaciones:
   - Tomar la acción A en el estado S, observar la recompensa R y el nuevo estado S'.
   - Si S' es terminal, actualizar los pesos w:

$$w \leftarrow w + \alpha \left[R - \hat{q}(S, A, w) \right] \Delta \hat{q}(S, A, w)$$

     donde $\Delta \hat{q}(S, A, w)$ es el gradiente de la función $\hat{q}$ con respecto a los pesos w.
   - Si S' no es terminal, elegir una nueva acción A' como función de la política derivada de $\hat{q}(S', ., w)$.
   - Actualizar los pesos w utilizando la siguiente fórmula:

$$w \leftarrow w + \alpha \left[R + \gamma \hat{q}(S', A', w) - \hat{q}(S, A, w) \right] \Delta \hat{q}(S, A, w)$$

   - Actualizar el estado y la acción:

$$S \leftarrow S', \quad A \leftarrow A'$$

4. **Fin del episodio:** El episodio se repite hasta que el agente llegue a un estado terminal.

El pseudocódigo correspondiente es:

Algorithm 5 SARSA Semi-gradiente

Input: Función de acción-valor diferenciable $\hat{q} : S \times A \times \mathbb{R}^d \to \mathbb{R}$
Algoritmo parámetros: Tasa de aprendizaje $\alpha > 0$, pequeño $\epsilon > 0$
Inicializar: Pesos de la función de valor $w \in \mathbb{R}^d$ arbitrariamente (por ejemplo, $w = 0$)
for cada episodio **do**
    Inicializar S, A (estado inicial y acción del episodio) (por ejemplo, ϵ-greedy)
    for cada paso del episodio **do**
        Tomar acción A, observar recompensa R y nuevo estado S'
        if S' es terminal **then**
            $w \leftarrow w + \alpha \left[R - \hat{q}(S, A, w)\right] \Delta\hat{q}(S, A, w)$
            Avanzar al siguiente episodio
        end if
        Elegir A' como función de $\hat{q}(S', ., w)$ (por ejemplo, ϵ-greedy)
        $w \leftarrow w + \alpha \left[R + \gamma\hat{q}(S', A', w) - \hat{q}(S, A, w)\right] \Delta\hat{q}(S, A, w)$
        $S \leftarrow S'$
        $A \leftarrow A'$
    end for
end for

3.6. Deep Q-Learning

El algoritmo Deep Q-Learning es una extensión del algoritmo Q-learning tradicional que emplea redes neuronales profundas para aproximar la función de valor de acción $Q(s, a)$, que representa la calidad de tomar una acción a en un estado s. Este enfoque es útil cuando el espacio de estados y/o el espacio de acciones es demasiado grande como para ser manejado por una tabla de Q-values, lo cual es común en tareas con entradas de alta dimensión, como en el caso de imágenes o secuencias temporales.

En Deep Q-Learning, se utiliza una red neuronal $Q(s, a; \theta)$ para aproximar la función de valor de acción, donde θ son los parámetros (pesos) de la red. A lo largo del entrenamiento, los parámetros θ se actualizan mediante el algoritmo de retropropagación, utilizando el error de la diferencia temporal, una forma de aprendizaje supervisado en el contexto de Q-learning.

El objetivo es que la red neuronal aprenda a predecir la recompensa futura de las acciones de manera precisa, permitiendo que el agente tome decisiones más informadas.

El algoritmo sigue los siguientes pasos:

1. **Inicialización:** Inicializar la memoria de repetición D con capacidad N y la función de acción-valor Q parametrizada por una red neuronal con pesos θ.
2. **Iteración por episodios:** En cada episodio, el agente realiza los siguientes pasos:
    - Inicializar la secuencia de estados s_1 y su versión preprocesada $\phi_1 = \phi(s_1)$.
    - Durante cada paso del episodio:
        - Con probabilidad ϵ, seleccionar una acción aleatoria a_t, de lo contrario seleccionar $a_t = \max_a Q(\phi(s_t), a; \theta)$, donde la función Q es la red neuronal que estima la acción-valor.
        - Ejecutar la acción seleccionada a_t en el entorno, observar la recompensa r_t y el nuevo estado x_{t+1}.
        - Establecer el nuevo estado $s_{t+1} = s_t, a_t, x_{t+1}$ y preprocesar $\phi_{t+1} = \phi(s_{t+1})$.
        - Almacenar la transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en la memoria de repetición D.
        - Muestrear aleatoriamente un minibatch de transiciones $(\phi_j, a_j, r_j, \phi_{j+1})$ de la memoria D.
        - Para las transiciones donde ϕ_{j+1} es terminal, establecer $y_j = r_j$. Para las transiciones no terminales, calcular:
        $$y_j = r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta)$$
        donde γ es el factor de descuento.
        - Realizar un paso de descenso de gradiente en la diferencia cuadrada:
        $$\theta \leftarrow \theta - \alpha \nabla_\theta \left(y_j - Q(\phi_j, a_j; \theta)\right)^2$$
        donde α es la tasa de aprendizaje.
3. **Fin del episodio:** El ciclo de episodios continúa hasta que el agente haya aprendido a aproximar de manera precisa las funciones de valor de acción para todas las posibles combinaciones de estados y acciones.

El principio fundamental detrás de Deep Q-Learning es que, en lugar de mantener una tabla de valores de acción, se utiliza una red neuronal que aprende a estimar estos valores a medida que el agente interactúa con el entorno. Esta aproximación permite resolver problemas con grandes espacios de estados y acciones, como juegos con imágenes de entrada o simulaciones complejas.

Aquí tenemos su pseudocódigo:

Algorithm 6 Deep Q-Learning

Inicializar: Memoria de repetición D con capacidad N
Inicializar: Función de acción-valor Q con pesos aleatorios
for episodio = 1, M **do**
  Inicializar secuencia $s_1 = \{x_1\}$ y secuencia preprocesada $\phi_1 = \phi(s_1)$
  for t = 1, T **do**
    if con probabilidad ϵ **then**
      Seleccionar acción aleatoria a_t
    else
      Seleccionar $a_t = \text{máx}_a\, Q^*(\phi(s_t), a; \theta)$
    end if
    Ejecutar acción a_t en el emulador, observar recompensa r_t y nueva imagen x_{t+1}
    Establecer $s_{t+1} = s_t, a_t, x_{t+1}$ y preprocesar $\phi_{t+1} = \phi(s_{t+1})$
    Almacenar transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en D
    Muestrear minibatch aleatorio de transiciones $(\phi_j, a_j, r_j, \phi_{j+1})$ de D
    if ϕ_{j+1} es terminal **then**
      Establecer $y_j = r_j$
    else
      Establecer $y_j = r_j + \gamma\, \text{máx}_{a'}\, Q(\phi_{j+1}, a'; \theta)$
    end if
    Realizar un paso de descenso de gradiente en $(y_j - Q(\phi_j, a_j; \theta))^2$
  end for
end for

4. Desarrollo del experimento

Este apartado describe varios experimentos de aprendizaje por refuerzo utilizando el algoritmo de Monte Carlo con políticas epsilon-soft, tanto on-policy como off-policy, así como el algoritmo SARSA, el Q-learning, el SARSA semi-gradiente y el Deep Q-learning.

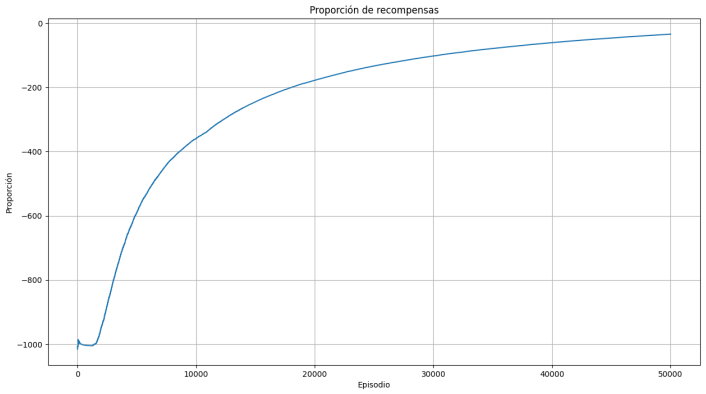
Para comprobar el aprendizaje se mostrará la función $f(t) = \frac{\sum_{i=1}^{t} R_i}{t}$ para $t = 1, 2, \ldots, NumeroEpisodios$. La justificación es la siguiente. Como sabemmos que el retorno en el estados inicial 1 (pues no hay descuento) o 9, si se divide por el número de episodios ejecutados se calcular el porcentaje de recompensas positivas obtenidas. Dicho de otra forma, nos dirá el porcentaje de veces que el agente ha llegado al estado terminal.

También se muestra una gráfica con la longitud de los episodios en cada estado. Esta gráfica nos será útil para analizar si realmente el algoritmo está aprendiendo a llegar a la meta en cada episodio, observando cómo evoluciona la duración de los episodios a lo largo del entrenamiento (lo ideal sería que vayan disminuyendo conforme vaya aprendiendo).

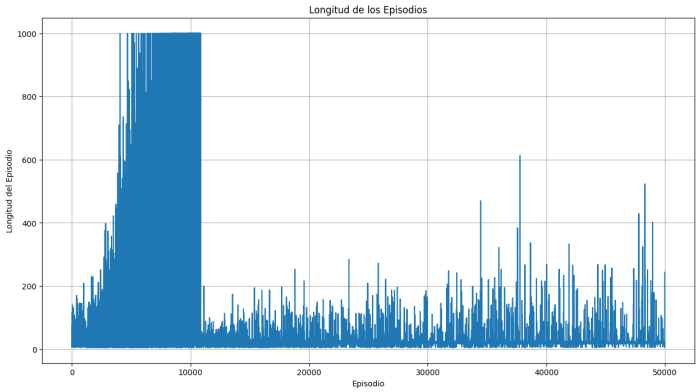
4.1. Frozen Lake

El propósito de este análisis es entrenar un agente en un entorno de gym con el juego "FrozenLake", un entorno estándar en el que el agente debe aprender a moverse a través de un mapa en busca de una meta, evitando caer en agujeros.

4.1.1. Monte Carlo on-policy

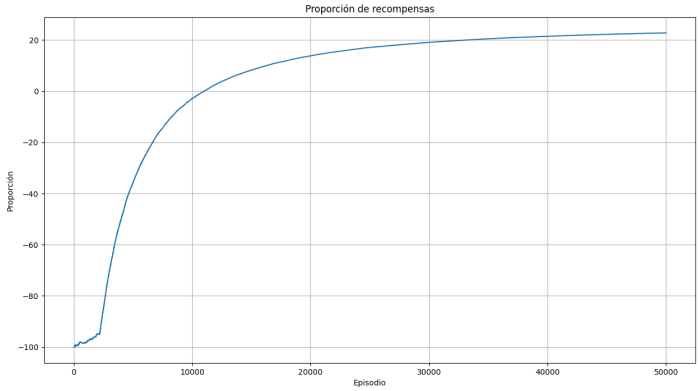


Las recompensas aumentan a lo largo de los episodios.

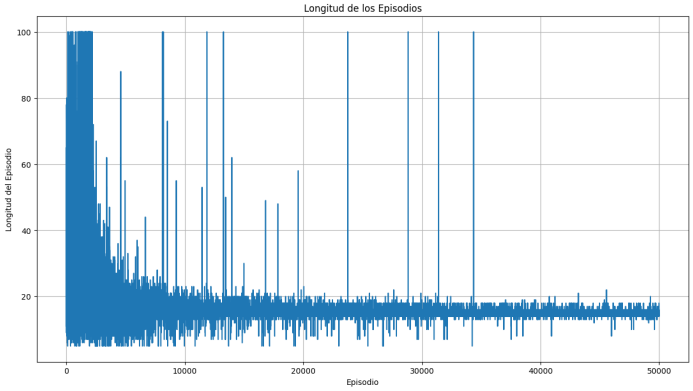


Al principio, la longitud de los episodios es bastante grande, aunque a partir de los 10000 episodios aproximadamente, disminuye y se estabiliza en una longitud mucho menor.

4.1.2. Monte Carlo off-policy

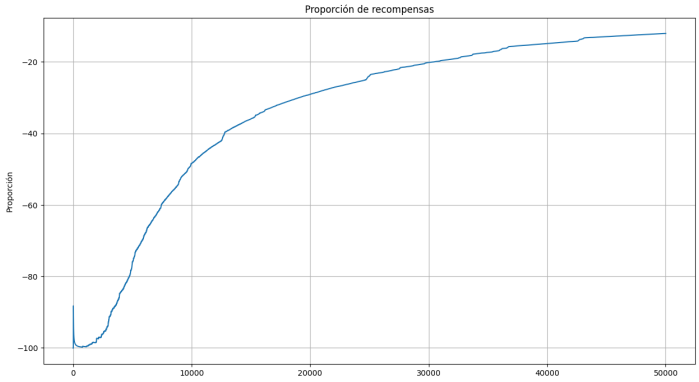


Observando la gráfica para la proporción de recompensas con Monte Carlo off-policy, esta parece mejorar con respecto a Monte Carlo on-policy, indicando que la estrategia de usar dos políticas, una política objetivo y una de comportamiento, ayuda a mejorar el rendimiento del algoritmo.

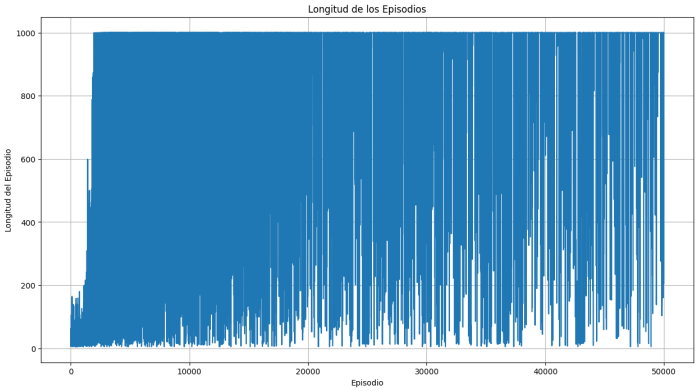


Al principio, los episodios son algo más largos, aunque en este caso hemos dejado una longitud máxima de 100 pasos por episodio, y aun así se han obtenido resultados relativamente buenos. Con otros algoritmos es necesario aumentar más el límite de pasos para que logre aprender correctamente, pero en este caso no ha sido necesario.

4.1.3. SARSA

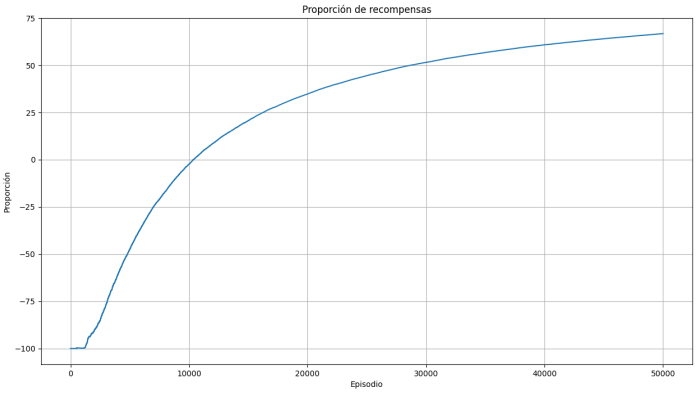


En este caso vemos que la proporción de recompensas aumenta, por lo que el agente está aprendiendo, pero no logra alcanzar valores positivos para dicha proporción. Esto quiere decir que en la mayoría de episodios no está alcanzando la meta.

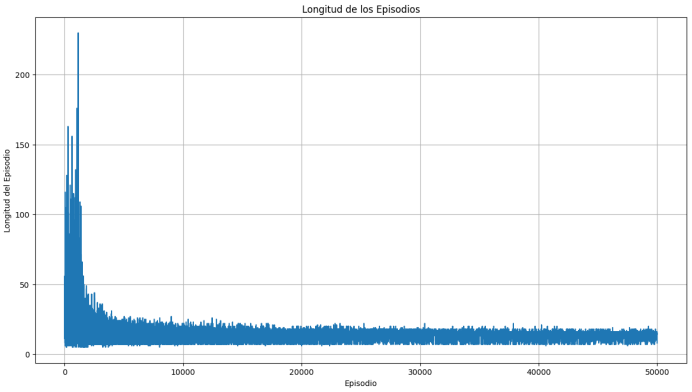


Al observar la gráfica con el número de pasos por episodio, descubrimos donde está el problema: el límite de pasos de 1000 es demasiado bajo, pues la mayoría de episodios no consiguen llegar a ningún estado terminal en 1000 pasos. Por esta razón, el agente no está consiguiendo aprender correctamente un camino óptimo hasta la meta.

4.1.4. Q-Learning



El algoritmo Q-learning da buenos resultados, como se observa en el gráfico de la proporción de recompensas. Al principio, como suele ocurrir, dicha proporción es muy baja y negativa, pero con el paso de los episodios, rápidamente empieza a aumentar hasta alcanzar un máximo de 66,77. Este valor es bastante elevado y muestra que el agente está logrando aprender un buen camino hasta la meta.

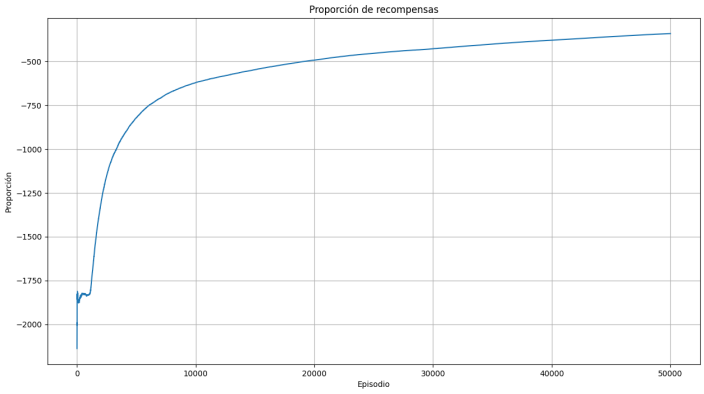


La longitud de los episodios comienza siendo algo más elevada, pero rápidamente disminuye y se estabiliza en valores bastante bajos, lo que junto con la anterior gráfica muestra que el agente está logrando encontrar un camino hasta la meta, y además encuentra un camino óptimo.

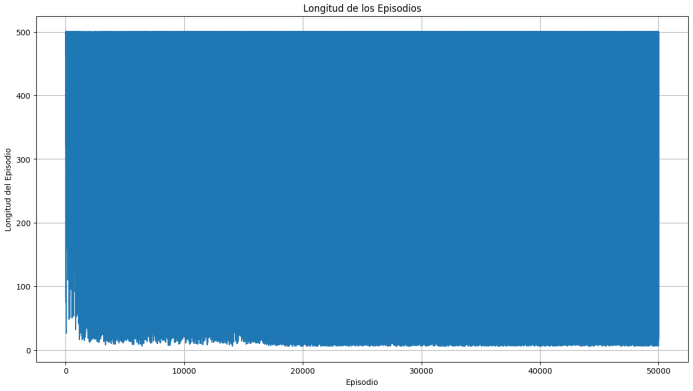
4.2. Taxi

El propósito de este análisis es entrenar un agente en un entorno de gym con el juego "Taxi", un entorno estándar en el que el agente debe aprender a moverse a través de un mapa recogiendo a pasajeros y llevándolos a otra localización.

4.2.1. Monte Carlo on-policy

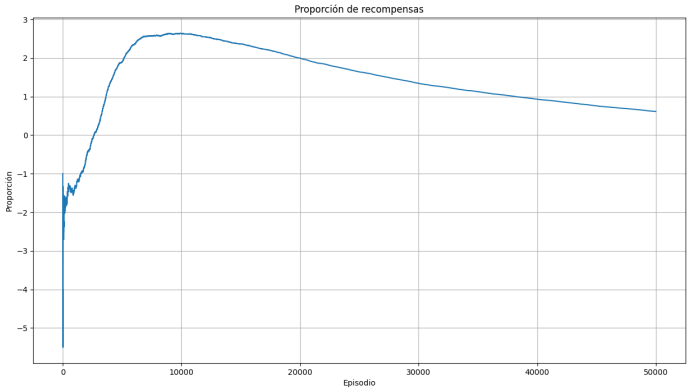


La proporción de recompensas va aumentando a lo largo de los episodios hasta alcanzar una máxima proporción de cerca de -340 de recompensa. Esto puede ser debido a que, a medida que el agente avanza por el entorno, las penalizaciones por cada acción (de -1) se acumulan, y si el agente no tiene una política eficaz, podría estar realizando muchos movimientos innecesarios. Esto provocaría que el agente acumule una gran cantidad de penalizaciones antes de llegar a la meta. Sin embargo, vemos como a lo largo de los episodios la política va mejorando relativamente.

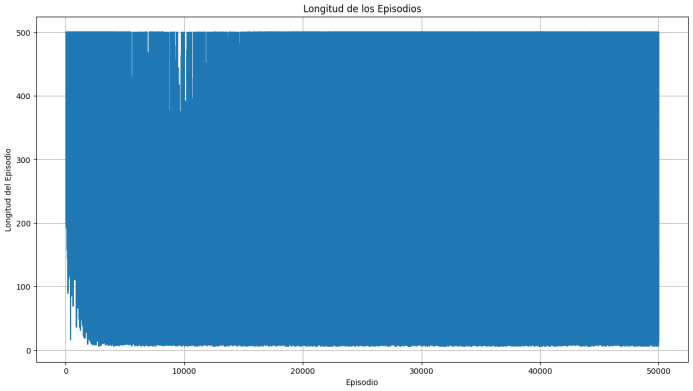


Con el paso de los episodios, la longitud de los mismos no consigue disminuir, pues se alcanza el máximo permitido en muchos de los episodios sin llegar a un estado terminal.

4.2.2. Monte Carlo off-policy

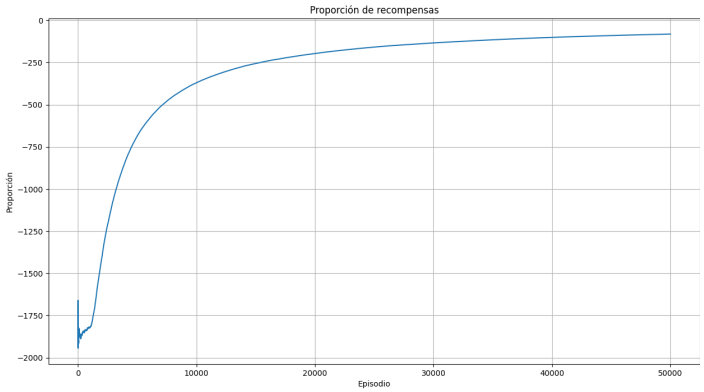


Observando la gráfica para la proporción de recompensas con Monte Carlo off-policy, esta claramente mejora con respecto a Monte Carlo on-policy, pues en este caso se logran alcanzar recompensas positivas, indicando que la estrategia de usar dos políticas, una política objetivo y una de comportamiento, ayuda a mejorar el rendimiento del algoritmo.

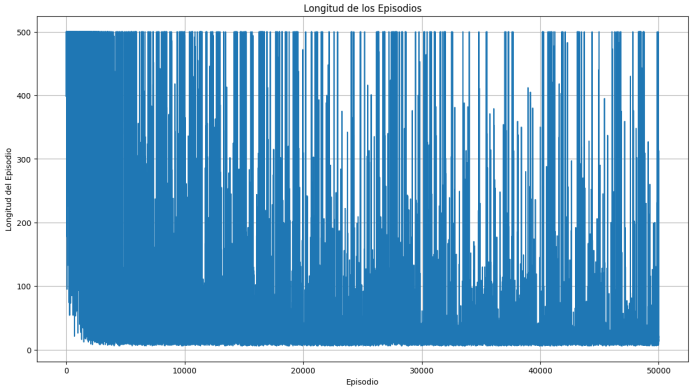


La longitud máxima permitida para los episodios sigue siendo algo baja, pues con el paso de los episodios, la longitud de los mismos no consigue disminuir. En muchos episodios se alcanza el máximo permitido sin llegar a un estado terminal.

4.2.3. SARSA



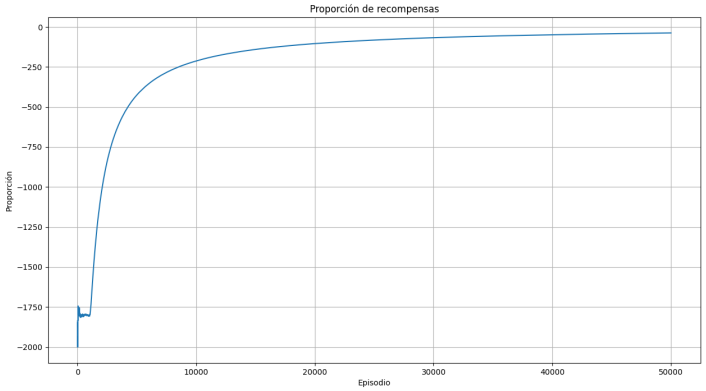
En este caso vemos que la proporción de recompensas aumenta, por lo que el agente está aprendiendo, pero no logra alcanzar valores positivos para dicha proporción. No obstante, la máxima proporción obtenida es mejor que con Monte Carlo on-policy.



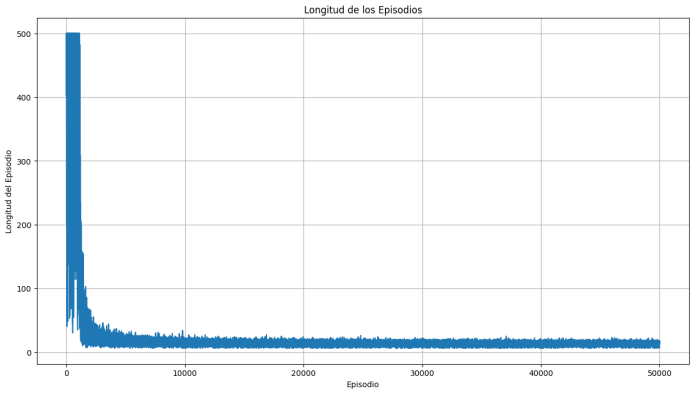
Al observar la gráfica con el número de pasos por episodio, vemos que, si bien no logra converger del todo a un número bajo de pasos, sí que se va disminuyendo el número de episodios que llegan a 500 pasos conforme nos acercamos más al final del entrenamiento. Parece que el agente está aprendiendo mejor que Monte Carlo, aunque el límite de pasos permitido

sigue siendo algo bajo.

4.2.4. Q-Learning



El algoritmo Q-learning da buenos resultados, como se observa en el gráfico de la proporción de recompensas. Al principio, como suele ocurrir, dicha proporción es muy baja y negativa, pero con el paso de los episodios, rápidamente empieza a aumentar hasta alcanzar un máximo de $-37{,}91$. A pesar de ser un valor negativo, es bastante alto en comparación con los obtenidos con algunos algoritmos anteriores, lo cual muestra el mejor rendimiento de Q-learning.

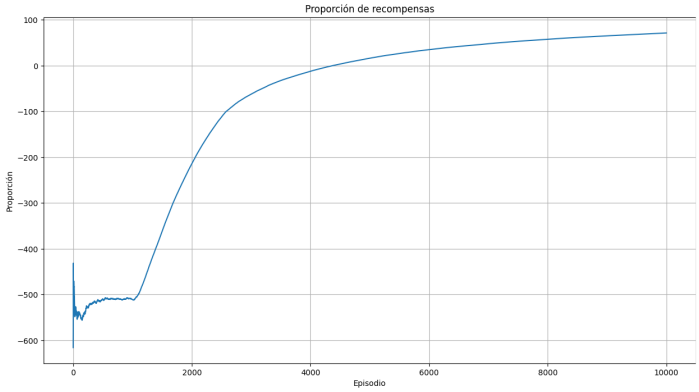


La longitud de los episodios comienza siendo algo más elevada, pero rápidamente disminuye y se estabiliza en valores bastante bajos, lo que junto con la anterior gráfica muestra que el agente está logrando encontrar una buena estrategia. Este es el único algoritmo que ha logrado que la longitud de los episodios se estabilice completamente en un valor bajo.

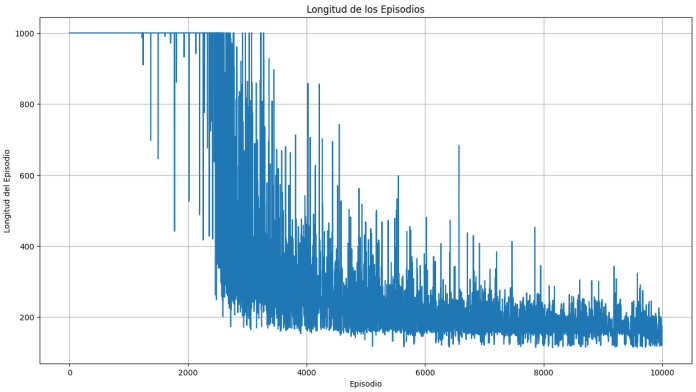
4.3. Mountain Car

Finalmente, se estudian algoritmos de Control con Aproximaciones, como SARSA semi-gradiente y Deep Q-Learning. Se probarán dichos algoritmos en el entorno continuo Mountain Car de Gymnasium.

4.3.1. SARSA semi-gradiente

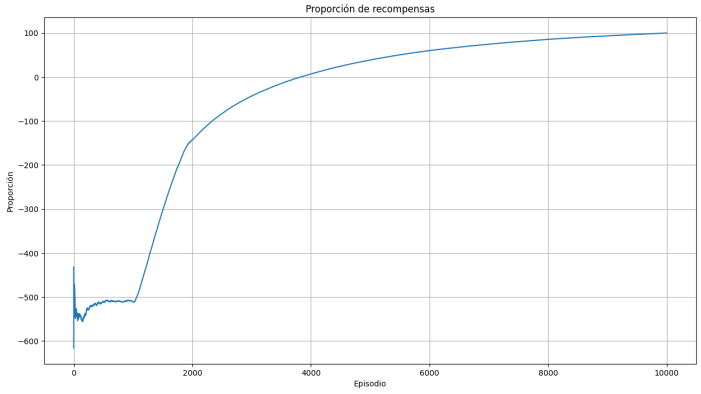


Al observar la gráfica de proporción de recompensas, se puede apreciar como al principio, en los primeros episodios, esta proporción es bastante baja, y de hecho, negativa, y además oscila, pero a partir de aproximadamente el episodio 1000, comienza a crecer considerablemente, alcanzando incluso valores positivos para la recompensa. Esto es una clara muestra de que el agente está aprendiendo, seguramente gracias al wrapper que hemos implementado para modificar las recompensas, especialmente favoreciendo el aumento de la velocidad.

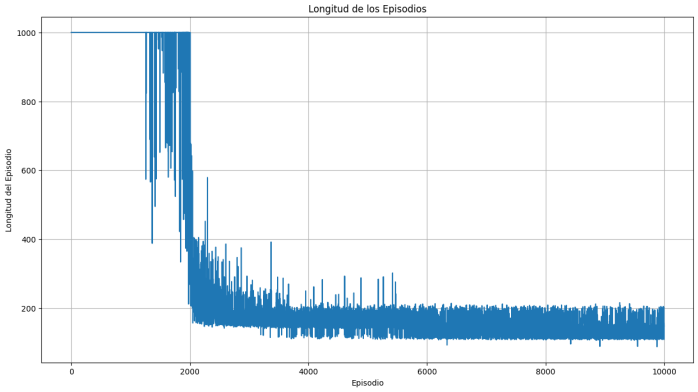


En la gráfica anterior se muestra la longitud de los episodios de entrenamiento. Se observa como los primeros episodios duran 1000 pasos, que de hecho es el máximo que hemos seleccionado para la longitud de los episodios. Sin embargo, a partir del episodio 2000 aproximadamente, la longitud de los episodios empieza a dismunuir considerablemnete, hasta que finalmente se estabiliza alrededor de los 100 pasos por episodio. Esto es una clara muestra de que a lo largo del entrenamiento, el agente está aprendiendo a llegar a la meta cada vez en menos pasos.

4.3.2. Deep Q-Learning



El algoritmo Q-learning da buenos resultados, como se observa en el gráfico de la proporción de recompensas. Al principio, como suele ocurrir, dicha proporción es muy baja y negativa, pero con el paso de los episodios, rápidamente empieza a aumentar hasta alcanzar un máximo de $-37{,}91$. A pesar de ser un valor negativo, es bastante alto en comparación con los obtenidos con algunos algoritmos anteriores, lo cual muestra el mejor rendimiento de Q-learning.



La longitud de los episodios comienza siendo algo más elevada, pero rápidamente disminuye y se estabiliza en valores bastante bajos, lo que junto con la anterior gráfica muestra que el agente está logrando encontrar una buena estrategia. Este es el único algoritmo que ha logrado que la longitud de los episodios se estabilice completamente en un valor bajo.

5. Conclusiones

En este experimento se analizaron varios algoritmos de aprendizaje por refuerzo aplicados en diferentes entornos de Gymnasium, incluyendo Monte Carlo, SARSA, Q-learning, SARSA semi-gradiente y Deep Q-learning. Los resultados muestran que, en general, los algoritmos lograron mejorar su rendimiento con el tiempo, tal como se observa en las gráficas, con el aumento en la proporción de recompensas y la disminución en la longitud de los episodios.

El algoritmo de Monte Carlo off-policy mostró una mejora notable en comparación con el on-policy, destacándose por un rendimiento superior en todos los entornos. El Q-learning también demostró un buen desempeño, con una mejora rápida en la proporción de recompensas y una disminución eficiente de la longitud de los episodios, especialmente en el entorno de "Taxi", donde fue el único que logró estabilizar la longitud de los episodios en un valor bajo. Por su parte, SARSA presentó un aprendizaje más lento y no logró alcanzar recompensas positivas en ninguno de los entornos, aunque mostró una mejora progresiva.

Una futura mejora en el entorno "Taxi"podría ser probar a aumentar el número máximo de pasos por episodio, ya que varios de los algoritmos utilizados han tenido dificultades para lograr alcanzar un estado terminal en muchos de los episodios. Gracias a esto podrían observarse mejoras en varios de los algoritmos utilizados. Sin embargo, con el límite de pasos que hemos usado en esta práctica, Q-learning ha sido capaz de aprender de forma bastante efectiva, y ha sido el único algoritmo que no ha tenido problemas con ello.

Por último, en el entorno continuo "Mountain Car", los algoritmos SARSA semi-gradiente y Deep Q-learning mostraron un desempeño destacable, con un crecimiento progresivo en la proporción de recompensas y una estabilización de la longitud de los episodios en valores bajos. Estos resultados demuestran la efectividad de las aproximaciones de control y la importancia de ajustes en las recompensas para facilitar el aprendizaje en entornos complejos.

En resumen, los algoritmos más avanzados como Q-learning y Deep Q-learning en general ofrecieron los mejores resultados, mientras que los enfoques más simples, como SARSA, requirieron más tiempo para mejorar y en algunos casos no alcanzaron un desempeño óptimo.

Referencias

[1] Gil, A. & García, J. & Malest, L. (2025). *Repositorio para la práctica II*. GitHub. Disponible en: https://github.com/JMGO-coding/RL_GGM/blob/main/

[2] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.