MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

Estudio y comparación de algoritmos de Aprendizaje por Refuerzo en Espacios Continuos

Autor: ***David Cocero Quintanilla***

Director: ***Miguel Ángel Bobi Sanz***

Madrid

2024

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Estudio y comparación de algoritmos de Aprendizaje por refuerzo en Espacios Continuos

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2023/24 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: David Cocero Quintanilla Fecha: 11/07/2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Firmado por SANZ BOBI MIGUEL ANGEL - ***6599** el día 11/07/2024 con un certificado emitido por AC FNMT

Fdo.: Miguel Ángel Bobi Sanz Fecha: ……/ ……/ ……

ESTUDIO Y COMPARACIÓN DE ALGORITMOS DE APRENDIZAJE POR REFUERZO EN ESPACIOS CONTINUOS

Autor: Cocero Quintanilla, David
Director: Bobi Sanz, Miguel Ángel
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN DEL PROYECTO

Este proyecto aborda el estudio y comparación de algoritmos de aprendizaje por refuerzo en espacios continuos, centrado en el análisis de los más usados como TD3, PPO o A2C. Se implementan y entrenan estos modelos en distintos entornos para evaluar su rendimiento en términos de recompensas acumuladas, velocidad de convergencia y estabilidad.

Palabras clave: Aprendizaje por refuerzo, Espacios Continuos, Stable Baselines, Gymnasium

1. Introducción

El aprendizaje por refuerzo (RL- Reinforcement Learning) es una técnica de machine learning que permite a un agente aprender comportamientos mediante la interacción con su entorno. No requiere datos etiquetados, sino que se basa en el proceso de prueba y error, donde el agente recibe recompensas o penalizaciones según sus acciones. Concretamente nos centramos en entornos que operan en espacios continuos, abordando la complejidad adicional que estos presentan en comparación con los espacios discretos. El aprendizaje por refuerzo tiene numerosas aplicaciones destacadas en robótica para control remoto, desarrollo de vehículos autónomos y otras más. Este trabajo pretende seguir abriendo camino en un área con tantas posibilidades de futuro como este.

2. Definición del proyecto

El proyecto tiene como objetivo explorar y comparar diversos algoritmos de aprendizaje por refuerzo. Se entrenarán y optimizarán estos algoritmos, utilizando para ello las implementaciones recogidas en la librería de Python de Stable Baselines 3. Estos se aplicarán en entornos de Gymnasium con espacios continuos de observaciones y acciones.

Se realizará a continuación una evaluación de los entrenamientos, viendo la evolución de las recompensas y un vídeo para mostrar el funcionamiento. Esta evaluación ayudará a identificar las ventajas y limitaciones de cada algoritmo en diferentes escenarios, proporcionando información valiosa para su aplicación en problemas reales cómo puede ser control de robots.

3. Descripción del sistema

El sistema presentado en la Ilustración 1 muestra una vista general del proyecto llevado a cabo. Inicialmente, se seleccionan algoritmos populares de aprendizaje por refuerzo, como TRPO, PPO, TD3, A2C y SAC. La implementación de estos se hace a través de la librería de Stable Baselines 3.

Los algoritmos se entrenan en distintos entornos de Gymnasium, tales como Mountain Car Continuous, Car Racing, Half Cheetah y Humanoid. Durante esta primera fase, se optimizan los hiperparámetros de cada algoritmo utilizando herramientas como RL Zoo y Optuna, con el objetivo de maximizar su rendimiento. Este proceso de optimización es crucial para adaptar cada algoritmo a las características particulares de los entornos, mejorando la eficacia del aprendizaje y la capacidad de generalización.

Aplicando los hiperparámetros óptimos, se entrenan los algoritmos con 500.000 iteraciones. A continuación, se obtienen y analizan los resultados del entrenamiento, evaluando el rendimiento de cada algoritmo a través de la evolución de las recompensas y los vídeos generados del agente. Estos resultados se visualizan a través de una interfaz gráfica desarrollada con Streamlit, que permite una interacción dinámica con los datos. La plataforma de Streamlit también ofrece la posibilidad de entrenar nuevos modelos, proporcionando una herramienta integral para la evaluación continua y la experimentación en aprendizaje por refuerzo.

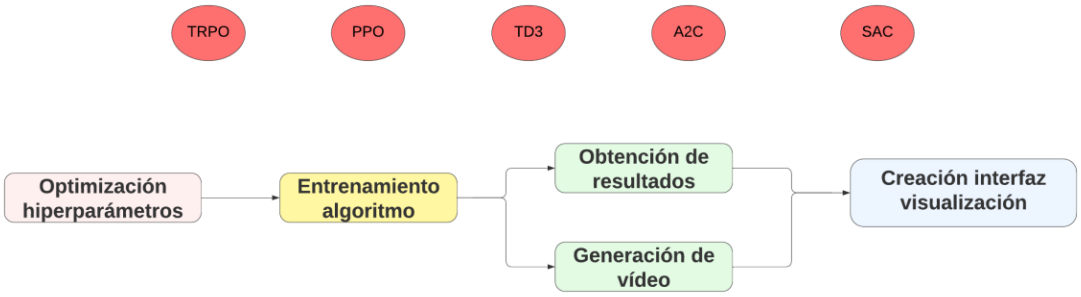


Ilustración 1: Vista general del proyecto

4. Resultados

Esta gráfica muestra las recompensas acumuladas durante el entrenamiento de diferentes algoritmos de aprendizaje por refuerzo en el entorno Half Cheetah. En el eje x se representan las iteraciones (o pasos de entrenamiento), y en el eje y se muestran las recompensas obtenidas. Los algoritmos evaluados son A2C, PPO, SAC, TD3 y TRPO.

SAC y TD3 son los algoritmos con las recompensas más altas entre todos. SAC, representado en rojo sólido, muestra un crecimiento rápido en las recompensas y alcanza valores superiores a 8000. Por otro lado, TD3, representado en rosa, también tiene un rendimiento alto, aunque un poco menos consistente que SAC, alcanzando valores superiores a 8000 pero con más variabilidad.

PPO también muestra un rendimiento decente, pero significativamente más bajo que SAC y TD3, alcanzando recompensas en torno a 4000. Finalmente, TRPO y A2C tienen recompensas más bajas e inestables, manteniéndose alrededor de 0-1000. En ellas no se aprecia además tendencia de mejoría en el entrenamiento.

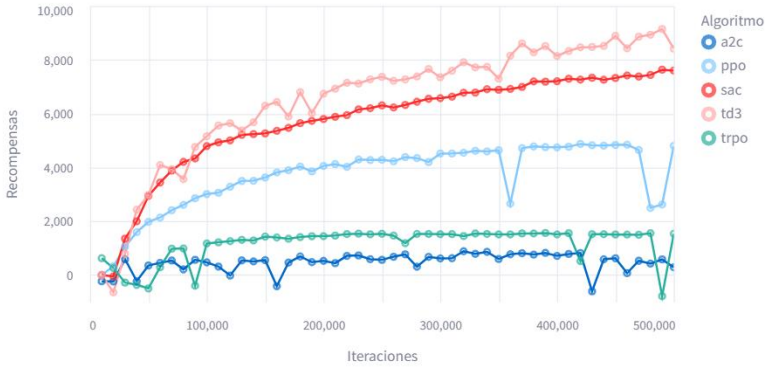


Ilustración 2 – Evolución de las recompensas en entrenamientos de distintos algoritmos para Half Cheetah

5. Conclusiones

Durante el desarrollo del proyecto, se han obtenido las siguientes conclusiones:

- La optimización de los hiperparámetros fue crucial para maximizar el rendimiento de cada algoritmo, adaptándolo al entorno.
- Algoritmos como SAC y TD3 funcionan mejor que los otros para entornos de cierta complejidad, mientras que otros como PPO o TRPO sufren más.
- Los resultados de este estudio son valiosos para aplicaciones prácticas en robótica, control autónomo y otros campos que requieren soluciones avanzadas de RL.

6. Referencias

[1] Vandelaer, C. “Reinforcement Learning: An introduction”, Agosto 2022, https://medium.com/@cedric.vandelaer/reinforcement-learning-an-introduction-part-1-4-866695deb4d1

STUDY AND COMPARISON OF REINFORCEMENT LEARNING ALGORITHMS IN CONTINUOUS SPACES

Author: Cocero Quintanilla, David
Supervisor: Sanz Bobi, Miguel Ángel
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This project explores and compares reinforcement learning (RL) algorithms in continuous spaces, focusing on popular algorithms such as TD3, PPO, and A2C. These models are implemented and trained in various environments to evaluate their performance in terms of cumulative rewards, convergence speed, and stability.

Keywords: Reinforcement Learning, Continuous Spaces, Stable Baselines, Gymnasium

1. Introduction

Reinforcement learning (RL) is a machine learning technique that enables an agent to learn behaviors through interaction with its environment. It does not require labeled data but relies on a trial-and-error process where the agent receives rewards or penalties based on its actions. This study specifically focuses on environments that operate in continuous spaces, addressing the additional complexity these spaces present compared to discrete ones. Reinforcement learning has numerous notable applications in robotics, development of autonomous vehicles, and more. This work aims to continue paving the way in a field with such vast future potential.

2. Project definition

The goal of this project is to explore and compare various reinforcement learning algorithms. These algorithms will be trained and optimized using implementations from the Python library Stable Baselines 3. They will be applied in Gymnasium environments with continuous observation and action spaces.

Next, the training results will be evaluated, observing the evolution of rewards and showcasing the agent's performance through video demonstrations. This evaluation will help identify the advantages and limitations of each algorithm in different scenarios, providing valuable insights for their application to real-world problems like robot control.

3. System description

The system presented in Figure 1 provides an overview of the project. Initially, popular reinforcement learning algorithms such as TRPO, PPO, TD3, A2C, and SAC are implemented. These implementations are carried out using the Stable Baselines 3 library.

The algorithms are trained in various Gymnasium environments, such as MountainCarContinuous, Car Racing, HalfCheetah, and Humanoid. During this first phase, the hyperparameters of each algorithm are optimized using tools like RL Zoo and Optuna, aiming to maximize performance. This optimization process is crucial for adapting each algorithm to the specific characteristics of the environments, enhancing learning efficiency and generalization capabilities.

With the optimal hyperparameters applied, the algorithms are trained for 500,000 iterations. The training results are then obtained and analyzed, evaluating the performance of each algorithm through the evolution of rewards and the generated videos of the agent. These results are visualized through a graphical interface developed with Streamlit, which allows dynamic interaction with the data. The Streamlit platform also offers the possibility of training new models, providing an integrated tool for continuous evaluation and experimentation in reinforcement learning.

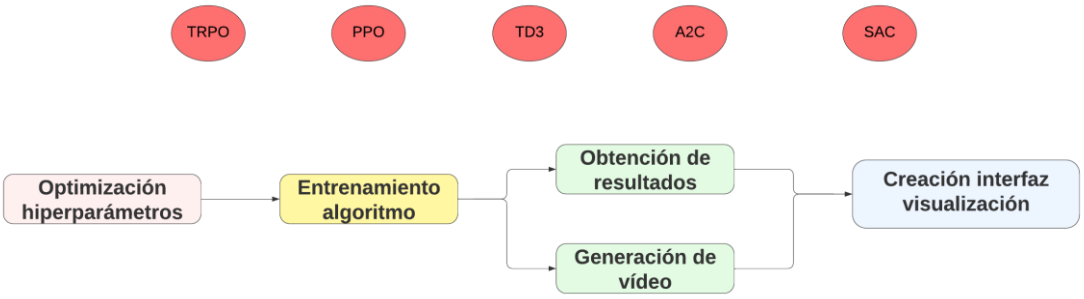


Figure 1 – Overall view of the project

4. Results

Figure 2 shows the accumulated rewards during the training of different reinforcement learning algorithms in the Half Cheetah environment. The x-axis represents the iterations (or training steps), and the y-axis shows the rewards obtained. The evaluated algorithms are A2C, PPO, SAC, TD3, and TRPO.

SAC and TD3 are the algorithms with the highest rewards among all. SAC, represented by the solid red line, shows rapid reward growth and reaches values above 8000. On the other hand, TD3, represented in pink, also performs highly, although slightly less consistently than SAC, reaching values above 8000 but with more variability.

PPO also shows decent performance but significantly lower than SAC and TD3, achieving rewards of around 4000. Finally, TRPO and A2C have lower and more unstable rewards, staying around 0-1000. Additionally, they do not show a clear trend of improvement during training.

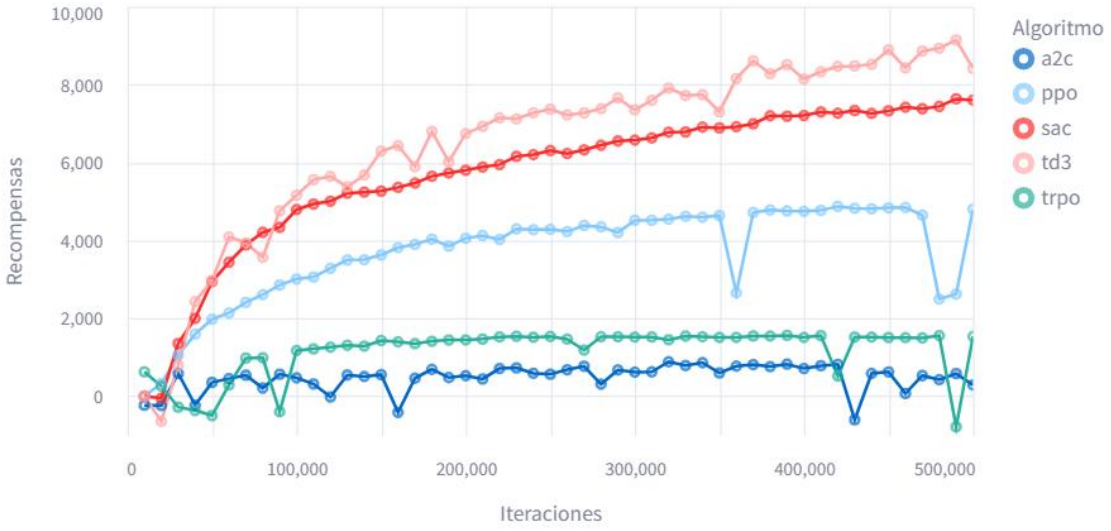


Figure 2 – Reward evolution during the training of several algorithms for Half Cheetah

5. Conclusions

The following conclusions have been drawn from the project's development:

- Hyperparameter optimization was crucial for maximizing the performance of each algorithm, adapting them to the environment.
- Algorithms like SAC and TD3 outperform others in complex environments, while algorithms like PPO and TRPO struggle more.
- The findings of this study are valuable for practical applications in robotics, autonomous control, and other fields that require advanced RL solutions.

6. References

[1] Vandelaer, C. “Reinforcement Learning: An introduction”,August 2022, https://medium.com/@cedric.vandelaer/reinforcement-learning-an-introduction-part-1-4-866695deb4d1

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En este capítulo se hace una introducción de este proyecto despertando el interés del lector por el proyecto y describiendo la motivación del proyecto.

Introducción al Aprendizaje por refuerzo

El aprendizaje por refuerzo (Reinforcement Learning- RL) es una rama del Machine Learning que se centra en cómo los agentes deben tomar acciones en un entorno para conseguir alcanzar uno o varios objetivos *[1].* A diferencia de otros tipos de aprendizaje, como el supervisado, donde el modelo aprende a partir de ejemplos etiquetados, el aprendizaje por refuerzo se basa en la interacción con el entorno. Aquí, un agente aprende mediante prueba y error, recibiendo retroalimentación en forma de recompensas o castigos en función de las acciones que toma.

Algunos de los conceptos más importantes del RL son:

- **Agente**: La entidad que toma decisiones dentro del entorno. El agente interactúa con el entorno realizando acciones y observando las consecuencias de estas acciones.
- **Entorno**: Todo lo que rodea al agente y con lo que interactúa. El entorno responde a las acciones del agente y proporciona nuevos estados y recompensas.
- **Acciones**: Las decisiones que puede tomar el agente en cada momento. Estas acciones afectan el estado del entorno y, por ende, las futuras recompensas que el agente puede recibir.
- **Recompensa**: La señal de retroalimentación que recibe el agente tras tomar una acción. Indica cómo de buena o mala fue una acción hacia la consecución del objetivo final.
- **Política**: La estrategia que sigue el agente para decidir qué acción toma en cada estado. Puede ser determinística (una acción específica para cada estado) o

probabilística (una distribución de probabilidad sobre las posibles acciones para cada estado).

- **Función de valor**: Evalúa la bondad de un estado dado, en términos de la recompensa esperada a largo plazo a partir de ese estado y siguiendo una política particular.

Los algoritmos del aprendizaje por refuerzo pueden clasificarse en dos categorías principales: tabulares y no tabulares *[2]*. Los algoritmos tabulares, como Q-learning, son adecuados para espacios de estados y acciones discretos donde es factible mantener una tabla que registre las valoraciones de todas las combinaciones posibles de estados y acciones. Sin embargo, en muchos problemas del mundo real, los espacios de estados y acciones son continuos, lo que hace inviable el uso de métodos tabulares.

Motivación del proyecto

La motivación de este trabajo surge de la creciente importancia del aprendizaje por refuerzo en la solución de problemas complejos. Particularmente resulta clave la toma de decisiones en espacios continuos, los cuales son comunes en aplicaciones del mundo real. Los espacios continuos presentan un mayor desafío en comparación con los espacios discretos debido al número infinito de posibles acciones y estados.

Esta complejidad hace crucial identificar y optimizar algoritmos que puedan manejar eficazmente dichos entornos. Las aplicaciones en robótica, vehículos autónomos y otros campos se beneficiarían significativamente de los avances en las técnicas de RL. Al comparar diferentes algoritmos, este proyecto busca proporcionar conocimientos que ayuden a seleccionar y ajustar métodos de RL para problemas prácticos del mundo real, contribuyendo en última instancia al avance de los sistemas autónomos y la inteligencia artificial.

Descripción

Este proyecto investiga y compara diversos algoritmos de aprendizaje por refuerzo en espacios continuos, centrándose en métodos populares como TD3 *[3]*, SAC *[4]*, TRPO *[5]*, PPO *[6]* y A2C *[7]*. Estos algoritmos se seleccionan debido a su prominencia en la literatura y su uso extendido en aplicaciones prácticas de RL.

Con el fin de facilitar la implementación de estos algoritmos, se emplea la librería Stable Baselines 3 *[8]*, una herramienta muy útil para trabajar en el área del aprendizaje por refuerzo. Además, para maximizar el rendimiento de cada algoritmo, se realiza una optimización de hiperparámetros. Esto se consigue con herramientas que son capaces de hacer una búsqueda eficiente de los mejores hiperparámetros a través de técnicas avanzadas como la optimización bayesiana.

Estos algoritmos se entrenan en una variedad de entornos muy diversos que incluyen desafíos de control clásico, simulaciones de conducción, y complejas tareas de mecánica en 3D. Cada entorno presenta un conjunto único de dinámicas, permitiendo una evaluación exhaustiva del rendimiento de los algoritmos en diferentes contextos.

El rendimiento de cada algoritmo se evalúa en términos de tres métricas clave:

- **Recompensa Máxima:** Esta métrica mide la capacidad del algoritmo para maximizar las recompensas recibidas a lo largo del tiempo. Un mayor valor de recompensas acumuladas indica un mejor desempeño del algoritmo en la tarea específica.
- **Velocidad de Convergencia:** Esta métrica evalúa cuán rápidamente un algoritmo alcanza un rendimiento estable y óptimo. Una mayor velocidad de convergencia es deseable ya que indica que el algoritmo puede aprender eficientemente con menos iteraciones.
- **Estabilidad:** La estabilidad se refiere a la consistencia del rendimiento del algoritmo a lo largo del tiempo y bajo diferentes condiciones. Un algoritmo estable mostrará

menos variabilidad en su desempeño, lo que es crucial para aplicaciones en el mundo real donde la consistencia es vital.

Finalmente, se crea una interfaz visual interactiva donde los usuarios pueden ver los resultados del entrenamiento de los algoritmos de manera dinámica y visual. Los gráficos y vídeos ayudan a identificar patrones y tendencias en el rendimiento de los algoritmos, proporcionando una comprensión más profunda de sus fortalezas y debilidades. Además, se incluye la posibilidad que los usuarios puedan ejecutar sus propios entrenamientos para el entorno de su elección. Se permite tunear los hiperparámetros y hacer un seguimiento en tiempo real de las recompensas obtenidas en este entrenamiento.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Gymnasium

Gymnasium *[9]* es un conjunto de herramientas desarrollado por OpenAI que proporciona una amplia gama de entornos para trabajar con algoritmos de aprendizaje por refuerzo. Estos entornos simulan diversas tareas y desafíos, desde juegos clásicos con instrucciones simples hasta complejas simulaciones robóticas. Gymnasium ofrece una interfaz estandarizada para interactuar con estos entornos, facilitando la implementación y prueba de estos algoritmos. La variedad de entornos disponibles estandarizados en Gymnasium permite una prueba integral, donde el agente realiza acciones y recibe recompensas en base a unas reglas predefinidas.

Concretamente, los dos entornos más complejos (Half Cheetah y Humanoid) son parte de MuJoCo. MuJoCo (Multi-Joint dynamics with Contact) *[10]*, es un motor de física de uso general diseñado para facilitar la investigación y el desarrollo en campos como la robótica, la biomecánica, los gráficos y la animación, áreas que requieren simulaciones rápidas y precisas de estructuras articuladas interactuando con su entorno. MuJoCo está desarrollado por Google DeepMind y está integrado dentro de Gymnasium.

Optuna

Optuna *[11]* es un marco de optimización de hiperparámetros de código abierto diseñado para automatizar el tedioso proceso de ajuste de hiperparámetros. Utiliza un enfoque basado en muestreo eficiente para buscar las mejores configuraciones de hiperparámetros, apoyando tanto la optimización bayesiana como los algoritmos evolutivos. En el contexto de este proyecto, Optuna intenta mejorar el rendimiento de los algoritmos de RL mediante la búsqueda de las configuraciones óptimas de diversos hiperparámetros. Este proceso de optimización es clave para adaptar los algoritmos a las características específicas de diferentes entornos.

Stable Baselines 3

Stable Baselines 3 *[12]* es una biblioteca que ofrece implementaciones de algoritmos de RL basadas en PyTorch. Esta biblioteca es una nueva versión de Stable Baselines, que a su vez se basa en OpenAI Baselines. Stable Baselines 3 proporciona un conjunto de algoritmos populares de RL, incluyendo los usados en este proyecto: A2C, PPO, SAC, TD3 y TRPO. La biblioteca está diseñada para ser fácil de usar y extender, permitiendo a los investigadores y profesionales entrenar y evaluar estos algoritmos en entornos estándares. En este proyecto, Stable Baselines 3 se utiliza para implementar y entrenar los modelos, aprovechando su integración con los entornos de Gymnasium. Resulta clave la simplicidad y comodidad de Stable Baselines 3 para no perder tiempo lidiando con problemas de versiones.

RL Zoo *[13]*, parte del conjunto de herramientas de Stable Baselines 3, es una colección de algoritmos de aprendizaje por refuerzo predefinidos y configuraciones de hiperparámetros. Proporciona un punto de referencia estandarizado para entrenar algoritmos de RL, facilitando además la optimización de los hiperparámetros para maximizar el rendimiento extraído.

Streamlit

Streamlit *[14]* es un framework permite transformar scripts de Python en aplicaciones web interactivas en minutos. Streamlit incluye características como construir *dashboards*, generar informes o crear aplicaciones de chat de manera rápida y sencilla. Una vez creada la aplicación, puede ser desplegada de manera gratuita para el acceso público. Entre sus principales ventajas destacan la simplicidad, su capacidad de crear interfaces interactivas con los datos y la posibilidad de la edición en vivo.

En este proyecto, Streamlit se utiliza para desarrollar una interfaz gráfica para visualizar los resultados del entrenamiento, proporcionando una forma interactiva de explorar y analizar el rendimiento de los diferentes algoritmos de RL.

Capítulo 3. ESTADO DE LA CUESTIÓN

Implementaciones de RL

FROM SCRATCH

Esta fue la primera opción que se exploró como posible implementación de los algoritmos. Se trata de hacerlo todo "a mano", siguiendo el proceso teórico de los mismos.

Para ilustrarlo mejor, mostramos un ejemplo de una implementación en código de TD3 realizada con TensorFlow. TensorFlow *[15]* es un framework desarrollado por Google para el Deep Learning, utilizado principalmente para trabajar con redes neuronales.

Empezamos con el ReplayBuffer, una memoria para almacenar experiencias de la interacción del agente con el entorno. Tiene 2 funciones principales: guardar experiencias formadas por 5 elementos (estado, acción, recompensa, siguiente estado y terminado) y poder seleccionar aleatoriamente experiencias para así romper el problema que pudiera surgir de presentar siempre el mismo conjunto de entradas durante el proceso de aprendizaje.

```
class ReplayBuffer():
    def __init__(self, maxsize, statedim, naction):
        self.cnt = 0
        self.maxsize = maxsize
        self.state_memory = np.zeros((maxsize, *statedim), dtype=np.float32)
        self.action_memory = np.zeros((maxsize, naction), dtype=np.float32)
        self.reward_memory = np.zeros((maxsize,), dtype=np.float32)
        self.next_state_memory = np.zeros((maxsize, *statedim), dtype=np.float32)
        self.done_memory = np.zeros((maxsize,), dtype=bool)

    def storexp(self, state, action, reward, next_state, done):
        index = self.cnt % self.maxsize
        self.state_memory[index] = state
        self.action_memory[index] = action
        self.reward_memory[index] = reward
        self.next_state_memory[index] = next_state
        self.done_memory[index] = done
        self.cnt += 1

    def sample(self, batch_size):
        max_mem = min(self.cnt, self.maxsize)
```

```
        batch = np.random.choice(max_mem, batch_size, replace= False)

        states = self.state_memory[batch]
        next_states = self.next_state_memory[batch]
        rewards = self.reward_memory[batch]
        actions = self.action_memory[batch]
        dones = self.done_memory[batch]
        return states, next_states, rewards, actions, dones
```

Se usó un esquema de aprendizaje basado en el concepto de Actor-Critic de RL. La clase Actor decide qué acción tomar en función del estado. Al ser en este caso el estado una imagen, primero se necesita colocar capas convolucionales para su procesamiento, seguido de 3 capas completas que son las que producen la acción.

```
class Actor(tf.keras.Model):
    def __init__(self, state_size: int, action_size: int,layer1_size: int,
layer2_size: int
    ):
        """Initialization."""
        super(Actor, self).__init__()

        self.state_size = state_size
        self.action_size = action_size
        # set the convolutional layers to process the image
        self.conv1 = tf.keras.layers.Conv2D(filters=16,
kernel_size=8,activation='relu')
        self.conv2 = tf.keras.layers.Conv2D(filters=8, kernel_size=4,
activation='relu')
        self.conv3 = tf.keras.layers.Conv2D(filters=4, kernel_size=4,
activation='relu')
        self.flatten = tf.keras.layers.Flatten()
        # set the hidden layers
        self.layer1 = tf.keras.layers.Dense(layer1_size, activation='relu')
        self.layer2 = tf.keras.layers.Dense(layer2_size, activation='relu')
        self.policy =  tf.keras.layers.Dense(action_size, activation='tanh')

    def call(self, state):
        conv1 = self.conv1(state)
        conv2 = self.conv2(conv1)
        conv3 = self.conv3(conv2)
        flatten = self.flatten(conv3)
        layer1 = self.layer1(flatten)
        layer2 = self.layer2(layer1)
        policy = self.policy(layer2)
        return policy
```

La clase *CriticQ* es la que vigila lo que hace el Actor, y evalúa cómo de buena es una acción para la consecución del objetivo. Para ello recibe el estado y la acción y devuelve un valor que mide la calidad del movimiento. De nuevo volvemos a tener las capas convolucionales para procesar el estado y las capas densas para obtener el resultado final.

```
class CriticQ(tf.keras.Model):
    def __init__(
        self,
        state_size: int,
        layer1_size: int,
        layer2_size: int
    ):
        """Initialize."""
        super(CriticQ, self).__init__()
         # set the convolutional layers to process the image
        self.conv1 = tf.keras.layers.Conv2D(filters=16,
kernel_size=8,activation='relu')
        self.conv2 = tf.keras.layers.Conv2D(filters=8,
kernel_size=4,activation='relu')
        self.conv3 = tf.keras.layers.Conv2D(filters=4,
kernel_size=4,activation='relu')
        self.flatten = tf.keras.layers.Flatten()

        self.layer1 = tf.keras.layers.Dense(layer1_size, activation='relu')
        self.layer2 = tf.keras.layers.Dense(layer2_size, activation='relu')
        self.value = tf.keras.layers.Dense(1, activation = None)

    def call(self, state, action):
        x = self.conv1(state)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.flatten(x)
        layer1 = self.layer1(tf.concat([x, action], axis=1))
        layer2 = self.layer2(layer1)
        value = self.value(layer2)
        return value
```

Para evitar extender demasiado este apartado, se excluye el resto del código del algoritmo. Cómo resumen, se puede decir que el procedimiento a seguir es el siguiente:

1. Se inicializan los parámetros y las redes neuronales. Habrá 2 de actores y 4 de críticos (2 de *policy* y 2 de *target*). Además, se inicializa el *ReplayBuffer*.
2. En bucle hasta que se llegue al número de iteraciones:

    a. Se selecciona una acción con el actor, añadiéndole ruido y *clipping*.
    b. Se ejecuta la acción en el entorno y se guarda la experiencia en el *ReplayBuffer*
    c. Para entrenar, se muestrea una experiencia y se elige la siguiente acción a través del actor. Luego utilizando el target *critics* se evalúa la acción, actualizando así los pesos de los *policy critics.*
    d. Cada cierto número de iteraciones (suele ser 2), se actualizan los pesos del actor.

Esta implementación from scratch de algoritmos ofrece la ventaja de una mayor flexibilidad y control sobre los detalles específicos del modelo y su entrenamiento, permitiendo optimizaciones personalizadas que pueden mejorar el rendimiento en tareas específicas. Además, es más fácil entender qué se está haciendo en cada punto del proceso, no es una caja negra. Sin embargo, presenta desventajas en términos de complejidad y tiempo de desarrollo, ya que requiere una comprensión profunda de los fundamentos del Reinforcement Learning y, además de ser más propensa a errores y difícil de depurar en comparación con el uso de librerías preconfiguradas y ampliamente utilizadas como Stable Baselines.

MATLAB

MATLAB tiene disponible un toolbox de Reinforcement Learning *[16]*. Es una herramienta poderosa diseñada para facilitar la creación, entrenamiento y simulación de agentes de aprendizaje por refuerzo. Te permite representar políticas y funciones de valor a través de redes neuronales y tablas, siendo utilizadas para el entrenamiento de los algoritmos. El toolbox incluye los algoritmos más populares, como DQN, PPO, SAC y DDPG, aunque también permite crear uno nuevo.

Una de las principales ventajas del toolbox es su capacidad para integrar fácilmente simulaciones de sistemas físicos o entornos personalizados creados con Simulink *[17]*, lo que es particularmente útil para aplicaciones en robótica, control de sistemas, y automatización. MATLAB también proporciona ejemplos y plantillas preconfiguradas que permiten a los usuarios empezar rápidamente y entender los conceptos fundamentales del aprendizaje por refuerzo, haciendo del toolbox una herramienta accesible tanto para investigadores como para ingenieros que buscan aplicar técnicas de aprendizaje por refuerzo en sus proyectos.

La mayor desventaja que conlleva el uso de este Toolbox es que requiere una licencia de Matlab. Actualmente la Universidad nos la da como estudiantes, pero sí sería un problema en el futuro si alguien quisiera continuar con el proyecto fuera de este ámbito, tanto particulares como empresas. Además, Matlab no te aporta la integración que tienes con herramientas como Python, siendo por ejemplo más complejo usar entornos Gymasium, recopilar y representar resultados o reutilizar ciertos bloques de código.

STABLE BASELINES 3

Como ya se comentó en el capítulo 2, Stable Baselines es una librería que reúne implementaciones de algoritmos de aprendizaje por refuerzo con el fin de que sea más fácil trabajar en esta área.

Este ejemplo muestra lo cómodo que resulta el uso de Stable Baselines, donde se entrena TD3 para el entorno de CarRacing. Se aprecia cómo no es necesario definir ni las redes de actores, ni de críticos, ni el ReplayBuffer. Basta con definir el entorno a utilizar, que en este caso es de Gymnasium, y crear la clase del algoritmo, pasándole los parámetros que se desean. En este caso, por ejemplo, fijamos una política MLP, con un ruido normal de desviación estándar 0.1.

A continuación, se llama al método *learn ()* para comenzar el entrenamiento, especificando algún parámetro como puede ser el número de iteraciones. Esta función además va generando logs cada cierto tiempo para informar cuál es el estado actual del entrenamiento, indicando valores como las recompensas medias. El modelo entrenado puede ser guardado en un archivo para luego ser recuperado más tarde.

Finalmente, se puede disfrutar del modelo usando el método *predict ()*, que devuelve la mejor acción en base a el estado actual. Poniendo el render mode del entorno a "human" se observa el comportamiento del agente entrenado.

```
import gymnasium as gym
import numpy as np

from stable_baselines3 import TD3
from stable_baselines3.common.noise import NormalActionNoise

env = gym.make("CarRacing-v2", render_mode="rgb_array")

n_actions = env.action_space.shape[-1]
action_noise = NormalActionNoise(mean=np.zeros(n_actions), sigma=0.1 *
np.ones(n_actions))

model = TD3("MlpPolicy", env, action_noise=action_noise, verbose=1)

model.learn(total_timesteps=100000, log_interval=100)

vec_env = model.get_env()
obs = vec_env.reset()
while True:
    action, _states = model.predict(obs)
    obs, rewards, dones, info = vec_env.step(action)
    vec_env.render("human")
```

Capítulo 4. DEFINICIÓN DEL TRABAJO

Justificación

En el capítulo anterior se mostraron diferentes alternativas posibles de implementación que hubieran podido usarse para el desarrollo de los algoritmos de aprendizaje por refuerzo requeridos en este trabajo. Son soluciones utilizadas por miles de usuarios y que resultan muy útiles para trabajar en este campo. Sin embargo, deben verse como bloques de construcción del proyecto más que como competencia.

Lo que se ofrece con este desarrollo es una plataforma integral que permite hacer un estudio del rendimiento de algoritmos de RL, comparando distintos entornos de espacios continuos. De cara a la comunidad, puede orientar a la gente a decidir qué tipo de algoritmo usar en función del problema a resolver. No sólo eso, sino que da al usuario la posibilidad de configurar los hiperparámetros como quiera y comenzar a entrenar un algoritmo desde cero. Ahora mismo no existen herramientas en el mercado con este tipo de características.

Además, este proyecto se debe ver de cara al futuro. En este momento se está trabajando solo con entornos recogidos en Gymnasium, por lo que la aplicación práctica no es tan directa. Sin embargo, en el futuro se puede explorar la posibilidad de introducir entornos personalizados, cómo por ejemplo de robótica reales, conducción autónoma, etc...

Objetivos

Establecemos los siguientes 3 objetivos principales para el proyecto:

- **Implementación de algoritmos en entornos de espacios continuos utilizando la librería Stable Baselines de Python**: Se empleará la biblioteca de Python Stable Baselines para facilitar la implementación y experimentación de estos algoritmos. Además, se explorará el fine-tuning de parámetros para optimizar el rendimiento de los algoritmos en contextos específicos.
- **Comparación de algoritmos en diferentes entornos evaluando estabilidad, rendimiento y convergencia**: Se analizará el rendimiento de cada algoritmo en términos de las recompensas acumuladas a lo largo del tiempo, considerando también otras características como la velocidad de convergencia y la capacidad de generalización. Esta evaluación permitirá identificar las principales fortalezas y debilidades de cada algoritmo, proporcionando información valiosa para la toma de decisiones en aplicaciones prácticas.
- **Desarrollo de plataforma con los resultados obtenidos:** Se creará una plataforma interactiva que permite al usuario revisar los entrenamientos realizados sobre los distintos entornos. La página incluirá información de cada entorno y de los parámetros usados para el entrenamiento, así como la evolución de las recompensas para los distintos algoritmos. Otra característica clave de esta herramienta es que permita realizar nuevos entrenamientos de los algoritmos, siendo posible configurar los hiperparámetros y seguir en directo este entrenamiento.

Metodología

En primer lugar, se comenzará por un estudio del aprendizaje por refuerzo, entendiendo conceptos e ideas claves. Se aprenderá también sobre los algoritmos más populares, como DDPG, SAC, A2C, HER, TD3, PPO y TRPO, de los que se hablará más adelante. Es necesario partir de estas bases teóricas, ya que no tendría sentido aplicarlos sin saber qué se está haciendo.

A continuación, se elegirá una serie de entornos donde se trabajará. Desde hace mucho tiempo la comunidad científica ha venido trabajando con entornos del registro de Gymnasium, ya que ofrece mayor diversidad, integración y capacidad de comparación entre diversos algoritmos. Además, recordamos que deben ser entornos de espacios continuos, tanto de acciones como de observaciones por decisión del tipo de estudios en los que se centra este trabajo. Por otro lado, como algoritmos cogemos algunos de los más populares, que nos permitan trabajar también en este tipo de entornos: PPO, TRPO, SAC, A2C y TD3. Para la implementación de estos algoritmos se utiliza Stable Baselines 3, como se ha comentado anteriormente. Los parámetros óptimos se obtienen a partir de configuraciones recogidas en RLZoo, optimizando con Optuna cuando sea necesario.

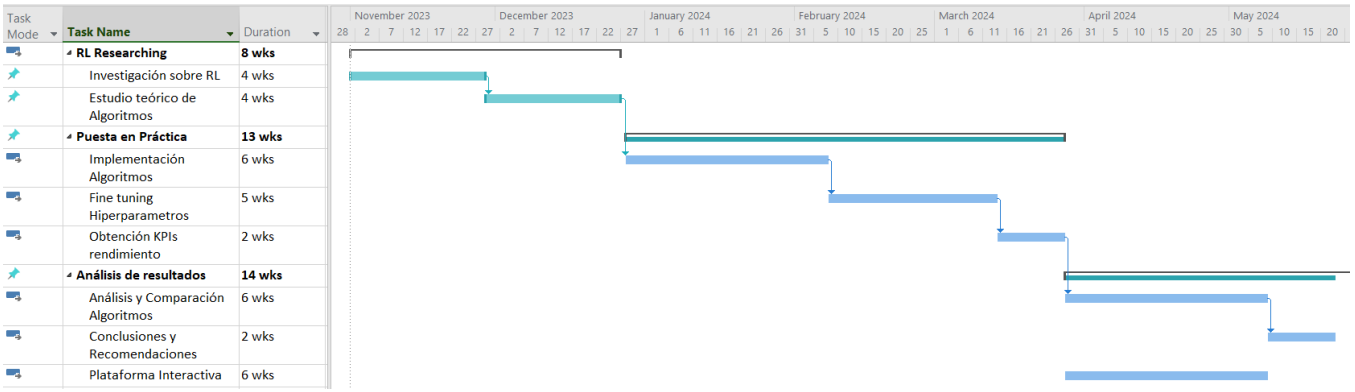
Debido a las altas necesidades computacionales de los entrenamientos, resulta imposible ejecutarlos en un entorno local, por lo que se utilizará el clúster de la Universidad. Para ello, se accederá al nodo Edge conectándose por Jupyter Lab, que permite el lanzamiento de procesos en el clúster. En algún caso, como en el de los entornos de MuJoCo, resulta más fácil trabajar en Google Collab *[18]* por motivo de versiones conflictivas. Adicionalmente, Collab pone a nuestra disposición GPUs, lo que puede acelerar considerablemente el entrenamiento de los modelos, aunque en la versión gratis su uso está limitado.

Finalmente, la plataforma producto final se desarrollará utilizando la librería de Streamlit. Se diseñará esta herramienta para que un usuario pueda interactuar con los resultados obtenidos, sin olvidar también la posibilidad de arrancar un nuevo entrenamiento con parámetros personalizados.

Planificación y Estimación Económica

Todas las librerías y herramientas utilizadas son open-source. Además, los recursos empleados para la ejecución fueron el Clúster de la universidad y la versión gratuita de Google Collab. Por lo tanto, el presupuesto necesario para el desarrollo del proyecto fue de 0 euros. Alternativamente, el uso de GPUs para acelerar el entrenamiento de los algoritmos, hubiese supuesto un desembolso económico importante.

Por otro lado, a continuación, se muestra la planificación temporal del proyecto, donde aparecen las tareas principales junto con su duración. Las tareas se han dividido en 3 fases: *RL Researching*, donde se investiga sobre la teoría del aprendizaje por refuerzo, *Puesta en Práctica*, que implica el proceso de implementación, parametrización y entrenamiento, y *Análisis de resultados*, que trata de realizar una comparación de los algoritmos y la creación de la plataforma integral.



Task Mode	Task Name	Duration
	RL Researching	**8 wks**
	Investigación sobre RL	4 wks
	Estudio teórico de Algoritmos	4 wks
	Puesta en Práctica	**13 wks**
	Implementación Algoritmos	6 wks
	Fine tuning Hiperparametros	5 wks
	Obtención KPIs rendimiento	2 wks
	Análisis de resultados	**14 wks**
	Análisis y Comparación Algoritmos	6 wks
	Conclusiones y Recomendaciones	2 wks
	Plataforma Interactiva	6 wks

…..

.

Capítulo 5. EXPLICACIÓN DEL PROYECTO

5.1 Algoritmos de Aprendizaje por Refuerzo implantados

Este apartado pretende introducir al lector a los principios básicos de los 5 algoritmos de RL empleados en el desarrollo del proyecto, con el fin de facilitar el entendimiento de los resultados.

PPO

Proximal Policy Optimization (PPO) es un algoritmo que se enfoca en mejorar la estabilidad y eficiencia del entrenamiento de agentes. PPO está desarrollado por OpenAI *[19]* y es una variante del método de optimización de políticas que pertenece a la familia de métodos de gradiente de política.

La principal innovación de PPO *[20]* es la utilización de una técnica que evita actualizaciones de políticas que son demasiado grandes, lo que podría desestabilizar el proceso de entrenamiento. Para lograr esto, PPO emplea una función de pérdida que incluye un término de penalización que limita la magnitud del cambio en la política. Esto se hace a través de un "*clip*" en la probabilidad de acciones, que asegura que las actualizaciones estén dentro de un rango confiable. Esta característica hace que PPO sea más fácil de implementar, y es una de las razones por las cuales es ampliamente utilizado en aplicaciones prácticas.

TD3

Twin Delayed Deep Deterministic Policy Gradient (TD3) es un algoritmo de aprendizaje por refuerzo diseñado para mejorar el rendimiento en entornos continuos. TD3 es una extensión del algoritmo DDPG (Deep Deterministic Policy Gradient) *[21]*, que es conocido por ser sensible a la sobreestimación de los valores de Q.

TD3 aborda este problema utilizando tres mejoras clave *[22]*:

- **El uso de dos redes críticas:** En el caso de DDPG, una sola red crítica puede llevar a sobreestimaciones de los valores de Q, lo cual puede degradar el rendimiento del agente. TD3 aborda este problema utilizando dos redes críticas independientes para estimar los valores de Q. Durante el entrenamiento, el algoritmo toma el mínimo de las dos estimaciones de Q para actualizar la política
- **Aplicación de un retraso en la actualización de la red de política:** En los algoritmos actor-critic, la red de política (actor) se actualiza en función de las estimaciones proporcionadas por la red crítica (crítico). Sin embargo, si la política se actualiza demasiado rápido, antes de que la red crítica haya convergido a una estimación estable, esto puede introducir ruido y llevar a un aprendizaje ineficaz. TD3 mitiga este problema aplicando un retraso en la actualización de la red de política.
- **Adición de ruido:** Durante el entrenamiento, los agentes pueden sobre aprender del ruido en las acciones para maximizar las recompensas, generando políticas subóptimas. TD3 introduce el "*target policy smoothing*" para contrarrestar este problema. Esta técnica implica añadir ruido a las acciones seleccionadas por la política objetivo. El ruido añadido es pequeño y está suavizado para mejorar la robustez.

SAC

Soft Actor-Critic (SAC) *[23]* es un algoritmo de aprendizaje por refuerzo basado en la maximización de una política suavizada de entropía, lo cual fomenta la exploración y asegura una robustez en la toma de decisiones. SAC es un algoritmo *off-policy* cuya característica distintiva es el uso de una función de entropía en su objetivo.

SAC maximiza no solo la recompensa esperada sino también la entropía, por lo que se incentiva la exploración al mantener un cierto grado de incertidumbre. Esto ayuda a evitar la convergencia prematura y mejora el rendimiento en entornos complejos. SAC también utiliza dos redes críticas y *target networks*, lo que contribuye a la estabilidad del entrenamiento.

A2C

Advantage Actor-Critic (A2C) *[24]* combina las ventajas de los métodos de actor-crítico con una estructura simplificada y sincronizada. En el marco de los métodos actor-crítico, el actor es responsable de seleccionar las acciones basadas en la política aprendida, mientras que el crítico estima los valores de las acciones o estados, que en este caso se representa mediante la función de ventaja. Esto permite al actor actualizar la política basándose en una estimación más precisa y estable.

A2C es una variante sincrónica del algoritmo A3C (Asynchronous Advantage Actor-Critic) *[24]*. En A3C, múltiples agentes interactúan con el entorno de manera independiente y asincrónica, cada uno actualizando una copia local de la red global después de un cierto número de pasos. Esta asincronía puede aprovechar múltiples núcleos de CPU para acelerar el proceso de entrenamiento y puede ayudar a explorar más efectivamente el espacio de estados. Sin embargo, la naturaleza asincrónica de A3C puede complicar la implementación y dificultar la reproducibilidad y estabilidad del entrenamiento.

A2C, por otro lado, simplifica este proceso al operar de manera sincronizada. En lugar de múltiples agentes que actualizan una red global de manera independiente, A2C utiliza un solo agente que recopila datos de varios entornos en paralelo y actualiza la política de manera sincronizada. Al evitar la asincronía, A2C puede reducir la varianza en las actualizaciones de la política, resultando en un aprendizaje más consistente y predecible.

TRPO

Trust Region Policy Optimization (TRPO) *[25]* es un algoritmo de aprendizaje por refuerzo que impone restricciones en la actualización de la política. La idea central detrás de TRPO es asegurarse de que cada actualización de la política no sea demasiado grande para evitar el desajuste entre la política actual y la nueva política, lo cual podría resultar en una pérdida significativa de rendimiento. Este enfoque se basa en la teoría de regiones de confianza, un concepto común en la optimización numérica, que limita el rango en el que se permiten los cambios durante el proceso de optimización.

Para implementar esta restricción, TRPO maximiza una función de objetivo con la condición de que la desviación de la nueva política esté dentro de un límite predefinido. Esta restricción se expresa matemáticamente mediante la métrica de la divergencia de *Kullback-Leibler* (KL) *[26]*, que mide la diferencia entre dos distribuciones de probabilidad. En el contexto de TRPO, se impone una restricción sobre la divergencia KL entre la política nueva y la anterior, asegurando que la política no cambie demasiado rápido y de manera descontrolada.

La principal ventaja que se obtiene es que TRPO puede permitir cambios más sustanciales en la política sin riesgo de inestabilidad, lo que le permite sobresalir en tareas complejas y garantizar un rendimiento consistente del agente.

5.2 Entrenamiento de los algoritmos

5.2.1 MOUNTAIN CAR CONTINUOUS

5.2.1.1 Descripción del entorno

El entorno Mountain Car Continuous *[27]* es una variación del clásico problema de Mountain Car del paquete Gymnasium, utilizado frecuentemente para experimentar con algoritmos de aprendizaje por refuerzo. En este entorno, un coche debe superar una colina para alcanzar la meta ubicada en la cima de otra colina opuesta. A diferencia de la versión discreta, en Mountain Car Continuous, las acciones disponibles son continuas, lo que añade una capa de complejidad al problema. El objetivo es que el coche acumule suficiente impulso para llegar a la cima de la colina a la derecha, pero el motor del coche es demasiado débil para lograrlo en un solo movimiento directo. La política del agente debe aprender a seleccionar el nivel adecuado de aceleración en cada momento para optimizar su trayectoria y alcanzar la cima de la colina en el menor tiempo posible.

Las 2 observaciones del entorno se muestran en la siguiente tabla:

Observación	Rango	Unidad
Posición del coche a lo largo del eje X	De -Infinito a Infinito	Metros
Velocidad del coche	De -Infinito a Infinito	Metros

Tabla 1: Observaciones del entorno Car Racing

En este entorno, la acción equivale con el valor de la fuerza direccional que se aplica sobre el coche. Esta se representa como un valor continuo en el rango [-1, 1], lo que permite un control más fino y preciso del movimiento del coche en comparación con las acciones discretas.

Además, la recompensa se calcula de la siguiente manera:

- En cada paso se resta $-0.1 * action^2$, para fomentar que se cumpla con el objetivo lo antes posible, sin dar pasos innecesarios
- Si se consigue el objetivo de subir la colina se suma +100. Esto se detecta porque la posición del eje X es mayor o igual a 0.45

El episodio termina cuando el vagón sube la colina o cuando se llega a 999 iteraciones.

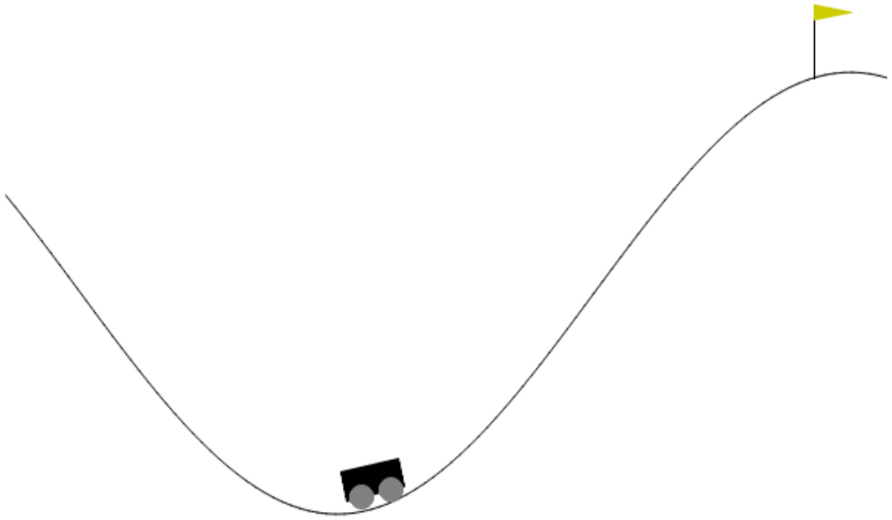


Figura 1: Entorno Mountain Car Continuous

5.2.1.2 Parametrización y entrenamiento

En este apartado se indica los parámetros usados para el entrenamiento de cada uno de los algoritmos. Los parámetros se obtuvieron en parte a través de RL Zoo y con optimizaciones de los mismos en Optuna.

El entrenamiento se hace con estos parámetros durante 500000 iteraciones, guardando la evolución de las recompensas y el mejor modelo obtenido.

A2C

Parámetro	Valor	Explicación
learning_rate	0.0007	Tasa de aprendizaje para el optimizador.
n_steps	5	Número de pasos por actualización de la red neuronal.
gamma	0.99	Factor de descuento para la recompensa.

ent_coef	0.0	Coeficiente de entropía para la regularización.
n_envs	4	Número de entornos paralelos utilizados durante el entrenamiento.
n_steps	100	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=0.0, ortho_init=False)	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal y la desviación estándar inicial.
sde_sample_freq	16	Frecuencia de muestreo para la exploración basada en el ruido de estado dependiente (SDE).
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.

Tabla 2: Hiperparámetros para A2C en Mountain Car Continuous

PPO

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
clip_range	0.1	Rango de recorte para la política PPO.

ent_coef	0.00429	Coeficiente de entropía para la regularización.
gae_lambda	0.9	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.9999	Factor de descuento para la recompensa.
learning_rate	7.77e-05	Tasa de aprendizaje para el optimizador.
max_grad_norm	5	Valor máximo para la normalización del gradiente.
n_envs	1	Número de entornos paralelos utilizados durante el entrenamiento.
n_epochs	10	Número de épocas de actualización por cada lote de datos.
n_steps	8	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-3.29, ortho_init=False)	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal y la desviación estándar inicial.
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.
vf_coef	0.19	Coeficiente de la pérdida de la función de valor.

Tabla 3: Hiperparámetros para PPO en Mountain Car Continuous

TRPO

Parámetro	Valor	Explicación
n_envs	2	Número de entornos paralelos utilizados durante el entrenamiento.
n_timesteps	500000	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
sde_sample_freq	4	Frecuencia de muestreo para la exploración basada en el ruido de estado dependiente (SDE).
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.

Tabla 4: Hiperparámetros para TRPO en Mountain Car Continuous

SAC

Parámetro	Valor	Explicación
batch_size	512	Tamaño del lote utilizado en el entrenamiento.
buffer_size	50000	Tamaño del buffer de reproducción.
ent_coef	0.1	Coeficiente de entropía para la regularización.
gamma	0.9999	Factor de descuento para la recompensa.
gradient_steps	32	Número de pasos de gradiente por actualización de la red neuronal.

learning_rate	0.0003	Tasa de aprendizaje para el optimizador.
learning_starts	0	Número de pasos antes de comenzar el aprendizaje.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-3.67, net_arch=[64, 64])	Argumentos adicionales para la configuración de la política, como la desviación estándar inicial y la arquitectura de la red.
tau	0.01	Factor de suavizado para la actualización del objetivo en algoritmos off-policy.
train_freq	32	Frecuencia de entrenamiento de la red neuronal en pasos de tiempo.
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.

Tabla 5: Hiperparámetros para SAC en Mountain Car Continuous

TD3

Parámetro	Valor	Explicación
n_timesteps	500000	Número total de pasos de tiempo para el entrenamiento.
noise_std	0.5	Desviación estándar del ruido utilizado para la exploración.
noise_type	ornstein-uhlenbeck	Tipo de ruido utilizado para la exploración, en este caso, el ruido de Ornstein-Uhlenbeck.

policy	MlpPolicy	Tipo de política utilizada en el modelo.

Tabla 6: Hiperparámetros para TD3 en Mountain Car Continuous

5.2.2 CAR RACING

5.2.2.1 Descripción del entorno

El entorno CarRacing *[28]* de Gymnasium es una simulación de conducción de coches en la que el objetivo del agente es completar un circuito de carreras lo más rápido posible. El coche debe navegar por una pista que puede tener curvas cerradas y diferentes tipos de desafíos, y el rendimiento del agente se mide en términos de tiempo y precisión al seguir la pista.

Este entorno utiliza una representación visual del estado, donde cada observación es una imagen de 96x96 píxeles en colores RGB que muestra la vista superior del coche y la pista. El agente debe aprender a interpretar estas imágenes para mantener el coche en la pista y evitar salirse de ella. El objetivo final es maximizar la recompensa acumulada, que corresponde a completar la mayor cantidad posible de la pista en el menor tiempo sin errores.

Tenemos 3 posibles acciones:

- **Volante (*Steering*):** Un -1 es girar todo a la izquierda, un 1 es girar todo a la derecha y un 0 es dejarlo recto
- **Acelerador (*Gas*):** Un 0 es no acelerar y un 1 es acelerar al máximo
- **Freno (*Breaking*):** Un 0 es no frenar y un 1 es frenar al máximo

La recompensa está pensada para motivar al agente a completar la pista en el menor tiempo posible y sin salirse del trazado. Por lo tanto, se resta 0.1 por cada *frame* y se suma $1000/N$ donde N es el número total de casillas visitadas de la pista. De esta manera, cuanto más tiempo se mantenga en la pista el coche, más recompensa tendrá.

El episodio termina cuando el coche completa una vuelta, o cuando se sale completamente de la superficie de juego (mucho más allá de la pista).

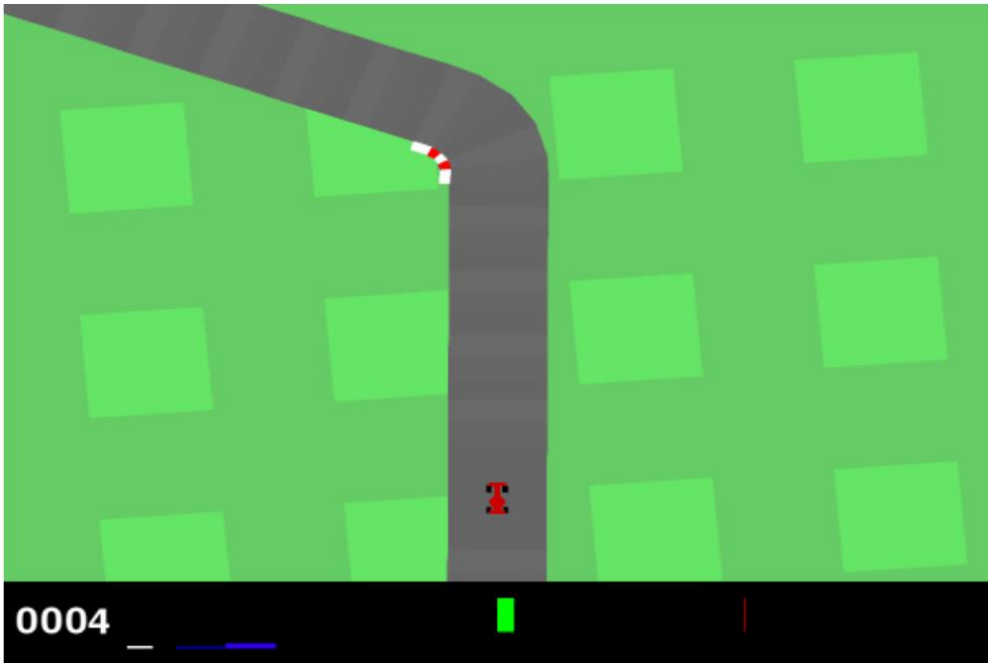


Figura 2: Entorno Car Racing

5.2.2.2 Parametrización y entrenamiento

En este apartado se indica los parámetros usados para el entrenamiento de cada uno de los algoritmos. Los parámetros se obtuvieron en parte a través de RL Zoo y con optimizaciones de los mismos en Optuna.

El entrenamiento se hace con estos parámetros durante 500000 iteraciones, guardando la evolución de las recompensas y el mejor modelo obtenido.

A2C

Parámetro	Valor	Explicación
ent_coef	1.5666550584860516e-06	Coeficiente de entropía para la regularización.

frame_stack	4	Número de fotogramas apilados para la entrada de la red neuronal.
gae_lambda	0.9549244758833236	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.9942776405534832	Factor de descuento para la recompensa.
learning_rate	1.8971962220405576e-05	Tasa de aprendizaje para el optimizador.
max_grad_norm	2.2574480552177127	Valor máximo para la normalización del gradiente.
n_envs	4	Número de entornos paralelos utilizados durante el entrenamiento.
n_steps	64	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	{'norm_obs': False, 'norm_reward': True}	Normalización de las observaciones y recompensas.
normalize_advantage	False	Indica si se normaliza la ventaja durante el entrenamiento.
policy	MlpPolicy	Tipo de política utilizada en el modelo.

policy_kwargs	dict(log_std_init=-4.84, ortho_init=True, activation_fn=nn.Tanh, net_arch=dict(pi=[256], vf=[256]),)	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal y la función de activación.
sde_sample_freq	16	Frecuencia de muestreo para la exploración basada en el ruido de estado dependiente (SDE).
use_rms_prop	False	Indica si se utiliza el optimizador RMSProp.
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.
vf_coef	0.12164696385898476	Coeficiente de la pérdida de la función de valor.

Tabla 7: Hiperparámetros para A2C en Car Racing

PPO

Parámetro	Valor	Explicación
batch_size	128	Tamaño del lote utilizado en el entrenamiento.
clip_range	0.2	Rango de recorte para la política PPO.
ent_coef	0.0	Coeficiente de entropía para la regularización.

frame_stack	2	Número de fotogramas apilados para la entrada de la red neuronal.
gae_lambda	0.95	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.99	Factor de descuento para la recompensa.
learning_rate	0.0001	Tasa de aprendizaje para el optimizador.
max_grad_norm	0.5	Valor máximo para la normalización del gradiente.
n_envs	8	Número de entornos paralelos utilizados durante el entrenamiento.
n_epochs	10	Número de épocas de actualización por cada lote de datos.
n_steps	512	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	{'norm_obs': False, 'norm_reward': True}	Normalización de las observaciones y recompensas.
policy	CnnPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-2, ortho_init=False, activation_fn=nn.GELU,	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal y la función de activación.

	net_arch=dict(pi=[256], vf=[256]),)	
sde_sample_freq	4	Frecuencia de muestreo para la exploración basada en el ruido de estado dependiente (SDE).
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.
vf_coef	0.5	Coeficiente de la pérdida de la función de valor.

Tabla 8: Hiperparámetros para PPO en Car Racing

TRPO

Parámetro	Valor	Explicación
batch_size	128	Tamaño del lote utilizado en el entrenamiento.
frame_stack	2	Número de fotogramas apilados para la entrada de la red neuronal.
gae_lambda	0.95	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.99	Factor de descuento para la recompensa.
learning_rate	0.0001	Tasa de aprendizaje para el optimizador.

n_envs	8	Número de entornos paralelos utilizados durante el entrenamiento.
n_steps	512	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	{'norm_obs': False, 'norm_reward': True}	Normalización de las observaciones y recompensas.
policy	CnnPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-2, ortho_init=False, activation_fn=nn.GELU, net_arch=dict(pi=[256], vf=[256]),)	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal y la función de activación.
sde_sample_freq	4	Frecuencia de muestreo para la exploración basada en el ruido de estado dependiente (SDE).
use_sde	True	Indica si se usa ruido de estado dependiente (SDE) para la exploración.

Tabla 9: Hiperparámetros para TRPO en Car Racing

5.2.3 HUMANOID

5.2.3.1 Descripción del entorno

El entorno Humanoid *[29]* de MuJoCo es un entorno avanzado de simulación utilizado en el aprendizaje por refuerzo, donde el objetivo del agente es controlar un humanoide articulado

para realizar diversas tareas como caminar, correr o mantener el equilibrio. Este entorno es altamente complejo debido a la alta dimensionalidad del espacio de estado y las dinámicas físicas detalladas que emulan movimientos humanos realistas. El humanoide está formado por un torso, cabeza, brazos y piernas. Las piernas están divididas en 3 partes, mientras que los brazos están en 2. Además, cuenta con múltiples articulaciones y grados de libertad, lo que hace que el control y la coordinación sean un desafío significativo,

La simulación incluye una variedad de fuerzas y restricciones físicas, como la gravedad y las colisiones, que el agente debe manejar adecuadamente. Este entorno es ideal para desarrollar y evaluar algoritmos de aprendizaje por refuerzo que pueden generalizar a problemas del mundo real que involucran control de alta precisión y coordinación compleja.

Nos encontramos hasta con 17 posibles acciones continuas del agente, cada una correspondiente al torque aplicado en las articulaciones del humanoide y con un valor que puede oscilar entre -0.4 y 0.4. Las articulaciones se muestran a continuación:

1. *abdomen_y*
2. *abdomen_z*
3. *abdomen_x*
4. *right_hip_x*
5. *right_hip_z*
6. *right_hip_y*
7. *right_knee*
8. *left_hip_x*
9. *left_hip_z*
10. *left_hip_y*
11. *left_knee*
12. *right_shoulder1*
13. *right_shoulder2*
14. *right_elbow*
15. *left_shoulder1*
16. *left_shoulder2*

17. left_elbow

Asimismo, tenemos hasta 45 valores que actúan para formar una observación, recogiendo métricas como la posición, ángulo, velocidad líneal y velocidad angular de las articulaciones. Todos ellos tienen un rango entre -infinito e infinito.

Por último, queda explicar cómo funcionan las recompensas. Cómo ya se ha comentado, el objetivo principal es que el humanoide pueda desplazarse hacia delante sin caerse y de forma eficiente, sin realizar movimientos innecesarios. Para ello tenemos hasta 4 tipos de recompensas:

- **Recompensa por salud**: Cada vez que el humanoide está vivo, recibe una recompensa de valor fijo llamada *healthy_reward*.
- **Recompensa por caminar hacia adelante**: Esta recompensa será positiva si el humanoide camina hacia adelante (en la dirección positiva del eje x).
- **Recompensa por control**: Una recompensa negativa para penalizar al humanoide si la fuerza de control es demasiado grande.
- **Recompensa por contacto**: Una recompensa negativa para penalizar al humanoide si la fuerza de contacto externa es demasiado grande

El cálculo de la recompensa total es

```
reward = healthy_reward + forward_reward - ctrl_cost - contact_cost
```

El final del episodio se da normalmente cuando el humanoide se cae (está *unhealthy*), lo que se mide con la posición z del torso. También es posible la finalización del mismo si se llega a 1000 iteraciones.

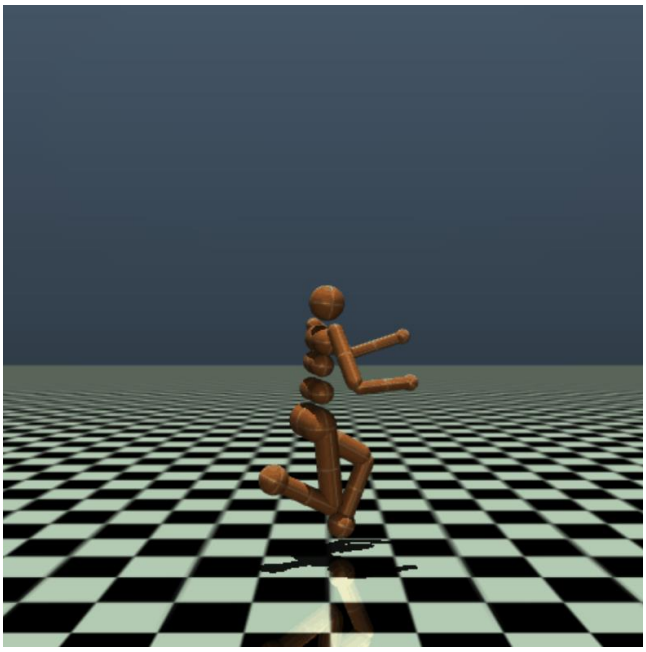


Figura 3: Entorno Humanoid

5.2.3.2 Parametrización y entrenamiento

En este apartado se indica los parámetros usados para el entrenamiento de cada uno de los algoritmos. Los parámetros se obtuvieron en parte a través de RL Zoo y con optimizaciones de los mismos en Optuna.

El entrenamiento se hace con estos parámetros durante 500000 iteraciones, guardando la evolución de las recompensas y el mejor modelo obtenido.

A2C

Parámetro	Valor	Explicación
learning_rate	0.0007	Tasa de aprendizaje para el optimizador.
n_steps	5	Número de pasos por actualización de la red neuronal.
gamma	0.99	Factor de descuento para la recompensa.

ent_coef	0.0	Coeficiente de entropía para la regularización.
n_envs	4	Número de entornos paralelos utilizados durante el entrenamiento.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.

Tabla 10: Hiperparámetros para A2C en Humanoid

PPO

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
clip_range	0.3	Rango de recorte para la política PPO.
ent_coef	0.00238306	Coeficiente de entropía para la regularización.
gae_lambda	0.9	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.95	Factor de descuento para la recompensa.
learning_rate	3.56987e-05	Tasa de aprendizaje para el optimizador.

max_grad_norm	2	Valor máximo para la normalización del gradiente.
n_envs	1	Número de entornos paralelos utilizados durante el entrenamiento.
n_epochs	5	Número de épocas de actualización por cada lote de datos.
n_steps	512	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-2, ortho_init=False, activation_fn=nn.ReLU, net_arch=dict(pi=[256, 256], vf=[256, 256]))	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal, función de activación y arquitectura de la red.
vf_coef	0.431892	Coeficiente de la pérdida de la función de valor.

Tabla 11 Hiperparámetros para PPO en Humanoid

TRPO

Parámetro	Valor	Explicación
batch_size	128	Tamaño del lote utilizado en el entrenamiento.
cg_damping	0.1	Amortiguación para el método de gradiente conjugado (CG).
cg_max_steps	25	Número máximo de pasos para el método de gradiente conjugado (CG).
gae_lambda	0.95	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.99	Factor de descuento para la recompensa.
learning_rate	0.001	Tasa de aprendizaje para el optimizador.
n_critic_updates	20	Número de actualizaciones del crítico por iteración.
n_envs	2	Número de entornos paralelos utilizados durante el entrenamiento.
n_steps	1024	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.

normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
sub_sampling_factor	1	Factor de submuestreo utilizado en algún aspecto específico del modelo.

Tabla 12: Hiperparámetros para TRPO en Humanoid

SAC

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
buffer_size	1000000	Tamaño del buffer de reproducción.
gamma	0.99	Factor de descuento para la recompensa.
gradient_steps	1	Número de pasos de gradiente por actualización de la red neuronal.
learning_rate	0.0003	Tasa de aprendizaje para el optimizador.
learning_starts	10000	Número de pasos antes de comenzar el aprendizaje.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
tau	0.005	Factor de suavizado para la actualización del objetivo en algoritmos off-policy.

train_freq	1	Frecuencia de entrenamiento de la red neuronal en pasos de tiempo.

Tabla 13: Hiperparámetros para SAC en Humanoid

TD3

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
gradient_steps	1	Número de pasos de gradiente por actualización de la red neuronal.
learning_rate	0.001	Tasa de aprendizaje para el optimizador.
learning_starts	10000	Número de pasos antes de comenzar el aprendizaje.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
noise_std	0.1	Desviación estándar del ruido utilizado para la exploración.
noise_type	normal	Tipo de ruido utilizado para la exploración, en este caso, ruido normal.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict (net_arch= [400, 300])	Argumentos adicionales para la configuración de la política, como la arquitectura de la red.
train_freq	1	Frecuencia de entrenamiento de la red neuronal en pasos de tiempo.

Tabla 14: Hiperparámetros para TD3 en Humanoid

5.2.4 HALF CHEETAH

5.2.4.1 Descripción del entorno

El entorno HalfCheetah *[30]* de MuJoCo es una simulación de un robot bípedo similar a un guepardo. El objetivo del agente en este entorno es aprenderse a mover hacia adelante lo más rápido posible, maximizando la distancia recorrida en el menor tiempo. El robot está formado por 9 partes, unidas por 8 articulaciones. Al estar tanto el torso como la cabeza fija, el torque para el movimiento se debe aplicar en las 6 articulaciones restantes.

El HalfCheetah debe aprender a coordinar sus extremidades para generar un movimiento fluido y rápido, maximizando una función de recompensa que generalmente está relacionada con la velocidad hacia adelante y la eficiencia energética. El entorno incluye diversas fuerzas y restricciones físicas, como la gravedad y las colisiones, que el agente debe manejar adecuadamente.

Las acciones entonces se entienden como los torques que se aplican a las siguientes articulaciones, y que pueden estar entre -1 y 1:

1. *Bthigh* (Muslo trasero)
2. *Bshin* (Espinilla trasera)
3. *Bfoot* (Pie trasero)
4. *Fthigh* (Muslo trasero)
5. *Fshin* (Espinilla trasera)
6. *Ffoot* (Pie trasero)

De manera similar al Humanoide, las observaciones representan 17 medidas distintas de las partes del cuerpo del guepardo, entre las que se incluyen velocidad, ángulo, posición y velocidad angular.

También cabe destacar cómo funciona la recompensa, que busca forzar al Half Cheetah a correr hacia delante con velocidad. Con este fin, la recompensa tiene 2 componentes principales:

- **Recompensa por velocidad hacia adelante**: Se mide como la diferencia en la posición x del centro de masa del Half Cheetah entre pasos de tiempo. Instiga al agente a avanzar lo más rápido posible por el eje x.
- **Recompensa por control:** Una recompensa negativa que penaliza por el uso excesivo de fuerzas de control. Se mide como el cuadrado de los torques aplicados a las articulaciones.

La recompensa total es calculada como:

```
reward = forward_reward - ctrl_cost
```

Para acabar, decir que el Half Cheetah no tiene una condición de finalización por caída como tenía el Humanoid. El episodio termina cuando la longitud del mismo llegue a 1000.

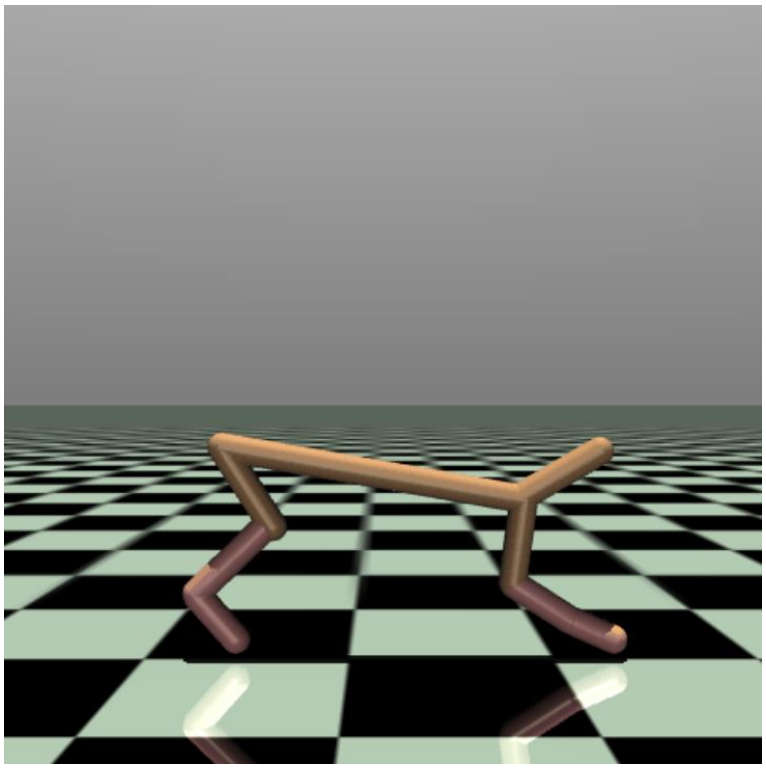


Figura 4: Entorno Half Cheetah

5.2.4.2 Parametrización y entrenamiento

En este apartado se indica los parámetros usados para el entrenamiento de cada uno de los algoritmos. Los parámetros se obtuvieron en parte a través de RL Zoo y con optimizaciones de los mismos en Optuna.

El entrenamiento se hace con estos parámetros durante 500000 iteraciones, guardando la evolución de las recompensas y el mejor modelo obtenido.

A2C

Parámetro	Valor	Explicación
learning_rate	0.0007	Tasa de aprendizaje para el optimizador.
n_steps	5	Número de pasos por actualización de la red neuronal.
gamma	0.99	Factor de descuento para la recompensa.
ent_coef	0.0	Coeficiente de entropía para la regularización.
n_envs	4	Número de entornos paralelos utilizados durante el entrenamiento.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.

Tabla 15: Hiperparámetros para A2C en Half Cheetah

PPO

Parámetro	Valor	Explicación
batch_size	64	Tamaño del lote utilizado en el entrenamiento.
clip_range	0.1	Rango de recorte para la política PPO.
ent_coef	0.000401762	Coeficiente de entropía para la regularización.
gae_lambda	0.92	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.98	Factor de descuento para la recompensa.
learning_rate	2.0633e-05	Tasa de aprendizaje para el optimizador.
max_grad_norm	0.8	Valor máximo para la normalización del gradiente.
n_envs	1	Número de entornos paralelos utilizados durante el entrenamiento.
n_epochs	20	Número de épocas de actualización por cada lote de datos.
n_steps	512	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.

normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict(log_std_init=-2, ortho_init=False, activation_fn=nn.ReLU, net_arch=dict(pi=[256, 256], vf=[256, 256]))	Argumentos adicionales para la configuración de la política, como la inicialización ortogonal, función de activación y arquitectura de la red.
vf_coef	0.58096	Coeficiente de la pérdida de la función de valor.

Tabla 16: Hiperparámetros para PPO en Half Cheetah

TRPO

Parámetro	Valor	Explicación
batch_size	128	Tamaño del lote utilizado en el entrenamiento.
cg_damping	0.1	Amortiguación para el método de gradiente conjugado (CG).
cg_max_steps	25	Número máximo de pasos para el método de gradiente conjugado (CG).
gae_lambda	0.95	Parámetro lambda para la ventaja generalizada (GAE).
gamma	0.99	Factor de descuento para la recompensa.
learning_rate	0.001	Tasa de aprendizaje para el optimizador.

n_critic_updates	20	Número de actualizaciones del crítico por iteración.
n_envs	2	Número de entornos paralelos utilizados durante el entrenamiento.
n_steps	1024	Número de pasos por actualización de la red neuronal.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
normalize	True	Indica si se normalizan las observaciones y recompensas.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
sub_sampling_factor	1	Factor de submuestreo utilizado en algún aspecto específico del modelo.
target_kl	0.04	Tasa de divergencia KL objetivo para la actualización de la política.

Tabla 17: Hiperparámetros para TRPO en Half Cheetah

SAC

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
buffer_size	1000000	Tamaño del buffer de reproducción.
gamma	0.99	Factor de descuento para la recompensa.

gradient_steps	1	Número de pasos de gradiente por actualización de la red neuronal.
learning_rate	0.0003	Tasa de aprendizaje para el optimizador.
learning_starts	10000	Número de pasos antes de comenzar el aprendizaje.
n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
tau	0.005	Factor de suavizado para la actualización del objetivo en algoritmos off-policy.
train_freq	1	Frecuencia de entrenamiento de la red neuronal en pasos de tiempo.

Tabla 18: Hiperparámetros para SAC en Half Cheetah

TD3

Parámetro	Valor	Explicación
batch_size	256	Tamaño del lote utilizado en el entrenamiento.
gradient_steps	1	Número de pasos de gradiente por actualización de la red neuronal.
learning_rate	0.001	Tasa de aprendizaje para el optimizador.
learning_starts	10000	Número de pasos antes de comenzar el aprendizaje.

n_timesteps	500000.0	Número total de pasos de tiempo para el entrenamiento.
noise_std	0.1	Desviación estándar del ruido utilizado para la exploración.
noise_type	normal	Tipo de ruido utilizado para la exploración, en este caso, ruido normal.
policy	MlpPolicy	Tipo de política utilizada en el modelo.
policy_kwargs	dict (net_arch= [400, 300])	Argumentos adicionales para la configuración de la política, como la arquitectura de la red.
train_freq	1	Frecuencia de entrenamiento de la red neuronal en pasos de tiempo.

Tabla 19: Hiperparámetros para TD3 en Half Cheetah

5.3 Guía de usuario de la Plataforma Interactiva

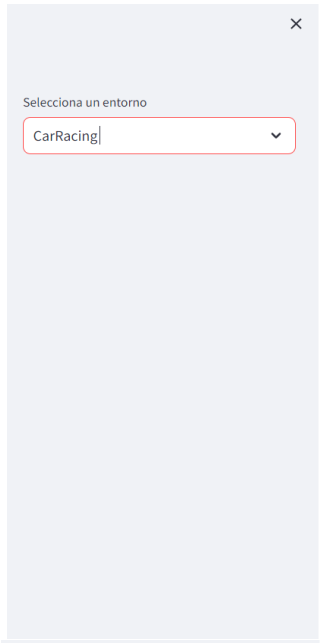
Finalmente se ha desarrollado una plataforma interactiva utilizando Streamlit que permite a los usuarios explorar los resultados de modelos de aprendizaje por refuerzo y también entrenar nuevos modelos ajustando los parámetros según sus preferencias. Streamlit ofrece una variedad de componentes útiles *[31]*, como sliders, botones y formularios, que facilitan la creación de interfaces de usuario intuitivas y dinámicas. Por ejemplo, mediante sliders, los usuarios pueden ajustar hiperparámetros y con solo un clic de botón, pueden iniciar el entrenamiento de un nuevo modelo.

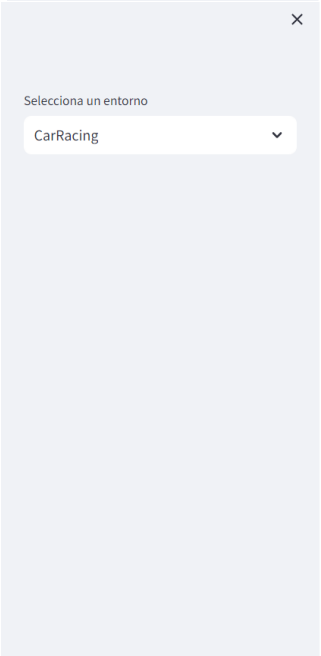
La integración de gráficos interactivos y tablas dinámicas permite visualizar los resultados y el rendimiento de los modelos, proporcionando una experiencia de usuario enriquecida y fluida. Esta plataforma no solo agiliza el proceso de experimentación con modelos de aprendizaje por refuerzo, sino que también democratiza el acceso a técnicas avanzadas de machine learning al hacerlas accesibles a través de una interfaz web simple y directa.

Al entrar a la misma, el usuario ve la pantalla que aparece debajo. Se le pide elegir uno de los cuatro entornos: Car Racing, Mountain Car Continuous, Humanoid y Half Cheetah.

Figura 5: Pantalla inicial de la plataforma

Una vez dentro del entorno se muestra una descripción breve del mismo, explicando también cómo funciona el reparto de las recompensas, las acciones y las observaciones. Se incluye además un breve video de cómo se comporta un agente antes del entrenamiento. Debajo podemos ver un ejemplo para el entorno de Car Racing.





CarRacing

El entorno CarRacing simula la experiencia de conducir un automóvil en una pista de carreras. Con gráficos realistas y controles precisos, los agentes de aprendizaje pueden experimentar diferentes condiciones de la carretera, como curvas pronunciadas, obstáculos y cambios de superficie. Este entorno desafía a los agentes a dominar la habilidad de conducir de manera eficiente, optimizando la velocidad, el agarre y la dirección para completar la pista en el menor tiempo posible sin salirse de la carretera o chocar.

Caracterización del entorno

El estado del entorno se describe mediante una matriz de píxeles que representan una vista aérea de la pista y el coche, lo que convierte el problema en uno de percepción visual y control continuo. La observación es una imagen RGB de dimensiones **96x96 píxeles**.

La recompensa está diseñada para incentivar al agente a seguir la pista de manera eficiente y rápida. El agente puede recibir:

- **Recompensa por avanzar**: Se otorgan pequeñas recompensas por cada segmento de pista avanzado. Esto anima al agente a intentar ir lo más rápido posible y manteniendo el coche en pista.
- **Penalización por salirse de la pista**: Se aplica una penalización cuando el coche sale de la pista, lo que desalienta el comportamiento subóptimo.

Las acciones disponibles son continuas y representan los comandos de control del coche, incluyendo la dirección del volante, la aceleración y el freno.

- **Dirección del volante** (*steer*): Un valor continuo que varía entre *-1* (giro completo a la izquierda) y *1* (giro completo a la derecha).
- **Aceleración** (*gas*): Un valor continuo que varía entre *0* y *1*.
- **Freno** (*brake*): Un valor continuo que varía entre *0* y *1*.

El episodio termina si el agente completa un giro de la pista. El episodio también puede terminar si el coche se sale de la pista durante un período prolongado o si se supera un número máximo de pasos.

Agente sin entrenar

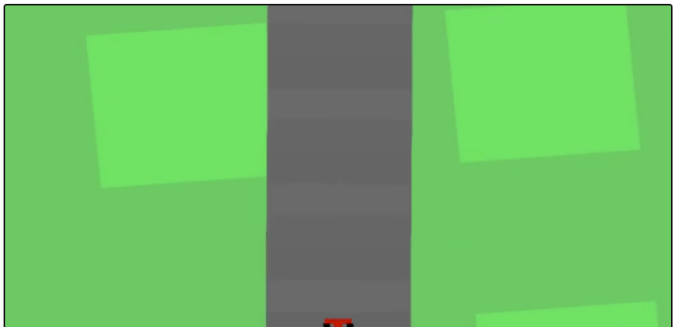


Figura 6: Explicación y caracterización del entorno Car Racing

Si se sigue bajando en la página, se le ofrece al usuario 2 opciones: explorar los resultados de entrenamientos previos o entrenar su propio modelo:

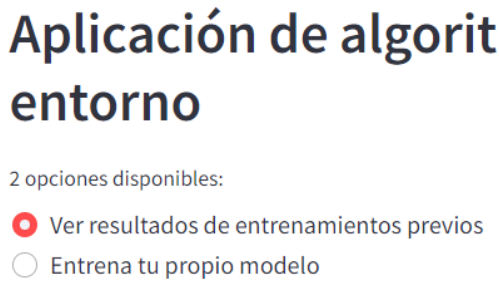


Figura 7: Elección del usuario entre modelos preentrenados o nuevo modelo

5.3.1 VISUALIZACIÓN DE RESULTADOS

Si el usuario elige la primera opción, lo primero que se muestra es una tabla con la mayor recompensa conseguida para cada uno de los algoritmos implementados. Es una forma rápida de ver cuál de ellos en principio ha sido el que mejor se ha comportado.

Ranking	Algoritmo	Mayor recompensa
1	td3	9142
2	sac	7633
3	ppo	4873
4	trpo	1561
5	a2c	882

Figura 8: Mayor recompensa obtenida para cada algoritmo

Además, para tener una mejor visión de los resultados, se dibujan las evoluciones de las recompensas para los distintos algoritmos. Aquí se pueden apreciar las tendencias, la velocidad de aprendizaje y convergencia.

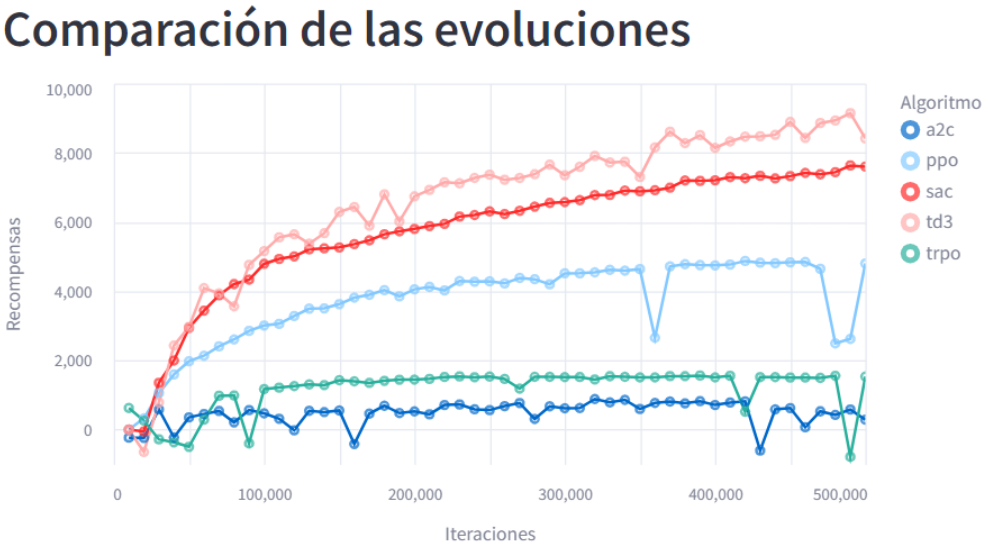


Figura 9: Comparación de las evoluciones de los 5 algoritmos implementados

Finalmente, se le da la opción al usuario de elegir un algoritmo en concreto para explorar los resultados más en detalle. En este caso se muestran los 5 algoritmos implementados: A2C, PPO, SAC, TD3 y TRPO.

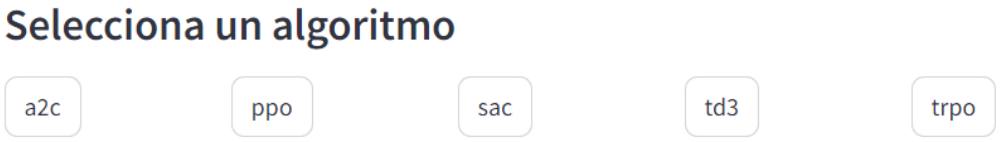


Figura 10: Elección del algoritmo por el usuario

Cuando se pulsa en un algoritmo (en este caso A2C), se muestra la mayor recompensa conseguida del mismo y los hiperparámetros que han sido utilizados para el entrenamiento del modelo.

A2C

```
Mayor recompensa conseguida:  882
```

Parámetros de entrenamiento usados:

```
[('learning_rate',0.0007),
             ('n_steps',5),
             ('gamma',0.99),
             ('ent_coef', 0.0),
             ('n_envs', 4),
             ('n_steps', 5),
             ('n_timesteps', 500000.0),
             ('normalize', True),
             ('policy', 'MlpPolicy')]
```

Figura 11: Recompensa e hiperparámetros del entrenamiento de A2C

Además, se dibuja individualmente la evolución de la recompensa media, para el entrenamiento de A2C en ese entorno.

Figura 12: Evolución de las recompensas de A2C

Por último, se incluye un vídeo de cómo el agente se comporta siguiendo el mejor modelo obtenido durante el entrenamiento:

Figura 13: Video del agente con el mejor modelo

5.3.2 ENTRENA TU PROPIO MODELO

En el caso de que el usuario tenga curiosidad y quiera probar su propio modelo, puede hacerlo con esta opción. Empezará eligiendo el algoritmo a implementar entre los disponibles.

Además, tiene la posibilidad de ajustar los hiperparámetros a su gusto. Los que aparecen por defecto, son los que se han empleado para los entrenamientos anteriores.

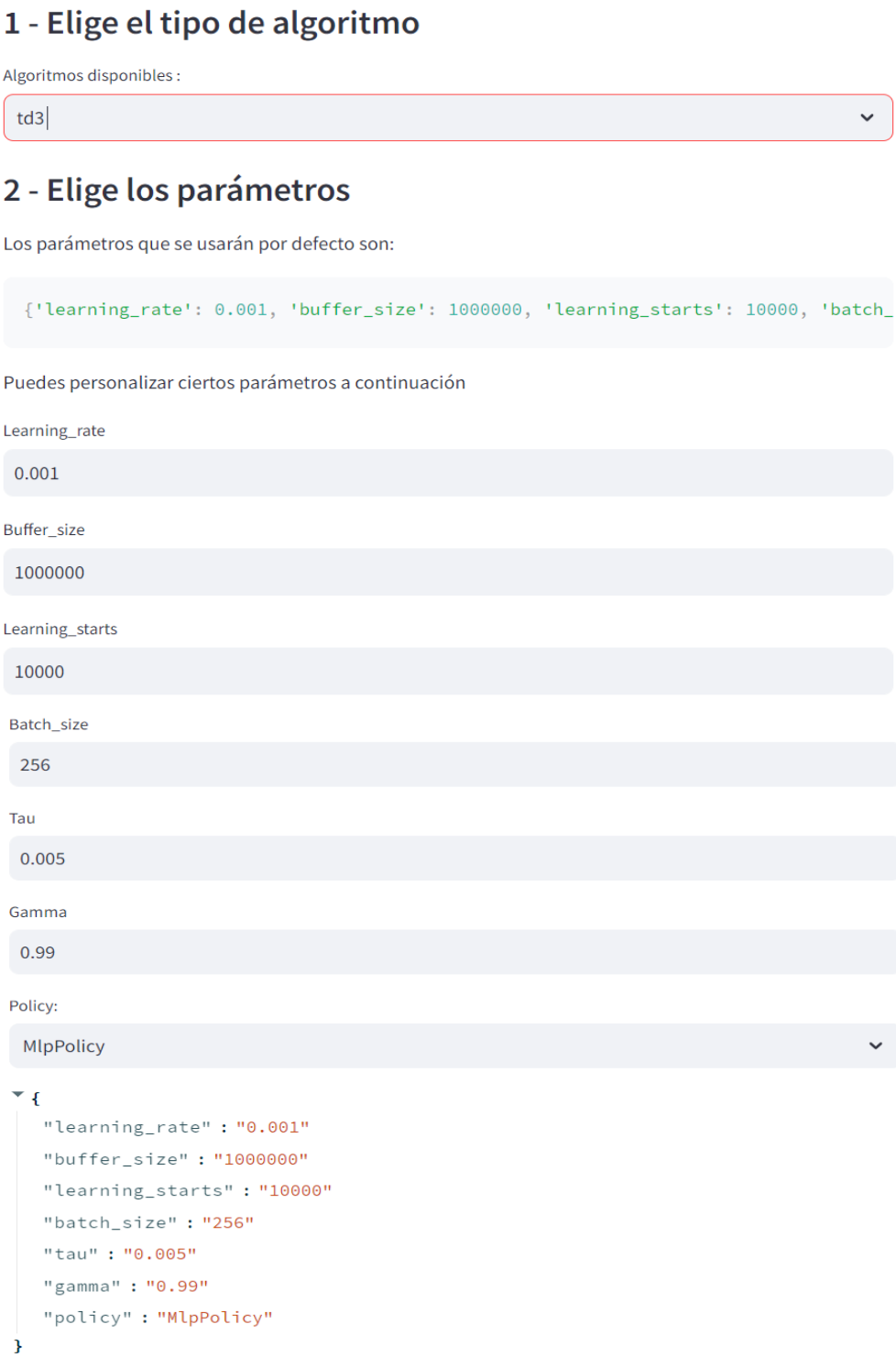


Figura 14: Personalización de hiperparámetros para Entrena tu propio Modelo

Finalmente se deberá elegir cómo de largo quiere que sea el entrenamiento, medido en el número de iteraciones (*timesteps*). Debe ser un valor entre 10000 y 1 millón.

El entrenamiento se iniciará cuando el usuario pulse el botón "*Empezar a entrenar!"*. Se podrá hacer un seguimiento en directo del mismo, viendo la evolución de las recompensas a medida que van pasando las iteraciones. Adicionalmente se incluye una barra de progreso que indica en qué punto del entrenamiento se está en cada momento.

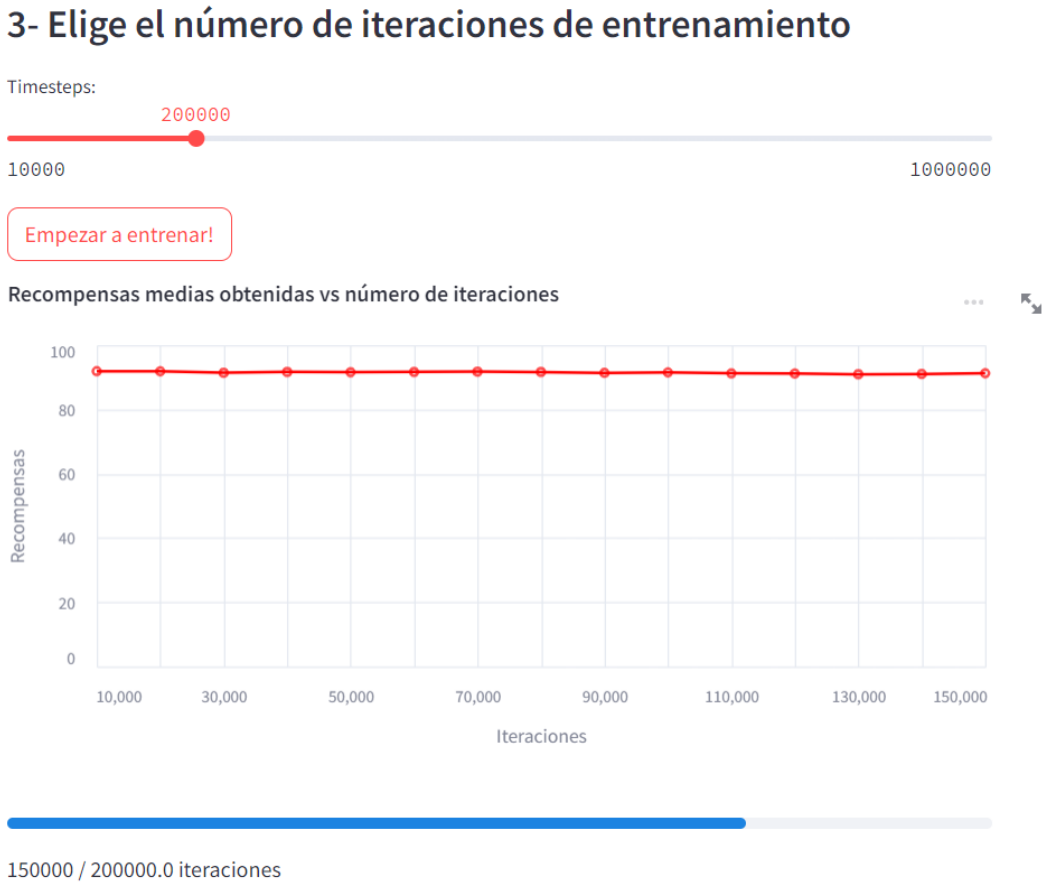


Figura 15: Inicio de entrenamiento y seguimiento para Entrena tu propio Modelo

Es preciso recordar al usuario que la velocidad de entrenamiento depende tanto de la combinación algoritmo/entorno, como de la capacidad de computación del sistema donde se ejecuta. Deberá ser paciente en algunos casos si parece que no avanza el entrenamiento.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se mostrarán los resultados de los entrenamientos realizados en términos de recompensa, comparando los distintos algoritmos para los 4 entornos. Se incluyen también los vídeos del agente entrenado. Al no poderlos incluirlos directamente en el Word por su elevado tamaño, se añaden como enlaces.

6.1 Mountain Car Continuous

En este caso recordemos que el objetivo era hacer subir el vagón a lo alto de la colina. La recompensa dada por conseguir esta meta era de 100, mientras que se restaba $-0.1 * action^2$ para evitar movimientos innecesarios. Por lo tanto, la mayor recompensa que se puede obtener estará cerca de 100, dependiendo de la eficiencia del agente.

En la siguiente tabla se muestra la mayor recompensa obtenida para cada uno de los algoritmos entrenados. Se aprecia como todos consiguen el objetivo, ya que es un entorno relativamente sencillo. Sin embargo, hay una diferencia en la forma de conseguirlo: SAC parece ser el más eficiente, subiendo la colina con menos acciones (recompensa de 98), seguido por TRPO (recompensa de 96), A2C y TD3 (ambos recompensa de 94). Por último, PPO parece tener algo más de problemas al tener la recompensa en 93, aunque tampoco hay una gran diferencia.

Ranking	Algoritmo	Mayor recompensa
1	sac	98
2	trpo	96
3	a2c	94
4	td3	94
5	ppo	93

Figura 16: Mayores recompensas obtenidas en Mountain Car Continuous

Aquí también es interesante ver cómo evoluciona esta recompensa a medida que avanza el entrenamiento para los 5 casos.

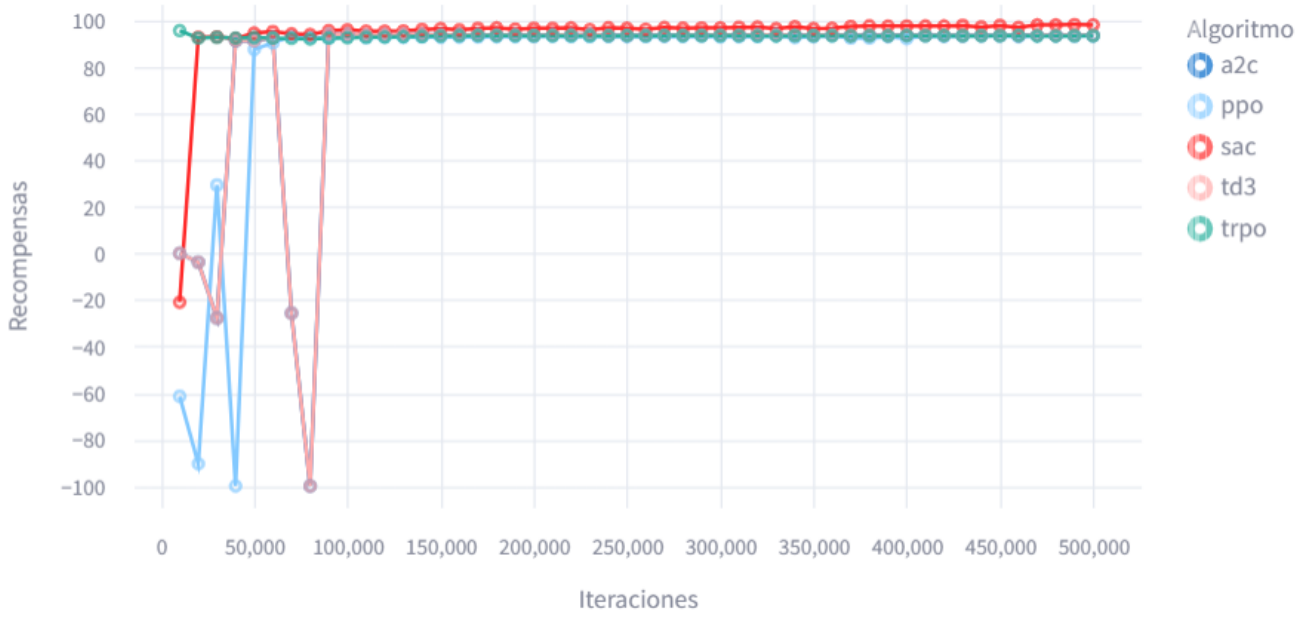


Figura 17: Evolución de recompensas para los algoritmos en Mountain Car Continuous

Es general vemos cómo a ninguno de ellos les cuesta especialmente acercarse a los 100 de recompensa, lo consiguen en menos de 100.000 iteraciones. Sin embargo, dentro de lo que cabe hay diferencias importantes en las convergencias.

TRPO parece conseguir el objetivo nada más comenzar el entrenamiento, en los 10000 primeros *timesteps*. SAC también parece ser bastante rápido, alcanzándolo en 20000. Recordemos además que fue SAC el que mayor recompensa tuvo de los 5. Para PPO y TD3, el entorno les resulta algo más complejo, llegando a converger en 60000 y 90000 iteraciones respectivamente. Esto unido a que son los dos con las recompensas máximas más bajas, nos dice que son los que peor parecen rendir.

Los enlaces a los videos de los 5 agentes entrenados resolviendo el entorno de Mountain Car Continuous se muestran a continuación:

- a2c-MountainCarContinuous.mp4

- ppo-MountainCarContinuous.mp4
- trpo-MountainCarContinuous.mp4
- sac-MountainCarContinuous.mp4
- td3-MountainCarContinuous.mp4

Se puede ver como en todos ellos se cumple el objetivo, y el vagón puede subir la cuesta sin dificultad. Tampoco se aprecia una gran diferencia en la cantidad de acciones requeridas para conseguir el objetivo.

Para resumir, Mountain Car Continuous es un entorno relativamente sencillo de Gymnasium, que los 5 algoritmos estudiados pueden resolver con facilidad y sin necesidad de un largo entrenamiento. Aun así, TD3 y PPO parecen tardar algo más en conseguir este objetivo que el resto.

Considero que la dificultad de TD3 y PPO para converger se debe a sus estrategias de exploración y actualización. TD3 utiliza ruido añadido a las acciones y actualizaciones diferidas, lo que puede ralentizar el aprendizaje inicial en entornos simples. PPO, por su parte, es conservador en sus actualizaciones debido a su objetivo de optimización con penalización por desviaciones grandes y es muy sensible a los ajustes de hiperparámetros. En comparación, algoritmos como A2C, TRPO y SAC pueden manejar mejor estos entornos debido a sus diferentes enfoques en la actualización de políticas y la exploración, resultando en una convergencia más rápida.

6.2 Car Racing

La recompensa en este entorno se reparte en función de la pista que recorre el coche, teniendo una penalización por el tiempo que tarda. Un coche que realiza la vuelta sin salirse tendrá una recompensa de 1000, aunque luego se debe tener en cuenta que se descuenta 0.1 por cada *frame* que tarda. Una muy buena vuelta estará entre 900 y 1000.

En la siguiente tabla se muestran las recompensas máximas obtenidas para los 3 casos. Se informa al lector que no se han podido implementar SAC y TD3 debido a su incompatibilidad con el *Wrapper* del entorno.

Ranking	Algoritmo	Mayor recompensa
1	ppo	799
2	trpo	592
3	a2c	-40

Figura 18: Mayores recompensas obtenidas en Car Racing

PPO parece conseguir un muy buen rendimiento, que no llega a ser perfecto, pero destaca por encima del resto con 799 de recompensa. TRPO no se queda demasiado atrás, con 592 se puede decir que, aunque con dificultades, el algoritmo consigue avances en este entorno. Por el contrario, A2C parece totalmente incapaz de desenvolverse en Car Racing. Una recompensa negativa de -40 indica que el coche es lento y se sale continuamente del trazado del circuito.

La evolución de las recompensas de la Figura 19 cuenta una historia similar. A2C tiene alguna desviación, pero se estabiliza rápido resultando siempre en recompensas entre -50 y -100, que indican un rendimiento bastante pobre. Este no es el caso de TRPO: después de un inicio lento (obteniendo recompensas negativas), a partir de cerca de las 150000 iteraciones existe un punto de inflexión donde el agente mejora significativamente su comportamiento. Desde las 200000 hasta el final del entrenamiento las recompensas obtenidas tienen bastante

variabilidad, fluctuando entre 300 y 600. No se aprecia una tendencia clara de mejora en este tramo.

Por último, nos fijamos en la línea de PPO. Desde las primeras iteraciones se aprecia un modelo bastante mejor que los otros dos estudiados. La curva de aprendizaje es bastante alta alcanzando ya los 750 en torno a 150000 *timesteps*. A partir de ese momento, las recompensas se estabilizan más, y salvo algunos picos (como el de 799), no parece que haya una mejora significativa. Es importante destacar que es más consistente que TRPO al tener menos variabilidad de las recompensas.

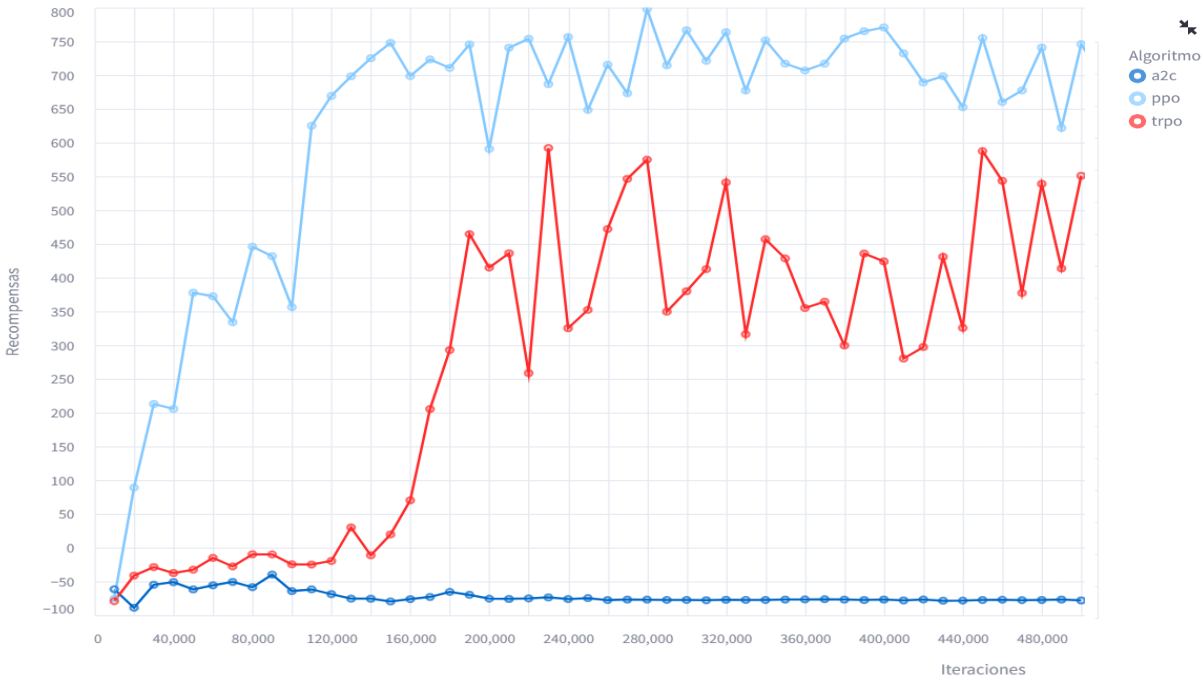


Figura 19: Evolución de recompensas para los algoritmos en Car Racing

Se incluyen ahora los enlaces a los videos de los 3 agentes entrenados resolviendo el entorno de Car Racing:

- a2c-CarRacing.mp4
- ppo-CarRacing.mp4

- trpo-CarRacing.mp4

Los vídeos del coche respaldan lo observado en las recompensas. Con A2C el coche no avanza prácticamente nada, y por ello obtiene recompensas negativas. TRPO mejora bastante, el coche va rápido e intenta seguir un poco mejor el circuito, aunque a veces tiene salidas de pista y no consigue adaptarse del todo.

PPO sí que parece conseguir que el coche mejore el tiempo por vuelta de los otros dos, limitando los fallos y haciendo relativamente bien las curvas. De todas formas, sigue sin ser perfecto, hay momentos donde el coche hace alguna excursión o trompo que perjudican la recompensa. Otro detalle que se observa en el vídeo de PPO es que en alguna ocasión donde el coche se sale del trazado, consigue volver a él, demostrando un buen conocimiento del entorno.

Para cerrar, podemos decir que, Car Racing es un entorno relativamente complejo, especialmente si lo comparamos con Mountain Car Continuous. PPO sobresale debido a sus actualizaciones estables y seguras, así como su exploración eficiente y flexibilidad, lo que le permite adaptarse bien a los distintos circuitos que se generan. TRPO también tiene un buen desempeño gracias a su optimización robusta y restricciones de región de confianza, aunque es menos flexible y resulta más costoso computacionalmente que PPO. Por otro lado, A2C tiene dificultades para aprender en este entorno debido a su exploración insuficiente donde parece que las actualizaciones no consiguen resolver nada. La variabilidad del circuito que se encuentra el agente parece jugar también en contra de A2C.

6.3 Humanoid

En el entorno de Humanoid el agente busca mantenerse en pie sin caerse, avanzando hacia delante lo más rápido posible y sin hacer movimientos innecesarios. Generalmente los episodios terminan porque se determina que el robot está *unhealthy*, que ocurre cuando no consigue mantener el equilibrio. De esta forma, cuanto más dure un episodio, mayor será la recompensa obtenida.

La tabla siguiente muestra el episodio con la mayor recompensa obtenida por cada uno de los 5 algoritmos en Humanoid.

Ranking	Algoritmo	Mayor recompensa
1	sac	5309
2	td3	4906
3	ppo	671
4	trpo	653
5	a2c	642

Figura 20:Mayores recompensas obtenidas en Humanoid

Lo que más destaca de estos valores es que encontramos dos grupos: SAC y TD3 con recompensas de cerca de 5000, y PPO, TRPO y A2C, que tienen recompensas de algo más de 600. Podríamos considerar a SAC y TD3 unos muy buenos modelos, que en teoría se corresponden con unos robots que parecen aguantar en equilibrio y desplazarse hacia delante durante un tiempo razonable, aunque será necesario ver los respectivos vídeos para confirmarlo.

De los otros tres no se puede decir lo mismo. Una recompensa de 600 en este entorno viene a decirnos que el agente no ha conseguido aprender a lidiar con el mundo que le rodea. No se espera que el robot cumpla su objetivo, durando poco tiempo de pie y sin hacer progreso

significativo. Hay que reconocer que puede llegar a ser curioso que, dentro de este mal rendimiento, las recompensas de los 3 algoritmos sean tan similares.

Ampliamos los resultados obtenidos para Humanoid con la evolución de las recompensas de los 5 algoritmos en la Figura 21.

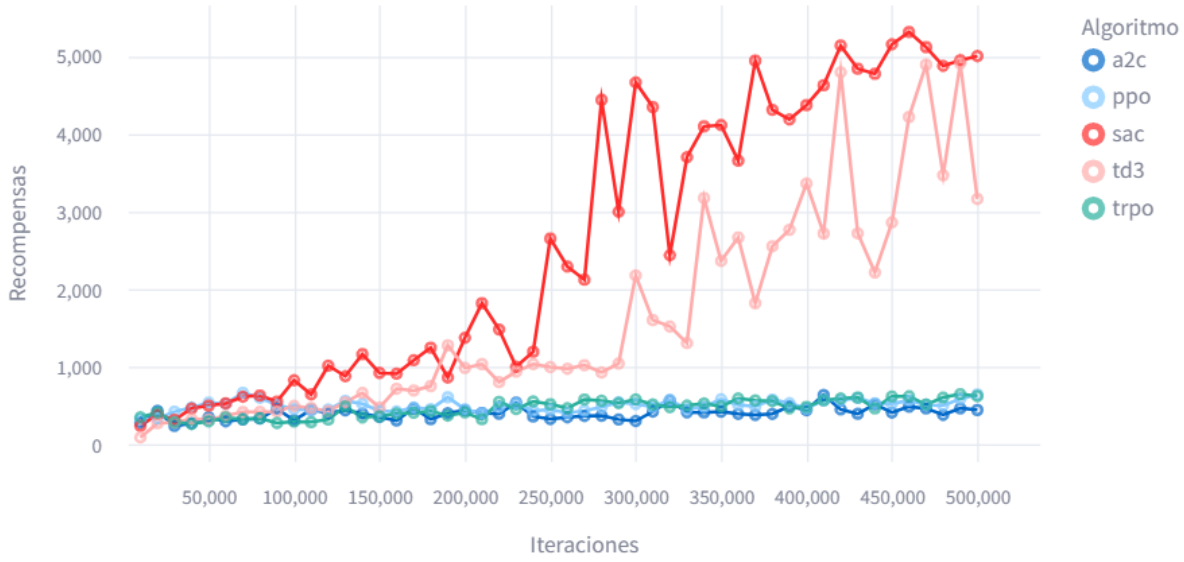


Figura 21: Evolución de recompensas para los algoritmos en Humanoid

No hay sorpresas en la gráfica. PPO, TRPO y A2C no consiguen aprender una buena política, las recompensas no experimentan ninguna subida y se mantienen en torno a 500 y 700. En principio no se aprecian grandes diferencias en la evolución de las 3, ya que las gráficas aparecen entrelazadas.

La otra historia la encontramos con SAC y TD3. TD3 parece tener un buen rendimiento, dando un salto de calidad a partir de las 300000 iteraciones. La recompensa tiene una tendencia ascendente llegando a un máximo de casi 5000. El problema que parece tener es la gran variabilidad de recompensas entre episodios. SAC mejora el rendimiento de TD3. Las recompensas son más consistentes y también algo superiores, continuando su tendencia positiva desde el inicio del entrenamiento. Otro punto a abordar es que parece que tanto SAC como TD3 se beneficiarían de continuar el entrenamiento, ya que no parecen converger del todo y hay también un margen de crecimiento, hasta llegar quizás a recompensas de 6000.

Por último, se dejan los enlaces a los vídeos de los 5 agentes para Humanoid:

- a2c-Humanoid.mp4
- ppo-Humanoid.mp4
- trpo-Humanoid.mp4
- sac-Humanoid.mp4
- td3-Humanoid.mp4

Los videos reflejan una realidad similar a la gráfica anterior. Los esfuerzos del agente en A2C, PPO y TRPO son en vano, ya que apenas dura unos segundos antes de caer, sin conseguir nada significativo. SAC y TD3, por el contrario, muestran un robot capaz de andar hacia delante manteniendo el equilibrio durante bastante tiempo. Entre estos dos también se encuentran diferencias. En TD3 el humanoide anda con el torso inclinado hacia atrás, lo que parece menos eficiente y más lento que el caso de SAC donde el paso del humanoide es en posición totalmente vertical. Esto parece explicar porque SAC tiene unas recompensas algo superiores y más consistentes comparada con TD3.

Para terminar, se puede decir que SAC y TD3 sobresalen en el entorno de Humanoid. SAC, con su enfoque basado en la entropía máxima, fomenta una exploración balanceada y consistente, crucial para entornos complejos y de alta dimensionalidad. TD3, por su parte, utiliza actualizaciones diferidas y una política determinista con redes críticas dobles, lo que reduce el sobreajuste y mejora la estabilidad del aprendizaje con respecto a otro tipo de algoritmos. En contraste, otros como TRPO y PPO, aunque son robustos, tienen actualizaciones más conservadoras y pueden ser menos eficientes en la exploración de un entorno de alta dimensionalidad. Además, A2C puede sufrir de problemas de estabilidad y convergencia lenta debido a su uso de una única red crítica y su menor capacidad para capturar y aprender de la variabilidad del entorno.

6.4 Half Cheetah

En el entorno Half Cheetah las recompensas incentivan que el robot se desplace lo más rápido posible hacia delante. A diferencia del humanoide, un episodio no termina por la caída del guepardo, sino por un límite de tiempo. Por lo tanto, la recompensa será mayor cuanto más terreno se recorra durante la duración del episodio.

Recogemos en la siguiente tabla la recompensa máxima obtenida durante el entrenamiento de los 5 principales algoritmos empleados.

Ranking	Algoritmo	Mayor recompensa
1	td3	9142
2	sac	7633
3	ppo	4873
4	trpo	1561
5	a2c	882

Figura 22: Mayores recompensas obtenidas en Half Cheetah

De manera similar al Humanoid, TD3 y SAC son los que mejor rendimiento obtienen. TD3 consigue un valor de recompensa de más de 9000, que se puede considerar como un muy buen modelo. SAC por su parte se sitúa en un respetable 7600, también un buen indicador para este entorno.

PPO sorprende respecto al Humanoid, y consigue una recompensa máxima de cerca de 5000 para situarse como el tercer mejor algoritmo. Por el contrario, TRPO y A2C vuelven a sufrir y a tener un comportamiento subóptimo, con recompensas cercanas a los 1000.

La Figura 23 muestra una vista más en detalle de los resultados, con las evoluciones de recompensas de los 5 casos.

Tanto TRPO como A2C empiezan aprendiendo, pero se estabilizan a partir de las 100000 iteraciones. A2C converge entorno a 1000 de recompensa mientras que TRPO lo hace alrededor de 1500. No parecen ser los algoritmos más indicados para este entorno. PPO sí que parece tener una tendencia de crecimiento superior y continúa aprendiendo durante bastante más tiempo, estabilizándose cerca de las 400000 iteraciones.

SAC y TD3 se confirman como los mejores modelos entrenados, aunque encontramos diferencias entre ellos. SAC parece muy consistente en las recompensas obtenidas, apenas hay variación entre episodios consecutivos y la progresión de crecimiento es muy constante. TD3 genera mejores recompensas que SAC, pero tiene una mayor variabilidad, con muchos picos y valles. Ambos algoritmos se beneficiarían de una continuación del entrenamiento, ya que se aprecia que las recompensas no han convergido y siguen creciendo ligeramente.

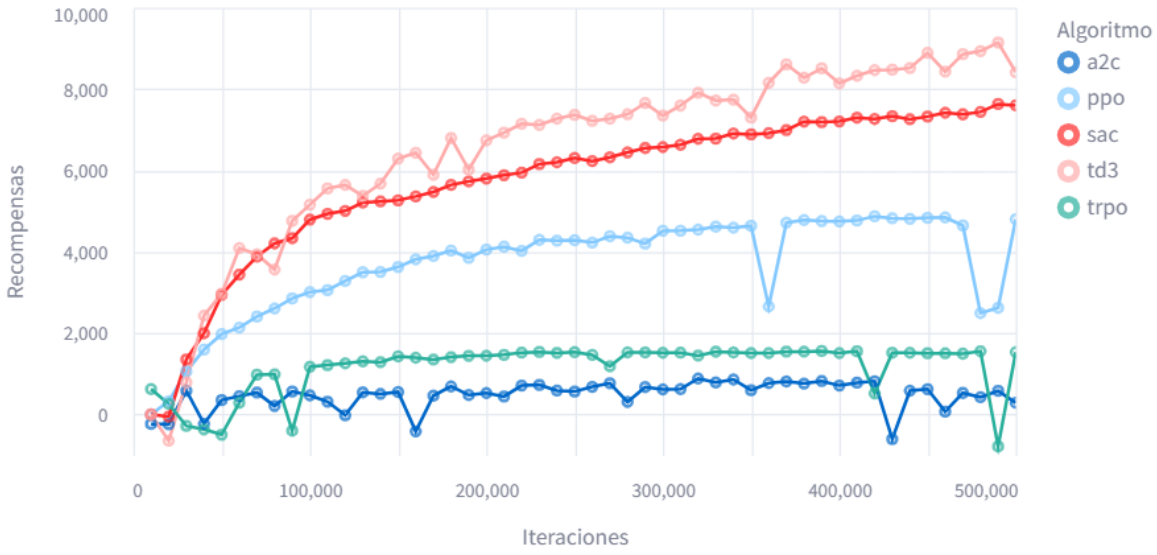


Figura 23: Evolución de recompensas para los algoritmos en Half Cheetah

Se dejan a continuación enlaces a los vídeos de los 5 agentes entrenados de Half Cheetah:

- a2c-HalfCheetah.mp4
- ppo-HalfCheetah.mp4
- trpo-HalfCheetah.mp4
- sac-HalfCheetah.mp4

- td3-HalfCheetah.mp4

En A2C y en TRPO, las bajas recompensas se explican porque el guepardo acaba del revés, haciendo su desplazamiento mucho más lento al no poder correr. En ninguno de los dos casos parecen ser capaces de aprender a mantener el equilibrio.

PPO ya deja una mejor imagen, con el guepardo corriendo de manera normal a una buena velocidad. En ningún momento se para ni se da la vuelta y no se aprecian problemas aparentes. Aun así, tanto en SAC como en TD3 se ve a el robot corriendo a una velocidad superior a PPO, lo que explica esa diferencia en las recompensas.

En general, en Half Cheetah encontramos una dinámica similar al Humanoid, que tiene bastante sentido, ya que estamos hablando de entornos similares. Ambos forman parte de Mujoco, con gran dimensionalidad en observaciones y acciones y complejos por su gran variabilidad. SAC y TD3 vuelven a destacar por encima del resto, consiguiendo que el robot corra adecuadamente y con un muy buen ritmo. TD3 en esta ocasión es el que mejores recompensas consigue entre los dos, con un excelso rendimiento gracias a características como sus dos redes críticas o la regularización del ruido que son muy potentes para espacios de alta dimensionalidad. Para SAC, la razón de su éxito vuelve a ser la maximización de la entropía, que le lleva a hacer una exploración balanceada y constante. Además, SAC parece ser más constante entre episodios que TD3, una ventaja a considerar en el futuro.

PPO juega un mejor papel que en Humanoid y el guepardo se mueve correctamente, aunque no consigue ir tan rápido como en SAC y TD3. Aquí se puede beneficiar de sus actualizaciones controladas que permiten una exploración eficiente sin grandes desviaciones de la política. La cara negativa la ponen TRPO y A2C, que demuestran su incapacidad para aprender en entornos de este tipo. En esta ocasión, en los vídeos se aprecia como el guepardo se pone boca arriba, inhabilitando su movimiento natural al correr.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El análisis de resultados en los cuatro entornos (Mountain Car Continuous, Car Racing, Humanoid y Half Cheetah) demuestra que los algoritmos SAC y TD3 destacan como los más eficientes y consistentes, especialmente en entornos de alta dimensionalidad y complejidad como son los de Mujoco. Estos algoritmos no solo alcanzaron las recompensas más altas, sino que también mostraron una tendencia de convergencia más rápida y estable. Su éxito puede atribuirse a sus estrategias avanzadas de exploración y actualización, con SAC aprovechando la maximización de la entropía para una exploración balanceada y TD3 utilizando redes críticas dobles y regularización del ruido para mejorar la estabilidad del aprendizaje.

Por otro lado, PPO mostró un rendimiento sólido en algunos entornos, destacando especialmente en Car Racing, donde logró recompensas significativamente superiores a las de TRPO y A2C, siendo el único modelo en conseguir una conducción consistente en los distintos circuitos. La capacidad de PPO para realizar actualizaciones controladas y eficientes le permite adaptarse bien a entornos de alta dimensionalidad, aunque quizás no tanto como SAC y TD3. Además, PPO tuvo dificultades en Humanoid, donde su rendimiento fue de los peores.

Finalmente, TRPO y A2C presentaron rendimientos subóptimos en la mayoría de los entornos analizados, especialmente en los más complejos. Estos algoritmos mostraron una incapacidad para adaptarse adecuadamente, con una convergencia lenta y recompensas bajas. Las estrategias de exploración conservadoras y la menor eficiencia en la actualización de políticas explican en parte su desempeño inferior.

En resumen, la elección del algoritmo óptimo depende en gran medida de las características específicas del entorno. En entornos sencillos como Mountain Car Continuous, todos los algoritmos dan buenos resultados y con una convergencia relativamente rápida. En estos casos en lo que se debe fijar uno es en los costes computacionales de los mismos, seleccionando por ejemplo un algoritmo como A2C para agilizar el entrenamiento.

Por otro lado, en entornos complejos de alta dimensionalidad como los de Mujoco, debe premiar más el rendimiento, y se recomienda utilizar SAC o TD3, aunque el entrenamiento sea más largo y costoso. En estos entornos también puede ser necesario aumentar el número de iteraciones hasta que llegue la convergencia, implementando algún tipo de *Early Stopping* *[32]* para detectar cuando la recompensa obtenida no mejora más.

Cabe destacar también la importancia que tiene el ajuste de hiperparámetros en los algoritmos de aprendizaje por refuerzo. Las recompensas obtenidas pueden cambiar mucho en función de los ajustes que se hagan, especialmente con valores como el *Learning rate*, *Gamma* o el tipo de política elegida. Por ello es clave buscar los parámetros que maximicen las recompensas, pese a el esfuerzo extra que esto puede suponer.

Como posibles trabajos y desarrollos futuros se sugieren los siguientes puntos:

- Extender los algoritmos existentes para **entornos multiagente** donde múltiples agentes interactúan y aprenden simultáneamente. Esto abrirá nuevas posibilidades para aplicaciones como simulaciones de tráfico y robots colaborativos, donde la coordinación y la competencia entre agentes son cruciales.
- Investigar e implementar **algoritmos de RL basados en modelos** que utilicen una representación del entorno para planificar y tomar decisiones más informadas. Hay situaciones donde se pueden utilizar este tipo de algoritmos para mejorar el rendimiento y la eficiencia de los agentes.
- Extender la aplicación de estos algoritmos a **nuevos dominios del mundo real**, sin limitarse a entornos prefabricados de Gymnasium. Esto puede demostrar el impacto potencial del aprendizaje por refuerzo en una amplia gama de industrias, promoviendo la adopción de estas técnicas en aplicaciones prácticas.
- Ampliación de **la plataforma interactiva de Streamlit**. Se pueden añadir nuevas características a esta herramienta, como la posibilidad de entrenar modelos en entornos customizados. También sería interesante desplegar la aplicación en algún entorno Cloud para facilitar el acceso a otros usuarios.

Capítulo 8. BIBLIOGRAFÍA

[1] Prasidini, K. "Algorithms in Reinforcement Learning", Medium, Noviembre 2020. https://medium.com/swlh/algorithms-in-reinforcement-learning-ec42a3826a0c

[2] Sutton, R. S., & Barto, A. G. "Reinforcement Learning: An Introduction." MIT Press, 2018. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[3] "Twin Delayed DDPG", Open AI Spinning Up, 2023, https://spinningup.openai.com/en/latest/algorithms/td3.html

[4] "Soft Actor-Critic", Open AI Spinning Up, 2023, https://spinningup.openai.com/en/latest/algorithms/sac.html

[5] "Proximal Policy Approximation", Open AI Spinning Up, 2023, https://spinningup.openai.com/en/latest/algorithms/ppo.html

[6] "Trust Region Policy Approximation", Open AI Spinning Up, 2023, https://spinningup.openai.com/en/latest/algorithms/trpo.html

[7] Simonini, T. "Advantage Actor Critic (A2C)", Hugging Face, Julio 2022, https://huggingface.co/blog/deep-rl-a2c

[8] "Stable-Baselines3 Docs - Reliable Reinforcement Learning Implementation", Stable Baselines3, 2021, https://stable-baselines3.readthedocs.io/en/master/

[9] "Basic Usage", Gymnasium Documentation, 2023, https://gymnasium.farama.org/content/basic_usage/

[10] "Mujoco", Google Deepmind Github 2021, https://github.com/google-deepmind/mujoco

[11] "Optuna: A hyperparameter optimization framework", Optuna 2018, https://optuna.readthedocs.io/en/stable/

[12] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. "Stable-Baselines3: Reliable reinforcement learning implementations". *Journal of Machine Learning Research, 22*(268), 1-8. 2021 http://jmlr.org/papers/v22/20-1364.html

[13] Ding, Z., Yu, T., Huang, Y., Zhang, H., Mai, L., & Dong, H. "RL zoo: A comprehensive and adaptive reinforcement learning library". *arXiv preprint arXiv:2009.08644* 2020

[14] "Basic concepts of Streamlit", Streamlit Documentation 2024, https://docs.streamlit.io/get-started/fundamentals/main-concepts

[15] "Por qué TensorFlow", TensorFlow 2023,https://www.tensorflow.org/about?hl=es-419

[16] "Reinforcement Learning Toolbox", Mathworks 2022 https://es.mathworks.com/products/reinforcement-learning.html

[17] "Simulink. Diseñe. Simule. Despliegue", Mathworks 2022, https://es.mathworks.com/products/simulink.html

[18] "Google Collaboratory", Google 2023,https://colab.google/

[19] "Pioneering research on the path to AGI", Open AI 2024, https://openai.com/research/

[20] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). "Proximal Policy Optimization Algorithms"

[21] Fujimoto, S., van Hoof, H., & Meger, D. (2018). "Addressing Function Approximation Error in Actor-Critic Methods." Proceedings of the 35th International Conference on Machine Learning (ICML)

[22] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2016). "Continuous control with deep reinforcement learning."

[23] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor" Proceedings of the 35th International Conference on Machine Learning (ICML)

[24] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., & Kavukcuoglu, K. (2016). "Asynchronous Methods for Deep Reinforcement Learning". Proceedings of the 33rd International Conference on Machine Learning (ICML)

[25] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). "Trust Region Policy Optimization." Proceedings of the 32nd International Conference on Machine Learning (ICML)

[26] "Divergencia de Kullback-Leibler (KL)", Amazon AWS Docs, 2024, https://docs.aws.amazon.com/es_es/sagemaker/latest/dg/clarify-data-bias-metric-kl-divergence.html

[27] "Mountain Car Continuous", Classic Control Gymnasium Documentation, 2023, https://gymnasium.farama.org/environments/classic_control/mountain_car_continuous/

[28] "Car Racing", Box 2D Gymnasium Documentation, 2023, https://gymnasium.farama.org/environments/box2d/car_racing/

[29] "Humanoid", Mujoco Gymnasium Documentation, 2023, https://gymnasium.farama.org/environments/mujoco/humanoid/

[30] "Half Cheetah", Mujoco Gymnasium Documentation, 2023, https://gymnasium.farama.org/environments/mujoco/half_cheetah/

[31] "Streamlit Components", Streamlit Documentation 2024, https://streamlit.io/components

[32] Brownlee, J. "Use Early Stopping to Halt the Training of Neural Networks At the Right Time", Machine Learning Mastery, 2020, https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

ANEXO I: ODS

Los Objetivos de Desarrollo Sostenible (ODS) son un conjunto de 17 metas globales adoptadas por todos los Estados Miembros de las Naciones Unidas en 2015, como parte de la Agenda 2030. Los ODS sirven como una hoja de ruta para lograr un desarrollo sostenible en todas las dimensiones: económica, social y ambiental, asegurando que nadie quede atrás en este proceso.

Podemos destacar tres principales ODS a los que contribuye este proyecto:

1. **ODS 4: Quality Education (Educación de Calidad):** El proyecto fomenta la educación de calidad a través del desarrollo de conocimientos avanzados en el campo del Aprendizaje por Refuerzo. Al profundizar en la investigación de estos algoritmos, se genera nuevo conocimiento y se promueve la formación en tecnologías emergentes. Además, proporciona recursos como la plataforma interactiva que pueden ser utilizados en programas educativos y formativos, mejorando la calidad de la educación en este area.
2. **ODS 8: Decent Work and Economic Growth (Trabajo Decente y Crecimiento Económico):** El avance en el Aprendizaje por Refuerzo y su aplicación en áreas como la robótica o procesos industriales puede generar nuevos empleos en campos como la ingeniería de software, la inteligencia artificial y la robótica, sustituyendo trabajos de menor calidad como los manuales. Además, la innovación tecnológica en diversos sectores industriales y de servicios pueden impulsar el crecimiento económico, haciendo que las empresas sean más competitivas y sostenibles.
3. **ODS 9: Industry, Innovation and Infrastructure (Industria, Innovación e Infraestructura):** El estudio de algoritmos como TD3, SAC, PPO, TRPO y A2C facilitan el desarrollo de soluciones innovadoras en robótica, vehículos autónomos y otros sistemas automatizados. Esto tiene un impacto directo en el avance de la industria y la mejora de infraestructuras inteligentes.