COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Grado en ingeniería en tecnologías industriales

TRABAJO FIN GRADO

Diseño de agentes artificiales autónomos en entornos de simulación mediante aprendizaje por refuerzo

Autor: Manuel González López

Director: Álvaro Jesús López López

Madrid

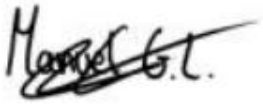
Diciembre 2024

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

Diseño de agentes artificiales autónomos en entornos de simulación mediante aprendizaje por refuerzo

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/25 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.: Manuel González López Fecha: 18/12/2024



Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Álvaro Jesús López López

Firmado digitalmente por Álvaro Jesús López López
Fecha: 2024.12.21 12:45:21 +01'00'

Fdo.: Álvaro Jesús López López Fecha: 18/12/2024

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

Grado en ingeniería en tecnologías industriales

TRABAJO FIN DE GRADO

Diseño de agentes artificiales autónomos en entornos de simulación mediante aprendizaje por refuerzo

Autor: Manuel González López

Director: Álvaro Jesús López López

Madrid

Diciembre 2024

DISEÑO DE AGENTES ARTIFICIALES AUTÓNOMOS EN ENTORNOS DE SIMULACIÓN MEDIANTE APRENDIZAJE POR REFUERZO

Autor: González López, Manuel

Director: López López, Álvaro Jesús

Entidad Colaboradora: ICAI-Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

INTRODUCCIÓN

La inteligencia artificial está cambiando el paradigma de la industria y la sociedad. Desde sistemas autónomos en la conducción hasta la robótica avanzada. Esto se debe a la capacidad de diseñar agentes que aprendan y se adapten de forma autónoma, a través de su propia experiencia con su interacción con el entorno. El aprendizaje por refuerzo (*Reinforcement Learning*, RL) se ha convertido en un pilar para resolver problemas de decisión que demandan autonomía, adaptabilidad y capacidad de toma de decisiones en entornos complejos. [1]

A diferencia de las técnicas tradicionales de aprendizaje supervisado, RL permite a los agentes aprender mediante la interacción directa con su entorno, utilizando un sistema de recompensas. Este enfoque se asemeja al aprendizaje humano basado en prueba y error permitiendo así que los sistemas no dependan de grandes conjuntos de datos con los problemas que conllevan. El aprendizaje por refuerzo está penetrando a la hora de dar soluciones en diversas áreas, como por ejemplo la conducción autónoma, la gestión energética y la robótica. Sin embargo, a pesar de sus avances, enfrenta desafíos significativos, como la estabilidad durante el proceso de entrenamiento y la capacidad de extrapolar estas técnicas a entornos cada vez más complejos. [1]

Este proyecto aborda el diseño de agentes autónomos entrenados mediante aprendizaje por refuerzo, centrándose en la comparación entre una técnica de método basado en valor y una técnica de aproximación directa de la política, *Deep Q-Learning* y *Proximal Policy Optimization* respectivamente. Para facilitar el análisis se evalúan en un entorno sencillo como es el LunarLander-v2, que es una versión del problema clásico de optimización de trayectoria de cohetes y en la librería de Box2D. LunarLander-v2 ofrece un marco ideal para explorar los aspectos técnicos de ambas técnicas en profundidad, debido a su simplicidad estructural y su capacidad para modelar problemas de control discreto.

Los objetivos principales del proyecto son:

a) Analizar y comparar el rendimiento entre las técnicas de DQN y PPO en un entorno sencillo.
b) Evaluar métricas clave como la recompensa media por episodio, la duración media de los episodios, los pasos globales y la tasa de aterrizajes exitosos.

c) Identificar las fortalezas y limitaciones de cada técnica, desarrollando una metodología de comparación para futuras investigaciones en este campo.

METODOLOGÍA

El proyecto implementa una rigurosa metodología estructurada en varias etapas, cada una diseñada para garantizar un análisis riguroso de las técnicas de aprendizaje por refuerzo escogidas.

En primer lugar, se escoge el entorno con el problema de decisión de Markov a resolver. El entorno escogido es el entorno de LunarLander-v2, proporcionado por la librería Box2D dentro de la librería de gym. Este entorno es un entorno estándar en la investigación de aprendizaje por refuerzo, conocido por su capacidad para modelar problemas de control con un equilibrio entre simplicidad y complejidad. Los agentes deben aprender a tomar decisiones para controlar la posición, velocidad y orientación de la naver para poder aterrizar en una zona obetivo de manera exitosa. Las características del entorno son las siguientes:

- **Recompensas:** Los agentes reciben recompensas por aterrizar correctamente, estabilizarse y mantenerse dentro de los límites del área designada. Se penalizan las colisiones, salirse del área proporcionada y los aterrizajes fuera del área segura.

- **Espacio de estados:** Un vector de 8 dimensiones que incluye información sobre la posición, velocidad, ángulo y contacto con el suelo.

- **Espacio de acciones:** Cuatro acciones discretas que incluyen encender el motor principal, activar los motores laterales y no realizar ninguna acción.

En segundo lugar, se escogen dos técnicas para comparar y entrenar nuestros agentes. Estas técnicas siguen las dos principales ramas del aprendizaje por refuerzo, técnicas basadas en valor y aproximación directa de la política. A continuación, se explican brevemente cada una de ellas.

- *Deep Q-Learning* (DQN)

Este algoritmo es una técnica basada en valor que se basa en aproximar los Q valores, que representan las recompensas esperadas de realizar ciertas acciones desde un estado dado. A partir de esta información se optimiza indirectamente la política que desarrolla el agente. Para aproximar estos valores Q se utiliza una red neuronal, lo que permite al agente explorar el entorno y desarrollar estrategias óptimas. Para estabilizar el entrenamiento, se incorporan técnicas como la experiencia de repetición, *replay buffer*, y redes objetivo separadas. [2] [3]

- *Proximal Policy Optimization* (PPO)

Este algoritmo es una técnica de *Actor*-Critic, ya que tiene un crítico que utiliza técnicas basadas en valor para evaluar la calidad de la acción tomada por el agente. Aun así, el desarrollo

de la política toma se desarrolla a partir de una aproximación directa de la política. El algoritmo de PPO ajusta directamente la política del agente mediante el ascenso de gradiente. Para ello se utilizan funciones objetivo recortadas, *Clipped Surrogate Function*, para limitar los cambios bruscos al ir actualizándose la política durante el entrenamiento, garantizando así un entrenamiento estable y eficiente. Este algoritmo combina elementos de métodos *Actor-Critic*, donde un actor genera políticas y un crítico evalúa su calidad. [2] [4]

En tercer lugar, ambos algoritmos se implementan utilizando Python y las librerías StableBaselines3 y Gymnasium. Se utilizan distintas arquitecturas de redes neuronales para adaptar mejor cada técnica a la tarea en cuestión, con funciones de activación y capas ocultas optimizadas para el entorno. Los hiperparámetros clave, como las tasas de aprendizaje y los tamaños de lote, se ajustaron mediante pruebas iterativas para optimizar el desempeño de los agentes.

Por último, a partir de los agentes entrenados se realiza una comparación de distintas métricas de vital importancia para comparar el rendimiento de los agentes. Estos análisis son los siguientes.

- Recompensa media por episodio: mide la calidad del aprendizaje y la capacidad del agente para maximizar las recompensas acumuladas a lo largo del entrenamiento.
- Duración media de los episodios: refleja la evolución y la eficiencia del agente en la resolución de problemas durante el entrenamiento.
- Duración total del entrenamiento: refleja el tiempo necesario por cada técnica para llevar a cabo su entrenamiento.
- Porcentaje de aterrizajes exitosos: evalúa el éxito de los agentes después de haber sido entrenados.

RESULTADOS

A partir de la metodología desarrollada en este proyecto se realiza el análisis comentada en el apartado anterior obteniéndose los siguientes resultados. A continuación, se muestran las gráficas con los resultados más relevantes.

- Recompensa media por episodio

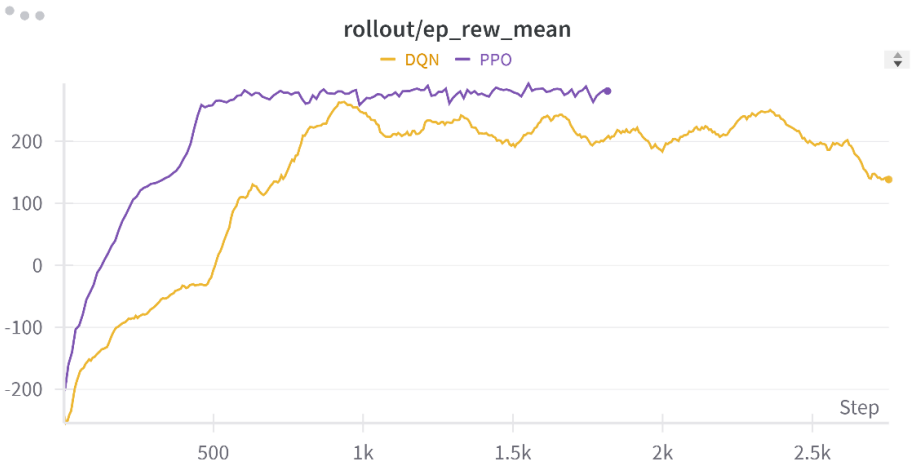


Figura R 1. Gráfica recompensa media por episodio de los agentes

Al inicio del entrenamiento, el agente de DQN mostró un aprendizaje lento comparándolo con el agente de PPO, ambos teniendo recompensas bajas debido a la exploración inicial. Las recompensas comenzaron a estabilizarse para el agente de DQN aproximadamente a los 100,000 pasos, alcanzando un valor promedio entorno a 200 puntos. Sin embargo, el agente de PPO alcanza un promedio de unos 250 puntos en la mitad de *steps*. Además, se observa que para la técnica de DQN tiene mucha más inestabilidad a lo largo de su entrenamiento que el modelo de PPO no presenta.

- Duración media de los episodios

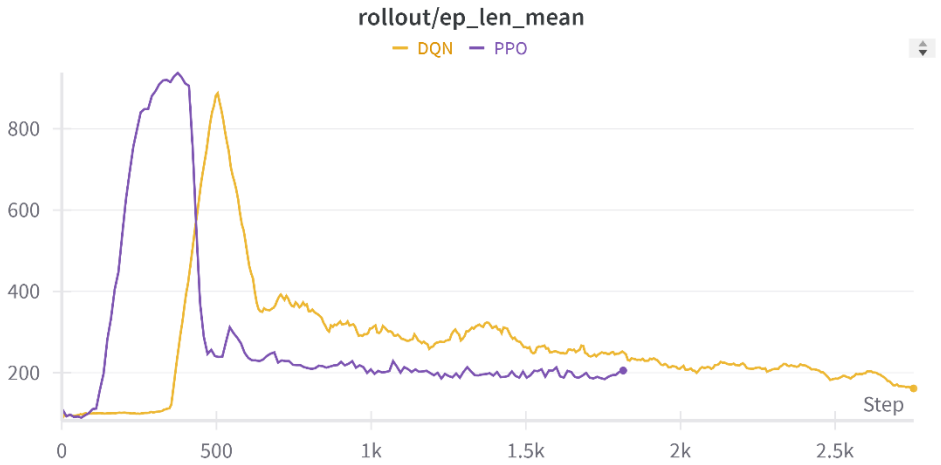


Figura R 2. Longitud media de los episodios durante el entrenamiento

Se observa que ambos agentes a medida que van desarrollando sus estrategias óptimas van reduciendo la duración de los episodios. Cabe destacar que para la técnica de DQN, aunque logró reducir la duración de los mismos a medida que aprendía, es más lento y menos constante que el agente con PPO.

- Duración total del entrenamiento (*Global Steps*)

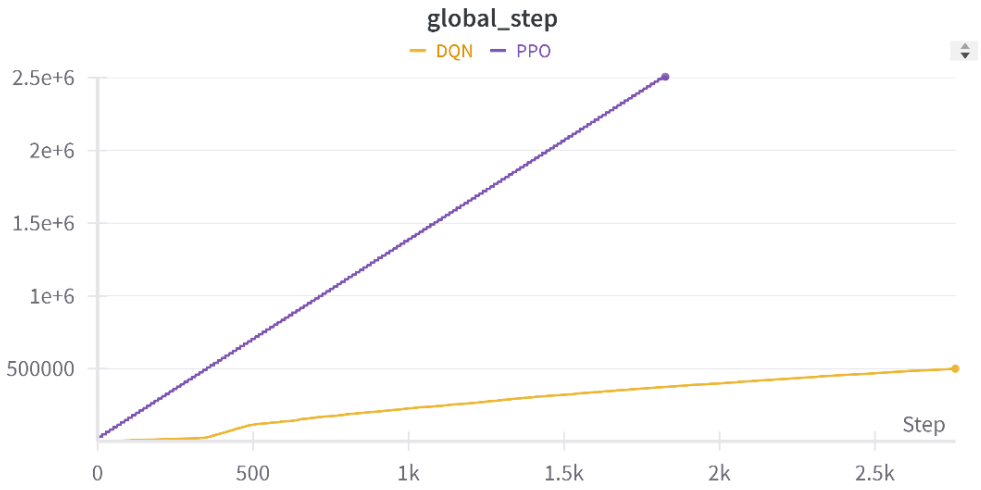


Figura R 3. Pasos globales vs pasos totales durante el entrenamiento

El entrenamiento de DQN requirió aproximadamente 500,000 pasos globales en los que tarda un poco menos del doble en realizarlos que el agente de PPO. Esto demuestra que debido a las características de la técnica de DQN necesita de más tiempo de entrenamiento. Estas diferencias se van agrandando a medida que el entorno es más complejo. Por otro lado, la técnica de PPO debido a su ascenso de gradiente y sus características es capaz de llegar a un óptimo en menos tiempo.

- Porcentaje de aterrizajes exitosos

Métrica	DQN	PPO
Media	271,37	281,90
Mediana	272,03	282,18
Desviación estándar	21,24	20,32
Valor mínimo	224,54	232,55
Valor máximo	310,86	321,10
Rango	86,32	88,55

Tabla R 1. Métricas porcentaje de éxito de los agentes

PPO superó a DQN en términos de éxito una vez ya entrenados, mostrando haber desarrollado una política más robusta frente a variaciones en el entorno y una mayor capacidad para adaptarse a situaciones complejas.

CONCLUSIONES

Este proyecto demuestra que las técnicas basadas en políticas, como PPO, ofrecen ventajas significativas frente a los métodos basados en valores como DQN. La técnica de PPO presenta una mayor estabilidad durante el entrenamiento, un aprendizaje más rápido y un mejor rendimiento en general, lo que lo hace más prometedor. Por otro lado, la técnica de DQN sigue siendo una herramienta valiosa para problemas de decisión donde es necesaria una exploración exhaustiva del entorno. Sin embargo, hay que tener en cuenta las limitaciones de tiempo necesario y problemas de a la hora de implementarse.

Este proyecto establece una metodología rigurosa para análisis de técnicas de aprendizaje por refuerzo que se puede aprovechar en futuras investigaciones. Este trabajo también refleja el potencial del aprendizaje por refuerzo como una herramienta clave en el desarrollo de sistemas autónomos más eficientes y robustos, abriendo nuevas oportunidades en diversos campos dentro de la industria.

REFERENCIAS

[1] V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature,* vol. 518, p. 529–533, 2015.

[2] Huggin Face, "Hugging Face," [Online]. Available: https://huggingface.co/learn/deep-rl-course/unit0/introduction.. [Accessed 15 October 2024].

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Cornell University, Ithaca, New York, 2013.

[4] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2ª ed. ed., Cambridge, MA.: MIT Press, 2018.

[5] M. González López, "Proyecto RL de Lunar Lander.," [Online]. Available: https://wandb.ai/TFG_RL/lunar_lander_rl?nw=nwusergolobama. [Accessed 26 October 2024].

DESIGN OF AUTONOMOUS ARTIFICIAL AGENTS IN SIMULATED ENVIRONMENTS REINFORCEMENT LEARNING

Author: González López, Manuel

Supervisor: López López, Álvaro Jesús

Collaborating Entity: ICAI-Universidad Pontificia Comillas.

PROJECT SUMMARY

INTRODUCTION

Artificial intelligence is changing the paradigm of industry and society, from autonomous driving to advanced robotics. This is due to the ability to design agents that can learn and adapt by a process if trial and error with their own experience by interacting with the environment. Reinforcement Learning has become a pilar for solving decision making problems that require autonomy, adaptability and the ability to make decisions in complex environments. [5]

In comparison with traditional supervised learning techniques, RL allows the agents to learn directly by their own interaction with the environment, using a reward system. This approach of trial and error resembles how humans learn, allowing the system to function without relying on big datasets and the challenges that come with. Reinforcement learning is increasingly being used to provide solutions in different areas of the industry such as autonomous driving, energy management and robotics. However, despite its advances, it faces significant challenges, such as stability during training and the ability to scale to increasingly complex environments. [5]

This project focuses on the design of autonomous agents trained using reinforcement learning, comparing a value-based method and a policy-based method, Deep Q-Learning and Proximal Policy Optimization respectively. For analysis purposes, the techniques are evaluated in the LunarLander-v2 environment, a simplified version of the classic rocket trajectory optimization problem within the Box2D library. LunarLander-v2 provides an ideal framework for exploring the technical aspects of both techniques due to its structural simplicity and ability to model discrete control problems.

The main objectives of this project are:

d) Analyze and compare the performance of DQN and PPO techniques in a simplified environment.
e) Evaluate key metrics such as average reward per episode, average episode duration, total steps and the success rate of landings.
f) Identify the strengths and limitations of each technique, developing a comparison methodology for future research in this field.

METHODOLOGY

The project follows a rigorous methodology structured into multiple stages to ensure a thorough analysis of the chosen reinforcement learning techniques.

Firstly, the chosen environment is LunarLander-v2, provided by the Box2D library within Gym. This environment is well known and used in the RL thanks to how it models control problems, balancing simplicity and complexity. In this environment, the agents must learn to make decisions to control the position, velocity and orientation of the spacecraft to successfully land in a designated area. The main characteristics of the environment are as follows:

- **Rewards:** Agents are rewarded for proper landings, stabilization and staying within the limits of the environment. Penalties are applied for collisions, leaving the area or landing outside the designated area.

- **State Space:** An 8-dimensional vector that includes position, velocity, angle and ground contact information.

- **Action space:** Four discrete actions that are firing the main engine, activating side engines or taking no action.

Secondly, the techniques for training and comparison are selected. In this project the techniques chosen are from each of the main branches of reinforcement learning, value-based methods and policy-based methods. The techniques chosen are the following.

- Deep Q-Learning (DQN)

This algorithm is a value-based method that approximates the Q-values, which represents the expected reward for a specific action in given state. With this information the policy of the agent is then optimized indirectly. To approximate these Q-values a neural network is implemented, enabling exploration and strategy development. Techniques such as replay buffer and different networks are implemented to have more stable training. [6] [3]

- Proximal Policy Optimization (PPO)

This algorithm is an Actor-Critic method, because it has a critic that uses value-based methods to evaluate the quality of the action taken by the agent. The agent mainly updates its policy implementing a policy-based method by using gradient ascent. To update the policy the agent utilizes the Clipped Surrogate Function to limit abrupt changes in the policy, ensuring stable and efficient training. This technique combines an actor that generates the policy and a critic that evaluates the quality of the actions taken. [6] [4]

Thirdly, both algorithms are implemented in Python using StableBaselines3 and Gymnasium libraries. Neural network architectures are optimized for each agent individually, with its activation functions and hidden layers to solve the environment. The hyperparameters, such as learning rates and batch sizes, are fine-tuned iteratively for optimal performance. This project has been developed using an ASUS TUF Gaming with windows 11 as operating system.

Lastly, with the agents already trained we evaluate key metrics between each other. These key metrics are the following.

- Mean reward per episode: measures the quality of the training process and the capacity for the agent to maximize the accumulated reward during the training process.
- Average episode duration per episode: reflects the evolution and efficiency of the agent to solve problems during the training process.
- Total training duration: reflects the time needed for each agent to finish their training.
- Successful landing rates: evaluates the success of each agent once trained.

RESULTS

With the methodology developed in this project and implemented on both agents selected. The analysis yields the following results.

- Mean reward per episode

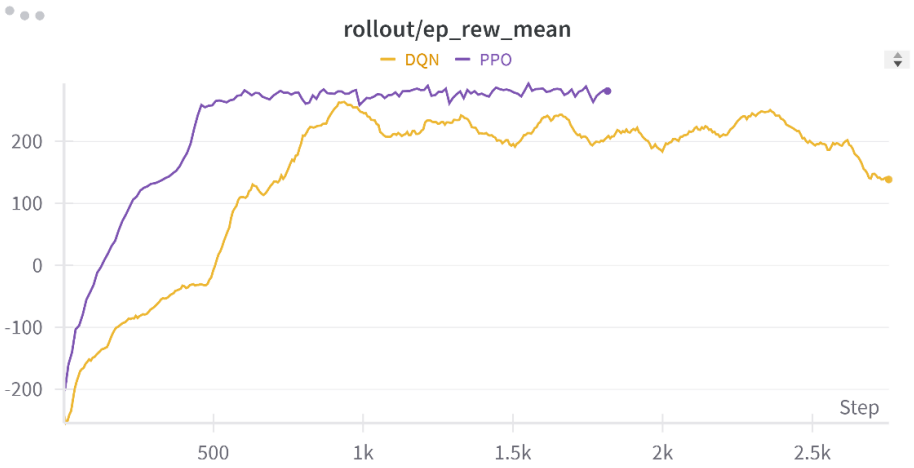


Figure 4. Mean reward per episode for each agent

At the beginning of training, the DQN agent exhibited a slower learning curve in comparison to the PPO agent, with both having low accumulative rewards in the early stages of training. The rewards stabilized for the DQN agent around 100,000 steps with an accumulative reward of 200 points. On the other hand, the PPO agent reaches 250 points with half of steps. It can also be observed that DQN technique presents more instability during training than PPO.

- Average episode duration per episode

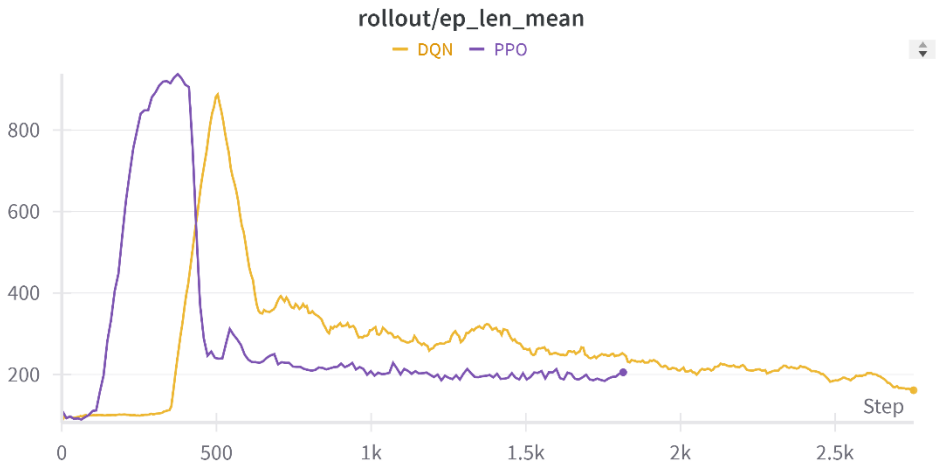


Figure 5. Mean length of the episodes during training

Both agents can reduce their episode duration as they are able to optimize their strategies. It can be observed that DQN technique, even though it can reduce the length per episode during training, is slower and less stable than PPO.

- Total training duration

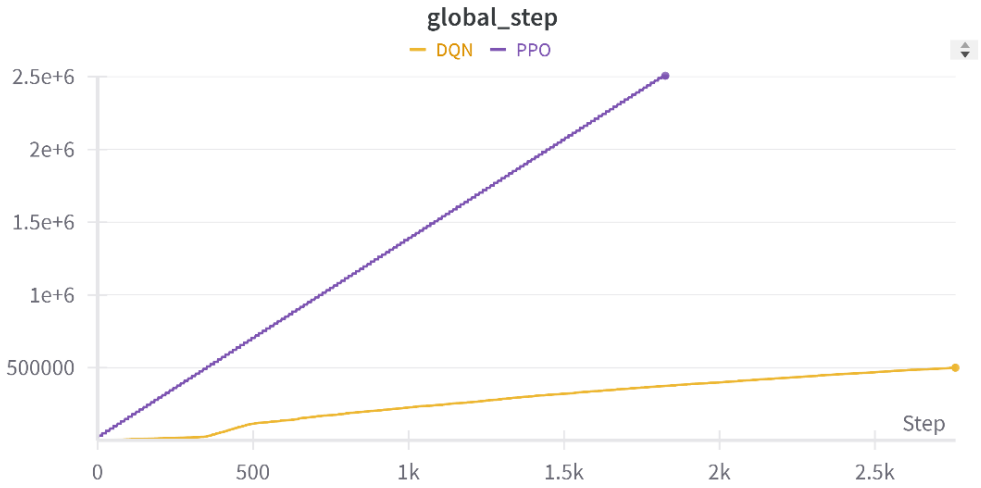


Figure 6. Global steps vs total steps during training

The DQN agent required approximately 500,000 global steps to finish the training process with a bit less than double the steps that the PPO agent required to finish the training. This demonstrates that because of the characteristics that the DQN technique has, it needs more time for training. The differences are bigger when the environment gets more complex.

- Succesful landing rates

Metric	DQN	PPO
Mean	271,37	281,90
Median	272,03	282,18
Standard deviation	21,24	20,32
Minimum	224,54	232,55
Maximum	310,86	321,10
Range	86,32	88,55

Table 2. Successful landing rates metrics for each agent

PPO outperformed DQN in success rates, once both agents finished training, demonstrating a more robust policy and greater adaptability to environmental variations.

CONCLUSIONS

This project shows that policy-based techniques like PPO offer significant advantages over value-based methods like DQN. PPO provides greater training stability, faster learning and superior overall performance, making it a promising approach. However, DQN remains a valuable tool for decision-making problems requiring extensive exploration of the environment, despite its time and implementation challenges.

The project establishes a rigorous methodology for analyzing reinforcement learning techniques, providing a foundation for future research. It highlights the potential of reinforcement learning as a key tool for developing efficient and robust autonomous systems, opening new opportunities across various industrial fields.

REFERENCES

[1] V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature,* vol. 518, p. 529–533, 2015.

[2] Huggin Face, "Hugging Face," [Online]. Available: https://huggingface.co/learn/deep-rl-course/unit0/introduction.. [Accessed 15 October 2024].

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Cornell University, Ithaca, New York, 2013.

[4] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2ª ed. ed., Cambridge, MA.: MIT Press, 2018.

[5] M. González López, "Proyecto RL de Lunar Lander.," [Online]. Available: https://wandb.ai/TFG_RL/lunar_lander_rl?nw=nwusergolobama. [Accessed 26 October 2024].

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo 1 Introducción y objetivos

1.1 Introducción

La inteligencia artificial se encuentra en un crecimiento exponencial debido a lo que puede aportar esta tecnología en diversos ámbitos de la industria y la sociedad en general. Tradicionalmente en el contexto de la inteligencia artificial, los algoritmos de aprendizaje supervisado eran el pilar en las aplicaciones de inteligencia artificial (IA), ya que estos sistemas aprendían de grandes volúmenes de datos. Con esta técnica se han podido resolver una variedad amplia de problemas, pero solo cuando los datos son claros y bien suministrados, ya que si no es así estos algoritmos fallan.

La necesidad de tener sistemas que no necesiten datos limpios para su aprendizaje ha fomentado un auge en el aprendizaje autónomo, en el que el aprendizaje por refuerzo (*Reinforcement Learning* RL), variante de *machine learning*, está siendo crucial. Esta técnica se basa en que los agentes autónomos puedan aprender a tomar decisiones y desarrollar estrategias óptimas a través de interactuar con el entorno, recibiendo recompensas dependiendo de las acciones que realiza el agente. De esta forma el agente acaba aprendiendo mediante prueba y error, como haría una persona.

Con estas nuevas técnicas se puede llegar a conseguir sistemas que son capaces de aprender y adaptarse de manera autónoma a entornos cada vez más dinámicos, complejos y grandes. Esto abre un abanico de posibilidades donde se mejora el rendimiento de sistemas autónomos y lo hace de forma más sostenible y segura en un gran campo de aplicaciones como la robótica, conducción autónoma o la gestión de energía inteligente, entre otras.

En la actualidad todavía un amplio campo de investigación para poder implementar modelos de aprendizaje por refuerzo a gran escala, esto se debe a la falta de infraestructura computacional y algoritmos adaptados. Los sistemas de IA tradicionales no tienen la capacidad de manejar problemas en los que se requiere procesar flujos de datos continuos, tomar decisiones en tiempo real o ser capaces de adaptarse a cambios en el entorno. Sin embargo, las técnicas de aprendizaje por refuerzo ofrecen grandes ventajas:

- **Beneficios técnicos:** permite que los sistemas inteligentes tengan una mayor capacidad de autonomía y adaptación en entornos más complejos e incluso cooperativos.
- **Beneficios económicos:** incremento de la eficiencia de los modelos, tanto a nivel de recursos computacionales y a largo plazo.
- **Beneficios en innovación:** permite el desarrollo de nuevas estrategias y técnicas experimentadas por el propio agente o agentes en su entrenamiento.

Uno de los mayores beneficios de la implementación del aprendizaje por refuerzo en sistemas de inteligencia artificial es la capacidad de adaptarse y aprender de manera continua a partir de la experiencia del agente al interactuar con el entorno. Aun así, es importante destacar que hay una serie de desafíos significativos en los que es necesario seguir desarrollando esta tecnología para poder seguir haciéndola más fiable. Algunos de estos consisten en la dificultad de prever todas las situaciones en la que el agente se puede encontrar en el entorno, la estabilidad del aprendizaje, la escalabilidad y los recursos computacionales. En este trabajo se pretende hacer un

estudio de investigación de estas técnicas, recalcando las más importantes y con el análisis de algunos agentes en diversos entornos. [1]

1.2 Motivación

En las últimas décadas se han logrado grandes avances en el aprendizaje por refuerzo, especialmente en problemas donde el agente tiene que aprender a tomar decisiones de manera autónoma a partir de interactuar con el entorno en el que se encuentra. La motivación principal para para este proyecto surge para conocer en primera instancia técnicas utilizadas en la actualidad para resolver problemas. A partir de esto, realizar un breve análisis de dos técnicas para ver las fortalezas y debilidades de cada una de ellas durante el entrenamiento, y una vez finalizado el mismo.

Con este análisis se persigue identificar en qué tipos de problemas cada una de las técnicas analizadas tienen ventajas y desventajas, para así poder tener una visión más clara de cuando utilizar cada una de las mismas. Además, con este análisis se pretende mejorar la comprensión de los algoritmos utilizados y aportar información para poder realizar mejores decisiones a la hora de elegir que técnica utilizar para distintos problemas. A partir de este proyecto se pretende aportar nuevos análisis para poder aplicar estas técnicas con mayor conocimiento.

1.3 Objetivos

Los principales objetivos de este proyecto de fin de grado son:

a) Comparación de las prestaciones durante el entrenamiento de técnicas basadas en funciones de valor y técnicas basadas en aproximación directa de la política.
b) Comparación de las prestaciones al finalizar el entrenamiento de técnicas basadas en funciones de valor y técnicas basadas en aproximación directa de la política.

Acorde con los objetivos principales, se sugieren los siguientes objetivos específicos:

- Revisión de los algoritmos empleados actualmente para tareas que se pueden resolver a partir del aprendizaje por refuerzo.
- Comparación de prestaciones y entrenamiento en un entorno sencillo de una técnica basada en funciones de valor y otra técnica basada en aproximación directa.
- Análisis tanto del entorno a analizar como de las herramientas a utilizar.

1.4 Estructura de la memoria.

A continuación, se describe la organización del presente proyecto.

El Capítulo 1 , se expone el contexto en el que se desarrolla el trabajo, esto es, inteligencia artificial y sobre todo el aprendizaje por refuerzo que es lo que se analiza en este proyecto.

El Capítulo 2 , se centra en el estado del arte de la cuestión. Se realiza una revisión de las técnica actuales, haciendo hincapié en los algoritmos *Deep Q-Learning* y PPO que son las escogidas para realizar este trabajo y análisis. Posteriormente, se describen los objetivos del presente trabajo.

El Capítulo 3 , se describe el entorno con el Problema de Decisión de Markov en el que se ha realizado el análisis y las herramientas utilizadas para entrenar y analizar los agentes. También se explica el funcionamiento del código para entrenar los agentes y las funciones implementadas para lo mismo.

El Capítulo 4 recoge los resultados del análisis en cuanto a prestaciones de los agentes en el entorno y su entrenamiento. También recoge el análisis a nivel de porcentaje de éxitos una vez ya entrenados los agentes.

El Capítulo 5 , muestra las conclusiones obtenidas a partir de la realización del proyecto, las aportaciones realizadas y futuras investigaciones que complementen este proyecto.

Capítulo 2 Estado de la cuestión

En este capítulo se han descrito las técnicas de IA con aprendizaje por refuerzo, su funcionamiento y algoritmos básicos. Además, se realiza una breve descripción del desarrollo de esta tecnología y la importancia que está cobrando en la actualidad.

2.1 Introducción

Las técnicas de aprendizaje empleadas en *machine learning*, especialmente en el aprendizaje por refuerzo, son métodos que se han ido desarrollando durante varias décadas. Originalmente, la mayoría de los algoritmos tienen enfoques estáticos y reglas predefinidas, lo que limita la capacidad adaptarse a entornos dinámicos y complejos. Uno de los grandes desafíos era lograr dicha adaptación para poder así hacer un aprendizaje más robusto y eficaz. Gracias a los avances tanto en software como hardware la capacidad de procesamiento ha aumentado de tal manera que se pueden hacer modelos más complejos. La técnica que está despuntando es el aprendizaje por refuerzo.

Dentro del aprendizaje por refuerzo hay dos grandes filosofías a la hora de realizar estos algoritmos de aprendizaje. Por un lado, existen los métodos de aprendizaje basados en políticas y por otro lado los métodos basados en valor. Estas permiten al agente aprender un comportamiento, llamado política, que maximiza a partir de la exploración y explotación del entorno. La implementación de estas técnicas está revolucionando diversos sectores dando soluciones más eficientes y adaptables. A continuación, se verán algunas de las técnicas más usadas y que se analizarán en este proyecto. [2]

2.2 Aprendizaje por refuerzo

2.2.1 Características principales

El aprendizaje por refuerzo es una técnica para resolver problemas de decisión en secuencia. Uno de los marcos preminentes para modelar matemáticamente estos problemas es el del Problema de Decisión de Markov (MDP, por sus siglas en inglés). El aprendizaje por refuerzo consiste en una analogía básica con el ser humano en el que se va aprendiendo mediante prueba y error explorando el entorno y sacando conclusiones de estas experiencias. Estas conclusiones se basan en un sistema de recompensa. Este consiste en que hay una serie de acciones, estados o ambas, que penalizan o recompensan al agente para que siga mejorando su comportamiento. Es el mismo funcionamiento que tenemos nosotros ya que aprendemos a realizar muchas tareas mediante: realización de la acción, evaluación de resultados obtenidos y aprendizaje de la experiencia.

Un concepto muy importante a la hora de diseñar un agente para poder entrenarlo mediante aprendizaje por refuerzo es la propiedad de Markov. Se dice que una señal es de Markov cuando la información que contiene permite conocer la evolución del sistema en el futuro independientemente de la trayectoria seguida para llegar al estado actual, i.e. una señal de Markov

hace al futuro incondicional de pasado dado el presente. En la Figura 1 se muestra el proceso que sigue cada agente en su proceso de entrenamiento. [2]

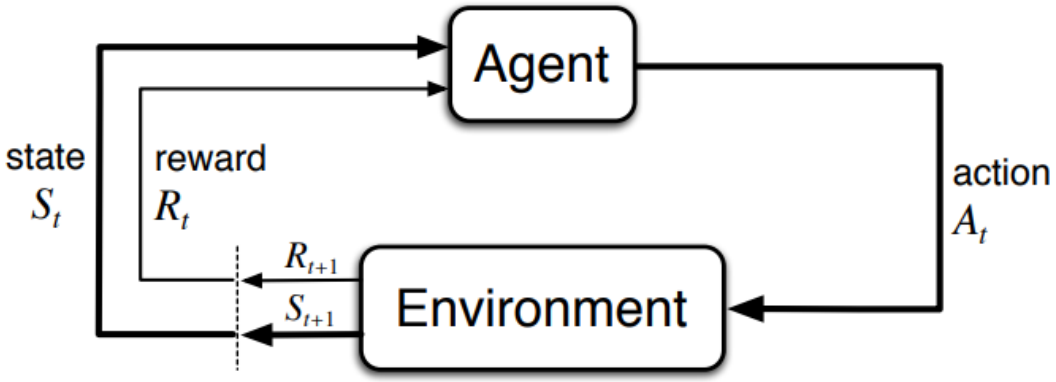


Figura 1. Interacción agente-entorno en aprendizaje por refuerzo

2.2.2 *Observaciones y estados*

Antes de definir cualquier concepto es importante conocer en qué entorno se encuentra nuestro agente. Debido a esta razón se definen las observaciones y estados de un agente, *state space* en inglés. Se trata de la información que recibe el agente del entorno en el que está. Esta información puede ser completa o incompleta, ya que el agente solo tiene la capacidad de percibir una parte parcial del entorno, aunque puede ser que la reciba completa. Dependiendo del caso se denomina de una de las siguientes maneras:

- Estado (s): es una descripción completa del entorno, es decir, de poder ver y saber todo acerca del entorno en cada situación en la que se encuentre el agente.
- Observación (o): solo se percibe una parte del entorno del estado completo en el que se encuentra el agente.

Además, dependiendo del entorno donde trabajemos, hay veces que un único *frame* o instante no nos sirve, ya que nos puede faltar información. Este concepto se denomina limitación temporal. Un ejemplo de este caso sería en un videojuego cualquiera en el que el agente necesita conocer la trayectoria para poder saber qué acción debe tomar. La solución es juntar un conjunto de *frames* seguidos para poder obtener esta información y a partir de esta seguir aprendiendo. Una vez conocemos la información que puede percibir el agente hay que saber qué normas rigen el posible comportamiento del agente, es decir, sus acciones. [2]

2.2.3 *Espacio de acciones*

Este concepto consiste en todas las posibles acciones que puede tomar el agente en el entorno. Dependiendo del entorno pueden ser discretas o continuas. Para ejemplificar el caso de acciones discretas podríamos pensar en el ajedrez; en este hay un número finito de acciones que el agente puede tomar. Por otro lado, en el caso de acciones continuas, hablaríamos de conducción

autónoma, el agente tiene una infinidad de acciones posibles. Este análisis es vital para poder elegir bien qué tipo de algoritmo se va a utilizar; para ello tendremos que considerar en qué entorno queremos entrenar a nuestro agente. A raíz de estas acciones hay que saber qué importancia dar a lo que hace nuestro agente a partir de su interacción con el entorno, equivalente a cómo aprendería un niño a partir de prueba y error hasta mejorar. [2]

2.2.4 Recompensas y descuento

Teniendo en cuenta el esquema mostrado en la Figura 1 el agente recibe una recompensa a partir de la acción que realiza en cada estado, siendo la única señal que guía el aprendizaje. La recompensa equivaldría a la evaluación por parte de un agente externo de la acción que se ha tomado ante la realización de una tarea. Es crucial por lo tanto una buena definición de esta ya que, de no definirla adecuadamente el agente podría estar muy restringido o, por el contrario, ilimitado haciendo que el entrenamiento no fuera óptimo.

Dentro de la política que va desarrollando el agente lo que se pretende es maximizar la recompensa acumulada, que es la suma de las recompensas de los estados hasta llegar al final del episodio siguiendo la política actual del agente. Además, este proyecto se va a centrar en problemas de MDPs episódicos, es decir, problemas que tienen un principio y final. Esto es importante recalcar ya que existen problemas en los que no paran nunca. siguiendo la política actual del agente. Esto genera un problema ya que puede haber acciones más arriesgadas a medida que avanza este. Debido a esto es necesario definir un parámetro de descuento, $\gamma \in [0,1)$. Este parámetro es un factor que acompaña a cada decisión tomada, que nos permite dar más o menos valor acciones futuras. Normalmente toma un valor entre 0.95 y 0.99, si el descuento es alto significa que le estamos dando más importancia a las acciones futuras, por lo que es útil para entornos que nos interesa más el largo plazo. Por otro lado, si hacemos más pequeño este estamos haciendo que el agente de más valor a las acciones iniciales haciendo que se centre más a corto plazo. La siguiente ecuación muestra el retorno descontado.

$$G(t) = r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k \cdot r_{t+k+1} \quad (1)$$

Esto es importante ya que dependiendo de la tarea que queramos que realice nuestro agente este parámetro lo tendremos que definir de una manera u otra. [2]

2.2.5 Tipos de MDPs

Al igual que pasa con las acciones que el agente puede tomar, también existe una diferencia dependiendo de la tarea que esté desarrollando el agente. El primer tipo se llaman tareas episódicas en las que hay un claro principio y final de las mismas. Por otro lado, existen las tareas continuas en las que la tarea se puede desarrollar infinitamente, es decir, no tiene un estado terminal. Un ejemplo de una tarea episódica sería una misión dentro de un juego, y como ejemplo de tarea continua podría considerarse un agente que se encarga de la compraventa de acciones. [2]

2.2.6 *Explotación vs Exploración*

Una vez conocemos que tarea va a realizar nuestro agente, el objetivo es maximizar la recompensa acumulada (el retorno descontado) y para ello hay que ajustar bien el compromiso entre exploración y explotación. Por un lado, la exploración consiste en que el agente sea capaz de realizar nuevas acciones para conseguir más información del entorno. A su vez, la explotación consiste en que con la información conocida se maximice la recompensa acumulada. Una política muy conocida es la política ε-greedy. Esta consiste en definir un parámetro $\varepsilon \in [0,1)$, que da la probabilidad de que el agente explore. Por consecuencia su complementario es la probabilidad de que el agente explote la información que ya conoce. Con todo esto en cuenta ya podemos empezar a ver qué va a regir el comportamiento de nuestro agente y cómo definimos el mismo.

2.2.7 *Métodos para resolver un MDP*

A la hora de resolver problemas de decisión existen dos variantes principales, métodos basados en política (π) y métodos basados en valores ($Q^*(s,a), V^*(s,a)$). A continuación, se explican las principales características de cada uno.

Empezando por los métodos basados en políticas, ambos consisten en que hay una función π que es el cerebro de nuestro agente. El objetivo consiste en encontrar una política que maximice la recompensa que obtenemos a partir de dicha función. Para este método se le enseña al agente qué acción tomar dado el estado actual, es decir, la función (π). Se dará como resultado la acción a tomar para cada estado. Dicha función se puede definir de dos maneras:

- Determinista: Son los tipos de función que para el mismo estado devuelve siempre la misma acción. Esto puede llegar a reflejarse en un mapa de acciones donde para cada estado está la acción más optima.

$$a = \pi(s) \ (2)$$

$$a: acción$$

$$s: estado$$

$$\pi: política$$

- Estocástica: Son los tipos de función que devuelven una distribución de probabilidad de las acciones para cada estado.

$$\pi(a \mid s) = P[a \mid s] \ (3)$$

$$a: acción$$

$$s: estado$$

$$\pi: política$$

Por otro lado, los métodos basados en valor consisten en realizar funciones que mapean cada estado de tal forma que nos dé el valor esperado para cada uno de estos estados contando con las siguientes acciones

El valor de cada estado es el retorno descontado que el agente puede llegar a obtener si empieza en ese estado y luego actúa según nuestra política, es decir, realiza la acción que lleva al siguiente estado con mayor valor.

$$V_\pi(s) = E_\pi[\gamma \cdot R_{t+1} + \gamma \cdot R_{t+2} + \gamma \cdot R_{t+3} + \cdots \mid S_t = s] \ (4)$$

En la función anterior V_π es la función valor y E_π es la recompensa total esperada descontada para el estado actual *s*. [2]

2.2.8 Métodos basados en valor

En todo método de aprendizaje por refuerzo el objetivo es maximizar la recompensa a raíz de nuestra política. La gran diferencia con los métodos basados en valores se encuentra en que en estos últimos no se busca de forma directa maximizar el comportamiento del agente, es decir, la política del mismo. Por esta razón se puede pensar que no tiene sentido ya que no se está optimizando la política directamente, pero a la hora de entrenar al agente por su función valor el agente tiene que decidir qué pareja de acción-estado debe tomar y así se consigue entrenar la política mediante exploración-explotación del entorno. Lo importante es que existe una relación entre el óptimo de la función de valor y el óptimo de la política del agente. La relación es la siguiente:

$$\pi^*(s) = \arg(max_a\, Q^*(s, a)) \ (5)$$

Una manera de llevar a cabo los problemas de decisión utilizando funciones de valor es a partir de la función de valor de estado. Esta función valor $V_\pi(s)$ devuelve la recompensa esperada si el agente empezando en ese estado sigue la política para siempre. Esta función solo nos sirve si tenemos el modelo de transiciones, es decir, la dinámica; ya que si no se puede recuperar la política a partir de la función valor. La función es la siguiente:

$$V_\pi(s) = E_\pi[G_t \mid S_t = s] \ (6)$$

Donde $E_\pi[G_t \mid S_t = s]$ es el valor esperado para el estado s, donde el agente empieza.

Por otro lado, existe la función de valor por acción ($Q_\pi(s,a)$) que es muy parecida a la anterior mencionada, pero con la diferencia de que ahora en vez de calcular el valor esperado por cada se calcula para la pareja acción-estado. La función es la siguiente:

$$Q_\pi(s,a) = E_\pi[G_t \mid S_t = s,\ A_t = a] \quad (7)$$

Donde $E_\pi[G_t \mid S_t = s,\ A_t = a]$ es el valor esperado para la pareja acción-estado s y a, donde el agente empieza. Esto puede llevar a un problema de cálculos repetitivos por lo que se necesita la ecuación de Bellman [2]

2.2.9 Ecuación de Bellman

Una problemática de estas dos funciones es que se requieren muchos cálculos que se repiten, ya que se acaban repitiendo suma de valores de estados al ir siguiendo la política hasta el final. Para reducir esta carga de cálculo se utiliza esta función:

$$V_\pi(s) = E_\pi[R_{t+1} + \gamma \cdot V_\pi(S_{t+1}) \mid S_t = s] \quad (8)$$

Con esta función calculamos una estimación del valor del estado inicial s a partir del valor del estado siguiente descontado un valor γ. De esta forma nos ahorramos muchos cálculos innecesarios y repetitivos. Esto es necesario para conseguir entrenamientos más eficiente y rápido. [2] [3]

2.2.10 Estrategias de entrenamiento

Dentro de los métodos de valor existen dos estrategias de entrenamiento distintas. Existe el método de Monte Carlo o el método de aprendizaje por diferencias temporales (DT). A continuación, se explican ambos.

El método de Monte Carlo se espera hasta el final del episodio para actualizar la función de valor. Una vez acabado el episodio a partir de lo sucedido en el mismo, se actualiza la función y se vuelve a empezar otro episodio para seguir aprendiendo. La idea es el siguiente:

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot [G_t - V(S_t)] \quad (9)$$

La expresión anterior actualiza el valor del estado estimado $V(S_t)$, teniendo en cuenta el valor estimado de la función valor anterior más un factor de aprendizaje α, siempre positivo entre

0 y 1, por la diferencia entre la recompensa y el valor de la función valor anterior estimado para un paso de tiempo *t*.

Por otro lado, el método de aprendizaje por diferencias temporales (DT) actualiza la función valor a cada interacción del agente para aprender. La diferencia con el anterior método es que, en este caso, desconocemos la recompensa final al no haber terminado el episodio. Debido a esta razón necesitamos realizar una estimación de la recompensa final de dicho episodio. Esta estimación consiste en sumar la recompensa del estado siguiente más la diferencia entre el valor de la función valor del siguiente estado multiplicado por un factor de descuento γ y el valor actual de la función valor a actualizar. A continuación, se muestra la fórmula completa:

$$V(S_t) \leftarrow V(S_t) + \alpha \cdot [R_{t+1} + \gamma \cdot V(S_{t+1}) - V(S_t)] \quad (10)$$

Al conjunto de $R_{t+1} + \gamma \cdot V(S_{t+1})$ se le considera el objetivo de diferencia temporal (objetivo TD) y es la estimación de la recompensa. A este método se le llama *bootstrapping*, debida a esta estimación. [2]

2.2.11 Algoritmo Q-Learning

Es un algoritmo basado en el aprendizaje por diferencias temporales que nos sirve para estimar la función de valor-acción óptima ($Q^*(s,a)$), es decir, que se define el valor de la función para cada estado y acción tomada en dicho estado. Se diferencia en lo comentado con anterioridad en que el algoritmo de *Q-Learning* se caracteriza por utilizar la ecuación óptima de Bellman para actualizar sus estimadores.

Esta función de valor $Q^*(s,a)$ consiste en una tabla con todas las posibles combinaciones y estados en los que se puede encontrar el agente. A raíz de esta información el agente interactúa con el entorno para ir buscando la función de valor óptima que por consecuencia maximice la política del mismo, ya que es un algoritmo basado en valor. A continuación, se muestra y explica el algoritmo que caracteriza a este método.

Algorithm 14: Sarsamax (Q-Learning)

Input: policy π, positive integer *num_episodes*, small positive fraction α, GLIE $\{\epsilon_i\}$
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)
Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$, and $Q(\textit{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ **to** *num_episodes* **do** Step 1
  $\epsilon \leftarrow \epsilon_i$
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action A_t using policy derived from Q (e.g., ϵ-greedy) Step 2
    Take action A_t and observe R_{t+1}, S_{t+1} Step 3
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$ Step 4
    $t \leftarrow t + 1$
  until S_t *is terminal*;
end
return Q

Figura 2. Algoritmo Q-Learning

- En el primer paso se inicializa la función de valor $Q^*(s,a)$ de manera arbitraria, normalmente a cero cada estado-acción de dicha función.
- En el segundo paso se escoge una acción, A_t, a partir de nuestra política según la política de exploración-explotación. Como podría ser la política ε-greedy, de las más utilizadas. Esta política consiste en que en las primeras iteraciones se busca explorar y a medida que pasa el tiempo se va reduciendo esta probabilidad para que el agente explote lo conocido del entorno y lo que ha aprendido.

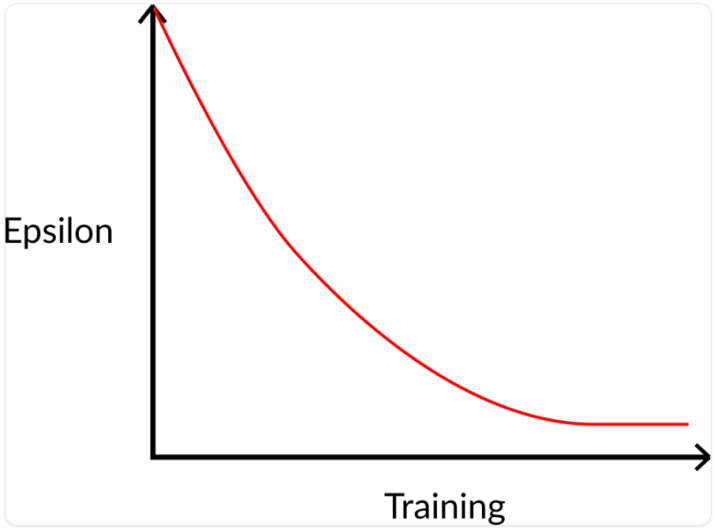


Figura 3. Gráfica política ε-greedy

- En el tercer paso el agente realiza dicha acción, pasando así al siguiente estado y obteniendo la recompensa de éste.
- En el cuarto paso se actualiza la función de valor a partir de lo ocurrido en esta interacción, ya que está basado en el método de diferencias temporales, pero con unas diferencias que se muestran a continuación.

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \cdot [R_{t+1} + \gamma \cdot max_a Q^*(S_{t+1},a) - Q(S_t,A_t)] \quad (11)$$

Como se puede observar la ecuación anterior es muy parecida a la vista en el algoritmo DT, pero cambia el DT objetivo. Esta estimación de la recompensa ahora es $R_{t+1} + \gamma \cdot max_a Q^*(S_{t+1},a)$.

Además, este método es un método llamado fuera de política, ya que como se puede observar del algoritmo se busca directamente la política óptima determinista, mientras que se emplea una política distinta para moverse por el entorno (ε-greedy).

Cabe destacar que este método requiere de muchos recursos computacionales ya que necesita ir almacenando todos los casos posibles e ir mapeando el entorno por completo para poder entrenar posteriormente al agente y conseguir alcanzar una política óptima. Por ello este

método no sirve para cualquier caso. Para solucionar este problema se necesitan realizar estimaciones ya que nos permite ahorrar muchos cálculos y recursos. Esto se consigue utilizando redes neuronales, dando así el algoritmo de *Deep Q-Learning*. [2] [3]

2.2.12 *Algoritmo Deep Q-Learning*

El termino de profundo, *Deep* en inglés, se debe al uso de redes neuronales para estimar los valores-Q para todas las posibles acciones que se pueden tomar. A continuación, se muestra un esquema comparativo entre ambos métodos:

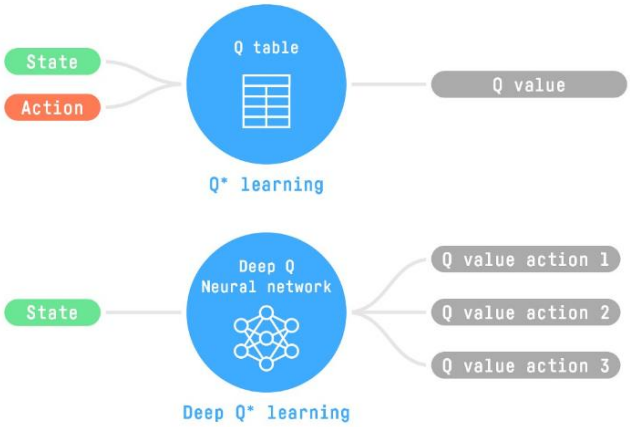


Figura 4. Comparación entre algoritmo Q-Learning y Deep Q-Learning

La estructura consiste en recibir el estado en el que se encuentra el agente como entrada a la red neuronal. Para que este entrenamiento sea lo más eficiente posible es imprescindible reducir el preprocesado de dicha entrada. Dependiendo del entorno y la tarea del agente se puede ir simplificando ciertos aspectos del mismo como, por ejemplo, los colores RGB para reducirlo a blanco y negro o incluso reducir las propias geometrías a geometrías más sencillas si la tarea no necesita de esta información. Otro aspecto importante de este preprocesado sería juntar *frames* para eliminar la limitación temporal comentada con anterioridad si la tarea lo necesita. A continuación, y dependiendo del caso, se pasaría la información a una red neuronal que tendrá una arquitectura optimizada para el problema que se esté resolviendo. En la Figura 5 se muestra un ejemplo de cómo funcionaria. [2]

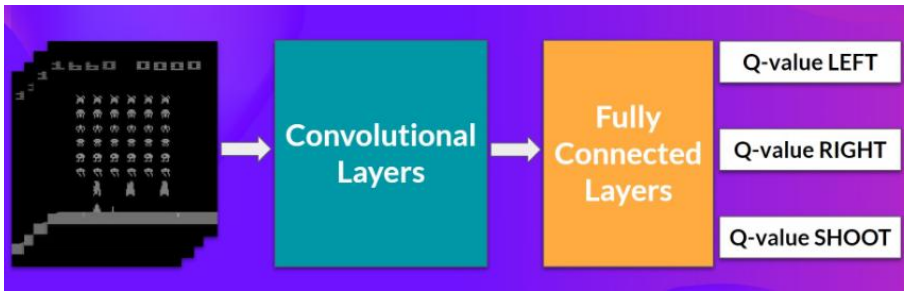


Figura 5. Estructura básica de un algoritmo Deep Q-Learning

Una vez se hace este análisis pasando por el resto de la red neuronal se obtienen un vector con los valores-Q para cada posible acción que se puede realizar en dicho estado. Con esta información ya se puede comenzar el proceso de entrenamiento con el algoritmo pertinente.

Debido a la red neuronal el algoritmo de aprendizaje no puede ser igual al anterior ya que no podemos actualizar nuestra función de valor-Q, sino que ahora vamos a analizar dos funciones distintas que son la pérdida de valor-Q y el objetivo-Q. Por un lado, tenemos la función objetivo-Q que contiene los valores-Q estimados por nuestra red neuronal y es la siguiente:

$$y_j = r_j + \gamma \cdot max_{a'} \hat{Q}(S_{t+1}, a'; \theta^-) \quad (12)$$

Donde r_j es la recompensa inmediata y $\gamma \cdot max_a Q^*(S_{t+1}, a; \theta^-)$ es la estimación futura de la mejor acción en el siguiente estado utilizando la red neuronal de parámetro θ^-. A partir de esta función se genera la función de pérdida de valor-Q que nos permite ajustar ya los pesos de nuestra red neuronal. La función es la siguiente:

$$y_j - Q(\phi_j, a_j; \theta) \quad (13)$$

Donde ϕ_j es una representación o característica de los datos que se utilizan como entrada para la red neuronal. Esta ecuación consiste en la diferencia entre la función objetivo-Q mencionada anteriormente y el valor-Q de nuestra red neuronal.

Con este planteamiento se puede desarrollar un algoritmo de entrenamiento que se divide en dos fases. Por un lado, tenemos la fase de muestro, en el que se realizan acciones a partir de la política de exploración definida, como puede ser (ε-greedy), y se guardan en una tupla de memoria (*s, a, r, s', done*); que consiste en un vector donde se guarda el estado actual *s*, la acción tomada *a*, la recompensa obtenida *r*, el estado siguiente *s'* y un índice que indica si el estado siguiente es el terminal *done*. Después, está la fase de entrenamiento donde se elige una pequeña muestra de tuplas aleatorias y aprende de cada muestra utilizando una política de descenso de gradiente, a esto se le llama repetición de experiencia. El algoritmo completo es el siguiente:

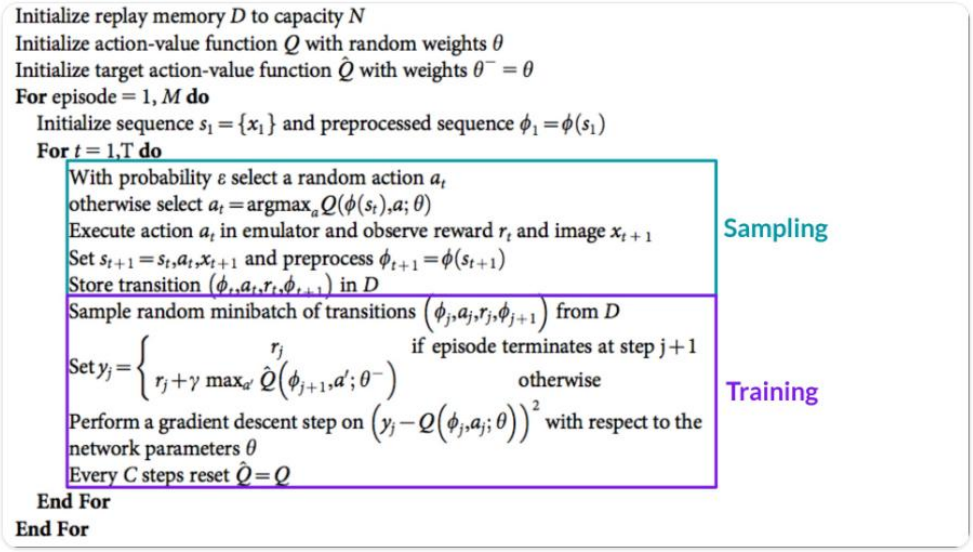


Figura 6. Algoritmo Deep Q-Learning

A diferencia de los algoritmos y técnicas de entrenamiento anteriores ahora hay un problema de estabilidad durante el proceso. Este se debe al combinar el *bootstrapping*, cuando actualizamos la función a partir de recompensas estimadas, y la función de valor-Q no lineal por las redes neuronales. Para poder estabilizar el entrenamiento se utiliza la experiencia de repetición, Q-objetivo fijo y *Double Deep Q-Learning*. A continuación, se explicará cada concepto con más detalle. [2] [3]

2.2.13 Experiencia de repetición

Tener una memoria de repetición permite al agente un uso de experiencias de entrenamiento más eficientes. Normalmente el agente cuando interactúa con el entorno consigue experiencias (estado, acción, recompensa y siguiente estado), aprende de ellas, actualiza los pasos de la red neuronal y elimina la información empleada. Esto no es eficiente ya que se utiliza información valiosa una sola vez. Para solucionar esto se guarda esta experiencia para que el agente pueda volver a aprender de ella durante el proceso. A continuación, se muestra donde se ve reflejado en el algoritmo mostrado anteriormente.

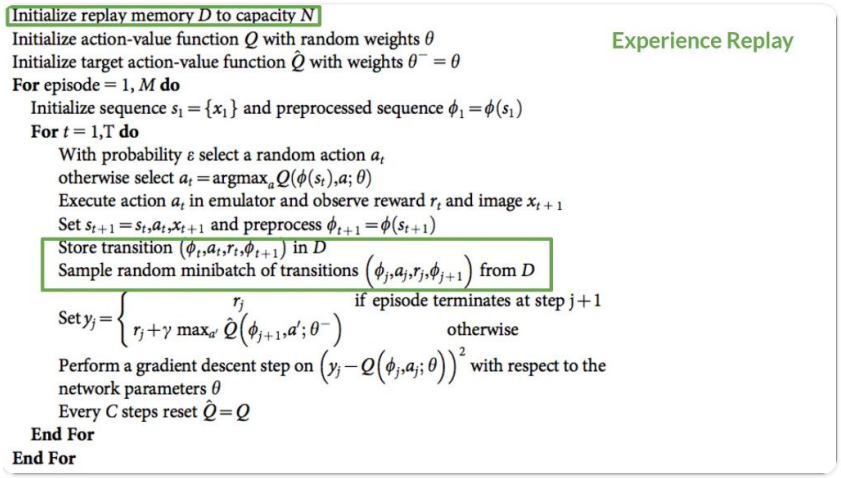


Figura 7. Experiencia de repetición

Por otro lado, también permite evitar olvidar experiencias previas, en *inglés catastrophic interference* o *catastrophic forgetting*, y reducir la correlación entre experiencias. El *catastrophic interference* es un concepto que se refiere a que al darle nuevas experiencias a la red neuronal tiende a olvidar las experiencias ya aprendidas de otras tareas. Un ejemplo de esto sería en una serie de tareas secuenciales cuando terminar de aprender bien la primera tarea y empieza a entrenar en la segunda la red neuronal se olvida de las experiencias de la primera tarea.

Este problema se arregla con una memoria de repeticiones que permiten al agente recordar lo hecho anteriormente y así no olvidar lo ya aprendido. Además, ayuda a eliminar la correlación en las secuencias de observación y evita que los valores de decisión oscilen o diverjan catastróficamente. [2] [3]

2.2.14 Q-objetivo fijo

Como se ha mencionado previamente este algoritmo, para aumentar su eficiencia, se basa en el cálculo de la diferencia entre el Q-objetivo y el actual Q-valor. Para realizar ambas estimaciones se utilizan los mismos parámetros de la red neuronal, esto implica que varían ambos a la vez. Esto condiciona que, a medida que se acerca al objetivo durante el entrenamiento, este se mueve constantemente; todo ello lleva a grandes oscilaciones durante el proceso. Esto implica un problema que puede solucionarse mediante el inicio de una segunda red neuronal para la estimación de la Q-objetivo que vaya actualizándose conforme avanza el entrenamiento. A continuación, se muestra en qué parte del algoritmo se encuentra. [2] [3]

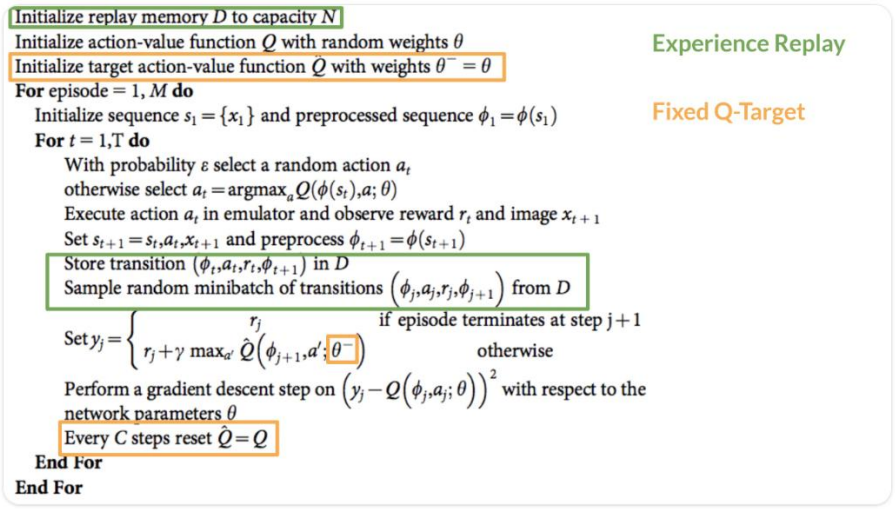


Figura 8. Q-objetivo fijo

2.2.15 Doble Deep Q-Learning

Puede darse el caso en el que haya una sobrestimación de parámetros que esté dando más importancia a acciones que no tienen por qué ser las óptimas para ese estado. Esto ocurre sobre todo al principio del entrenamiento ya que la precisión de la estimación depende de cuantos estados hayamos explorado y la experiencia del agente. Para solucionar esto se necesitan dos redes neuronales como en el problema de Q-objetivo fijo. Una de las redes neuronales para seleccionar la mejor acción, la acción con un Q-valor más alto, y otra para calcular el Q-valor objetivo de realizar dicha acción en el siguiente estado. Esto nos ayuda a reducir la sobreestimación de parámetros y consecuentemente tener un entramiento más corto, rápido y eficiente. Estos son los métodos más comunes de métodos basados en valor, a continuación, se explicarán los métodos basados en políticas. [2]

2.2.16 Métodos basados en políticas

El objetivo final del aprendizaje por refuerzo consiste en encontrar una política óptima (π^*) que maximiza la recompensa acumulada. Para conseguir esto se puede hacer de varias maneras diferentes, ya que se puede optimizar directa o indirectamente. En el caso de los métodos basados en políticas se trata de hacerlo directamente sobre la función π^*. Para conseguir esto se parametriza dicha función, ya sea con una red neuronal θ, que nos da una distribución de probabilidad de cada acción a tomar para cada estado o con una política estocástica.

$$\pi_\theta(s) = P[A|s;\theta] \quad (14)$$

Nuestro objetivo es maximizar dichos parámetros para maximizar el rendimiento de la política parametrizada, esto se puede hacer de diferentes formas como *Hill Climbing*, *Simulated*

Annealing, *Evolution Strategies* o utilizando un ascenso de gradiente. Este último es en un subconjunto de los métodos basados en políticas llamados métodos de gradiente de política, en los que nos vamos a centrar en este proyecto. [2]

2.2.17 Métodos de gradiente de política (Policy-Gradient methods)

Esta familia de métodos ataca directamente el problema de control, esto es, la optimización de una política objetivo para obtener el máximo retorno descontado. En este caso, al estar entrenando directamente la política del agente y esta ser estocástica, buscamos que durante el entrenamiento se vaya dando más preferencia a las acciones que maximicen la recompensa. Para eso hará más o menos probable a las acciones dependiendo de si la recompensa para el episodio ha sido positiva o negativa.

Para poder saber cómo de buena es nuestra política estocástica necesitamos definir una función objetivo $J(\theta)$. Esta función nos indica el rendimiento del agente dada una secuencia de estados-acciones (τ), pero sin contar la recompensa y nos devuelve la recompensa acumulada esperada $(R(\tau))$.

$$J(\theta) = E_{\tau\sim\pi}[R(\tau)] = \sum_{\tau} P(\tau;\theta)\cdot G(\tau) \quad (15)$$

$$G(\tau) = r_{t+1} + \gamma\cdot r_{t+2} + \gamma^2\cdot r_{t+3} + \gamma^3\cdot r_{t+4} + \cdots (16)$$

$$P(\tau;\theta) = [\prod_{t=0} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)] \quad (17)$$

La recompensa acumulada esperada $J(\theta)$ es la media ponderada de las recompensas, donde los pesos vienen dados por $P(\tau;\theta)$ de todos los valores posibles que la recompensa acumulada puede dar. Como ya se ha visto en otras técnicas, $R(\tau)$ es la recompensa de una trayectoria τ arbitraria y $P(\tau;\theta)$ es la probabilidad de cada trayectoria, que normalmente dependerá de un parámetro ya que este viene definido por la política del agente. Con estas funciones se puede analizar el funcionamiento del gradiente ascendente hasta que converja en un máximo. [2]

2.2.18 Ascenso del gradiente

El objetivo del entrenamiento es encontrar los valores de los parámetros θ que maximicen la recompensa acumulada esperada $J(\theta)$. Esto al ser un problema de optimización se puede resolver a partir del ascenso de gradiente. El proceso iterativo de gradiente ascendente es el siguiente.

$$\theta \leftarrow \theta + \alpha\cdot\nabla_\theta J(\theta) \ (18)$$

Aun así, no se puede aplicar directamente ya que al buscar el máximo global necesitaríamos calcular todas las probabilidades de todas las trayectorias posibles. Este cálculo consume una cantidad sustancial de recursos lo que lo hace muy complicado implementarse directamente; para solucionarlo se realiza una estimación para que sea factible. Por otro lado, para poder derivar la función objetivo, necesitamos derivar la distribución de estados, a lo que se llama la dinámica de un Proceso de Decisión de Markov. Este problema va ligado al entorno y nos da la probabilidad de que el entorno pase al siguiente estado, dadas el estado y acción actual del agente. El problema es que hay situaciones que no lo podemos diferenciar porque no tenemos dicha información y depende de la política.

Gracias al teorema de gradiente de política, *policy gradient theorem* en inglés, se nos permite reformular nuestra función política objetivo de tal manera que la hace diferenciable y no involucra la diferenciación de una distribución de estados. El teorema nos indica que para cualquier política diferenciable y para cualquier función política objetivo, el gradiente de la política es el siguiente:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\pi_\theta(a_t|s_t)R(\tau)] \ (19)$$

Una vez ya tenemos nuestra función diferenciable pasamos a analizar el proceso de entrenamiento del agente, llamado el método de refuerzo de Monte Carlo. [2]

2.2.19 Reinforce (Ascenso del gradiente de MC)

Este algoritmo de entrenamiento utiliza la recompensa esperada de un episodio completo para actualizar el parámetro θ de nuestra red neuronal. Se puede aplicar a una única trayectoria o a múltiples trayectorias a la vez. Primero analizaremos la manera de resolver una trayectoria.

El algoritmo de entrenamiento consiste en un bucle obtener un episodio τ a partir de la política π_θ. A partir de este episodio estimar el gradiente $\hat{g} = \nabla_\theta J(\theta)$ y por último actualizar los pesos de la política del agente, $\theta \leftarrow \theta + \alpha \cdot \hat{g}$. A continuación, se muestra la estimación del gradiente para una sola trayectoria:

$$\nabla_\theta J(\theta) \approx \hat{g} = \sum_{t=0} \nabla_\theta log\pi_\theta(a_t|s_t)R(\tau) \ (20)$$

En la expresión anterior el $\nabla_\theta log\pi_\theta(a_t|s_t)$ nos da la dirección de mayor probabilidad del algoritmo. Con esta información sabemos cómo debemos cambiar los parámetros para aumentar o disminuir la probabilidad logarítmica de escoger la acción a_t en el estado s_t. Por otro lado, $R(\tau)$ es la función de puntuación para los episodios en la que si el resultado es alto se aumenta la probabilidad de la acción a_t en el estado s_t. Por el contrario, si el resultado es bajo se disminuye dicha probabilidad.

Además, esto se puede extrapolar a múltiples trayectorias permitiendo al agente hacer la media de los gradientes de diversos episodios y así conseguir un entrenamiento que es más estable y reduce la varianza en las actualizaciones de los pesos. La diferencia con la estimación de una trayectoria es que es necesario añadir un factor de escala que es inverso al número de trayectorias a tener en cuenta, m. A continuación, se muestra la estimación del gradiente $\hat{g}$:

$$\nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \cdot \sum_{i=1}^{m} \sum_{t=0} \nabla_\theta log \pi_\theta({a_t}^{(i)} | s^{(i)}) R(\tau^{(i)}) \quad (21)$$

Una vez tenemos la función objetivo del algoritmo *Reinforce* se puede mejorar dicho algoritmo utilizando el algoritmo de *Proximal Policy Optimization*. [2]

2.2.20 Proximal Policy Optimization (PPO)

El algoritmo PPO no es un método basado en política puro ya que hay una parte de método basado en valor. A esto se le conoce como algoritmos *Actor-Critic*, en los que existe una mezcla de estos dos. El proceso de estos algoritmos consiste en tener nuestra política ($\pi_\theta(s)$) que define al agente como debe actuar y, además, tenemos una función de valor ($\hat{q}_w(s,a)$)) que nos mide como de buena es la acción que el agente escoge. El flujo del proceso es el siguiente:

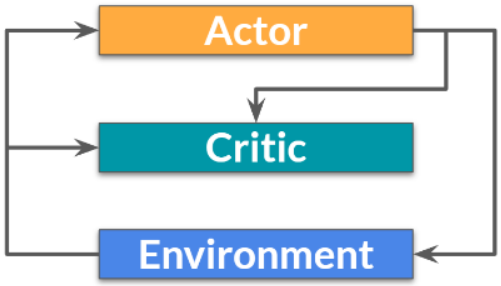


Figura 9. Esquema agentes estructura Actor-Critic

Para una instancia, t, dentro de un episodio se obtiene el estado actual S_t del entorno que es la entrada del Actor y del Crítico. A partir de la política el Actor realiza una acción A_t que a su vez es otra entrada para el Crítico, teniendo así el par de acción-estado (A_t, S_t). Con este par el Crítico nos da el Q-valor para dicho par. A continuación, el agente realiza la acción (A_t) en el entorno obteniendo así el siguiente estado (S_{t+1}) y la recompensa (R_{t+1}). A partir de esto, el Actor actualiza sus parámetros utilizando el Q-valor obtenido del Crítico. Después el Actor vuelve a producir una nueva acción (A_{t+1}) para este nuevo estado y el Crítico actualiza sus parámetros a partir del nuevo par de acción-estado. Se puede ver claramente que la política y la estimación de la función de valor son independientes.

Esto consigue estabilizar el entrenamiento más todavía ya que con una función de ventaja permite al agente saber cómo de bueno es tomar una acción para un estado en comparación con el valor medio de ese estado. La función ventaja es la siguiente.

$$A(s,a) = Q(s,a) - V(s) \ (22)$$

Donde $A(s,a)$ es la función ventaja, $Q(s,a)$ es el Q-valor para para estado-acción y $V(s)$ es el valor medio para dicho estado. Como se están utilizando dos funciones de valor diferentes es necesario estimar una de ellas. Por ello para estimar la función Q-valor utilizamos el error TD $(r + \gamma \cdot V(s') - V(s))$, ya que es un muy buen aproximador estocástico de la función de ventaja. En la práctica, se suele usar el error TD (lambda), con lambda ejerciendo como un hiperparámetro de equilibrio entre sesgo y varianza que permite estabilizar el método y mejorar su eficiencia muestral. Quedando dicha estimación de la siguiente forma.

$$A(s,a) = r + \gamma \cdot V(s') - V(s) \ (23)$$

Analizando la parte del método basada en política PPO consigue mejorar la estabilidad del agente durante el entrenamiento, ya que su característica principal es que trata de acotar las actualizaciones en la política para que no sean muy grandes, específicamente en un rango entre $[1-\varepsilon, 1+\varepsilon]$. Esto es muy interesante ya que empíricamente está demostrado que teniendo actualizaciones más pequeñas en la política se consigue un entrenamiento más estable. Por otro lado, puede conllevar que el agente acabe con una política en la que sea complicado recuperarse durante el entrenamiento, haciendo que necesite más tiempo o incluso que no sea efectivo.

Teniendo en cuenta la función objetivo del algoritmo de *Reinforce*, ecuación (15), la idea es que a partir del aumento de gradiente conseguimos que durante el entrenamiento el agente vaya realizando acciones que maximicen su recompensa y minimicen acciones negativas. Debido a su estructura si la función tiene actualizaciones muy grandes el proceso es demasiado lento y si no son lo suficientemente grandes hay demasiada variabilidad en el entrenamiento. Para solucionar esto la función objetivo del algoritmo PPO, llamada *Clipped Surrogate Function*, limita los cambios de política en un intervalo pequeño a partir de recortes, clips en inglés. La función es la siguiente:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\cdot \hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\cdot \hat{A}_t\right)\right] \ (24)$$

Para entender mejor esta función vamos a ir desglosando cada parte. Por un lado, tenemos la función ratio $(r_t(\theta))$. Esta es el cociente de la probabilidad de escoger la acción a_t en el estado s_t de la política actual y la probabilidad de la política para el mismo par de acción-estado. La función ratio es la siguiente:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{antigua}}(a_t|s_t)}$$

Este cociente nos permite determinar la variación en la probabilidad del agente de tomar dicha acción en el estado de la política anterior con la actual y así determinar la tasa de cambio del agente. Por otro lado, tenemos la parte sin recortar (min $(r_t(\theta) \cdot \hat{A}_t)$ que con la función ratio podemos sustituir la probabilidad logarítmica de la función objetivo del algoritmo de refuerzo de Montecarlo y multiplicarlo por la ventaja, la puntuación en el algoritmo de Montecarlo. Obteniendo así la siguiente ecuación:

$$L^{CPI}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{antigua}}(a_t|s_t)} \cdot \hat{A}_t\right] = \hat{E}_t[r_t(\theta) \cdot \hat{A}_t] \quad (25)$$

A pesar de esto no desaparecería el problema de las grandes actualizaciones en la política, por lo que es necesaria la segunda parte de la ecuación, la parte recortada ($clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon) \cdot \hat{A}_t$)). Con esto definimos cuanto restringimos la tasa de actualización del agente durante el entrenamiento, normalmente se restringe para que la función ratio este entre $[0.8, 1.2]$. Gracias a la nueva definición de la función objetivo se tienen dos ratios de probabilidad, uno recortado y otro sin recortar. Utilizando un criterio pesimista para reducir la tasa de actualización nos quedamos con el valor menor de estos. Aun así, nos falta por definir en que situaciones de las que se pueden dar queremos o no actualizar los parámetros para cumplir con el objetivo de obtener cambios lo menos bruscos posibles en la política del agente. A continuación, se muestra una tabla donde se explican las posibles situaciones que se pueden dar y qué hacer con el gradiente.

$p_t(\theta) > 0$	A_t	**Return Value of** *min*	**Objective is Clipped**	**Sign of Objective**	**Gradient**
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	−	$p_t(\theta)A_t$	no	−	✓
$p_t(\theta) < 1-\epsilon$	+	$p_t(\theta)A_t$	no	+	✓
$p_t(\theta) < 1-\epsilon$	−	$(1-\epsilon)A_t$	yes	−	**0**
$p_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$	yes	+	**0**
$p_t(\theta) > 1+\epsilon$	−	$p_t(\theta)A_t$	no	−	✓

Tabla 1. Situaciones posibles y como actualizar el gradiente para el algoritmo PPO

Como se puede observar en la tabla anterior solo nos interesa actualizar la política cuando el resultado de nuestra función es la parte sin recortar, si es la parte recortada no se actualiza. Esto se debe a que cuando el mínimo es la parte recortada su derivada es 0 ya que está fijada en el máximo o mínimo de nuestro intervalo. Además, nos interesa actualizar la política si la ventaja nos acerca a estar dentro del intervalo después de la actualización, es decir, si estamos por debajo del ratio, pero la ventaja es positiva o si estamos por encima del ratio, pero la ventaja es negativa.

Una vez entendida la parte de la política del algoritmo de PPO vemos la función objetiva final de dicho algoritmo en la que se ve reflejada la parte la función de valor ya que es un método Actor-Critic. La función objetivo es la siguiente.

$$L_t^{CLIP+VF+S}(\theta) = \hat{E}_t[L_t^{CLIP}(\theta) - c_1 \cdot L_t^{VF}(\theta) + c_2 \cdot S[\pi_\theta](s_t)] \quad (26)$$

Como se puede observar la función objetivo recortada explicada anteriormente ($L_t^{CLIP}(\theta)$), como se vio en los métodos basados en valor, la función de pérdida de valor al cuadrado ($L_t^{VF}(\theta) = (V_\theta(s_t) - V_+^{objetivo})^2$) y un bonus de entropía extra ($S[\pi_\theta](s_t)$) para asegurar suficiente exploración del agente. Donde van acompañadas estas dos últimas por sus respectivos coeficientes a ajustar c_1 y c_2. [6] [4]

Capítulo 3 Metodología

En este capítulo se explican las librerías utilizadas para el entorno, el Problema de Decisión de Markov, las técnicas de los agentes utilizados y el análisis de los mismos.

Se describe el entorno con todas sus características, la función objetivo, las recompensas, las acciones y la definición de estado. También se describe las estructuras de la Red Neuronal, la optimización de la misma con sus hiperparámetros y política de exploración-explotación para los algoritmos utilizados, PPO y DQN.

3.1 Descripción General de la Metodología Implementada

En este trabajo se ha desarrollado una metodología para realizar un análisis en un único entorno sencillo con un Problema de Decisión de Markov para facilitar la comparación entre las siguientes técnicas:

- PPO
- DQN

Para realizar dicho análisis se define el Problema de Markov a resolver, definiendo bien sus estados, acciones y recompensas. Además, es necesario definir un programa para hacer dicho entrenamiento y análisis en python. El entorno y las herramientas son las siguientes:

- Entorno LunarLander-v2 de la librería Box2D-py 2.3.5
- Librería StableBaslines3 2.3.2
- Librería Gymnasium 0.29.1
- Librería Weights and Biases
- Python 3.9.19

En la sección 3.2 se describen el entorno, en la 3.3 los algoritmos con sus estructuras individuales e hiperparámetros y en la 3.4 el código con el que se ha realizado dicho proyecto.

3.2 Entorno LunarLander-v2 de la librería Box2D-py

El entorno LunarLander-v2 de gym es un entorno perteneciente a la librería de Box2D. Box2D es una librería de simulaciones basados en física clásica que tiene diferentes entornos sencillos de videojuegos con distintos parámetros a cambiar como la gravedad, fricción o colisiones. A continuación, se muestra una figura del agente actuando en dicho entorno. [6]

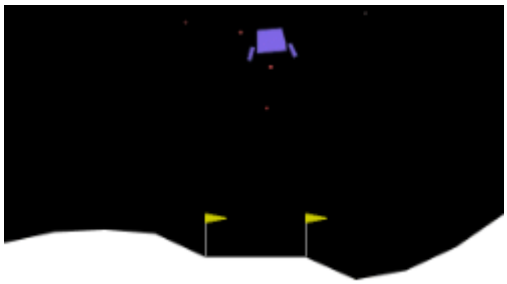


Figura 10. Agente y entorno LunarLander-v2

El entorno LunarLander-v2 es una versión del problema clásico de optimización de trayectoria de cohetes y en la librería de Box2D existen dos versiones, discreta y continua. La versión utilizada es la discreta por sencillez. Además, debido al principio mínimo de Pointriaguin está demostrado que el control óptimo en la trayectoria de cohetes consiste en activar los motores al máximo cada vez que sean necesarios, por lo que el agente solo puede encender o apagar los motores. Para este proyecto se utiliza este entorno para el análisis debido a su sencillez y así facilitar una comparación entre distintos agentes. A continuación, se muestra un resumen del entorno con sus características y como se inicializan dichas variables por defecto. [7]

Action Space	Discrete(4)
Observation Shape	(8,)
Observation High	[1.5 1.5 5. 5. 3.14 5. 1. 1.]
Observation Low	[-1.5 -1.5 -5. -5. -3.14 -5. -0. -0.]
Import	`gym.make("LunarLander-v2")`

Tabla 2. Características entorno LunarLander-v2

```python
import gym
env = gym.make(
    "LunarLander-v2",
    continuous: bool = False,
    gravity: float = -10.0,
    enable_wind: bool = False,
    wind_power: float = 15.0,
    turbulence_power: float = 1.5,
)
```

Figura 11. Argumentos inicializados por defecto del entorno LunarLander-v2

3.2.1 Recompensas del entorno LunarLander-v2

El objetivo principal del agente, una nave espacial, es aterrizar de forma segura en la zona delimitada por las banderas. El agente comienza en el centro de la ventana con una fuerza inicial aleatoria en su centro de masa. Tanto la zona de aterrizaje como la superficie cambia en cada episodio, esto implica que el agente debe aprender a mantener el equilibrio controlando su velocidad y posición para aterrizar en la zona permitida y sin chocarse. A continuación, se muestra la estructura de las recompensa del agente:

- Recompensa por que el agente sea capaz desde el estado inicial y aterrizar en la zona permitida es de $[100, 140]$ puntos, ya que si el agente se aleja de la zona permitida pierde una pequeña cantidad de recompensa.
- Recompensa por que el agente se choque recibe -100 puntos.
- Si el agente es capaz de aterrizar y quedarse quieto en la zona permitida recibe 100 puntos.
- Por realizar contacto con la superficie con uno de sus apoyos recibe 10 puntos.
- Cada vez que se utiliza el motor principal el agente recibe una recompensa de -0.3 puntos.
- Por utilizar los motores laterales el agente recibe una recompensa de -0.03 puntos.

El episodio se termina cuando la nave aterriza y se detiene, colisiona o si el agente esta dormido, es decir, que no se mueve o interactúa con ningún elemento. Se considera exitoso si el agente ha obtenido una recompensa de 200 puntos. Una vez definida las recompensas del agente es necesario conocer el espacio de acciones que puede realizar. [7]

3.2.2 Espacio de acciones y espacio de observaciones

Para poder resolver cualquier Problema de Markov es necesario entender tanto que información recibe el agente, como las acciones que puede tomar el mismo. Además, es importante saber, cómo se describe en el Capítulo 2 , si este espacio de acciones es continuo o discreto. Como se ha comentado al inicio de este capítulo el entorno que utilizamos tiene un espacio de acciones discreto, con 4 acciones posibles para el agente que son las siguientes:

- No realizar ninguna acción, es decir, dejar que la nave caiga por la fuerza de la gravedad.
- Encender el motor principal.
- Encender motor izquierdo.
- Encender motor derecho.

Por otro lado, el espacio de observaciones del agente para el entorno LunarLander-v2 es un vector de 8 dimensiones que representa información del estado de la nave. Dicho vector es el siguiente:

- Posición horizontal de la nave x.
- Posición vertical de la nave y.
- Velocidad horizontal $\frac{dx}{dt}$.
- Velocidad vertical $\frac{dy}{dt}$.

- Ángulo de la nave θ.
- Velocidad angular $\frac{d\theta}{dt}$.
- Dos variable booleanas que indican si el apoyo izquierdo o derecho están en contacto con el suelo respectivamente.

Con esta información el agente tiene que tomar las decisiones para así poder resolver el problema de manera segura y exitosa. Una vez conocido la estructura del entorno que vamos a utilizar para entrenar los agentes, se analizan las estructura utilizadas para cada agente. En primer lugar, se va a comenzar analizando el agente que utiliza el algoritmo de *Deep Q-Learning* (DQN). En el siguiente apartado se explica en detalle dicha estructura. [7]

3.3 Estructura Deep Q-Learning (DQN)

A partir de la librería utilizada de StableBaselines3 se define la arquitectura del agente con sus hiperparámetros. A continuación, se muestra la estructura del mismo.

```
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 500000,
    "env_name": "LunarLander-v2",
    "learning_starts": 0,
    "batch_size": 128,
    "buffer_size": 100000,
    "learning_rate": 7e-4,
    "target_update_interval": 250,
    "train_freq": 1,
    "gradient_steps": 4,
    "exploration_fraction": 0.08,
    "exploration_final_eps": 0.05,
    "policy_kwargs": dict(net_arch=[256, 256])
}
```

Figura 12. Arquitectura del agente que utiliza DQN

Como se observa en la figura 12 hemos definido los diferentes parámetros, pero sobre todo es importante destacar que utilizamos una ratio de exploración-explotación de $\varepsilon - greedy$. Como se comento en el capitulo dos es una política muy utilizada y en este caso permite resolver el problema de una manera correcta y eficiente. El resto de hiperparámetros se definen mediante prueba y error con analizando la eficiencia del agente durante el entrenamiento.

3.3.1 Red neuronal

La red neuronal utilizada tiene un tipo de política "$MlpPolicy$" que indica que la red neuronal es un perceptrón multicapa totalmente conectada. Este tipo de estructura consiste en que cada neurona de la capa anterior está conectada a cada neurona de la siguiente capa. Esta arquitectura viene definida en "$policy\_kwargs$" con "$net\_arch = [256, 256]$" y por defecto sino

se especifica nada tiene como función de activación ReLu. Esto indica que la red neuronal tiene dos capas ocultas y cada una de estas 256 neuronas.

Como se definió en el Capítulo 2 la red neuronal tiene como capa de entrada las observaciones recibidas por el entorno, es decir, el espacio de observaciones. Esta capa solo manda dicha información a la primera capa oculta. La segunda y la tercera capa son capas ocultas que contienen las 256 neuronas cada una conectadas a las salidas de la capa anterior. Estas aplican una transformación lineal a los datos y luego pasan el resultado a través de la función ReLU (Rectified Linear Unit). Esta función es común en redes neuronales profundas, ya que introduce no linealidades sin que aparezcan problemas de gradiente que desaparece como otras funciones de activación. La cuarta y última capa es la capa de salida que proporciona los valores Q de cada acción a tomar para cada estado. Esta está formada solo por cuatro neuronas que son el espacio de acciones mencionados. [8]

3.3.2 Optimizador

El optimizador es un algoritmo que se encarga de ir ajustando los pesos de la red neuronal para así ir minimizando la función de pérdida. Se utiliza el optimizador por defecto de StableBaselines3, que es el optimizador Adam. Este optimizador se basa en otros dos, AdaGrad y RMSProp, y sirve para ajustar la tasa de aprendizaje de la red neuronal en función de la frecuencia y la magnitud de los gradientes que se van calculando durante el entrenamiento. Esto permite calcular la tasa de aprendizaje de cada parámetro de la red, permitiendo una optimización más estable y eficiente.

Adam para su funcionamiento mantiene dos vectores para cada neurona de la red. El primer vector es el primer momento de los gradientes, es decir, realiza el promedio de los gradientes acumulados así obteniendo una especie de velocidad y dirección general de estos. El segundo vector es el segundo momento de los gradientes. Esto es necesario para poder contrarrestar el sesgo que se introduce al inicializar los parámetros a cero. Con los momentos Adam actualiza los parámetros restando una fracción del primer momento de cada parámetro, dividida por la raíz cuadrada del segundo momento. Esta fracción está controlada por la tasa de aprendizaje. Una ventaja de Adam es que cada parámetro tiene su tasa de aprendizaje individual. Esta adaptabilidad permite que la red converja de manera rápida y fiable. A continuación, se comentan los parámetros de Adam y explicando sus valores:

- Tasa de aprendizaje, α $(learning\_rate = 1e-4)$

La tasa de aprendizaje define el paso con el que el optimizador va ajustando los pesos de la red neuronal a partir del cálculo del gradiente. El valor por defecto utilizado es un valor frecuente en entornos donde los gradientes pueden ser ruidosos.

- Parámetro 1, β_1 $(beta\_1 = 0.9)$

Este parámetro define la tasa de decaimiento exponencial para el promedio móvil del primer momento, es decir, define con que rango de gradientes pasados se va quedando para realizar la media. Con el valor por defecto nos permite almacenar un buen historial

- Parámetro 2, β_2 $(beta\_1 = 0.999)$

Define el decaimiento exponencial para el promedio móvil del segundo momento. Es igual que el β_2 pero con la varianza de los gradientes. En este caso el valor es más alto que β_1 ya que el entrenamiento de aprendizaje por refuerzo suele ser mucho más ruidoso y así conseguir un entrenamiento más estable.

- ε $(epsilon = 1e - 7)$

Epsilon es un parámetro necesario pequeño para que no haya divisiones por cero durante la actualización de la red. Debido a esto el valor por defecto que se utiliza es lo suficiente pequeño para que no afecte, pero si cumplir su función, proporcionando más estabilidad durante el proceso.

- AMSGrad $(amsgrad = false)$

Es una variante del optimizador Adam que nos permite poner una condición estricta en el promedio móvil del segundo momento, lo que lleva a un entrenamiento más estable y largo. En nuestro caso la red usada es muy simple así que no es necesario activar esta variante. [8] [9]

3.4 Estructura Proximal Policy Optimización (PPO)

A partir de la librería utilizada de StableBaselines3 se define la arquitectura del agente con sus hiperparámetros. A continuación, se muestra la estructura del mismo.

```
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 2500000,
    "env_name": "LunarLander-v2",
    "learning_rate": 0.001,
    "n_steps": 1024,
    "batch_size": 64,
    "n_epochs": 5,
    "gamma": 0.999,
    "gae_lambda": 0.98,
    "ent_coef": 0.01,
    "vf_coef": 0.5,
    "max_grad_norm": 0.5
}
```

Figura 13. Arquitectura del agente que utiliza PPO

Como se observa en la figura 12 hemos definido los diferentes parámetros. Como se comentó en el capítulo dos es una política muy utilizada y en este caso permite resolver el problema de una manera correcta y eficiente. El resto de hiperparámetros se definen mediante prueba y error con analizando la eficiencia del agente durante el entrenamiento.

3.4.1 Red neuronal

La red neuronal utilizada tiene un tipo de política "$MlpPolicy$" que indica que la red neuronal es un perceptrón multicapa totalmente conectada. Este tipo de estructura consiste en que cada neurona de la capa anterior está conectada a cada neurona de la siguiente capa. Esta arquitectura viene definida por defecto en "$policy\_kwargs$" con "$net\_arch = [64, 64]$" y por defecto sino se especifica nada tiene como función de activación ReLu. Esto indica que la red neuronal tiene dos capas ocultas y cada una de estas 64 neuronas.

Como se definió en el Capítulo 2 la red neuronal tiene como capa de entrada las observaciones recibidas por el entorno, es decir, el espacio de observaciones. Esta capa solo manda dicha información a la primera capa oculta. La segunda y la tercera capa son capas ocultas que contienen las 64 neuronas cada una conectadas a las salidas de la capa anterior. Estas aplican una transformación lineal a los datos y luego pasan el resultado a través de la función ReLU (Rectified Linear Unit). Esta función es común en redes neuronales profundas, ya que introduce no linealidades sin que aparezcan problemas de gradiente que desaparece como otras funciones de activación. La cuarta y última capa es la capa de salida que proporciona los valores Q de cada acción a tomar para cada estado. Esta está formada solo por cuatro neuronas que son el espacio de acciones mencionados. [8]

3.4.2 Optimizador

El optimizador es un algoritmo que se encarga de ir ajustando los pesos de la red neuronal para así ir minimizando la función de pérdida. Se utiliza el optimizador por defecto de StableBaselines3, que es el optimizador Adam. Este optimizador se basa en otros dos, AdaGrad y RMSProp, y sirve para ajustar la tasa de aprendizaje de la red neuronal en función de la frecuencia y la magnitud de los gradientes que se van calculando durante el entrenamiento. Esto permite calcular la tasa de aprendizaje de cada parámetro de la red, permitiendo una optimización más estable y eficiente. Los hiperparámetros de está son los mismo que los utilizados en la arquitectura de DQN vista en el apartado anterior. [9]

3.5 Metodología de Análisis

El problema de decisión de Markov como ya se ha comentado en este capítulo se puede resolver con ambos agentes. Para poder comparar estas dos técnicas se va a analizar la evolución durante del entrenamiento de la recompensa media durante los pasos de los mismos. A su vez también se analizará el porcentaje de éxito una vez terminado el entrenamiento teniendo como límite de éxito una recompensa de 200, como se comentó en la definición del entorno.

Este análisis se ha realizado utilizando dos cuadernos de jupyter utilizando un contenedor que tiene las siguientes librerías mencionadas en el Anexo A. A continuación, se muestran los códigos utilizados para entrenar y analizar tanto el entrenamiento y el éxito de cada agente.

3.5.1 Metodología para entrenar los agentes

El funcionamiento de los códigos utilizados para entrenar a los agente a partir de lo mencionado en los apartados 3.3. Dichos códigos se encargan de entrenar a cada agente utilizando el algoritmo DQN o PPO en el entorno LunarLander-v2 de la librería Gymnasium. Se utiliza Weights & Biases para guardar el progreso del entrenamiento, parámetros clave y guardar los resultados. Los códigos para el entrenamiento de ambos agentes se encuentran en el Anexo A [10] [11]

3.5.2 Metodología para analizar el éxito de los agentes

El código desarrollado se encarga de comparar el desempeño de ambos agentes ya entrenados, siendo esto PPO y DQN, en el entorno LunarLander-v2 de Gymnasium. Se evalúa a ambos modelos en 100 episodios y se calcula el porcentaje de aterrizajes exitosos, siendo el éxito 200 como se ha definido anteriormente en este capítulo. El código utilizado para analizar el éxito de los agentes se encuentra en el Anexo A .

3.5.3 Descripción de los estudios a realizar

Para evaluar la eficiencia del entrenamiento de los distintos agentes se va a analizar la capacidad de aprendizaje, eficiencia y capacidad de converger para el entorno de gym LunarLander-v2. Para ver estas diferencias se van a realizar los siguientes análisis:

- *Recompensa media por episodio:* Con este análisis podemos comparar la calidad del aprendizaje en cuanto a los niveles de recompensa que llega cada agente. Se compara la recompensa media acumulada frente al número de episodios.
- *Longitud media de episodios:* Es el análisis para poder ver la capacidad que tiene el agente para resolver la tarea de forma eficiente por cada episodio.
- *Pasos globales:* Con este análisis se puede ver el tiempo total que necesita cada agente para terminar su entrenamiento y las diferencias entre los mismos.
- *Porcentaje de aterrizajes exitosos:* Una vez ya se tienen los modelos entrenados es necesario evaluar la eficacia de cada una a la hora de resolver el problema en cuestión. Para esto se compara en una muestra de 100 repeticiones de cada agente ya entrenado viendo su tasa de éxitos.

3.5.4 Diagrama de Flujo de la Metodología.

El diagrama de flujos de este proyecto para realizar ambos análisis es el mismo para ambos agentes. Esta metodología es la mostrado en la siguiente figura.

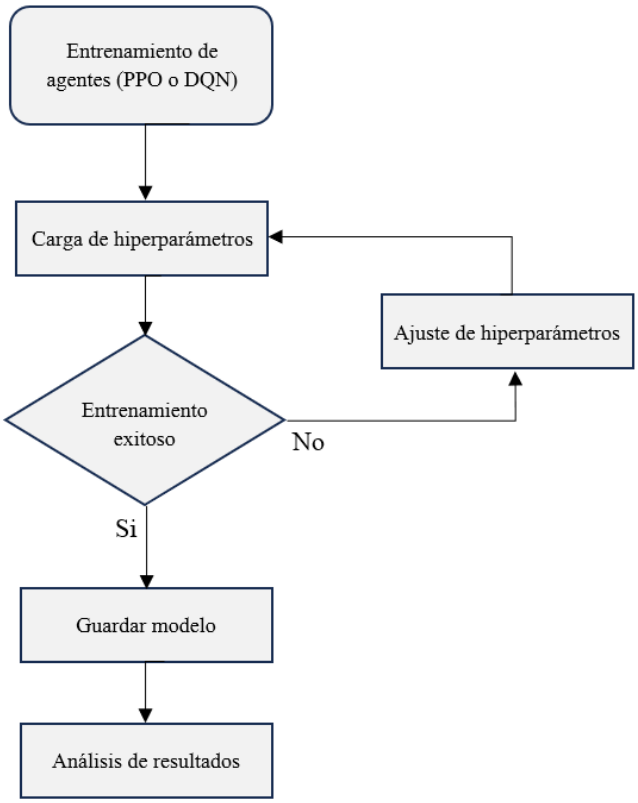


Figura 14. Diagrama de flujo de la metodología de los análisis.

Capítulo 4 Resultados

En este capítulo se realiza el análisis de prestaciones durante el entrenamiento y el porcentaje de éxitos de ambos agentes una vez ya han acabado el entrenamiento. Se compararán los resultados entre dichos agentes con el fin de evaluar su eficiencia a la hora de resolver nuestro problema de Markov. Se compararán los resultados entre dichos agentes con el fin de evaluar su eficiencia a la hora de resolver nuestro problema de Markov.

4.1.1 Análisis recompensa media por episodio

Para poder realizar los análisis de comparación con los agentes, PPO y DQN, se van a entrenar con los hiperparámetros y estructuras comentadas en el Capítulo 3 . Estas estructuras son las mostradas en las siguiente figuras.

```
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 500000,
    "env_name": "LunarLander-v2",
    "learning_starts": 0,
    "batch_size": 128,
    "buffer_size": 100000,
    "learning_rate": 7e-4,
    "target_update_interval": 250,
    "train_freq": 1,
    "gradient_steps": 4,
    "exploration_fraction": 0.08,
    "exploration_final_eps": 0.05,
    "policy_kwargs": dict(net_arch=[256, 256])
}
```

Figura 15. Arquitectura del modelo DQN

```
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 2500000,
    "env_name": "LunarLander-v2",
    "learning_rate": 0.001,
    "n_steps": 1024,
    "batch_size": 64,
    "n_epochs": 5,
    "gamma": 0.999,
    "gae_lambda": 0.98,
    "ent_coef": 0.01,
    "vf_coef": 0.5,
    "max_grad_norm": 0.5
}
```

Figura 16. Arquitectura del modelo PPO

Con nuestros agentes ya definidos y a partir de sus códigos han sido entrenados obteniendo la siguiente gráfica con los resultados.

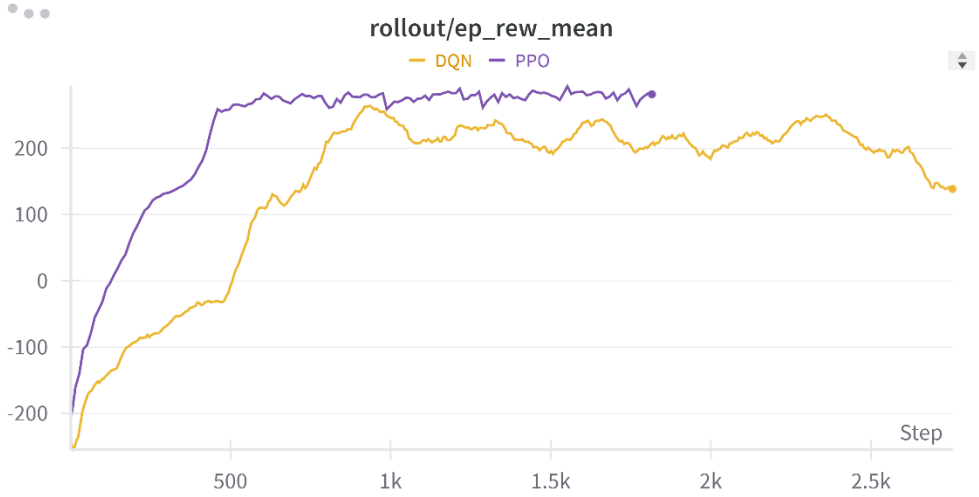


Figura 17. Gráfica recompensa media por episodio de los agentes

Como se puede observar a partir de la gráfica anterior para el modelo de DQN el proceso es notablemente más lento durante el inicio del entrenamiento. Las recompensas son bajas inicialmente para ambos modelos, esto se debe a la política definida de $\varepsilon - greedy$ para el modelo de DQN y que los agentes se centran en ir explorando el entorno para ir conociendo que acciones son mejores. Cabe destacar que para algoritmos basados en *Q-Learning*, como el algoritmo de *Deep Q-Learning*, al inicio tienen sus valores de la red neuronal iniciados por defecto según el método de Xavier Glorot. Esta es una manera muy común para inicializar los pesos con funciones de activación ReLU, como es nuestro caso. Debido a esto ambos agentes necesitan explorar gran parte del entorno para poder empezar a construir su estrategia.

Para el modelo de DQN se puede observar que en torno a 500 pasos ya empieza a tener recompensas positivas y en torno a 800 pasos ya llega a una estrategia óptima situada entorno a una recompensa de 200, que es el éxito considerado en este entorno. A partir de este punto el modelo se estanca con oscilaciones bastante más grandes intentando perfeccionar su estrategia, pero no se estabiliza de la mejor manera.

En contraste, el modelo de PPO es claramente superior que el modelo de DQN a lo largo del entrenamiento. Se puede observar que la pendiente del aprendizaje de este modelo es mayor llegando a recompensas positivas en unos 100 pasos. Además, se estabiliza en una estrategia óptima con una recompensa de unos 265 y con muchas menos oscilaciones. Con estos hiperparámetros se puede observar que el modelo de PPO es bastante más rápido y eficiente a la hora de resolver este entorno, ya que tiene un entrenamiento mucho más estable y le permite actualizarse y adaptarse ante las complejidades que se pueden plantear en el entorno.

Estas diferencias entre ambos modelos se deben a las características intrínsecas de cada uno de ellos como se comentaron en el Capítulo 2 , ya que el algoritmo de PPO es una técnica que utiliza en enfoque de optimización directo de la política y el algoritmo de DQN actualiza su política a partir de los valores Q estimados. Además, como se ha comentado la capacidad de mantenerse estable y tener un equilibrio de exploración y explotación es superior para el modelo

de PPO debido a los límites que se imponen para cumplir con sus actualizaciones, así obteniendo un rendimiento general superior.

4.1.2 Análisis longitud media de episodios

A continuación, se muestra la gráfica de la longitud media de episodios durante el entrenamiento de los agentes para poder analizar como los agentes se van desenvolviendo a la hora de resolver los problemas del entorno.

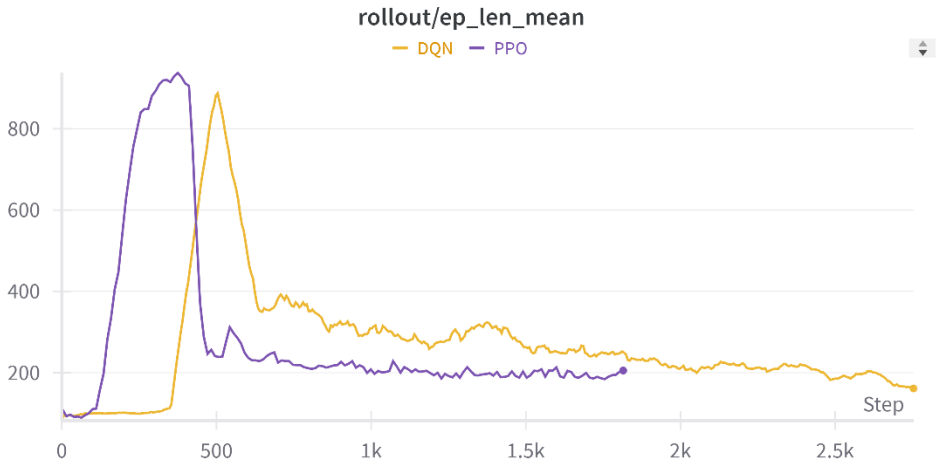


Figura 18. Longitud media de los episodios durante el entrenamiento

Como se puede observar en la gráfica ambos modelos inicialmente se comportan de forma similar y esto se debe a que inicialmente los agentes no tienen una estrategia definida y necesitan explorar el entorno para poder desarrollar una. Sin embargo, al cabo de unos pasos la longitud de los episodios para ambos empieza a decaer a medida que el agente ya va definiendo su estrategia. Cabe destacar que, como se refleja en la gráfica de la Figura 17, el modelo de PPO es capaz de ir reduciendo la longitud de los episodios antes debido a que es capaz de desarrollar una estrategia en menos pasos.

Por otro lado, a pesar de que el modelo DQN también consigue reducir la duración de sus episodios lo hace en un mayor tiempo y con una menor reducción de la duración. Esto se debe a que la estrategia desarrollada por este modelo no es igual de óptima que la del PPO, tardando más en adaptarse a las distintas situaciones. También se puede ver como el modelo de DQN es más inestable en sus soluciones ya que la duración de sus episodios va variando debido a que llega a su estrategia óptima más tarde que el modelo de PPO.

A partir de este análisis se puede observar una clara ventaja para el agente que utiliza PPO ya que tiene un proceso más eficiente. Además, esto lo consigue sin tener que reducir la calidad obtenida en su solución al problema como se ve en la Figura 17. Por otro lado, el modelo de DQN necesita de más tiempo para poder llegar a una solución peor, pero aun así aceptable dentro del entorno debido al funcionamiento de su algoritmo de aprendizaje.

4.1.3 Análisis de pasos globales

Otro punto importante que analizar es el tiempo total que tardan los agentes en realizar el entrenamiento completo. Para ello se ha analizado el número de pasos globales frente al tiempo en llevarlos a cabo. En la siguiente gráfica se muestra dicha evolución.

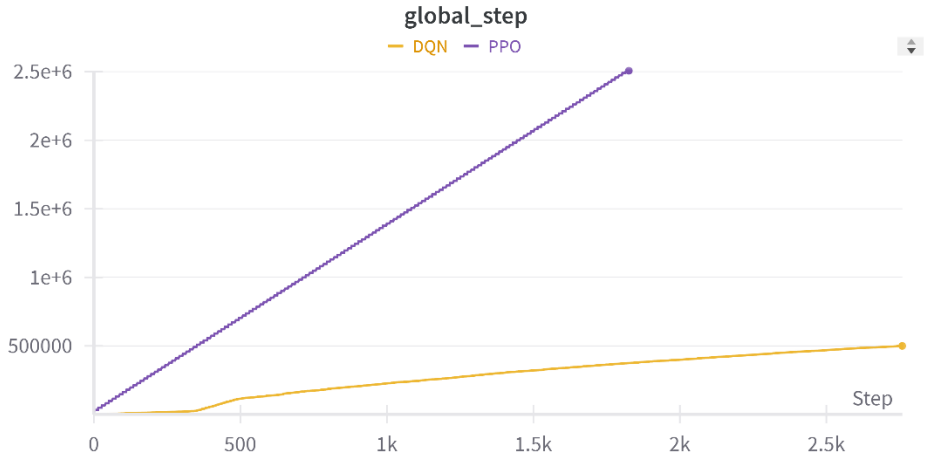


Figura 19. Pasos globales vs pasos totales durante el entrenamiento

Como se puede observar en la gráfica de la Figura 19 existe una gran diferencia entre ambos algoritmos. El algoritmo de DQN solo llega a realizar 500000 pasos, mientras que el modelo de PPO alcanza 2.5 millones. Esta gran diferencia permite a PPO tener muchos mas datos y poder así optimizar mejor su estrategia. Además, como se ha comentado en la Gráfica 18, el algoritmo de DQN tarda mucho más en realizar su entrenamiento ya que la longitud de sus episodios es más larga.

El algoritmo de PPO al poder obtener un mayor número de datos en menos tiempo de entrenamiento, le permite tener una exploración más amplia lo que con lleva una mejor generalización de la política. Esto se debe a su capacidad de estabilizarse más rápidamente una vez ya tiene definida la política.

4.1.4 Porcentaje de aterrizajes exitosos por agente

Una vez analizados ambos agentes durante sus respectivos entrenamientos se pasa a analizar la efectividad de cada uno en realizar la tarea. En la tabla del Anexo A se muestran 100 episodios realizados por cada agente con su recompensa. Cabe destacar que, como se mencionó en el Capítulo 3 , el éxito del agente se consigue a partir de 200 puntos de recompensa. En la siguiente tabla se muestra un resumen de los resultados obtenidos para ambos agentes.

Métrica	DQN	PPO
Media	271,37	281,90
Mediana	272,03	282,18
Desviación estándar	21,24	20,32
Valor mínimo	224,54	232,55
Valor máximo	310,86	321,10
Rango	86,32	88,55

Tabla 3. Métricas porcentaje de éxito de los agentes

Como se puede observar tanto en la Tabla 3 como en la Tabla 4 del Anexo A se observa que ambos agentes resuelven el problema de manera exitosa ya que ambos tienen una tasa del 100 % de éxito. Cabe destacar que, a pesar de que ambos agentes resuelven el entorno de manera exitosa, la técnica de PPO tiene unos valores un poco más altos. Estas diferencias se deben a lo comentado en este capítulo ya que el algoritmo de DQN actúa mejor en entorno más simples y estables. Por otro lado, la técnica de PPO es mucho más robusta para cualquier tipo de entornos ya que por su funcionamiento permite definir una política de manera muy estable.

Capítulo 5 Conclusiones

En los próximos años se espera que la inteligencia artificial experimente un incremento en su uso en gran variedad de sectores, ya que tiene multitud de aplicaciones y es una tecnología muy novedosa. Todavía hace falta más investigación y desarrollo de estas técnicas, como ya se está realizando en una gran cantidad de países y empresas. Por ello, es necesario analizar estas técnicas y saber cuáles son más eficientes y prometedoras para poder emplearlas de una manera más adecuada.

Por ende, en este proyecto se ha desarrollado un análisis comparando dos técnicas de las principales formas para resolver problemas de toma optimizada de decisiones en secuencia: los métodos de aprendizaje por refuerzo basados en valor y los métodos basados en aproximación directa de la política. En nuestro caso se ha comparado uno de los métodos más prometedores, como es la técnica de PPO, con otra técnica bastante utilizada que es el algoritmo de DQN para entornos sencillos. Para facilitar esta comparación y su posterior análisis el entorno escogido es bastante sencillo. Finalmente, después de los respectivos entrenamientos de los agentes se han obtenido los resultados de cada uno de ellos durante el entrenamiento y posterior al mismo para realizar el análisis mencionado.

5.1 Agentes

En esta sección se discuten los agentes desarrollados para el entorno con el problema de decisión de Markov a resolver.

El modelo de DQN es una técnica basada en valor que empieza a ser más eficiente que las primeras técnicas originales y de ahí su mayor utilidad, ya que gracias a la red neuronal puede ir estimando los valores y objetivos facilitando el cálculo para entornos más complejos. A la hora de resolver el problema de decisión logra un aprendizaje y una política aceptable y lo suficientemente robusta. Por otro lado, demuestra ser bastante ruidosa en su entrenamiento lo que hace que su política final sea menos robusta en cuanto a ese aspecto.

El modelo de PPO es una técnica principalmente basada en aproximación directa de la política, que es de las más eficientes en la actualidad. Esto se debe a la estabilidad y robustez presentada durante el entrenamiento debido a las restricciones para los cambios en la política. A su vez presenta una mejor política final que DQN ya que es capaz de tener una recompensa mayor por lo general. Debido a estas razones PPO es de las técnicas más prometedoras en la actualidad.

5.2 Conclusiones finales

Las principales conclusiones que se han extraído del desarrollo de este proyecto son las siguientes:

- Una metodología rigurosa para comparar dos técnicas, que es aplicable a la comparación de otras distintas a las empleadas en este proyecto, para analizar la eficiencia a nivel de entrenamiento y a la hora de resolver el problema en cuestión en profundidad.

Permitiendo así ver las fortalezas y debilidades de cada técnica para su futuro uso en otros entornos.

- Un análisis completo del estado actual de las principales técnicas a nivel de usuario para comprender y entender esta tecnología. Esto facilita la extrapolación de estos conocimientos para poder aplicar estas técnicas en diversos entornos y problemas de decisión.
- A partir del análisis comparativo realizado en este proyecto se puede concluir que la técnica de PPO es más eficiente tanto en la exploración como en la explotación del entorno en cuestión. Por otro lado, el algoritmo de DQN presenta muchas variaciones que afectan al desarrollo de su política, aunque consigue desarrollar una política que funciona.
- El algoritmo de PPO demuestra tener una solución más estable y robusta que la técnica de DQN en menor tiempo, lo que es una gran ventaja para la técnica de PPO.

5.3 Trabajos futuros

En la temática del aprendizaje por refuerzo se pueden seguir diferentes vías; además se trata de una herramienta en una etapa temprana en su desarrollo e implementación en las diferentes industrias.

En este proyecto se ha realizado un breve análisis de varias técnicas y la capacidad que tienen para resolver diversos problemas. Alguno de los aspectos que deberían continuar esta investigación son:

- Extender el análisis a entornos más complejos o con espacios continuos.
- Explorar entornos de multiagentes. Es una de las aplicaciones más interesantes de esta tecnología.
- Realizar estudios a nivel de recursos de cada técnica utilizada.

Anexo A

En esta sección se encuentran datos de los agentes actuando sobre el entorno ya finalizado su entrenamiento. En la siguiente tabla se muestran los datos obtenidos de la recompensa acumulada para cada agente.

Episodio	DQN	PPO
1	275,06	306,83
2	255,75	284,71
3	256,06	300,39
4	299,26	276,28
5	261,53	310,11
6	283,08	253,92
7	241,33	274,34
8	266,12	276,89
9	263,34	267,17
10	251,33	313,12
11	278,28	291,65
12	287,33	245,50
13	269,36	292,26
14	249,52	296,51
15	246,99	294,10
16	267,97	269,29
17	271,56	244,24
18	281,09	274,50
19	292,93	312,13
20	305,45	287,77
21	259,22	319,70
22	292,76	294,12
23	291,09	289,37
24	264,90	279,04
25	264,23	297,55
26	257,08	261,77
27	275,64	310,03
28	292,04	270,01
29	267,75	266,79
30	294,92	282,05
31	297,90	274,12
32	238,78	302,14
33	249,25	291,02
34	257,45	270,67
35	247,03	288,58
36	277,95	280,47
37	286,75	264,98
38	280,80	264,27

39	254,49	321,10
40	286,52	320,01
41	260,62	283,19
42	225,15	307,72
43	262,17	297,25
44	285,40	267,78
45	256,56	289,74
46	266,23	246,28
47	296,87	271,29
48	300,60	301,84
49	300,44	289,83
50	265,64	235,54
51	287,90	286,70
52	266,20	236,00
53	226,67	317,09
54	303,31	260,62
55	281,60	247,36
56	263,32	302,80
57	298,92	308,52
58	272,53	281,67
59	254,63	297,95
60	245,14	292,64
61	272,49	297,94
62	309,32	254,96
63	259,55	279,97
64	229,52	273,76
65	244,59	286,05
66	310,86	279,36
67	285,25	288,09
68	278,34	286,70
69	300,28	309,42
70	248,72	281,57
71	263,62	279,92
72	304,14	297,19
73	285,21	232,55
74	303,64	280,11
75	309,07	307,18
76	287,59	278,86
77	228,95	276,91
78	253,37	306,42
79	283,87	301,44
80	269,85	269,33
81	240,16	308,10
82	224,54	289,50
83	267,24	304,74
84	295,68	267,15

85	256,43	279,33
86	291,98	268,35
87	273,88	266,52
88	297,67	303,38
89	274,41	264,09
90	295,77	278,97
91	256,98	282,31
92	254,61	245,82
93	273,43	271,30
94	262,72	308,44
95	282,92	269,36
96	245,72	245,50
97	254,52	262,97
98	299,13	289,14
99	289,82	288,98
100	295,19	261,15
Éxitos	100%	100%

Tabla 4. Recompensa acumulada de los agentes entrenados para 100 episodios

A continuación, se muestra el código utilizado para entrenar a nuestro agente con la técnica de DQN.

```
# Import necessary libraries
import gymnasium as gym

from stable_baselines3 import DQN
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.vec_env import VecVideoRecorder
from stable_baselines3.common.env_util import make_vec_env

import wandb
from wandb.integration.sb3 import WandbCallback

 # Step 1: Login to Weights & Biases (You only need to do this once)
wandb.login()

# Step 2: Initialize W&B project and configure settings
# Initialize a W&B run
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 500000,
    "env_name": "LunarLander-v2",
    "learning_starts": 0,
    "batch_size": 128,
    "buffer_size": 100000,
    "learning_rate": 7e-4,
    "target_update_interval": 250,
    "train_freq": 1,
    "gradient_steps": 4,
    "exploration_fraction": 0.08,
```

```
    "exploration_final_eps": 0.05,
    "policy_kwargs": dict(net_arch=[256, 256])
}

# Start a W&B run and track model configuration
wandb.init(
    project="lunar_lander_rl",  # Project name in W&B
    config=config,              # Configuration to track in the W&B UI
    sync_tensorboard=True        # Syncing with TensorBoard logs
)

# Step 3: Create the environment
env = gym.make(config["env_name"])
env = Monitor(env) # Monitor is used to keep track of stats like
rewards
env = make_vec_env("LunarLander-v2", n_envs=8) # Create a vectorized
environment - a method for stacking multiple independent environments
into a single environment
env = VecVideoRecorder(env, f"videos/{wandb.run.id}",
record_video_trigger=lambda x: x % 2000 == 0, video_length=200) #
Record videos

# Step 4: Create the RL model (PPO in this case)
model = DQN(
    config["policy_type"],
    env,
    verbose=1,
    learning_starts=config["learning_starts"],
    batch_size=config["batch_size"],
    buffer_size=config["buffer_size"],
    learning_rate=config["learning_rate"],
    target_update_interval=config["target_update_interval"],
    train_freq=config["train_freq"],
    gradient_steps=config["gradient_steps"],
    exploration_fraction=config["exploration_fraction"],
    exploration_final_eps=config["exploration_final_eps"],
    policy_kwargs=config["policy_kwargs"],
    tensorboard_log=f"runs/{wandb.run.id}"
)

# Step 5: Set up W&B callback for logging
wandb_callback = WandbCallback(
    gradient_save_freq=100, # Frequency to save gradient info
    model_save_path=f"models/{wandb.run.id}", # Where to save the
model
    model_save_freq=100, # Frequency to save the model
    verbose=1, # How much info you want during training
    log="all", # Log all info
)

# Step 6: Train the model and log the progress to W&B
model.learn(total_timesteps=config["total_timesteps"],
callback=wandb_callback)

# Step 7: End the W&B run after training

wandb.finish()
```

A continuación, se muestra el código utilizado para entrenar a nuestro agente con la técnica de PPO.

```
# Import necessary libraries
import gymnasium as gym

from stable_baselines3 import PPO
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.env_util import make_vec_env

import wandb
from wandb.integration.sb3 import WandbCallback

# Step 1: Login to Weights & Biases (You only need to do this once)
wandb.login()

# Step 2: Initialize W&B project and configure settings
# Initialize a W&B run
config = {
    "policy_type": "MlpPolicy",
    "total_timesteps": 2500000,
    "env_name": "LunarLander-v2",
    "learning_rate": 0.001,
    "n_steps": 1024,
    "batch_size": 64,
    "n_epochs": 5,
    "gamma": 0.999,
    "gae_lambda": 0.98,
    "ent_coef": 0.01,
    "vf_coef": 0.5,
    "max_grad_norm": 0.5
}

# Start a W&B run and track model configuration
wandb.init(
    project="lunar_lander_rl",  # Project name in W&B
    config=config,              # Configuration to track in the W&B UI
    sync_tensorboard=True        # Syncing with TensorBoard logs
)

# Step 3: Create the environment
env = gym.make(config["env_name"])
env = Monitor(env) # Monitor is used to keep track of stats like
rewards
env = make_vec_env("LunarLander-v2", n_envs=16) # Create a vectorized
environment - a method for stacking multiple independent environments
into a single environment

# Step 4: Create the RL model (PPO in this case)
model = PPO(
    config["policy_type"],
    env,
    verbose=1,
    learning_rate=config["learning_rate"],
    n_steps=config["n_steps"],
    batch_size=config["batch_size"],
    n_epochs=config["n_epochs"],
    gamma=config["gamma"],
```

```
    gae_lambda=config["gae_lambda"],
    ent_coef=config["ent_coef"],
    vf_coef=config["vf_coef"],
    max_grad_norm=config["max_grad_norm"],
    tensorboard_log=f"runs/{wandb.run.id}"
)

# Step 5: Set up W&B callback for logging
wandb_callback = WandbCallback(
    gradient_save_freq=100, # Frequency to save gradient info
    model_save_path=f"models/{wandb.run.id}", # Where to save the
model
    model_save_freq=100, # Frequency to save the model
    verbose=1, # How much info you want during training
    log="all", # Log all info
)

# Step 6: Train the model and log the progress to W&B
model.learn(total_timesteps=config["total_timesteps"],
callback=wandb_callback)

# Step 7: End the W&B run after training
wandb.finish()
```

A continuación, se muestra el código utilizado para ver el porcentaje de éxitos en 100 episodio por agente.

```
# Import the necessary libraries
import gymnasium as gym

from stable_baselines3 import PPO, DQN

# Step 1: Load your models
model_PPO =
PPO.load('C:/Users/manug/OneDrive/Escritorio/ICAI/AgenteCICLAB/Atari_G
ames-Deep_Reinforcement_Learning/Notebooks/LunarLander-v2/PPO-
wandb/models/em62l5a4/model.zip')
model_DQN =
DQN.load('C:/Users/manug/OneDrive/Escritorio/ICAI/AgenteCICLAB/Atari_G
ames-Deep_Reinforcement_Learning/Notebooks/LunarLander-v2/DQN-
wandb/models/3jxsmxvm/model.zip')

# Step 2: Create the environment
env = gym.make('LunarLander-v2')

# Step 3: Run the agent for 100 episodes (PPO model)
num_episodes = 100
successful_threshold = 200
successful_landings_ppo = 0
episode_rewards = []

# PPO model
for episode in range(num_episodes):
    obs_ppo, _ = env.reset()
    total_reward_ppo = 0
    done_ppo = False
    while not done_ppo:
```

```
        action_ppo = model_PPO.predict(obs_ppo, deterministic=True)[0]
        step_result_ppo = env.step(action_ppo)
        if len(step_result_ppo) == 5:
            obs_ppo, reward_ppo, done_ppo, info_ppo, _ =
step_result_ppo
        else:
            obs_ppo, reward_ppo, done_ppo, info_ppo = step_result_ppo
        total_reward_ppo += reward_ppo

    # Record the reward for the episode
    episode_rewards.append(total_reward_ppo)

    # Check if the episode is a success based on the reward threshold
    if total_reward_ppo >= successful_threshold:
        successful_landings_ppo += 1

# Display the rewards for each episode
for i, reward in enumerate(episode_rewards, 1):
    print(f"Episode {i}: Reward = {reward}")

# Step 4: Run the agent for 100 episodes (DQN model)
num_episodes = 100
successful_threshold = 200
max_steps_per_episode = 1000
successful_landings_dqn = 0
episode_rewards_dqn = []

# DQN model
for episode in range(num_episodes):
    obs_dqn, _ = env.reset()
    total_reward_dqn = 0
    done_dqn = False
    step_count = 0
    while not done_dqn and step_count < max_steps_per_episode:
        action_dqn = model_DQN.predict(obs_dqn, deterministic=True)[0]
        step_result_dqn = env.step(action_dqn)

        if len(step_result_dqn) == 5:
            obs_dqn, reward_dqn, done_dqn, info_dqn, _ =
step_result_dqn
        else:
            obs_dqn, reward_dqn, done_dqn, info_dqn = step_result_dqn

        total_reward_dqn += reward_dqn
        step_count += 1

    # Record the reward for the episode
    episode_rewards_dqn.append(total_reward_dqn)

    # Check if the episode is a success based on the reward threshold
    if total_reward_dqn >= successful_threshold:
        successful_landings_dqn += 1

# Display the rewards for each episode
for i, reward in enumerate(episode_rewards_dqn, 1):
    print(f"Episode {i}: Reward = {reward}")

# If an episode takes too long, it will automatically stop after
max_steps_per_episode.
```

```
# Step 5: Calculate the percentage of successful landings for each
model
percentage_success_ppo = (successful_landings_ppo / num_episodes) *
100
print(f"Percentage of successful landings (PPO):
{percentage_success_ppo}%")

percentage_success_dqn = (successful_landings_dqn / num_episodes) *
100
print(f"Percentage of successful landings (DQN):
{percentage_success_dqn}%")
```

Referencias

[1] V. Mnih, et al., "Human-level control through deep reinforcement learning," *Nature,* vol. 518, p. 529–533, 2015.

[2] Hugging Face, "Hugging Face," [Online]. Available: https://huggingface.co/learn/deep-rl-course/unit0/introduction.. [Accessed 15 October 2024].

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," Cornell University, Ithaca, New York, 2013.

[4] D. Bick, "Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization," University of Groningen, Groningen, The Netherlands, 2021.

[5] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2ª ed. ed., Cambridge, MA.: MIT Press, 2018.

[6] Farama Foundation, "Gymnasium Documentation," [Online]. Available: https://gymnasium.farama.org/. [Accessed 11 November 2024].

[7] Farama Foundation, "Gym Library," [Online]. Available: https://www.gymlibrary.dev/environments/box2d/lunar_lander/. [Accessed 11 November 2024].

[8] Stable Baselines3, "Stable Baselines3 Documentation," [Online]. Available: https://stable-baselines3.readthedocs.io/en/master/guide/install.html. [Accessed 8 November 2024].

[9] EITCA Academy, "Deep Learning con TensorFlow," 2024. [Online]. Available: https://es.eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow/neural-network-model/examination-review-neural-network-model/how-does-the-adam-optimizer-optimize-the-neural-network-model/. [Accessed 4 November 2024].

[10] TensorFlow, "API Documentation," [Online]. Available: https://www.tensorflow.org/api_docs. [Accessed 9 November 2024].

[11] Weights & Biases, "WandB Documentation," [Online]. Available: https://docs.wandb.ai/. [Accessed 8 November 2024].

[12] M. González López, "Proyecto RL de Lunar Lander.," [Online]. Available: https://wandb.ai/TFG_RL/lunar_lander_rl?nw=nwusergolobama. [Accessed 26 October 2024].