UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Posicionamiento de estaciones base en tiempo real basado en aprendizaje de refuerzo profundo para futuras redes 6G.

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Rico Ibáñez, Mario

Tutor/a: Naranjo Ornedo, Valeriana

Cotutor/a: López Pérez, David

CURSO ACADÉMICO: 2023/2024

Resumen

Las comunicaciones móviles son esenciales en la sociedad actual, definiendo cómo nos comunicamos y relacionamos, e influyendo en cómo vivimos. La optimización de este bien tan valioso es esencial para proporcionar la mejor experiencia posible de forma sostenible. Para mejorar las prestaciones de la red y su eficiencia energética, se espera que las futuras generaciones de comunicaciones incorporen estaciones base capaces de reposicionarse de forma sencilla, sin intervención humana, mediante vehículos terrestres o aeronaves no tripuladas, como drones.

En esta tesis, proponemos una solución inteligente que permite a dichas estaciones base móviles adaptarse y aprender los patrones de movimiento de los usuarios, y encontrar su posición óptima en tiempo real. Para cuantificar la experiencia del usuario, utilizamos la tasa de datos media de los usuarios. Para guiar dichas decisiones y aprender políticas óptimas, hemos utilizado técnicas avanzadas de aprendizaje, en más detalle, aprendizaje reforzado profundo (DRL, del inglés deep reinforcement learning).

Como primer punto de trabajo, es importante enfatizar el uso, desarrollo y optimización de un entorno de simulación avanzado de redes de comunicaciones móviles 4G/5G/6G, llamado Giulia, basado en Python. Dicha optimización implicó la transformación de un número sustancial de operaciones a métodos más eficientes, como el uso de GPU a través de la biblioteca PyTorch. Como mayor contribución de este trabajo, implementamos técnicas avanzadas de DRL, adaptándolas a nuestro problema específico, que combinan la potencia de las redes neuronales profundas (DNN, del inglés deep neural networks) con la optimización del aprendizaje reforzado. A través del trabajo de esta tesis hemos sentado unas bases sólidas y conocimiento acerca de la integración de DRL con entornos de simulación complejos para la toma de decisiones en tiempo real. Hemos demostrado que el uso de estadísticas acerca de la potencia de señal recibida, el ángulo de llegada y la tasa de datos de los usuarios es información suficiente para aprender patrones de movimiento de grupos de usuarios, y optimizar en tiempo real el movimiento de estaciones base móviles que pretenden maximizar la experiencia de dichos usuarios. Los resultados de la tesis indican que DRL puede ser utilizado en las siguientes generaciones de redes de comunicaciones que pueden ser controladas por agentes autónomos en tiempo real.

Resum

Les comunicacions mòbils són essencials en la societat actual, definint com ens comuniquem i ens relacionem, i influint en com vivim. L'optimització d'aquest bé tan valuós és essencial per a proporcionar la millor experiència possible de manera sostenible. Per a millorar les prestacions de la xarxa i la seua eficiència energètica, s'espera que les futures generacions de comunicacions incorporen estacions base capaces de reposicionar-se de forma senzilla, sense intervenció humana, mitjançant vehicles terrestres o aeronaus no tripulades, com drons.

En aquesta tesi, proposem una solució intel·ligent que permet a aquestes estacions base mòbils adaptar-se i aprendre els patrons de moviment dels usuaris, i trobar la seua posició òptima en temps real. Per a quantificar l'experiència de l'usuari, utilitzem la taxa de dades mitjana dels usuaris. Per a guiar aquestes decisions i aprendre polítiques òptimes, hem utilitzat tècniques avançades d'aprenentatge, en més detall, aprenentatge reforçat profund (DRL, de l'anglés deep reinforcement learning).

Com a primer punt de treball, és important enfatitzar l'ús, desenvolupament i optimització d'un entorn de simulació avançat de xarxes de comunicacions mòbils 4G/5G/6G, anomenat Giulia, basat en Python. Aquesta optimització va implicar la transformació d'un nombre substancial d'operacions a mètodes més eficients, com l'ús de GPU a través de la biblioteca PyTorch. Com a major contribució d'aquest treball, implementem tècniques avançades de DRL, adaptant-les al nostre problema específic, que combinen la potència de les xarxes neuronals profundes (DNN, de l'anglés deep neural networks) amb l'optimització de l'aprenentatge reforçat. A través del treball d'aquesta tesi hem assentat unes bases sòlides i coneixement sobre la integració de DRL amb entorns de simulació complexos per a la presa de decisions en temps real. Hem demostrat que l'ús d'estadístiques sobre la potència de senyal rebuda, l'angle d'arribada i la taxa de dades dels usuaris és informació suficient per a aprendre patrons de moviment de grups d'usuaris, i optimitzar en temps real el moviment d'estacions base mòbils que pretenen maximitzar l'experiència d'aquests usuaris. Els resultats de la tesi indiquen que DRL pot ser utilitzat en les següents generacions de xarxes de comunicacions que poden ser controlades per agents autònoms en temps real.

Abstract

Mobile communications are essential in today's society, defining how we communicate and interact, and influencing how we live. The optimization of this valuable asset is crucial to provide the best possible experience in a sustainable way. To improve network performance and energy efficiency, it is expected that future generations of communications will incorporate base stations capable of repositioning easily, without human intervention, using ground vehicles or unmanned aerial vehicles (UAVs), such as drones.

In this thesis, we propose an intelligent solution that allows these mobile base stations to adapt and learn user movement patterns, and find their optimal position in real-time. To quantify the user experience, we use the average data rate of users. To guide these decisions and learn optimal policies, we have used advanced learning techniques, specifically, deep reinforcement learning (DRL).

As the first point of work, it is important to emphasize the use, development, and optimization of an advanced simulation environment for 4G/5G/6G mobile communication networks, called Giulia, based on Python. This optimization involved transforming a substantial number of operations into more efficient methods, such as using GPU through the PyTorch library. As the main contribution of this work, we implemented advanced DRL techniques, adapting them to our specific problem, which combine the power of deep neural networks (DNN) with the optimization of reinforcement learning. Through the work of this thesis, we have laid a solid foundation and knowledge about the integration of DRL with complex simulation environments for real-time decision-making. We have demonstrated that the use of statistics on received signal strength, angle of arrival, and user data rate is sufficient information to learn movement patterns of user groups and optimize in real-time the movement of mobile base stations that aim to maximize the experience of these users. The results of the thesis indicate that DRL can be used in the next generations of communication networks that can be controlled by autonomous agents in real-time.

RESUMEN EJECUTIVO

La memoria del TFG del GTIST debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la IT

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	1.
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	2-5, 9-36, 37-45.
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	6-7.
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	45-53, 57-66.
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	S	45-53, 57-66, 70-74.
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	53, 69-74.
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	77-78.

A mis padres y mi abuela,

por mantener siempre en mi esas ganas de mejorar, apoyo incondicional y un cariño único.

A Alejandro, Sergio y Martina,

los mejores hermanos que he podido tener, gracias por hacerme feliz.

A mis amigos,

por ser un pilar fundamental y una gran fuente de diversión.

A mis tutores, Valery y David,

por confiar en mi y apoyarme en todo lo que pueden.

Todo esto es gracias a vosotros.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

3GPP 3rd Generation Partnership Project..

6G Sixth-generation Technology for Wireless Communication.

A3C Asynchronous Advantage Actor Critic..

BS Base Station..

CNN Convolutional Neural Network..

DDPG Deep Deterministic Policy Gradient..

DQN Deep-Q-Network..

DRL Deep Reinforcement Learning..

GAE Generalized Advantage Estimation..

GAN Generative Adversarial Network..

ITU International Telecommunication Union..

MAC Media Access Control..

MDP Markov Decision Process..

ML Machine Learning..

MPC Multipath Component..

PDP Power Delay Profile..

POMDP Partially Observable Markov Decision Process..

PPO Proximal Policy Optimization..

ReLU Rectified Linear Unit..

RL Reinforcement Learning..

RNN Recurrent Neural Network..

SARSA State-Action-Reward-State-Action..

SINR Signal-to-Interference-plus-Noise Ratio..

TDMA Time Division Multiple Access ..

TRPO Trust Region Policy Optimization..

UAV Unmanned Aerial Vehicle..

UES User Equipment..

Parte I

Memoria

Capítulo 1

Introducción

1.1. Motivación y Descripción del Trabajo

Las situaciones de emergencia y accidentes, tales como los desastres naturales, plantean desafíos significativos para las infraestructuras de comunicaciones. Durante estos eventos, las redes de comunicación se vuelven vitales para una variedad de tareas críticas, incluyendo las operaciones de rescate, la coordinación de recursos y la comunicación entre equipos de emergencia y afectados. Sin embargo, las redes de comunicación actuales presentan limitaciones importantes. A menudo, no pueden desplegarse fácilmente en cualquier localización o cualquier momento, lo que dificulta su uso efectivo en situaciones de emergencia.

En este contexto, las tecnologías emergentes de la sexta generación (6G) de comunicaciones móviles ofrecen una solución prometedora. Una de las innovaciones más destacadas es el uso de drones, los cuales permiten el despliegue de redes de comunicación casi en cualquier lugar y en cualquier momento, superando así las limitaciones de las infraestructuras fijas tradicionales. Estos drones pueden portar estaciones base móviles, proporcionando cobertura de red en áreas afectadas por desastres donde la infraestructura tradicional ha sido destruida o es inaccesible.

Sin embargo, la mera disponibilidad de drones no es suficiente para resolver todos los problemas asociados con la comunicación en situaciones de emergencia. Es esencial que estos drones sean capaces de adaptarse de manera dinámica y eficiente a las necesidades cambiantes de los equipamiento de usuario (UE, del inglés *user equipment*) y las condiciones del entorno. Estos drones pueden ser realmente necesarios en operativos móviles, donde el seguimiento de estos equipos de operación es crucial para un correcto funcionamiento de la red. Para lograr esto, se requiere una inteligencia avanzada que permita a los drones tomar decisiones en tiempo real, como se observa en la Figura 1.1.

En este trabajo, proponemos el desarrollo de un agente de aprendizaje reforzado (RL, del inglés *reinforcement learning*) capaz de adaptarse dinámicamente a los movimientos de los UEs y a las condiciones del entorno en tiempo real. Este agente empleará técnicas avanzadas de aprendizaje prrofundo por refuerzo (DRL, del inglés *deep reinforcement learning*) para aprender el movimiento de los UEs y optimizar los patrones de movimiento de los drones. Con esta capacidad de adaptación, los drones podrán no solo posicionarse de manera óptima, sino también ajustarse continuamente a los cambios en el entorno y las necesidades de los UEs, maximizando así la eficiencia de la red y mejorando significativamente la experiencia del UE.

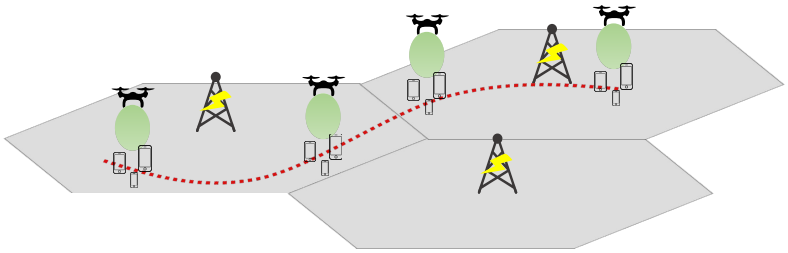


Figura 1.1: Ejemplo situación con UEs en movimiento y red terrestre sin servicio.

1.2. Introducción Aprendizaje Reforzado

El RL [1] es un subcampo del aprendizaje automático (ML, del inglés *machine learning*) que se inspira en el comportamiento humano y animal. En este campo, se desarrollan algoritmos capaces de aprender a seleccionar las mejores acciones a tomar en un entorno determinado, basándose en la maximización de recompensas acumuladas a largo plazo. En el marco típico del aprendizaje reforzado, se identifican los siguientes componentes clave:

- **Entorno**: Es el dominio en el que reside la dinámica del problema que se quiere resolver. El entorno puede ser, tanto simulado, como real, y define las reglas y condiciones bajo las cuales el agente opera. Incluye todos los posibles estados y las transiciones entre ellos.
- **Agente**: Es el componente que interactúa con el entorno, tomando acciones en función de las decisiones del algoritmo de aprendizaje reforzado. Aprendiendo a través de la experiencia qué acciones son las más beneficiosas en los diferentes estados del entorno.

El esquema típico del RL se observa en la Figura 1.2. Se puede apreciar como el agente y el entorno se comunican. El agente toma una acción sobre el entorno y observa el nuevo estado además de la recompensa de haber tomado la acción.

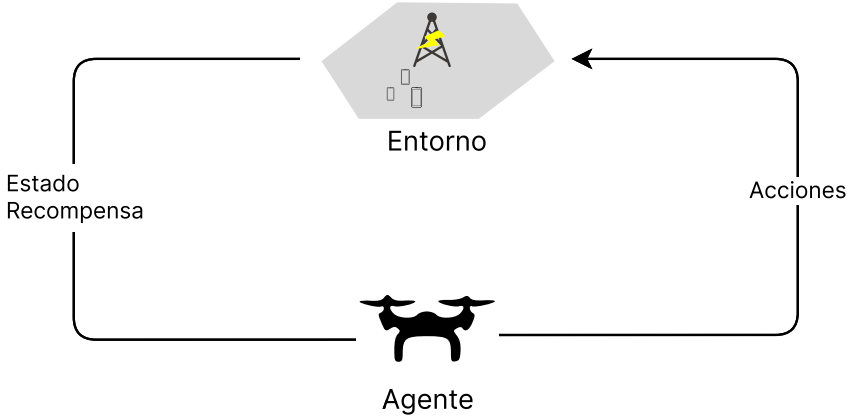


Figura 1.2: Esquema básico entorno RL.

Durante los últimos años, se ha empezado a aplicar el ML a casos de uso de telecomunicaciones relacionados con drones. Sin embargo el uso del RL no ha sido tan común. A continuación, se describen algunos de ellos.

1.2.1. Estado del Arte de La Inteligencia Artificial Aplicada a Drones

En la última década, los Vehículos Aéreos No Tripulados (UAVs) han proporcionado soluciones eficientes y rentables para la recolección de datos y comunicaciones. Su movilidad, flexibilidad y rápida implementación han permitido su uso extensivo en agricultura, misiones de rescate, ciudades inteligentes y sistemas de transporte inteligente. El ML ha demostrado ser capaz de mejorar la automatización y precisión operativa de los UAVs y de muchas de sus aplicaciones.

A medida que los UAVs continúan evolucionando, se han explorado diversas técnicas de ML para mejorar sus capacidades operativas. Estas técnicas no solo han optimizado las funciones básicas de los UAVs, sino que también han abierto nuevas oportunidades para aplicaciones cada dia mas avanzadas y especializadas.

El artículo [2] proporciona una visión general y comprensible de las técnicas de ML utilizadas en las operaciones y comunicaciones de UAVs, identificando las áreas de crecimiento potencial y las brechas de investigación. Se destacan cuatro componentes clave en las operaciones y comunicaciones de UAVs donde el ML puede contribuir significativamente: percepción y extracción de características, interpretación y regeneración de características, planificación de trayectorias y misiones, y control y operación aerodinámica.

En el ámbito de la percepción y extracción de características, el ML permite la interpretación del entorno y la extracción de información relevante. Las técnicas supervisadas, como las redes neuronales convolucionales (CNN, del inglés *Convolutional Neural Network*) y las redes neuronales recurrentes (RNN, del inglés *Recurrent Neural Network*), se utilizan para la clasificación y segmentación de imágenes y la predicción de trayectorias, respectivamente. Por ejemplo, en el trabajo realizado en la Reserva Natural de las Cinco Islas, Australia, [3] se evaluaron tres enfoques para mapear la vegetación usando imágenes recogidas por un UAV. Se utilizó un algoritmo de clasificación de imágenes basado en CNN, LeNet, para detectar matojos plantados de Lomandra longifolia, con una precisión del 85 %.

La interpretación y regeneración de características involucra el modelado y digitalización del entorno basado en los datos recogidos. Los *Autoencoders* y redes generativas adversarias (GAN, del inglés *Generative Adversarial Networks*) se han aplicado para la reducción de dimensionalidad y clasificación de características visuales en imágenes aéreas. En [4], se ha propuesto una técnica para el modelado del canal aire-tierra en frecuencias de onda milimétrica (mmWave) en una red inalámbrica de UAVs. Un enfoque cooperativo basado en GAN distribuido permite a cada UAV aprender la distribución del canal mmWave de manera totalmente distribuida, mejorando la precisión del aprendizaje y la tasa de datos.

El uso de DRL en comunicaciones móviles se centra principalmente en las capas física y la de control de acceso al medio (MAC, del inglés *Media Access Control*) [5]. En la capa física, la mayoría de los trabajos abordan problemas como la optimización del *beamforming*, control de potencia e interferencias y consumo energético [6]. Se ha demostrado cómo la optimización conjunta de *beamforming* y control de potencia e interferencias puede mejorar significativamente el rendimiento de la red en términos de señal a interferencia más ruido (SINR, del inglés *Signal to Interference plus Noise ratio*) y tasa de datos en bandas de frecuencia sub-6 GHz y mmWave. En la capa MAC, los esfuerzos se enfocan en desarrollar *schedulers* que distribuyan eficientemente los recursos de la red. Un trabajo notable presenta el modelo LEASCH; un *scheduler* basado en DRL que ha demostrado ser eficiente independientemente de las diferentes configuraciones de numerología en redes 5G [7].

En cuanto a la capa de red, se han creado agentes que optimicen el enrutamiento de paquetes. Un ejemplo de ello, es un agente de DRL que se adapta automáticamente a las condiciones actuales del tráfico y minimiza el retraso en la red [8]. Otro estudio relevante es el presentado en [9], que explora la aplicación de los algoritmos *Deep-Q-Network* (DQN) y *Deep Deterministic Policy Gradient* (DDPG) en un entorno de comunicaciones marítimas. En este caso, se establece una red de comunicación marítima con UAVs que actúan como estaciones base aéreas. El estudio se centra en la minimización de la latencia del sistema.

La optimización de la posición de UAVs es otro campo significativo de investigación. En [10], se utiliza el algoritmo DDPG para controlar múltiples UAVs. Este enfoque busca maximizar la cobertura de la red, mientras se minimiza el consumo energético. Los UAVs son controlados para adaptarse dinámicamente a las necesidades de cobertura, manteniendo una alta eficiencia energética. Este trabajo destaca por su capacidad de equilibrar cobertura y consumo energético de manera efectiva. Es importante enfatizar que este algoritmo mejora la cobertura tomando como dato la distribución de UEs, y no pretenden aprender y seguir en tiempo real el movimiento de unos UEs determinados dentro del escenario, como es nuestro caso.

En [11], se aborda la optimización de la trayectoria de un dron en un modelo tridimensional, utilizando DDPG. El objetivo es maximizar el número de UEs cubiertos durante el vuelo. Además de considerar la eficiencia energética, este estudio introduce la asignación autónoma de frecuencias, mejorando aún más la eficiencia del sistema. En este caso, el dron tiene conocimiento previo de la posición de los UEs, a diferencia de nuestro caso, donde dichas posiciones no se conocen y deben inferirse a través del ángulo de llegada de las señales.

1.3. Establecimiento del Problema

En esta sección se introduce el modelo de sistema y se definen matemáticamente el problema a resolver en este trabajo.

1.3.1. Modelo del Sistema

Se considera una red celular en la que se comunica tanto en uplink como en downlink de forma no simultánea, utilizando *time division multiple access* (TDMA). En esta red, se encuentran UEs terrestres y drones (UAVs), cuyos conjuntos son $\mathcal{U}$ y $\mathcal{D}$, respectivamente. Los drones portan una estación base (BS, del inglés *Base Station*), y se comportan como tal.

El conjunto, $\mathcal{U}$, está compuesto por U UEs de manera que:

$$\mathcal{U} = \{u_1, \ldots, u_u, \ldots, u_U\}. \tag{1.1}$$

La posición geográfica del UE, u, se determina por:

$$\rho_u^{\mathrm{U}} = \{x_u^{\mathrm{U}}, y_u^{\mathrm{U}}, z_u^{\mathrm{U}}\}, \tag{1.2}$$

y la matriz de posiciones de todos los UEs se define de la siguiente manera:

$$\boldsymbol{\rho}^{\mathrm{U}} = \{\rho_1^{\mathrm{U}}, \ldots, \rho_u^{\mathrm{U}}, \ldots, \rho_U^{\mathrm{U}}\}. \tag{1.3}$$

El conjunto, $\mathcal{D}$, está compuesto por D drones de manera que:

$$\mathcal{D} = \{d_I, \ldots, d_d, \ldots, d_D\}, \tag{1.4}$$

donde la posición del dron, d, se determina por:

$$\rho_d^{\mathrm{D}} = \{x_d^{\mathrm{D}}, y_d^{\mathrm{D}}, z_d^{\mathrm{D}}\}, \tag{1.5}$$

y la matriz de posiciones de todos los UAVs está definida de la siguiente forma:

$$\boldsymbol{\rho}^{\mathrm{D}} = \{\rho_I^{\mathrm{D}}, \ldots, \rho_d^{\mathrm{D}}, \ldots, \rho_D^{\mathrm{D}}\}. \tag{1.6}$$

La red trabaja en una banda de 10 MHz ubicada en 2 GHz. Todos los enlaces radio que forman la red experimentan ganancias por distancia y lognormal *shadowing* . Denotamos la ganancia, $G_{u,d}$, como la ganancia entre el UE, u, y el dron, d. En esta ganancia se tienen en cuenta la ganancia de la antena (3.2.1), G^{a}, la ganancia por distancia (3.2.2), G^{p}, la ganancia *outdoor to indoor* (3.2.4), G^{e}, el *shadowing* (3.2.3), G^{s}, y efectos del multicamino (3.2.5), G^{ff}.

Por tanto, la ganancia del enlace se define como:

$$G_{u,d,k} = G_{u,d}^{\mathrm{a}} \cdot G_{u,d}^{\mathrm{p}} \cdot G_{u,d}^{\mathrm{e}} \cdot G_{u,d}^{\mathrm{s}} \cdot \left| G_{u,d,k}^{\mathrm{ff}} \right|^2, \tag{1.7}$$

donde k hace referencia al recurso frecuencial utilizado en la comunicación. Cabe destacar que esta ganancia es dependiente tanto de la posición del dron, ρ_d^{D}, como de la posición del UE, ρ_u^{U}. Por lo que se define $G_{u,d,k}(\rho_d^{\mathrm{D}}, \rho_u^{\mathrm{U}})$.

A su vez la potencia recibida por el UE, u, proveniente del dron, d, en el recurso frecuencial, k, se expresa en la siguente ecuación:

$$P_{u,d,k}^{\mathrm{rx}} = P_{d,k}^{\mathrm{tx}} \cdot G_{u,d,k}(\rho_d^{\mathrm{D}}, \rho_u^{\mathrm{U}}), \tag{1.8}$$

donde P_d^{tx} es la potencia transmitida por el dron, d, en el recurso frecuencial, k.

El parámetro que indica la calidad de la señal recibida por el UE, u, del dron, d, en el recurso de frecuencia, k, se denomina *signal to interference plus noise ratio* (SINR), $\gamma_{u,d,k}$, expresado en la siguiente ecuación:

$$\gamma_{u,d,k} = \frac{P_{u,d,k}^{\mathrm{r}x}}{\sum_{\substack{d\prime=0 \\ d\prime \neq d}}^{D} P_{u\prime,d,k}^{\mathrm{r}x} + \sigma^2}, \tag{1.9}$$

donde σ^2 es la potencia del ruido.

Una vez definida la SINR, y utilizando el teorema de Shannon-Hartley, la tasa de transmisión de datos del UE, u, conectado al dron, d, en el recurso de frecuencia, k, se define como:

$$R_{u,d,k} = B_k \log_2(1 + \gamma_{u,d,k}), \tag{1.10}$$

donde B es el ancho de banda de la señal.

Si se utiliza un *scheduler* que reparte los recursos disponibles de manera equitativa entre los UEs de la red, como podría ser el Round-Robin [12], la tasa de transmisión será por tanto la expresada en la ecuación siguiente.

$$R_u = \frac{B}{U} \log_2(1 + \bar{\gamma}_{u,d,k}), \tag{1.11}$$

donde $\frac{B}{U}$ indica la porción del ancho de banda que cada UE utiliza, de media, en la comunicación. Añadir que se utiliza la SINR media, $\bar{\gamma}_{u,d,k}$, del UE en los recursos de frecuencia.

1.3.2. Objetivo

El objetivo de este trabajo es encontrar, en tiempo real, la posición de los drones que maximice la tasa de transmisión total justa presentada en la ecuación 1.12, R_{fair}, que se define como la suma de los logaritmos de las tasas de transmisión de todos los UEs, y a su vez depende de las posiciones de los drones, $\boldsymbol{\rho}^{\text{D}}$, como de las de los UEs, $\boldsymbol{\rho}^{\text{U}}$, como se indicó anteriormente, es decir:

$$R_{\text{fair}}(\boldsymbol{\rho}^{\text{U}}, \boldsymbol{\rho}^{\text{D}}) = \sum_{u \in \mathcal{U}} \log_{10}(R_{u,d,k}(\rho_u^{\text{U}}, \rho^{\text{D}})). \tag{1.12}$$

La implementación de una suma de los logaritmos de las tasas de transmisión individuales, R_{fair}, es una estrategia que hace el cálculo más justo. Esto se debe a que el logaritmo es una función cóncava, lo que implica que aumentos adicionales en la tasa de transmisión de un UE con una tasa ya alta contribuyen menos a R_{fair} en comparación con aumentos en la tasa de transmisión de un UE con una tasa baja.

El problema de optimización se formula formalmente entonces como:

$$\max_{\boldsymbol{\rho}^{\text{D}}} R_{\text{fair}}(\boldsymbol{\rho}^{\text{U}}, \boldsymbol{\rho}^{\text{D}}). \tag{1.13}$$

Este problema se caracteriza por tener una alta dimensionalidad, estocasticidad y no linealidad. Además, las características temporales del problema, como la necesidad de adaptarse continuamente a las condiciones cambiantes del entorno (por ejemplo, cambios en la posición de los UEs y en la topología de la red), hacen que los métodos tradicionales de optimización sean inadecuados. Estos métodos no pueden manejar eficazmente la complejidad y la dinámica del sistema. Por tanto, hemos optado por utilizar técnicas de RL en esta trabajo, que son capaces de aprender y adaptarse a través de la interacción con el entorno, para resolver este problema de manera efectiva. El RL permite encontrar políticas óptimas que maximicen la tasa de transmisión total justa, $R_{\text{fair}}(\boldsymbol{\rho}^{\text{U}}, \boldsymbol{\rho}^{\text{D}})$, ajustándose dinámicamente a los cambios en las condiciones de la red y proporcionando soluciones robustas y eficientes.

1.4. Modelado del Entorno

Para abordar y testear con garantías la problemática establecida, se han llevado a cabo los ensayos en un entorno de simulación capaz de replicar con precisión los eventos más importantes que ocurren en una red. Este entorno es el simulador Giulia, desarrollado en Python, el cual destaca por su gran modularidad y flexibilidad.

Giulia permite la adaptación específica a nuestro problema y facilita la conexión con diversas librerías de Python dedicadas al DRL. Esta capacidad es crucial, ya que permite integrar fácilmente algoritmos avanzados y técnicas de DRL para optimizar la toma de decisiones en la red.

No obstante, antes de poder utilizar Giulia de manera efectiva, fue necesario realizar una optimización del simulador. El DRL es extremadamente sensible al tiempo de simulación, el entrenamiento de un agente requiere la realización de miles de iteraciones en el entorno simulado para alcanzar un rendimiento efectivo. En consecuencia la eficiencia en la simulación no solo reduce el tiempo de cómputo, sino que también mejora la precisión y la estabilidad del modelo entrenado.

La optimización del simulador incluyó la mejora de algoritmos internos, la reducción de la complejidad computacional y la implementación de técnicas avanzadas de paralelización. Estos esfuerzos aseguran que Giulia pueda manejar simulaciones de gran escala y alta fidelidad, permitiendo un entrenamiento más rápido y preciso de los agentes de DRL.

En resumen, el uso del simulador Giulia optimizado proporciona un entorno robusto y eficiente para probar y validar nuestras soluciones de DRL, garantizando que se pueda abordar la problemática de manera efectiva y con resultados confiables.

1.5. Estructura del Trabajo

A continuación, se presenta la estructura de esta tesis, organizada en varios capítulos que detallan cada aspecto de la investigación realizada.

En el capítulo 2, se presenta una introducción detallada al campo del RL, comenzando con los conceptos fundamentales y avanzando hasta el estado del arte. Este capítulo proporciona el contexto necesario para comprender la relevancia y aplicación de RL en la resolución de problemas complejos.

El capítulo 3 se enfoca en la implementación del simulador Giulia. Aquí, se discuten en detalle las optimizaciones llevadas a cabo para asegurar que el simulador pueda entrenar un agente de RL de manera eficiente y en tiempos razonables. Este capítulo también cubre los desafíos técnicos enfrentados y las soluciones implementadas para mejorar el rendimiento del simulador.

En el capítulo 4, se exploran los diferentes algoritmos de RL utilizados en esta investigación. Además, se describen las diversas situaciones y escenarios en los que se realizaron los entrenamientos. Este capítulo es crucial para entender la metodología empleada y cómo se configuraron los experimentos para evaluar el rendimiento de los agentes de RL.

Los resultados obtenidos de los entrenamientos y simulaciones se presentan en el capítulo 5. Aquí, se analizan los datos y se discuten las observaciones clave derivadas de los experimentos. Este análisis proporciona una visión clara del rendimiento de los algoritmos y su eficacia en la resolución de la problemática planteada.

Finalmente, en el capítulo de 6, se sintetizan los hallazgos más importantes de la investigación, además de aclarar cual es la futura línea de investigación en el proyecto.

Capítulo 2

Aprendizaje Reforzado Profundo

2.1. Aprendizaje Reforzado

En esta sección vamos a profundizar en los fundamentos del aprendizaje reforzado, con el objetivo de poder comprender aquellos algoritmos más complejos como los utilizados en este proyecto.

2.1.1. Proceso de Decisión de Markov

Para adentrarnos en el aprendizaje reforzado es necesario entender lo que es un proceso de decisión de Markov.

Un proceso de decisión de Markov (MPD, del inglés *Markov Decision Process*) [13] se define como un proceso de control estocástico en tiempo discreto. Esto significa que es un proceso que incorpora elementos de aleatoriedad (estocásticos) y se evalúa en instantes discretos de tiempo. Además, incluye un agente que puede interactuar con el entorno. En cada instante de tiempo, el agente observa el estado actual del sistema y selecciona una acción. La elección de esta acción influye en el próximo estado del sistema, pero debido a la naturaleza estocástica del proceso, esta transición no es determinista, sino que está descrita por una distribución de probabilidad. Por lo que partiendo de un mismo estado y tomando la misma acción es posible acabar en diferentes estados. El objetivo del agente es maximizar una recompensa acumulada a lo largo del tiempo mediante la toma de decisiones óptimas en cada paso.

Todos los MDP deben respetar la propiedad de Markov presentada en la Ecuación 2.1 la cual establece que el futuro es independiente del pasado una vez conocido el presente. Esto quiere decir que, una vez conozcamos la situación actual de nuestro entorno, no nos es necesaria información pasada para poder caracterizar el proceso. Los MDP están definidos por las siguientes características:

- El espacio de estados, S y S^+. El primer subespacio, S, incluye todos los estados menos los terminales, el segundo incluye también los terminales.
- El espacio de acciones, A.
- La probabilidad de transición entre estados, P.
- La función de recompensa, r.

- El factor de descuento, que representa la importancia de la recompensa inmediata con respecto a las futuras recompensas, γ.
- La trayectoria, τ.

La propiedad de Markov se expresa matemáticamente de la siguiente manera:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, s_2, \ldots, s_t] \tag{2.1}$$

Como hemos dicho previamente, esta propiedad indica que la probabilidad de transicionar al estado, s_{t+1}, depende únicamente del estado actual, s_t, no de los pasados, $s_1, s_2, \ldots, s_{t-1}$.

El agente tiene acceso a la información de los estados, de manera total o parcial. A partir de ahora, para una mejor comprensión, se asumirá que el agente tiene acceso a la información completa de los estados. Dado el estado actual $s_t \in S$ para cada instante de tiempo t, el agente selecciona una acción $a_t \in A$ y observa el siguiente estado s_{t+1}, el cual se obtiene de acuerdo a la probabilidad de transición $P(s_{t+1}|s_t, a_t)$ y recibe una recompensa inmediata $r_{t+1} = r(s_t, a_t, s_{t+1})$.

El objetivo del agente es aprender una política π que le permita seleccionar la mejor acción en cada estado, maximizando la recompensa esperada acumulada a largo plazo . La política π es una función que mapea los estados a las acciones, y puede ser determinista o estocástica. La recompensa esperada se define como:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, s_{t+1}) \mid a \sim \pi(\cdot \mid s_t), s_0\right] \tag{2.2}$$

Esta ecuación es la dedicada para entornos con horizonte de tiempo infinito, pero también puede ser que nos encontremos en un problema con horizonte finito, la ecuación para estos casos es:

$$\mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t, a_t, s_{t+1}) \mid a \sim \pi(\cdot \mid s_t), s_0\right]. \tag{2.3}$$

2.1.2. Periodicidad y Estados Terminales

En el contexto de los entornos de simulación, podemos diferenciar entre episodios y pasos. Los pasos se refieren a las interacciones entre el agente y el entorno. Por ejemplo, en un mundo mallado (comúnmente conocido como *gridworld*), cada vez que el agente realiza una acción, se considera un paso. Un ejemplo de *gridworld* se aprecia en la Figura 2.1.

En este *gridworld* en específico, el agente comienza en una posición inicial y puede moverse en cuatro direcciones (arriba, abajo, izquierda, derecha) hasta llegar a un estado terminal. Hay dos posiciones terminales: $[4, 0]$ y $[0, 4]$. La posición $[4, 0]$ otorga una recompensa negativa (-1) y la posición $[0, 4]$ otorga una recompensa positiva (+1). Estos estados terminales finalizan la simulación, lo que significa que una vez el agente alcanza uno de estos estados, el episodio concluye.

Un episodio se define como la secuencia completa de pasos que realiza el agente desde el estado inicial hasta alcanzar uno de los estados terminales. El objetivo del agente en estos episodios es maximizar la recompensa total acumulada. En el contexto del *gridworld* específico que hemos

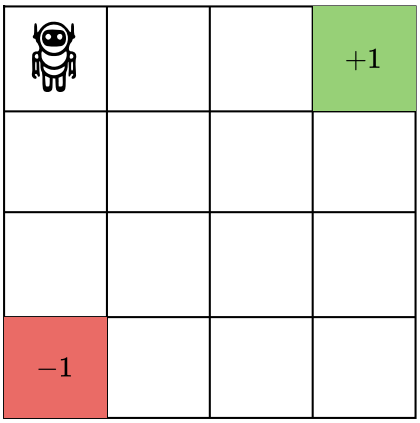


Figura 2.1: Estado inicial.

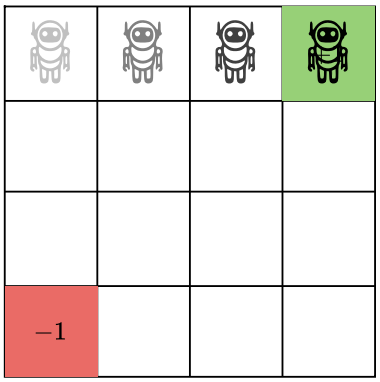


Figura 2.2: Episodio óptimo.

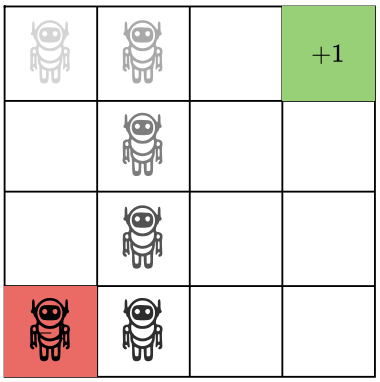


Figura 2.3: Episodio subóptimo.

mencionado, el agente debe aprender a llegar al estado terminal que le otorga la recompensa positiva (+1) lo más rápidamente posible, mientras evita el estado terminal que le otorga una recompensa negativa (-1).

La Figura 2.2 ilustra un episodio óptimo, en el cual el agente navega eficazmente por el *gridworld* y logra llegar al estado $[0, 4]$, obteniendo la recompensa positiva (+1). Este episodio demuestra que el agente ha aprendido una estrategia eficaz para maximizar su recompensa total.

En contraste, la Figura 2.3 muestra un episodio subóptimo. En este caso, el agente termina en el estado $[4, 0]$, recibiendo una recompensa negativa (-1). Esto indica que el agente no ha seguido una estrategia efectiva.

Es importante destacar que en algunos entornos de simulación, no hay estados terminales definidos,

lo que significa que la simulación podría continuar indefinidamente sin concluir.

2.1.3. Procesos de Decisión de Markov Parcialmente Observables

Como hemos mencionado anteriormente, el agente puede tener acceso a la información completa o parcial de los estados. En el caso de que sea parcialmente nos encontramos con un proceso de decisión de Markov parcialmente observable (POMDP, del inglés *Partially Oservable Markov Decision Process*) [14]. En este caso, el agente no observa toda la información que el entorno provee, simplemente una parte de ello. En la literatura las observaciones se denotan como $O_t \in O$, y la definición de estas suele confundirse con la definición de los estados. Sin embargo, es importante tener en cuenta que los estados son todas las variables que definen el entorno, mientras que las observaciones son las variables que el agente puede observar.

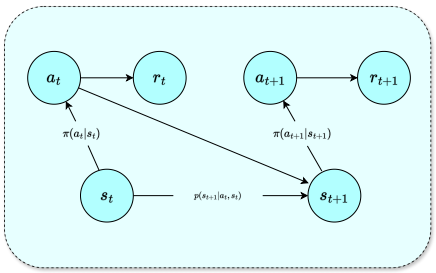


Figura 2.4: Proceso de decisión de Markov (MDP) con observación completa.

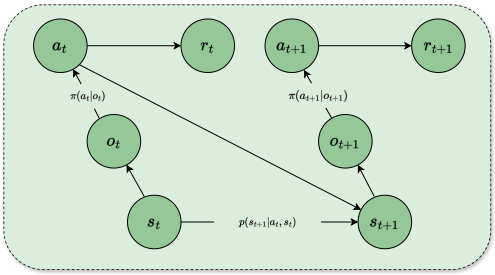


Figura 2.5: Proceso de decisión de Markov parcialmente observable (POMDP).

2.1.4. Política

La política, π, es la encargada de seleccionar la acción que toma el agente en el entorno dado un estado. La política se puede definir como una aplicación $\pi : S \rightarrow A$ que asigna a cada estado $s \in S$ una acción $a \in A$. Esta aplicación puede ser:

- **Determinista** : $\pi(s) = a$, donde cada estado s_t se asigna a una acción única a_t.
- **Estocástica** : $\pi(a \mid s) = p(a \mid s)$, donde $p(a \mid s)$ representa la probabilidad de seleccionar la acción a_t dado el estado s, con $p(a \mid s) \in [0, 1]$.

2.1.5. Probabilidad de Trayectoria

La probabilidad de que un agente, interactuando con un entorno, siga una determinada trayectoria se puede definir de la siguiente manera:

$$\underbrace{p(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p(\tau)} = p(\mathbf{s}_1) \underbrace{\prod_{t=1}^{T} \pi(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)}_{\text{Cadena de Markov } (\mathbf{s},\mathbf{a})} . \tag{2.4}$$

De esta manera, observamos que la probabilidad de que el agente siga la trayectoria τ depende tanto de la probabilidad de comenzar en el estado inicial , $p(\mathbf{s}_1)$, como del producto de las probabilidades de la política, $\pi(\mathbf{a}_t \mid \mathbf{s}_t)$, y de las probabilidades de transición, $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, a lo largo de todos los pasos $t \in T$.

2.1.6. Funciones de Valor y Q

El objetivo del agente es encontrar la política, π, que maximice la recompensa acumulada esperada descontada por:

$$r_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}. \tag{2.5}$$

El factor de descuento, γ, toma valores en el intervalo $[0, 1]$ y se utiliza para dar mayor importancia a las recompensas inmediatas en comparación con las futuras [1]. Si el factor de descuento es 0, el agente se enfoca únicamente en la recompensa inmediata, mientras que si el factor de descuento es 1, el agente considerará todas las recompensas futuras por igual.

Para simplificar la demostración, en esta sección asumiremos que el factor de descuento está fijado en 1. No obstante, estas demostraciones podrían realizarse manteniendo el factor de descuento variable. Por lo tanto, necesitamos buscar una política que maximice:

$$\mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{2.6}$$

Esta esperanza se puede descomponer como:

$$\mathbb{E}_{\mathbf{s}_1 \sim p(\mathbf{s}_1)} \left[\mathbb{E}_{\mathbf{a}_1 \sim \pi(\mathbf{a}_1 | \mathbf{s}_1)} \left[r(\mathbf{s}_1, \mathbf{a}_1) + \mathbb{E}_{\mathbf{s}_2 \sim p(\mathbf{s}_2 | \mathbf{s}_1, \mathbf{a}_1)} \left[\mathbb{E}_{\mathbf{a}_2 \sim \pi(\mathbf{a}_2 | \mathbf{s}_2)} \left[r(\mathbf{s}_2, \mathbf{a}_2) + \cdots \mid \mathbf{s}_2 \right] \mid \mathbf{s}_1, \mathbf{a}_1 \right] \right] \right]. \tag{2.7}$$

De lo que podemos definir:

$$Q(s_1, a_1) = r(s_1, a_1) + \mathbb{E}_{s_2 \sim p(s_2 | s_1, a_1)} \left[\mathbb{E}_{a_2 \sim \pi(a_2 | s_2)} \left[r(s_2, a_2) + \ldots \mid s_2 \right] \mid s_1, a_1 \right]. \tag{2.8}$$

Sustituyendo en la ecuación 2.7:

$$\mathbb{E}_{\tau \sim p(\tau)} \left[\sum_{t=1}^{T} r(s_t, a_t) \right] = \mathbb{E}_{s_1 \sim p(s_1)} \left[\mathbb{E}_{a_1 \sim \pi(a_1 | s_1)} \left[Q(s_1, a_1) \mid s_1 \right] \right]. \tag{2.9}$$

Por tanto la función Q determina la recompensa acumulada esperada que supondría tomar una determinada acción en un estado:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi} \left[\sum_{t}^{T} r(s_{t+1}, a_{t+1}) \mid s_t, a_t \right] : \text{Recompensa total de tomar } a_t \text{ en } s_t. \tag{2.10}$$

De igual manera podemos entender otra de las funciones importantes, la función de valor de los estados. Esta representa la recompensa acumulada que el agente, por estar en ese estado, recibe. Estos valores dependerán de la actual política, ya que es la que controla la decisión de acciones como de la probabilidad de transición del propio entorno.

$$V^{\pi}(s_t) = \mathbb{E}_{\pi}\left[\sum_{t}^{T} r(s_{t+1}, a_{t+1}) \mid s_t\right] : \text{Recompensa total de } s_t \tag{2.11}$$

$$V^{\pi}(s_t) = \mathbb{E}_{a_t \sim \pi(a_t|s_t)}\left[Q^{\pi}(s_t, a_t)\right] = \sum_{a \in A} \pi(s_t, a) Q^{\pi}(s_t, a). \tag{2.12}$$

En la siguiente Figura 2.6, se observa un ejemplo para calcular la función de vaor del estado, V^{π}, en un entorno en el que en el estado s_t únicamente tenemos tres acciones disponibles a_0, a_1, a_2.

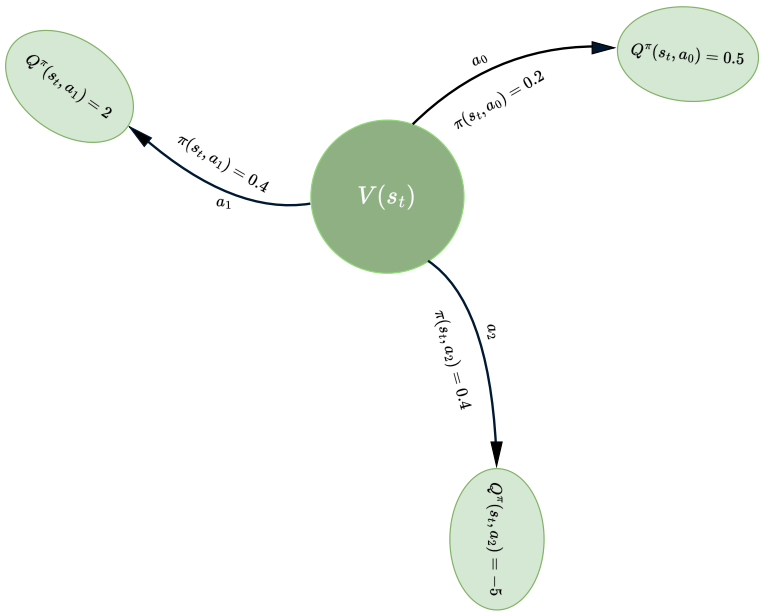


Figura 2.6: Ejemplo cálculo valor V.

Para este ejemplo el cálculo exacto sería:

$$V^{\pi}(s_t) = \sum_{a \in A} \pi(s_t, a) Q^{\pi}(s_t, a) = 0{,}2 \cdot 0{,}5 + 0{,}4 \cdot 2 + 0{,}4 \cdot -5 = -1{,}1.$$

2.1.6.1. Funciones Q y de Valor con Factor de Descuento

Para hacer una aproximación real a como la mayoría de algoritmos funcionan es necesario tener en cuenta el factor de descuento.

La función Q quedaría de la siguiente manera:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left\{r_t \mid s_t, a_t\right\} = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t, a_t\right\}. \tag{2.13}$$

Y la función de valor de la siguiente manera:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left\{r_t \mid s_t\right\} = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid s_t\right\}. \tag{2.14}$$

2.1.6.2. Funciones Óptimas

Las funciones óptimas se definen como objetivo del aprendizaje reforzado. Son los valores que la función de valor de estado y Q tendrían si encontrásemos la mejor política posible.

$$V^{*}(s) = \max_{\pi} V^{\pi}(s). \tag{2.15}$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{2.16}$$

2.1.7. Ecuaciones de Bellman

La ventaja de las ecuaciones previas es que se pueden escribir de manera recursiva y, por tanto, recurrir a técnicas de programación dinámica [15] para poder resolver estos entornos y encontrar la mejor política posible.

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s' \mid s,a)\left[r(s,a,s') + \gamma V^{\pi}(s')\right]. \tag{2.17}$$

Partiendo de la ecuación 2.17 y utilizando la definición de 2.12, tenemos:

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s' \mid s,a)\left[r(s,a,s') + \gamma V^{\pi}(s')\right]. \tag{2.18}$$

De igual manera podemos hacerlo con la función Q. Obteniendo:

$$Q^{\pi}(s,a) = \sum_{s' \in S} P(s' \mid s,a)\left[r(s,a,s') + \gamma \sum_{a' \in A} \pi(s',a') Q^{\pi}(s',a')\right]. \tag{2.19}$$

2.2. Métodos de Aprendizaje Reforzado

Antes de adentrarnos en los diversos métodos de aprendizaje reforzado, es fundamental establecer las diferentes clasificaciones que estos métodos pueden tener.

2.2.1. Clasificación según El Objetivo

Es la clasificación más común entre los diferentes métodos, se pueden identificar tres clases [16]:

- **Aprendizaje por valor** : El objetivo es aprender la función de valor óptima, $V^*(s)$ o $Q^*(s, a)$. Esta función de valor permite al agente seleccionar la mejor acción en cada estado al evaluar las posibles recompensas futuras esperadas.
- **Aprendizaje por política** : El objetivo es aprender la política óptima, $\pi^*(s)$. A diferencia de los anteriores, en los que se aprende la función de valor aquí directamente tenemos conocimiento de cual es la mejor acción a decidir.
- **Actor-crítico** : Este método combina los enfoques de aprendizaje por valor y aprendizaje por política, permitiendo al agente aprender tanto la función de valor como la política óptima simultáneamente.

2.2.2. Clasificación según El Tipo de Política

Además de los objetivos, también se pueden clasificar los métodos de aprendizaje reforzado según la información con la que son entrenados:

- ***On-policy*** : El agente aprende y actualiza la política basándose en la política que está siguiendo actualmente. Este enfoque se centra en mejorar la política en uso mediante la experiencia adquirida.
- ***Off-policy*** : El agente aprende la política óptima mientras sigue una política diferente, conocida como política de comportamiento. Este enfoque permite al agente explorar con una política mientras mejora otra, a menudo utilizando técnicas de aprendizaje más complejas y exploración extensa. O también puede darse el caso en el que para conseguir aprender la función valor $V(s)$ se utilicen todas las muestras de las que el agente dispone, dando igual si se utilizaba otra política.

2.2.3. Clasificación según El Uso de Modelo

Otra forma de clasificar los métodos de aprendizaje reforzado es según el uso de modelos del entorno:

- **Métodos basados en modelo** : Estos métodos utilizan un modelo explícito del entorno para tomar decisiones y planificar. El modelo puede predecir las transiciones de estados y las recompensas futuras, lo que permite al agente simular diferentes escenarios y estrategias antes de ejecutarlas en el entorno real. De manera que en este algoritmo también se tiene una cierta estimación de como funciona el entorno sin tener que interactuar con el [17].
- **Métodos libres de modelo** : Estos métodos no utilizan un modelo explícito del entorno. En lugar de ello, el agente aprende directamente a partir de la experiencia e interactuar con el entorno, actualizando sus estimaciones de la función de valor o su política basándose en la retroalimentación recibida por las interacciones directas [18].

Por tanto una clasificación de diferentes modelos se puede entender como en la siguiente Figura 2.7. En la cual estudiamos los modelos dependiendo de el tipo de objetivo que siguen, añadiendo matices sobre la política utilizada.

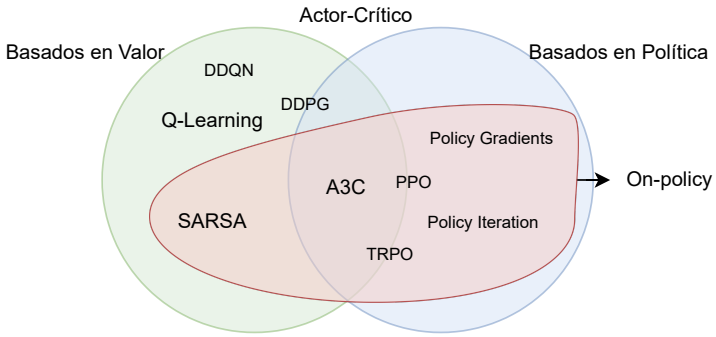


Figura 2.7: Clasificación de diferentes modelos.

2.3. Aprendizaje por Valor

Este tipo de técnicas se centran en seleccionar las acciones a tomar basándose en las funciones $V(s)$ y $Q(s,a)$. De manera que el agente tiene que aprender a aproximar dichas funciones y, posteriormente, decidir cual es la acción que mayor recompensa a largo plazo le va a otorgar.

2.3.1. Q-Learning

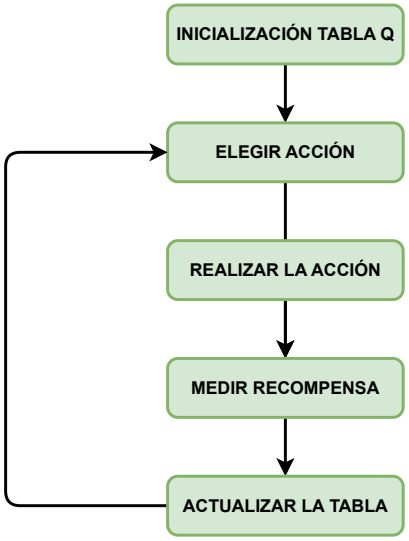
El algoritmo Q-Learning [19] se basa en la construcción de una tabla en la que se registran los resultados de las interacciones del agente con el entorno. La tabla Q almacena los valores de recompensa obtenidos al realizar una acción específica en un estado determinado. Tras múltiples iteraciones la tabla Q converge hacia una tabla óptima, lo que permite al agente seleccionar las acciones ideales para maximizar las recompensas acumuladas como se puede ver en la Figura 2.8. Se caracteriza por ser *off-policy* y libre de modelo.

En cada iteración, la tabla Q se actualiza mediante:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \text{max}_a Q(s_{t+1},a)\right], \tag{2.20}$$

donde tenemos:

- α es la tasa de parendizaje.Un parámetro que determina la influencia del nuevo conocimiento en la actualización de la tabla Q.
- r_{t+1}, es la recompensa inmediata, después de realizar la acción actual, a_t en el estado actual, s_t.
- γ, es el factor de descuento, el cual determina la importancia de las recompensas futuras.



$Q(s,a)$	a_0	a_1	a_2	...
s_0	$Q(s_0,a_0)$	$Q(s_0,a_1)$	$Q(s_0,a_2)$	...
s_1	$Q(s_1,a_0)$	$Q(s_1,a_1)$	$Q(s_1,a_2)$	...
s_2	$Q(s_2,a_0)$	$Q(s_2,a_1)$	$Q(s_2,a_2)$	...
...	...	...	...	...

Figura 2.8: Algoritmo de aprendizaje Q y tabla Q.

- $\max_a Q(s_{t+1}, a)$, es el valor máximo de la función Q, para el estado siguiente, s_{t+1}, evaluado sobre todas las acciones posibles, $a \in A$.

La intuición detrás de este algoritmo es la siguiente:

1. El agente se encuentra en el estado s_t.
2. El agente toma la acción a_t en el estado s_t.
3. Como resultado de esta acción, el agente recibe una recompensa inmediata r_{t+1} y transita al nuevo estado s_{t+1}.
4. El valor $Q(s_t, a_t)$ se actualiza para reflejar tanto la recompensa inmediata r_{t+1} como la recompensa futura estimada.
5. La actualización de $Q(s_t, a_t)$ se realiza utilizando la Ecuación 2.20.
6. Esta fórmula de actualización asegura que el valor $Q(s_t, a_t)$ se ajusta para reflejar no solo la recompensa inmediata r_{t+1}, sino también la recompensa futura esperada al seguir la mejor política posible desde el estado s_{t+1}.
7. El proceso se repite para cada paso en el tiempo, ajustando continuamente los valores de Q para mejorar la política del agente.

Lo que se sintetiza en: se elige una acción, se analiza la recompensa inmediata, y a partir del conocimiento previo se selecciona la acción que va a reportar una mayor recompensa. Una explicación más extensa se observa en el algoritmo 1.

La tabla Q se ajusta iterativamente, combinando recompensas inmediatas y a largo plazo para aprender la política óptima de una manera implícita.

Algorithm 1 Algoritmo Q-learning

Parámetros del algoritmo : tamaño de paso $\alpha \in (0, 1]$, $0 < \epsilon < 1$
Inicializar $Q(s, a)$, para todos $s \in S^+$, $a \in \mathcal{A}(s)$, arbitrariamente excepto que $Q(\text{terminal}, \cdot) = 0$
repeat
  Inicializar s_t
  repeat
    Elegir a_t de s_t usando una política como ϵ-greedy
    Tomar acción a_t, observar r_t, s_{t+1}
    $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha \left[r_t + \gamma \text{max}_a Q(s_{t+1}, a)\right]$
    $s \leftarrow s_{t+1}$
  until s sea terminal
until Q converja

La tasa de aprendizaje debe satisfacer las siguientes condiciones:

$$\sum_{t=0}^{\infty} \alpha_t(s, a) = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2(s, a) < \infty. \tag{2.21}$$

2.3.1.1. Exploración versus Explotación

En estos algoritmos no siempre es óptimo seleccionar la acción con el mayor valor de Q, ya que, dependiendo de la inicialización, los valores en la tabla Q pueden ser subóptimos y lejanos del valor real. Por tanto, se utiliza una técnica llamada ϵ-greedy [20].

La estrategia ϵ-greedy consiste en tomar decisiones aleatorias con una pequeña probabilidad ϵ y seguir la política óptima con una probabilidad $1 - \epsilon$. Es decir, con una probabilidad ϵ el agente explora el entorno eligiendo una acción aleatoria. Con una probabilidad $1 - \epsilon$, el agente explota su conocimiento actual seleccionando la acción que maximiza el valor Q.

Matemáticamente, la acción a_t se selecciona de la siguiente manera:

$$a_t = \begin{cases} \text{acción aleatoria} & \text{con probabilidad } \epsilon \\ \arg\max_a Q(s_t, a) & \text{con probabilidad } 1 - \epsilon. \end{cases} \tag{2.22}$$

La técnica ϵ-greedy ayuda a equilibrar la exploración y la explotación. La exploración permite al agente descubrir nuevas acciones que podrían proporcionar mejores recompensas en el futuro, mientras que la explotación permite al agente utilizar su conocimiento actual para maximizar las recompensas inmediatas. Básicamente, esta técnica ayuda a descubrir nuevas estrategias que sin exploración no se podrían obtener. Con el tiempo, el valor de ϵ puede disminuir gradualmente para que el agente explore menos y explote más a medida que adquiere más conocimiento sobre el entorno. Sin embargo, esta técnica solamente se utiliza en la selección de la primera acción. Una vez seleccionada la primera acción el algoritmo Q-learning elegirá la acción con mayor recompensa.

De esta manera se puede asegurar que estos algoritmos lleguen a converger a una solución más cercana a la óptima.

2.3.2. SARSA

El algoritmo SARSA [21], que significa *State-Action-Reward-State-Action*, sigue la misma intuición que el Q-learning. La diferencia radica en la manera de actualizar la tabla Q. En lugar de utilizar el valor máximo de Q en el estado siguiente, SARSA vuelve a utilizar la misma política ϵ-greedy para seleccionar la siguiente acción. SARSA actualiza la tabla Q utilizando la acción real tomada en el estado siguiente, en lugar de la mejor acción posible.

Por eso mismo se caracteriza por ser *on-policy*, ya que la política que se sigue para seleccionar la siguiente acción es la misma que la que se está actualizando. Por tanto, este algoritmo sigue:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})\right]. \tag{2.23}$$

Algorithm 2 Algoritmo SARSA

Parámetros del algoritmo : tamaño de paso $\alpha \in (0, 1]$, $0 < \epsilon < 1$
Inicializar $Q(s, a)$, para todos $s \in S^+$, $a \in A(s)$, arbitrariamente excepto que $Q(\text{terminal}, \cdot) = 0$
repeat
    Inicializar s_t
    Elegir a_t de s_t usando una política como ϵ-greedy
    repeat
        Tomar acción a_t, observar r_t, s_t
        Elegir a_{t+1} de s_{t+1} usando una política como ϵ-greedy
        $Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1})]$
        $s \leftarrow s_{t+1}$
        $a \leftarrow a_{t+1}$
    until s_{t+1} sea terminal
until Q converja

2.4. Aprendizaje por Política

Los algoritmos de aprendizaje por política se enfocan en optimizar directamente una política, π, que puede ser estocástica o determinista. Con el mismo objetivo de maximizar la recompensa acumulada a largo plazo. A diferencia de los métodos basados en el valor, que primero estiman una función de valor y luego derivan una política a partir de ella, los métodos de aprendizaje por política buscan directamente la mejor política sin necesidad de pasar por una función de Q intermedia. Este enfoque permite manejar de manera más efectiva políticas en espacios de acción continuos y en problemas donde las políticas deben ser suavemente ajustadas.

2.4.1. Iteración de Política

La iteración de política es uno de los métodos fundamentales en el aprendizaje por política, que alterna entre dos fases principales: la evaluación de la política actual y la mejora de dicha política. Este proceso iterativo permite al agente converger hacia una política óptima. Este algoritmo es similar a los evaluados tabularmente, como el *Q-learning* o el SARSA.

2.4.1.1. Evaluación de la Política

La evaluación de la política implica calcular el valor esperado de la política actual, π. Esto se realiza mediante un proceso iterativo que ajusta los valores de los estados basándose en las recompensas y las transiciones esperadas. El algoritmo se observa en 3. El cual se detiene cuando las diferencias entre las iteraciones sucesivas son menores que un umbral, θ.

Algorithm 3 Evaluación iterativa de la política, para estimar $V \approx V^{\pi}$

Entrada : π, la política a evaluar
Parámetro del algoritmo : un pequeño umbral $\theta > 0$ que determina la precisión de la estimación
Inicializar $V(s)$, para todos $s \in S^{+}$, arbitrariamente excepto que $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for cada $s \in S$ **do**
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_{a} \pi(a|s) \sum_{s',r} p(s', r|s, a) \left[r + \gamma V(s')\right]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$

2.4.1.2. Mejora de la Política

Una vez que la política actual ha sido evaluada, el siguiente paso es mejorarla. La mejora se realiza ajustando su valor para que seleccione acciones que maximicen el valor esperado en cada estado, utilizando la función de valor $V(s)$ obtenida en la fase de evaluación. Este proceso de alternar entre evaluación y mejora se repite hasta que la política converge a una política óptima. El algoritmo 4, muestra como se actualiza está política.

Este proceso asegura que el agente iterativamente mejora su política hasta que alcanza la óptima, donde ningún cambio adicional puede proporcionar una recompensa mayor.

2.4.2. Política de Gradientes

Estos algoritmos, en comparación a los de iteración de política y los basados en valor, no se centran en ningún tipo de estimación de la función de valor [22]. Computan una política π, a través de la que interactuar con el entorno. En este caso la política estará determinada por una serie de parámetros,$\theta \in \mathbb{R}^{d'}$, a través de los cuales aproximaremos nuestra política. De manera que la política ahora quedará como $\pi(a \mid s, \theta) = Pr(a \mid s, \theta)$ para cada acción $a \in A$ tomada en el intervalo de tiempo t, en el estado $s \in S$ y distribución de parámetros θ.

Al tener unos parámetros, θ, es necesario optimizarlos y para ello se utilizan técnicas como el cálculo del gradiente, el cual veremos más adelante en la subsección 2.5.4.3. El objetivo de esto es aumentar la función de rendimiento escalar $J(\theta)$. El cual definimos como la recompensa acumulada esperada. Los pesos se actualizan en base a:

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta). \tag{2.24}$$

Algorithm 4 Iteración de Política (usando evaluación iterativa de la política) para estimar $\pi \approx \pi^*$

Inicialización
$V(s) \in \mathbb{R}$ y $\pi(s) \in A(s)$ arbitrariamente para todos $s \in S$

Evaluación de la Política
Mejora de la Política
policy-stable $\leftarrow$ true
for cada $s \in S$ **do**
  old-action $\leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$
  if old-action $\neq \pi(s)$ **then**
    policy-stable $\leftarrow$ false
  end if
end for
if policy-stable **then**
  detener y retornar $V \approx v^*$ y $\pi \approx \pi^*$
else
  ir a 2
end if

Para que estos teoremas se puedan utilizar es necesario que la política, $\pi(a \mid s, \theta)$, sea diferenciable con respecto a los parámetros, θ,para todos $s \in S$, $a \in A$ y $\theta \in \mathbb{R}^{d'}$. Además, para asegurar la explotación en este tipo de métodos es necesario que esta nunca se convierta en una política determinista. Ya que aprovecha la naturaleza de las distribuciones estocásticas haciendo que cada vez que está en el estado s_t elija una de las acciones en $\pi(a \mid s, \theta)$. Sin embargo, si la naturaleza del problema requiere que la política sea determinista, se pueden elegir acciones mediante ϵ-greedy, cuyo parámetro ϵ puede ir decayendo a través de las iteraciones. De esta manera, si la política óptima es una determinista, las actualizaciones de las acciones suboptimas serán infinitamente más pequeñas que las óptimas.

2.4.2.1. El Teorema de La Política de Gradiente

Dentro de este algoritmo, y gracias a las propiedades de la parametrización, las probabilidades de las acciones cambian de manera suave. Esto contrasta con otras técnicas que pueden producir cambios bruscos en la probabilidad de las acciones con una mínima variación en la estimación de los valores V. Esto se debe principalmente a que se pueden utilizar técnicas como el ascenso de gradiente 2.24. Para realizar modificaciones en la actuación de la política de manera más gradual.

Esta técnica presenta ciertas complicaciones para determinar cuál es la modificación de los parámetros que hace que la recompensa, $J(\theta)$, aumente. Además la modificación de los parametros 2.24 impacta directamente en cómo el agente actúa en el entorno y, por tanto, modifica el valor de los estados. Esto ocurre porque la política está parametrizada por θ, y cualquier cambio en estos parámetros altera la probabilidad de seleccionar ciertas acciones en ciertos estados, lo que a su vez afecta las transiciones y recompensas en el entorno. Por ejemplo, si en una primera iteración en 2.1 mi agente decide desplazarse hacia abajo, el valor de los estados va a ser negativo ya que no conoce otra solución, y al evaluar la política tendrán valor negativo. Sin embargo, si en la siguiente

iteración este agente decide ir hacia la derecha, el valor de los estados será positivo. Lo cual hace que sea inestable.

Esta interacción compleja entre los parámetros y el entorno puede ser manejada efectivamente mediante el uso de técnicas de gradiente. En particular, el teorema de la política de gradiente nos proporciona una forma de calcular el gradiente de la recompensa esperada con respecto a los parámetros de la política. Consiguiendo evitar el problema de las modificaciones repentinas en la función de valor.

Demostración de Algoritmos de Política de Gradiente

$$
\begin{aligned}
\nabla_\theta v_{\pi_\theta}(s) &= \nabla_\theta \left[\sum_a \pi_\theta(a|s) Q_{\pi_\theta}(s,a) \right], \quad \text{para todo } s \in S \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta Q_{\pi_\theta}(s,a) \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} p(s',r|s,a)(r + v_{\pi_\theta}(s')) \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s'} p(s'|s,a) \nabla_\theta v_{\pi_\theta}(s') \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s'} p(s'|s,a) \right. \\
&\quad \left. \sum_{a'} \left[\nabla_\theta \pi_\theta(a'|s') Q_{\pi_\theta}(s',a') + \pi_\theta(a'|s') \sum_{s''} p(s''|s',a') \nabla_\theta v_{\pi_\theta}(s'') \right] \right] \\
&= \sum_{x \in S} \sum_{k=0}^{\infty} \Pr(s \to x, k, \pi_\theta) \sum_a \nabla_\theta \pi_\theta(a|x) Q_{\pi_\theta}(x,a), \\
\nabla_\theta J(\theta) &= \nabla_\theta v_{\pi_\theta}(s_0) \\
&= \sum_s \left(\sum_{k=0}^{\infty} \Pr(s_0 \to s, k, \pi_\theta) \right) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a), \\
&= \sum_{s'} \eta(s') \left(\sum_{s'} \frac{\eta(s')}{\sum_{s'} \eta(s')} \right) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) Q_{\pi_\theta}(s,a).
\end{aligned}
$$

Como hemos nombrado antes, al modificar los parámetros de la aproximación cambia la dinámica en la que se interactúa con el entorno. Esto no supone un problema a la hora de calcular cómo se

modifica la recompensa que se obtiene, ya que si se modifica la política, π_θ, de manera directa se puede obtener cómo la recompensa es modificada. Pero no sucede lo mismo con la función valor, no sabemos directamente como un cambio en los parámetros puede afectar a su valor.

Por tanto, en la primera parte de la prueba se intenta llegar a una solución para el gradiente en la que solo se computa el gradiente con respecto a la política multiplicando al valor Q de cada tupla estado, acción. Donde $\Pr(s \to x, k, \pi_\theta)$ representa la probabilidad de acabar en el estado x después de k iteraciones.

La manera de medir la mejor política es la función valor en el estado inicial s_0, de forma que cuanto mayor sea dicho valor mejor será la política π_θ que estamos usando. Volviendo al ejemplo del *gridworld*, si el valor de mi estado estado inicial es positivo indica que la política utilizada es mejor que la política en el caso en el que ese valor es negativo.

En la segunda parte de la demostración se pone este cálculo del gradiente en función de $\eta(s)$ y $\mu(s)$, lo que representan el número esperado de visitas al estado s por episodio y la probabilidad de estar en el estado s respectivamente. De manera que $\mu(s) \geq 0$ y $\sum_s \mu(s) = 1$. La relación que tienen ambas es:

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}. \tag{2.25}$$

Esto se hace a que si es posible estimar en media la probabilidad de que el agente se encuentre en ese estado, no pasa así con las probabilidades de transición entre estados que requieren un conocimiento completo del entorno.

2.4.2.2. Algoritmo REINFORCE

Para lograr el objetivo de aprender mediante el gradiente 2.24, utilizamos [23]:

$$\begin{aligned} \nabla J(\theta) &\propto \sum_s \mu(s) \sum_a Q_\pi(s,a) \nabla \pi(a \mid s, \theta) \\ &= \mathbb{E}_\pi \left[\sum_a Q_\pi(s_t, a) \nabla \pi(a \mid s_t, \theta) \right]. \end{aligned} \tag{2.26}$$

El teorema de la política de gradiente establece una relación de proporcionalidad necesaria para que el algoritmo funcione correctamente. Para el ascenso de gradiente, necesitamos obtener muestras cuya esperanza sea proporcional al gradiente real del rendimiento en función de los parámetros, $J(\theta)$, de esta manera se puede cumplir 2.4.2.1. La parte derecha del teorema de la política de gradiente es una suma sobre los estados, ponderada por la frecuencia con la que ocurren los estados bajo la política π. Sin embargo, para hacer posible utilizar esta ecuación sin necesidad de conocer parámetros como la probabilidad de estar en un estado, $\mu(s)$, se utilizan las propiedades del muestreo. La cual indica que si se sigue la política π, en media, las interacciones con el entorno seguirán aquella distribución que cumple el valor real de $\mu(s)$. Por tanto, en lugar de utilizar un sumatorio sobre todos los estados, interactuamos con el entorno y calculamos la esperanza de esto.

De la misma manera que en 2.26 con s_t lo hacemos para a_t, siguiendo la política π, las acciones que se muestreen van a seguir la misma distribución que si hacemos media sobre todos las acciones.

$$\begin{aligned}\nabla J(\theta) &= \mathbb{E}_\pi \left[\sum_a \pi(a \mid s_t, \theta) Q_\pi(s_t, a) \frac{\nabla \pi(a \mid s_t, \theta)}{\pi(a \mid s_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[Q_\pi(s_t, a_t) \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)} \right] \\ &= \mathbb{E}_\pi \left[R_t \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)} \right].\end{aligned} \tag{2.27}$$

Además, por definición, tenemos que $Q_\pi(s_t, a_t) = \mathbb{E}\pi \left[r_t \mid s_t, a_t \right]$.

Por tanto, la expresión final de 2.24, termina con la expresión que se encuentra dentro de los paréntesis.

$$\theta_{t+1} = \theta_t + \alpha R_t \frac{\nabla \pi(a_t \mid s_t, \theta)}{\pi(a_t \mid s_t, \theta)}. \tag{2.28}$$

Esta expresión es intuitiva por dos razones principales. Primero, hace que el parámetro se mueva principalmente en las direcciones que favorecen las acciones que generan la mayor recompensa total esperada, al estar multiplicando el gradiente por estas. Segundo, la ponderación inversa a la probabilidad de la acción asegura que las acciones seleccionadas con mayor frecuencia no tengan una ventaja desproporcionada, evitando que dominen la política incluso si no proporcionan la mayor recompensa.

En la implementación del algoritmo 2.4.2.2 se usa la siguiente transformación $\nabla \ln(x) = \frac{\nabla x}{x}$. De manera que la ecuación 2.28 se puede reescribir de la siguiente forma:

$$\theta_{t+1} = \theta_t + \alpha R_t \nabla \ln \pi(a_t \mid s_t, \theta). \tag{2.29}$$

Algorithm 5 REINFORCE:

Entrada: una parametrización diferenciable de la política $\pi(a \mid s, \theta)$
Parámetro del algoritmo: tamaño de paso $\alpha > 0$
Inicializar el parámetro de la política $\theta \in \mathbb{R}^d$ (por ejemplo, a 0)
Bucle infinito (para cada episodio):
    Generar un episodio $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$, siguiendo $\pi(\cdot \mid \cdot, \theta)$
for cada paso del episodio $t = 0, 1, \ldots, T-1$ **do**
    $R \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k$
    $\theta \leftarrow \theta + \alpha \gamma^t R \nabla \ln \pi(a_t \mid s_t, \theta)$
end for

2.4.3. El Problema de La Expansión Exponencial del Espacio de Estados

Consideremos un ejemplo sencillo de un *gridworld*. Si definimos un entorno de tamaño 4x4, tenemos un total de 16 estados. Con 4 acciones posibles en cada estado, el número total de valores para la función Q sería $16 \times 4 = 64$. Sin embargo, al incrementar el tamaño del *gridworld* a 8x8, el número de estados aumenta a 64, y el número total de valores para la función Q se incrementa a $64 \times 4 = 256$.

A medida que trabajamos en entornos más complejos, como imágenes o videojuegos, el número de estados posibles crece exponencialmente. Por ejemplo, en el caso de una imagen de 100x100 píxeles en un entorno binario, el número de posibles configuraciones de estados sería 2^{10000}, un número extremadamente grande y difícil de manejar.

Este crecimiento exponencial del espacio de estados, conocido como la explosión del espacio de estados, representa un desafío significativo en el aprendizaje por refuerzo. Manejar y aprender en un espacio de estados tan vasto requiere técnicas avanzadas para la generalización y la eficiencia computacional.

Por lo tanto, se utilizan técnicas que puedan reducir la dimensionalidad de estados tan grande e implementarlas dentro de los algoritmos, como las redes neuronales artificiales. Estas técnicas permiten representar de manera compacta y eficiente el espacio de estados, facilitando el aprendizaje y la toma de decisiones en entornos complejos.

2.5. Redes Neuronales Artificiales

Las redes neuronales artificiales son un tipo de modelo de aprendizaje automático inspirado en la estructura y funcionamiento del cerebro humano. Están compuestas por múltiples capas de neuronas interconectadas, que procesan y transforman la información a medida que se propaga a través de la red. Las redes neuronales son capaces de aprender patrones y relaciones complejos en los datos, lo que las convierte en una herramienta poderosa para la clasificación, la regresión y la toma de decisiones en entornos complejos.

2.5.1. Perceptrón

El perceptrón es la unidad básica de una red neuronal, introducida en [24], que simula el comportamiento de una neurona biológica. En la Figura 2.9, observamos que el perceptrón consiste en un conjunto de entradas representadas como un vector $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ que se multiplica por un vector de pesos $\mathbf{w} = [w_1, w_2, \ldots, w_N]$. El producto interno de estos vectores, más un sesgo b, se pasa a través de una función de activación $\phi(\cdot)$, la cual determina la salida del perceptrón.

El modelo matemático de este perceptrón se define como:

$$y = \phi\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right). \tag{2.30}$$

2.5.2. Funciones de Activación

Dentro de las redes neuronales, las funciones de activación juegan un papel crucial al introducir no linealidades en el modelo [25]. Esto permite a las redes neuronales aprender y modelar relaciones complejas. Una característica importante de muchas funciones de activación es su diferenciabilidad, que facilita la aplicación de métodos de optimización como el descenso de gradiente 2.51.

- **Función Escalón**: La función escalón es una función de activación simple que produce una salida binaria. Si la entrada es mayor que un umbral, la salida es 1; de lo contrario, la salida

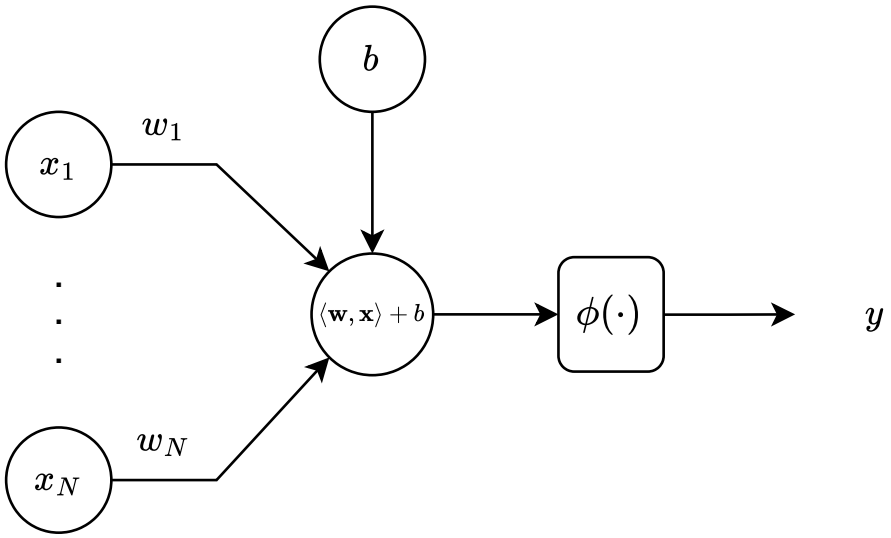


Figura 2.9: Esquema de un perceptrón.

es 0. La función escalón es una función no diferenciable, lo cual arrastra consigo una serie de problemas en la propagación del gradiente.

$$f(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0. \end{cases}$$

- **Función Sigmoide**: La función sigmoide es una función de activación suave que produce una salida en el rango de 0 a 1. La función sigmoide es diferenciable, lo que la hace útil para el entrenamiento de redes neuronales utilizando métodos basados en gradientes.

$$f(x) = \frac{1}{1 + e^{-x}}.$$

- **Función ReLU**: La función ReLU (*Rectified Linear Unit*) es una función de activación no lineal que produce una salida de 0, si la entrada es negativa, y la entrada misma, si esta es positiva. Es diferenciable en casi todo su dominio menos en 0, lo que facilita el uso de métodos de optimización basados en gradientes.

$$f(x) = \begin{cases} x & \text{si } x \geq 0 \\ 0 & \text{si } x < 0. \end{cases}$$

2.5.3. Perceptrón Multicapa

Cuando nos enfrentamos a problemas complejos en los que es necesario aprender a resolver tareas de clasificación y regresión no lineales, el perceptrón simple no es suficiente. Añadir varias capas de perceptrones, junto con la posibilidad de seleccionar distintos tipos de funciones de activación, permite al modelo aprender relaciones más complejas [26]. En la Figura 2.10, se muestra un esquema de un perceptrón multicapa.

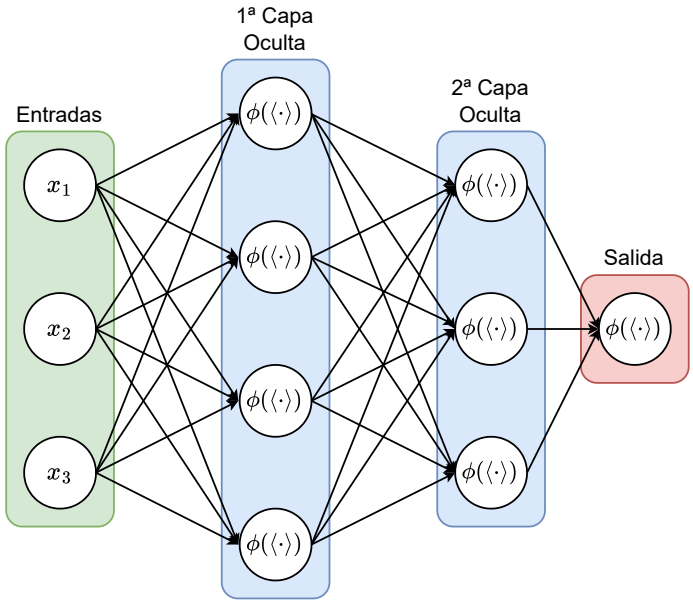


Figura 2.10: Esquema de un perceptrón multicapa.

Dentro de este perceptrón multicapa se pueden observar tres tipos de capas:

- **Capa de entrada**: Esta capa recibe el vector de entrada, **x**, y lo propaga a través de la red.
- **Capas ocultas**: Estas capas intermedias procesan la información y aprenden representaciones complejas de los datos. Reciben la salida de las capas anteriores denominada como activación, a. Gracias a las características no lineales de la función de activación, la red puede procesar relaciones más abstractas de los datos.
- **Capa de salida**: Esta capa produce la salida final de la red, $\hat{y}$, que puede ser una clasificación, una regresión, o cualquier otro tipo de tarea.

En la Figura 2.10, se observa el esquema de un perceptrón multicapa completamente conectado, pero también puede darse el caso que no todas las neuronas estén conectadas entre sí, o que haya conexiones entre neuronas de capas no consecutivas.

2.5.4. Proceso de Entrenamiento

Como se mencionó previamente, las redes neuronales han demostrado ser un aproximador de funciones excepcional, capaz de adaptarse a cualquier tipo de dato y aprender relaciones complejas entre ellos, ya sean lineales o no. Ejemplos claros de su funcionamiento son los grandes modelos de lenguaje, o *Large Language Models*, los cuales pueden aprender las relaciones entre palabras y llegar a imitar el comportamiento humano en cuanto a la generación de lenguaje [27]. Otros ejemplos, más relacionados con el campo del aprendizaje reforzado, son modelos como AlphaGo, desarrollado por Google, que logró vencer al campeón mundial del famoso juego asiático Go [28].

Para que las redes neuronales alcancen este nivel de desempeño excepcional, es necesario llevar a cabo un proceso de entrenamiento previo, el cual consta de tres etapas principales.

2.5.4.1. Feedforward

El proceso de propagación hacia adelante, o *feedforward*, se puede comprender como la aplicación lineal de la Ecuación 2.30 en cada capa de la red. La salida de una capa se convierte en la entrada de la siguiente, y así sucesivamente hasta llegar a la capa de salida. Este proceso se puede expresar matemáticamente de la siguiente manera:

$$\mathbf{a}^{(l)} = \phi\left(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right), \tag{2.31}$$

donde $\mathbf{a}^{(l)}$ es la salida (o activación) de la capa l, $\mathbf{W}^{(l)}$ es la matriz de pesos de la capa l, $\mathbf{b}^{(l)}$ es el vector de sesgos de la capa l, y $\phi(\cdot)$ es la función de activación de la capa l.

Cada columna de la matriz de pesos, $\mathbf{W}^{(l)}$, corresponde con las conexiones que tiene con la siguiente capa, y cada fila, con el numero de neuronas de la capa anterior. En el ejemplo de la figura 2.10, tendríamos tres matrices de pesos, la que conecta la capa de entrada con la primera capa oculta, la que conecta la primera capa oculta con la segunda capa oculta y la que conecta la segunda capa oculta con la capa de salida. Además, a cada capa ha de añadirsele un vector de sesgos, $\mathbf{b}$. Estas tendrían la siguiente forma:

$$\mathbf{W}^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} & w_{14}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} & w_{24}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} & w_{34}^{(1)} \end{pmatrix}, \quad \mathbf{b}^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{pmatrix}, \tag{2.32}$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} & w_{13}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} & w_{23}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} & w_{33}^{(2)} \\ w_{41}^{(2)} & w_{42}^{(2)} & w_{43}^{(2)} \end{pmatrix}, \quad \mathbf{b}^{(2)} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix}, \tag{2.33}$$

$$\mathbf{W}^{(3)} = \begin{pmatrix} w_{11}^{(3)} \\ w_{21}^{(3)} \\ w_{31}^{(3)} \end{pmatrix}, \quad \mathbf{b}^{(3)} = \left(b_1^{(3)}\right). \tag{2.34}$$

De manera que un completo feedforward se puede representar por las operaciones en la Ecuacion 2.35, aprovechando la propiedad de la composición de funciones:

$$\hat{\mathbf{y}} = \phi^{(3)}\left(\mathbf{W}^{(3)}\phi^{(2)}\left(\mathbf{W}^{(2)}\phi^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right) + \mathbf{b}^{(2)}\right) + \mathbf{b}^{(3)}\right). \tag{2.35}$$

2.5.4.2. Cálculo de Pérdida

Una vez se ha propagado la entrada por toda la red, y usando un ejemplo de clasificación o regresión, debemos comparar la salida de la red con el valor real. La manera de hacerlo es mediante

una función de pérdida, que mide la diferencia entre la salida predicha, $\hat{y}$, y el valor real, y. La elección de la función de pérdida depende del tipo de problema que se esté abordando. Algunas de las funciones de pérdida más comunes son:

- **Error Cuadrático Medio (MSE)**: La función de pérdida MSE es una medida común para problemas de regresión. Calcula el promedio de las diferencias al cuadrado entre las predicciones y los valores reales.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{2.36}$$

  donde n es el número de muestras, y_i es el valor real de la muestra i y $\hat{y}_i$ es el valor predicho por la red en la muestra i. Esta función de pérdida penaliza fuertemente los errores grandes, haciendo que el modelo se ajuste para reducir estos errores.

- **Error Absoluto Medio (MAE)**: Utilizada para problemas de regresión, la función MAE calcula el promedio de las diferencias absolutas entre las predicciones y los valores reales.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|. \tag{2.37}$$

  La MAE es más robusta a valores atípicos en comparación con el MSE, ya que no penaliza los errores grandes de manera tan severa.

- **Entropía Cruzada**: La entropía cruzada es una función de pérdida común para problemas de clasificación. Mide la discrepancia entre la distribución de probabilidad predicha por el modelo y la distribución real de las etiquetas.

$$\text{CE} = -\sum_{i} y_i \log(\hat{y}_i), \tag{2.38}$$

  donde y_i es la etiqueta verdadera (generalmente un valor binario 0 o 1) y $\hat{y}_i$ es la probabilidad predicha por la red de que la etiqueta sea 1. La entropía cruzada penaliza más las predicciones incorrectas que están seguras (valores de probabilidad predicha cercanos a 0 o 1) que las predicciones incorrectas que están inseguras (valores de probabilidad predicha cercanos a 0.5).

2.5.4.3. Cálculo de Gradiente y Backpropagation

Una vez calculado el error que la red ha cometido, es necesario actualizar los parámetros para minimizar dicho error. Este proceso se realiza mediante el algoritmo de retropropagación, o *backpropagation* [26]. Está busqueda de parámetros se representa como:

$$\theta^* = \text{argmin}_\theta \mathcal{L}(\theta), \tag{2.39}$$

donde θ representa los parámetros de la red y $\mathcal{L}(\theta)$ es la función de pérdida. El objetivo es encontrar los valores de θ que minimicen $\mathcal{L}(\theta)$.

Antes de actualizar los pesos para reducir esta función de pérdidas, primero debemos entender cómo cada parámetro (tanto pesos, w, como sesgos, b) afecta a la función de pérdida. Para ello, se calcula el gradiente de la función de pérdida con respecto a los parámetros de la red.

Gradiente

El gradiente es una herramienta fundamental en cálculo, ya que nos indica la dirección y la magnitud del cambio más pronunciado de una función en un punto dado. Por ejemplo, para una función sencilla como $f(x) = x^2$, el gradiente en un punto x es simplemente $2x$.

En el caso de funciones más complejas, como las que encontramos en redes neuronales, necesitamos calcular la derivada parcial de la función para cada variable y punto de la red.

De manera matemática, el gradiente se define como:

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right) \tag{2.40}$$

De manera que cada componente del vector del gradiente nos indica la pendiente de la función en el mismo punto.

Sin embargo, cuando tenemos funciones muy complicadas, como es el caso de una red neuronal, es necesario el mecanismo de la regla de la cadena para calcular el gradiente.

Regla de La Cadena

La regla de la cadena es una herramienta fundamental en el cálculo diferencial que se utiliza para encontrar la derivada de una función compuesta. Si tenemos una función compuesta, es decir, una función que resulta de aplicar una función a otra, la regla de la cadena nos dice que la derivada de la función compuesta es el producto de las derivadas de las funciones individuales.

Por ejemplo, si tenemos una función y que depende de una variable u, y a su vez u depende de otra variable x, la regla de la cadena nos dice que la derivada de y con respecto a x es el producto de la derivada de y con respecto a u y la derivada de u con respecto a x. Matemáticamente, esto se expresa como:

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \tag{2.41}$$

Esto significa que para encontrar la tasa de cambio de y con respecto a x, primero encontramos cómo cambia y con respecto a u y luego cómo cambia u con respecto a x, y multiplicamos esos dos resultados.

En la Figura 2.11, vemos el grafo de computación para la función $L = g(a, b, c, f) = (a \cdot b + c)f$, el cual vamos a utilizar como un ejemplo sencillo para entender como funciona la regla de la cadena en el *backpropagation* (El ejemplo no hace referencia directa a las operaciones utilizadas en una red neuronal).

Para saber cuanto influye cada hoja del grafo, o entrada de la función, en la salida final es necesario calcular el gradiente. El cual, en este caso, se puede calcular de manera bastante sencilla por la naturaleza de la función, como se enseña a continuación:

$$L = (a \cdot b + c) \cdot f \tag{2.42}$$

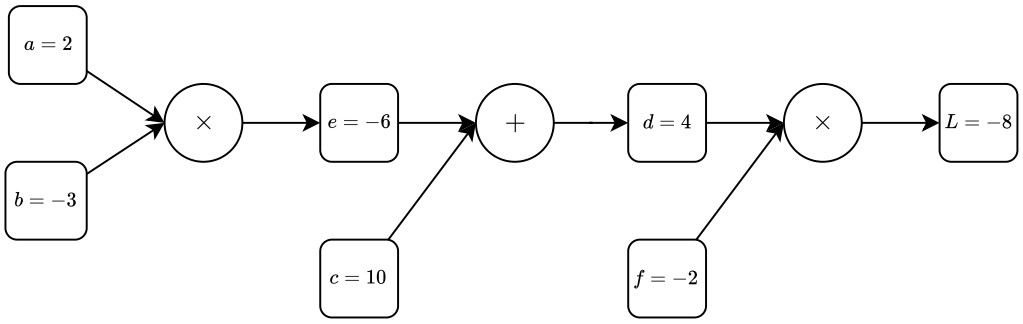


Figura 2.11: Grafo función g.

$$\frac{\partial L}{\partial a} = f \cdot b = 6. \tag{2.43}$$

Este gradiente nos indica que si modificamos el valor de a en una unidad, el resultado final de la función cambiará en 6, lo cual tiene lógica ya que la variable a está multiplicada primeramente por -3 y luego por -2.

Sin embargo, este proceso no siempre es tan sencillo de calcular, ya que en las redes neuronales el número de parámetros puede ascender al millón, por tanto la manera de hacerlo es usando la regla de la cadena. Y gracias a la visualización mediante grafos se hace mucho más sencillo comprenderla.

Comenzando en la salida L, su derivada parcial consigo misma es:

$$\frac{\partial L}{\partial L} = 1. \tag{2.44}$$

A continuación vamos hacia atrás (de ahí backpropagation) y calculamos la derivada parcial de L con respecto a f y d, que es:

$$\frac{\partial L}{\partial d} = \frac{\partial (d \cdot f)}{\partial d} = f, \tag{2.45}$$

$$\frac{\partial L}{\partial f} = \frac{\partial (d \cdot f)}{\partial f} = d. \tag{2.46}$$

De esto se concluye que, si modificamos el parámetro f en una unidad, el resultado final de la función cambiará en proporción a d, y viceversa. Esto revela una propiedad importante: cuando dos variables se combinan en un nodo de multiplicación, la derivada parcial de la salida con respecto a cada una de las variables es el producto del valor de la otra variable y el gradiente proveniente del nodo anterior.

Ahora continuamos hacia atrás, y calculamos la derivada parcial de L con respecto de e y c, que son:

$$\frac{\partial L}{\partial e} = \frac{\partial L}{\partial d} \cdot \frac{\partial d}{\partial e} = f \cdot \left(\frac{\partial (e + c)}{\partial e} \right) = f, \tag{2.47}$$

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial d} \cdot \frac{\partial d}{\partial c} = f \cdot \left(\frac{\partial (e + c)}{\partial c} \right) = f. \tag{2.48}$$

Si dos variables se juntan en un nodo de suma, la derivada parcial de la salida con respecto a las variables es uno, y por tanto, el gradiente de cada variable es el gradiente que viene del nodo anterior, en este caso $f = -2$.

Seguimos haciendo estos cálculos y al final obtenemos el grafo presentado en la Figura 2.12.

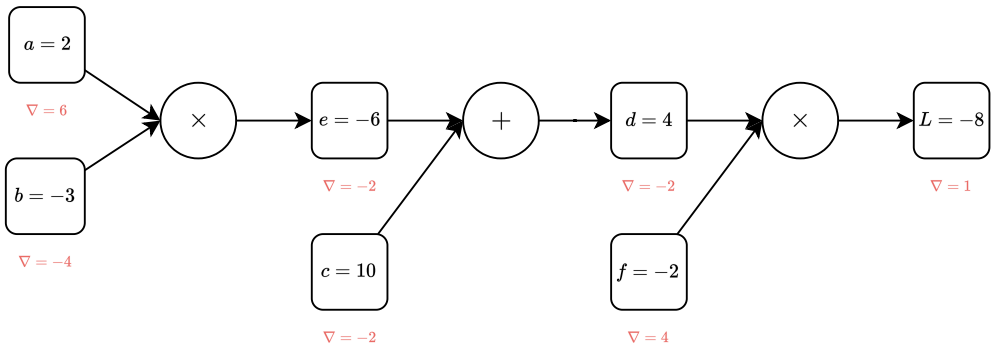


Figura 2.12: Grafo computación con gradientes.

Si nos fijamos en las hojas de nuestro grafo, que coinciden con las variables de la función, tendremos una idea de lo que cada una de ellas afecta al resultado.

Este es un ejemplo sencillo que explica la intuición detrás de la *backpropagation* que se utiliza en las redes neuronales. Sin embargo, recordar que grafo de computación de dichas redes suele muchísimo más complejo y su *backpropagation* resulta inabarcable para realizarlo de manera manual, teniendo en cuenta que pueden incluir funciones no lineales en las neuronas, lo cual incrementa la complejidad de las derivadas, ya que nos alejamos de simples multiplicaciones y sumas.

Backpropagation en Redes Neuronales

El gradiente de la función de pérdida de una red neuronal se puede expresar de la siguiente forma:

$$\nabla \mathcal{L} = \left[\frac{\partial \mathcal{L}}{\partial W^{(1)}} \quad \frac{\partial \mathcal{L}}{\partial b^{(1)}} \quad \cdots \quad \frac{\partial \mathcal{L}}{\partial W^{(L)}} \quad \frac{\partial \mathcal{L}}{\partial b^{(L)}} \right]^T. \tag{2.49}$$

Utilizando la regla de la cadena, Ecuación 2.31, obtenemos las siguientes expresiones para los gradientes con respecto a los pesos y los sesgos de la última capa:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W^{(L)}} &= \frac{\partial \mathcal{L}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial W^{(L)}} \\ \frac{\partial \mathcal{L}}{\partial b^{(L)}} &= \frac{\partial \mathcal{L}}{\partial a^{(L)}} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial z^{(L)}}{\partial b^{(L)}}, \end{aligned} \tag{2.50}$$

donde:

- $a^{(L)}$ es la salida de la última capa de la red.
- $z^{(L)}$ es la entrada a la última capa de la red (pre-activación).
- $\mathcal{L}$ es la función de pérdida.

Estas ecuaciones descomponen el gradiente en tres componentes: el cambio de la función de pérdida con respecto a la salida de la última capa, el cambio de la salida de la última capa con respecto a la entrada de la última capa, y el cambio de la entrada de la última capa con respecto al peso o sesgo correspondiente.

Este calculo se va haciendo de manera iterativa y retropropagando por la red hasta conseguir calcular los gradientes de todos los parámetros de nuestra red.

Actualización de Los Pesos

Una vez conocemos los gradientes de los parámetros, lo siguiente a realizar es modificar estos parámetros de manera que el error total que ha cometido con respecto a los datos de entrenamiento se reduzca. Para ello, se utiliza un algoritmo de optimización, como el descenso de gradiente, que actualiza los pesos en la dirección opuesta al gradiente, como se ve en la siguiente ecuación:

$$\theta \leftarrow \theta - \alpha \nabla \mathcal{L}. \tag{2.51}$$

La razón por la que se actualizan los pesos en el descenso del gradiente es para minimizar la función de pérdida. El gradiente indica la dirección en la que debemos movernos para aumentar la función de pérdida, por lo que restar el gradiente a los pesos nos asegura que nos movemos en la dirección correcta para alcanzar este objetivo.

Pongamos un ejemplo, si el gradiente de un peso es positivo, significa que la función de pérdida aumentará si aumentamos el peso. Por lo tanto, para reducir la función de pérdida, debemos disminuir el peso, lo que se logra restando el gradiente al peso. Si el gradiente es negativo, significa que la función de pérdida aumentará si disminuimos el peso, por lo que debemos aumentar el peso para reducir la función de pérdida.

El parámetro α se conoce como la tasa de aprendizaje y controla la magnitud de la actualización de los pesos. Un valor pequeño de α puede hacer que el entrenamiento sea más lento, pero puede ayudar a evitar oscilaciones y divergencias. Por otro lado, un valor grande de α puede hacer que el entrenamiento sea más rápido, pero puede hacer que el modelo sea inestable y diverja. También existe el riesgo de que el modelo no converja a un mínimo global, sino a un mínimo local, por tanto es importante encontrar un equilibrio en el valor de α.

2.6. Estado del Arte Aprendizaje Reforzado Profundo

En los últimos años, ha habido una tendencia creciente a incorporar técnicas de aprendizaje profundo dentro de las ya existentes de aprendizaje reforzado. Esto se debe principalmente al gran número de estados de los problemas a tratar y a la gran capacidad de las redes neuronales para representar funciones complejas y generalizar datos.

Uno de los estudios más importantes que allanaron el camino para este nuevo campo fue el presentado en [29]. En esta investigación, se introdujeron redes neuronales convolucionales (CNNs) para aproximar la función de valor de las tuplas estado-acción, Q. El artículo muestra cómo el agente, equipado con capacidad de visión gracias a las propiedades de las CNNs, y usando dicha CNN, es capaz de superar la destreza humana en la mayoría de los videojuegos de Atari 2600. Esto representaba un desafío significativo debido al problema de la explosión de estados presentado en la sección 2.4.3.

En [30], además de utilizar la red convolucional del estudio original de DQN, se introduce otra red neuronal con el objetivo de reducir la sobreestimación de las recompensas que realiza el algoritmo inicial. Esta segunda red se emplea en la etapa de toma de decisiones, eligiendo la acción con la mayor recompensa a largo plazo. Sin embargo, debemos de tener en cuneta que no se toma el valor directo que esta red proporciona. En su lugar, se estima la recompensa del estado elegido utilizando la red neuronal original.

Otra mejora significativa fue el desarrollo de las *Dueling-DQN* [31]. En esta *Dueling-DQN*, se implementa un nuevo término, la ventaja. Este término, se implementa dentro de la función valor de las tuplas estado-acción, Q. De manera que tenemos $Q(s,a) = V(s) + A(s,a)$. Por tanto se entiende la ventaja como la recompensa esperada a largo plazo que te aporta tomar la acción a parte del valor base de estar en el estado, $V(s)$. Esta técnica ha demostrado converger hacia mejores políticas gracias a esta descomposición. La idea es que no es necesario tomar acciones en todos los estados; a veces, no realizar ninguna es la mejor opción, y el término de la ventaja ayudaba a identificar estos casos.

La manera en la que los algoritmos similares a la *DQN* eran entrenados, era con un muestreo uniforme de las interacciones con el entorno. Lo que se estaba haciendo era crear una base de datos de las interacciones con ese entorno, guardando la recompensa que estas tenían, y entrenar la red neuronal que aproxima la función de valor de las tuplas estado-acción, $Q(s,a)$, con esa base de datos. Pero la manera en la que se elegían las muestras para entrenar era un muestreo uniforme. Sin embargo, en [32], se demuestra como un muestreo en función de la frecuencia en la que suceden las muestras hace que los algoritmos converjan a mejores aproximaciones.

Otra innovación importante en el campo de las DQN aborda el problema del espacio de acciones discreto. En Deep Deterministic Policy Gradient (DDPG) [33], se propone una versión de DQN con un enfoque actor-crítico, donde el actor puede tomar acciones en un espacio continuo, y el crítico evalúa el valor de cada tupla estado-acción. Esta aproximación permite a los agentes aprender tareas complejas como el balanceo del *cartpole swing-up* y la conducción autónoma. En un enfoque similar, el Asynchronous Advantage Actor-Critic (A3C) [34] logra reducir el costo computacional utilizando múltiples agentes entrenando en paralelo.

En los últimos años, los trabajos de *Trust Region Policy Optimization* (TRPO) [35] y *Proximal Policy Optimization* (PPO) [36] representaron un cambio de paradigma en el campo del DRL. En estos algoritmos, en lugar de implementar una red neuronal que aproxime la función del valor de los estados, $V(s)$, o de las tuplas estado-acción, $Q(s,a)$, la red neuronal es usada para aproximar la política, π, teniendo como base el algoritmo de los *Policy Gradients* 2.4.2 llamado REINFORCE 2.4.2.2.

En TRPO, se introduce un nuevo tipo de pérdida, explicada en [35], pero una idea básica es modificar la pérdida de la ecuación 2.26, de manera que se consiga una mayor estabilidad, para ello se implementa una técnica que mide las distancias entre las diferentes distribuciones de probabilidad

de la política anterior y la nueva. Este enfoque demostró un rendimiento robusto tanto en espacios de acciones continuos como discretos, siendo efectivo en tareas como juegos de Atari y control continuo en robótica simulada (natación, salto, etc.). A pesar de ser un método *on-policy* que puede sufrir colapsos de rendimiento durante el entrenamiento, TRPO garantiza adaptarse rápido a las nuevas tareas con un ajuste mínimo de hiperparámetros.

PPO, por otro lado, mejora el cálculo de las restricciones de la función de pérdida. En TRPO, para calcular que la distancia de las distribuciones entre las diferentes políticas es necesario aproximaciones de segundo orden, lo cual es bastante costoso, mientras que en PPO se toma una aproximación de primer orden. PPO ha demostrado un rendimiento superior en tareas de locomoción robótica simulada y juegos de Atari, ofreciendo un equilibrio favorable entre complejidad de muestra, simplicidad y tiempo de entrenamiento.

No solo existen demostraciones de como estos algoritmos funcionan bien en tareas como juegos Atari o de control básicas como *cart pole*. Sino que también se demuestran una destreza increíble en juegos como el Go.

En resumen, la combinación de técnicas de aprendizaje profundo y refuerzo ha revolucionado el campo del aprendizaje automático, permitiendo a los agentes superar retos complejos en diversos dominios. Los avances en DQN y sus variantes, como Double DQN y Dueling DQN, han mejorado significativamente la estabilidad y precisión en la toma de decisiones. Innovaciones como el muestreo prioritario, DDPG y A3C han ampliado las capacidades de los algoritmos a espacios de acción continuos y escenarios de alta demanda computacional. Por último, enfoques recientes como TRPO y PPO han optimizado la eficiencia y robustez del entrenamiento, consolidando su aplicación en robótica y juegos de alta complejidad.

Capítulo 3

Giulia

3.1. Naturaleza del Simulador

Para desarrollar un agente de aprendizaje reforzado capaz de ubicar dinámicamente el UAV, es fundamental contar con un entorno capaz de simular las redes de comunicación. Con este propósito, se ha utilizado un simulador denominado Giulia, desarrollado en Python, que simula con precisión la complejidad de los entornos reales.

Existen diferentes tipos de simuladores: a nivel de enlace y a nivel de sistema. Los simuladores a nivel de enlace se enfocan en la conexión punto a punto, proporcionando un análisis detallado, bit a bit, de las pérdidas por efectos de canal, como la propagación, la interferencia y el multicamino. Por otro lado, los simuladores a nivel de sistema se centran en la simulación de la red en su totalidad, con la incorporación de múltiples UEs y estaciones base. Estos simuladores utilizan modelos estandarizados que representan las características de la red, incluyendo modelos de despliegue de red y de UEs, operación de las distintas capas de protocolos y el canal, que capturan los efectos del *slow* y el *fast fading*.

La simulación a nivel de sistema es crucial en la evaluación del rendimiento de redes celulares complejas. Este enfoque permite modelar los elementos y operaciones de una red celular a través de software, proporcionando una solución más precisa y económica en comparación con las implementaciones reales.

El modelado de la ganancia de antena y la pérdida por distancia es fundamental para evaluar el rendimiento de la red. Las ganancias de antena, medidas en decibelios isotrópicos (dBi), y la pérdida por distancia, modelada mediante enfoques deterministas y estadísticos como los modelos de Okumura-Hata y del *3rd Generation Partnership Project* (3GPP), son elementos clave en esta evaluación. Además, el *shadow fading* y el desvanecimiento multi-trayectoria afectan significativamente la calidad de la señal recibida. Estos fenómenos se modelan como variables log-normales y mediante *Power Delay Profiles* (PDP), respectivamente.

Para evaluar la calidad y la fuerza de la señal recibida, se emplean modelos de SINR. Para evaluar la calidad de servicio, se utilizan modelos de tasa de transmisión, basados en el teorema de Shannon-Hartley-

Asimismo, los modelos de tráfico de UE y los modelos de movilidad, ayudan a evaluar el rendimiento de la red bajo diferentes condiciones de demanda y movimiento de los UEs.

En resumen, el simulador Giulia incorpora una amplia variedad de modelos para representar de manera precisa las condiciones de operación de una red celular, lo cual es esencial para el desarrollo y la optimización de tecnologías de redes celulares avanzadas.

3.2. Cálculo de La Tasa de Transmisión

Para poder calcular la tasa de transmisión de la que dispone cada UE, es necesario previamente calcular todas las ganancias relacionadas con el canal entre el UE y la estación base. En primer lugar, se calcula la ganancia de antena, G^{a}, que depende de la orientación de la antena. A continuación, se calcula la pérdida por distancia, G^{p}, que depende de la distancia entre el UE y la estación base. La ganancia de penetración *outdoor to indoor*, G^{e}, es debida a los obstáculos que tiene que atravesar la señal. Además, se tienen en cuenta el *shadow fading*, G^{s}, y el desvanecimiento multi-camino G^{ff}, que afectan a la calidad de la señal recibida y dependen en gran medida de la presencia de obstáculos en el entorno. Una vez calculadas todas estas ganancias, se puede determinar la tasa de transmisión de cada UE en función de la calidad de la señal recibida.

3.2.1. Ganancia de Antena

La antena típica en un entorno de simulación esta conformada por un *array* vertical de antenas. El cual tiene los siguientes diagramas de radiación vertical 3.2 y horizontal 3.1 respectivamente:

$$G^{\mathrm{a}}_{\mathrm{H}}(\varphi)[\mathrm{dB}] = -\,\mathrm{mín}\left[12\left(\frac{\varphi}{\varphi_{3\mathrm{dB}}}\right)^{2}, \kappa\right], \tag{3.1}$$

y

$$G^{\mathrm{a}}_{\mathrm{V}}(\theta)[\mathrm{dB}] = 20\log_{10}\left|\frac{\sin\left(\mathrm{K}\left(\theta-\theta_{\mathrm{etilt}}\right)\right)}{\mathrm{K}\left(\theta-\theta_{\mathrm{etilt}}\right)}\right|, \tag{3.2}$$

donde:

- φ y θ son los ángulos de llegada, medidos en grados, en los planos horizontal y vertical respecto a la dirección del haz principal.
- $G^{\mathrm{a}}_{\mathrm{H}}(\varphi)$ y $G^{\mathrm{a}}_{\mathrm{V}}(\theta)$ son las atenuaciones, medidas en dB, introducidas por los patrones de antena horizontal y vertical con respecto a la ganancia máxima de la antena en la dirección de φ y θ, respectivamente.
- $\varphi_{3\mathrm{dB}}$ y $\theta_{3\mathrm{dB}}$ son los anchos del haz principal, medidos en grados, de los patrones de antena horizontal y vertical dentro de los cuales se transmite la mitad de la potencia máxima.
- κ es la relación delante-detrás, medida en dB, en el plano horizontal.
- θ_{etilt} es el ángulo, medido en grados, entre la dirección del haz principal y el horizonte en el plano vertical (también conocido como la inclinación eléctrica de la antena).
- $K = \frac{164}{\theta_{3\mathrm{dB}}}$.

Según [37], para optimizar el tilt θ_{etilt}, este se puede calcular con respecto a la altura efectiva de la antena transmisora, $h_{\text{T,eff}}$, y la distancia entre la estación base y el límite de la celda, d_c, como sigue (ver Figura 3.1):

$$\theta_{\text{etilt}} = \arctan\left(\frac{h_{\text{T,eff}}}{d_c}\right) + z \cdot \theta_{\text{3dB}}, \tag{3.3}$$

donde z fue determinado empíricamente para un despliegue celular regular, y se estableció en 0.7 para lograr un buen equilibrio entre la potencia recibida y la interferencia entre celdas, donde $h_{\text{T,eff}}$ y d_c están expresados en metros.

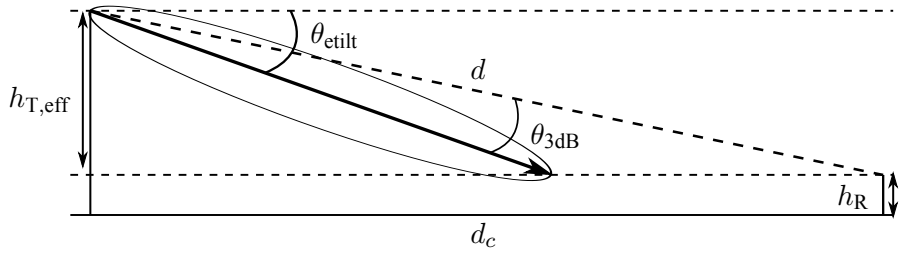


Figura 3.1: Inclinación de una estación base [38].

Para considerar el efecto conjunto de los patrones de antena horizontal y vertical y crear un patrón de antena 3-D, se utiliza el siguiente modelo:

$$G^{\text{a}}(\varphi,\theta)[\text{dB}] = G^{\text{a}}_{\text{M}}[\text{dBi}] + G^{\text{a}}_{\text{H}}(\varphi)[\text{dB}] + G^{\text{a}}_{\text{V}}(\theta)[\text{dB}], \tag{3.4}$$

donde G^{a}_{M} es la ganancia máxima de la antena en dBi. La Tabla 3.1 presenta valores típicos de este modelo de antena.

Parámetro	Valor
	3 sectores
G^{a}_{M}	15.5 dBi
φ_{3dB}	65 deg
θ_{3dB}	11.5 deg
κ	30 dB

Tabla 3.1: Parámetros típicos de una antena sectorial.

Los patrones de la antena, tanto vertical como horizontal, se observan en la Figura 3.2.

3.2.2. Modelado de La Pérdida por Distancia

Un importante factor en la ganancia total del canal entre un transmisor y un receptor es la pérdida por distancia. Esta pérdida se debe a la atenuación de la señal a medida que se propaga a través del espacio. La pérdida por distancia se puede modelar de diferentes maneras, dependiendo de las características del entorno y de la frecuencia de operación. En general, la pérdida por distancia se puede modelar como una función logarítmica de la distancia entre el transmisor y el receptor. En situación de visión directa, la pérdida por distancia se puede modelar como:

$$G^{p,fs}[\text{dB}] = -20\log_{10}(d) - 20\log_{10}(f_c) + 147{,}55, \tag{3.5}$$

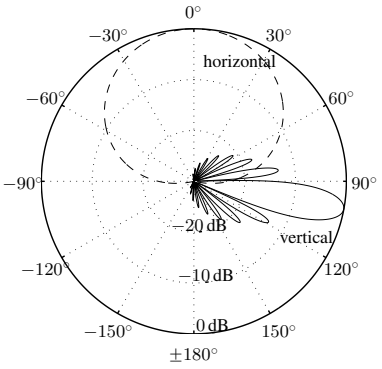


Figura 3.2: Diagrama de radiación de la antena [38].

donde d es la distancia entre el transmisor y el receptor en metros, y f_c es la frecuencia de operación en Hz. Sin embargo, esta pérdida por distancia no se mantiene para todos los entornos debido a que asume condiciones ideales, como la ausencia de obstáculos y la propagación en línea de visión. Por lo tanto, es necesario considerar otros factores, como la presencia de obstáculos, vegetación y edificios, que pueden afectar la propagación de la señal y aumentar la pérdida por distancia.

Para aumentar la precisión del cálculo de la pérdida por distancia a escenarios más realistas, modelados deterministas o estadísticos han de ser utilizados [39]. Los modelos deterministas están basados en trazado de rayos, y por tanto resultan muy precisos. Pero tienen un contrapunto, además de requerir un alto coste computacional, necesitan una descripción detallada del entorno, lo que puede ser complicado de obtener. Por otro lado, los modelos estadísticos son más sencillos de implementar y están basados en medias de pérdida por distancia en entornos típicos (rural, urbano, etc.).

Un modelo estadístico ampliamente utilizado es el modelo de Okumura-Hata [40], el cual fue construido a partir de mediciones recolectadas en la ciudad de Tokio y mejorado mediante mediciones en otras ciudades. Sin embargo, el modelo de Okumura-Hata tiene algunas limitaciones intrínsecas. Por ejemplo, no considera el perfil del terreno entre el transmisor y el receptor, ya que asume que los transmisores están ubicados en colinas, y solo proporciona soporte hasta 1.9 GHz.

Para superar tales limitaciones y adaptarse a todo tipo de condiciones, se han desarrollado una gran cantidad de modelos estadísticos basados en diferentes campañas de medición. En nuestro caso hemos utilizado el modelo 3GPP TR 36.814 caso 1. Cuya función de pérdida por distancia está modelada en la siguiente ecuación:

$$G^{p,3GPPTR36,814}[\text{dB}] = -128{,}1 - 37{,}6 \log_{10}(R)\,, \tag{3.6}$$

donde R es la distancia entre el transmisor y el receptor en kilómetros.

Se pueden observar las diferencias entre el modelo ideal y el modelo 3GPP TR 36.814 en la Figura 3.3. El modelo 3GPP TR 36.814 presenta una mayor pérdida por distancia en comparación con el modelo ideal. Esto se debe a que el modelo 3GPP TR 36.814 tiene en cuenta la presencia de obstáculos y la propagación no ideal de la señal.

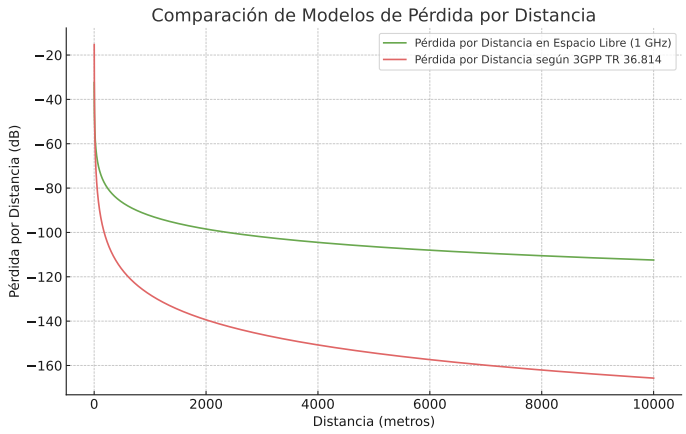


Figura 3.3: Comparación entre el modelo ideal y el modelo 3GPP TR 36.814 de pérdida por distancia.

3.2.3. Modelado de Shadowing

El *shadowing* modela las fluctuaciones aleatorias de la potencia de señal recibida en el lado del receptor, causadas por las obstrucciones de objetos que una señal de radio experimenta en su trayectoria de propagación. Las ubicaciones, tamaños y propiedades dieléctricas de los objetos obstructores, así como las superficies reflectantes y los obstáculos dispersores, generalmente son desconocidos. Debido a tales incertidumbres, se utilizan modelos estadísticos para modelar el *shadowing*. Un modelo ampliamente utilizado es el modelo de *shadowing* log-normal [41], que ha demostrado ser capaz de modelar las atenuaciones del *shadowing* con buena precisión en entornos exteriores e interiores.

En consecuencia, el *shadowing* en la trayectoria entre un transmisor y un receptor se puede modelar a priori utilizando una variable aleatoria log-normal, $G^{\mathrm{s}} \sim \mathcal{N}(\mu_{\mathrm{s}}, \sigma_{\mathrm{s}}^{\in})$, donde μ_{s} y σ_{s} son la media y la desviación estándar de la variable aleatoria log-normal en dB, respectivamente. Sin embargo, el modelado del *shadowing* cuando se consideran múltiples transmisores (por ejemplo, estaciones base) y receptores (por ejemplo, UEs) es más complicado debido a las propiedades de auto-correlación espacial y correlación cruzada del *shadowing*.

La auto-correlación espacial se debe a que un UE en movimiento puede experimentar atenuaciones de *shadow fading* similares de una estación base dada, mientras que estas atenuaciones pueden diferir significativamente en posiciones alejadas.

La correlación cruzada, por otro lado, se debe a que múltiples enlaces de diferentes estaciones base a un mismo UE pueden observar un *shadow fading* altamente correlado, dado que el entorno del UE es común. Por ejemplo, si dos estaciones base están muy cerca una de la otra, los dos caminos entre las estaciones base y un UE pueden atravesar un entorno similar, y por tanto el *shadow fading* será similar en los enlaces.

3.2.4. Ganancia de Penetración

La ganancia de penetración modela la atenuación que sufre una señal de radio al atravesar un obstáculo, como edificios o paredes. Esta pérdida depende de la frecuencia de la señal y de las propiedades del material (permisividad, permeabilidad y conductividad). Para evaluar el rendimiento de un receptor dentro de un edificio esta perdida es fundamental. Los modelos estadísticos de pérdida por distancia solo capturan el efecto promedio de los obstáculos, y asumen que el receptor esta fuera de edificios.

Una forma de modelar estos efectos es usar herramientas deterministas como el trazado de rayos, Sin embargo, estas herramientas son computacionalmente complejas y pueden requerir mucho tiempo de simulación.

Como compromiso, se puede usar un modelo que asocie una ganancia de penetración, G^{e}, a cada UE dentro de un edificio y agregarlo a la pérdida por distancia. La ganancia de penetración total se calcula sumando las pérdidas individuales de cada pared entre el transmisor y el receptor. Si ambos, transmisor y receptor, están en interiores pero en edificios diferentes, la ganancia de penetración es $2 \times G^{\text{e}}$; de lo contrario, es 0 dB. Según [42], la ganancia de penetración promedio es $G^{\text{e}} = -20$ dB.

3.2.5. Modelado del Desvanecimiento Multicamino

Los objetos que interfieren en la trayectoria de propagación desde el transmisor al receptor también pueden producir copias reflejadas, difractadas y dispersas de la señal, resultando en componentes de múltiples trayectorias (MPC, del inglés *Multi Path Component*). Los MPC pueden llegar al receptor atenuados en potencia, retrasados en el tiempo y desplazados en frecuencia (y/o fase) con respecto al primer y más fuerte MPC, sumándose así de manera constructiva o destructiva. Como consecuencia, la potencia recibida en el receptor puede variar significativamente sobre distancias muy pequeñas del orden de unas pocas longitudes de onda [41].

Dado que diferentes MPC viajan por diferentes caminos de distintas longitudes, un único impulso, enviado desde un transmisor, y que sufre de múltiples trayectorias, resultará en múltiples copias de dicho impulso único, siendo recibidas en el receptor en diferentes momentos. Como resultado, la respuesta al impulso del canal de múltiples trayectorias se puede modelar usando una línea de retardo con *taps* modelados de la siguiente forma:

$$h(\tau, t) = \sum_{i=0}^{\nu-1} h_i(t) \cdot \delta(\tau - \tau_i), \tag{3.7}$$

donde ν es el número de taps o MPC resolubles, y $h_i(t)$ y τ_i son la ganancia compleja del canal y el retardo del tap i, respectivamente.

Los PDPs se utilizan a menudo para modelar estas líneas de retardo con taps. Un perfil de retardo de potencia se caracteriza por el número de taps, ν, el retardo temporal relativo al primer tap, la potencia promedio relativa al tap más fuerte y el espectro Doppler de cada tap. Los PDPs más frecuentemente utilizados son los especificados por ITU 3.2,

Es importante señalar que las pérdidas de las muestras con retardo suelen ser relativamente pequeñas, pero ocasionalmente y dependiendo del escenario, pueden ser significativos. Estos retardos se pueden observar en la Tabla 3.2.

	Retardo (ns)	Potencia Relativa (dB)	Retardo (ns)	Potencia Relativa (dB)
Peatón ($\leq$3 km/h)				
Tap	Canal A (40 %)		{Canal B} (55 %)	
1	0	0	0	0
2	110	-9.7	200	-0.9
3	190	-19.2	800	-4.9
4	410	-22.8	1200	-8.0
5	-	-	2300	-7.8
6	-	-	3700	-23.9

Tabla 3.2: PDPs para UEs peatones (Pedestrian, $\leq$3 km/h).

Sin embargo, los cálculos de todas estos *taps* pueden ser realmente costosos para los ordenadores actuales, ya que hay que calcularlos por cada señal se envía en el sistema. Por tanto, se usan matrices precalculadas que no tienen ningún significado geográfico, simplemente un comportamiento general de como estas reflexiones multicamino se comportan. En la Figura 3.4, se puede ver un ejemplo de dichos modelos precalculados.

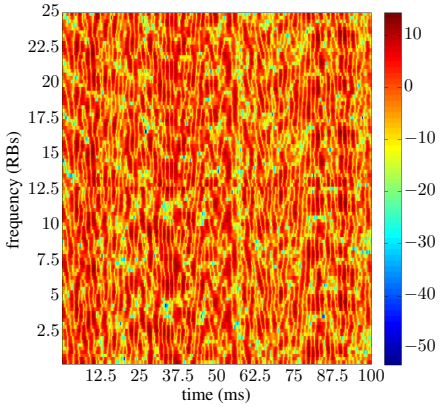


Figura 3.4: Mapa desvanecimiento multicamino [38].

3.2.6. Modelado de Potencia de Señal Recibida

Habiendo definido la ganancia de la antena, G^{a}, la pérdida por distancia, G^{p}, el desvanecimiento por *shadowing*, G^{s}, la ganancia de penetración, G^{e}, y el desvanecimiento por múltiples trayectorias, G_{ff} en las secciones anteriores, la potencia de la señal recibida, $P^{\mathrm{rx}}_{t,r,k}$, de la señal transmitida por un transmisor, T_t, en un receptor, R_r, en un recurso de frecuencia, K_k, se puede calcular como:

$$P^{\mathrm{rx}}_{t,r,k} = P^{\mathrm{tx}}_{t,k} \cdot G^{\mathrm{a}}_{t,r} \cdot G^{\mathrm{p}}_{t,r} \cdot G^{\mathrm{e}}_{t,r} \cdot G^{\mathrm{s}}_{t,r} \cdot \left| G^{\mathrm{ff}}_{t,r,k} \right|^2, \tag{3.8}$$

donde:

- $P^{\mathrm{tx}}_{t,k}$ es la potencia de transmisión aplicada por el transmisor T_t en el recurso de frecuencia K_k,
- $G^{\mathrm{a}}_{t,r}$ es la ganancia de la antena desde el transmisor T_t hasta el receptor R_r,

- $G^{\mathrm{p}}_{t,r}$ es la pérdida por distancia desde el transmisor T_t hasta el receptor R_r,
- $G^{\mathrm{e}}_{t,r}$ es la ganancia de penetración desde el transmisor T_t hasta el receptor R_r,
- $G^{\mathrm{s}}_{t,r}$ es el desvanecimiento por *shadowing* desde el transmisor T_t hasta el receptor R_r y
- $\left|G^{\mathrm{ff}}_{t,r,k}\right|^2$ es el desvanecimiento por múltiples trayectorias desde el transmisor T_t hasta el receptor R_r en el recurso de frecuencia K_k

(nótese que $G^{\mathrm{ff}}_{t,r,k}$ se definió como una ganancia compleja). Todas estas variables están expresadas en unidades lineales.

3.2.7. Modelado de La Calidad de Señal

Habiendo definido la potencia de la señal recibida , $P^{\mathrm{rx}}_{t,r,k}$, de la señal transmitida por un transmisor, T_t, a un receptor, R_r, y un recurso de frecuencia, K_k. Se define la SINR, $\gamma_{t,r,k}$, de esta señal ,en un caso de entrada única y salida única, se puede modelar como ya se introdujo en 1.9, sin embargo aquí se utiliza notación de transmisor, tx, y receptor, rx:

$$\gamma_{t,r,k} = \frac{P^{\mathrm{rx}}_{t,r,k}}{\sum_{\substack{t\prime=0 \\ t\prime\neq t}}^{T} P^{\mathrm{rx}}_{t\prime,r,k} + \sigma^2}. \tag{3.9}$$

Para ilustrar, la Figura 3.5 muestra los mapas de SINR para una disposición de macroceldas hexagonales simulada en la ciudad de Dublín, Irlanda, donde los mapas se han calculado como se indica en la Ecuación 3.9.

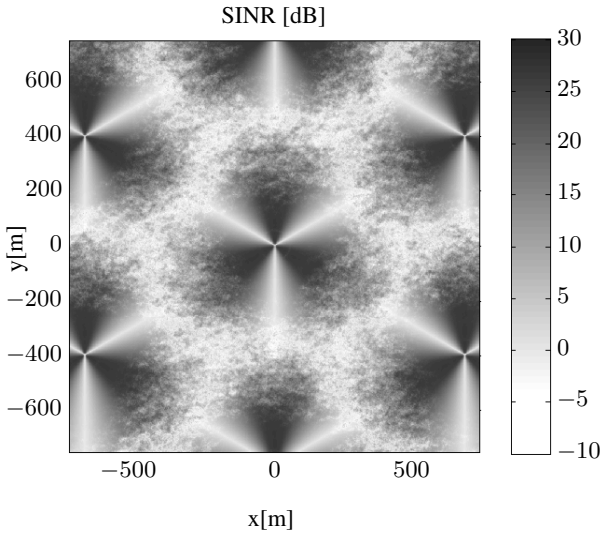


Figura 3.5: SINR ciudad de Dublín [38].

3.2.8. Calculo de La Tasa de Transmisión

El teorema de Shannon-Hartley [43] indica que la capacidad máxima de un canal de comunicación se puede calcular según:

$$C_{r,k,t} = B \log_2(1 + \gamma_{t,r,k}), \tag{3.10}$$

donde, si utilizamos la capacidad máxima del canal, tenemos que la tasa de transmisión en la comunicación entre el transmisor y receptor es $R_{r,k,t} = C_{r,k,t}$.

3.3. Creación de Un Entorno

Para trabajar de manera efectiva con Giulia, es fundamental asegurar que todas las simulaciones se realicen en un entorno uniforme. Esto garantiza que, independientemente del sistema utilizado ya sea un ordenador local, un clúster de ordenadores, o si otra persona accede al simulador, se mantengan las mismas versiones de todos los componentes. Esto es crítico en Giulia debido a la naturaleza estocástica de muchos de sus cálculos.

La estocasticidad en programación es un aspecto delicado, ya que las versiones de los paquetes utilizados pueden influir en las semillas que generan los números aleatorios. Por lo tanto, es vital mantener las mismas versiones de los paquetes, así como los drivers instalados en el sistema.

Para resolver esta problemática, se ha optado por utilizar Docker.

Docker es una herramienta poderosa que permite la creación de contenedores que encapsulan todas las dependencias necesarias, asegurando un entorno replicable y consistente. Para configurar este entorno utilizando Docker, se debe crear un archivo `Dockerfile` que especifique las versiones exactas de los paquetes, librerías y drivers necesarios.

El `Dockerfile` actúa como una receta detallada para construir la imagen del contenedor. Una vez creado este archivo, se procede a construir la imagen, un proceso que genera una versión fija del entorno de trabajo. Esta imagen puede ser almacenada y reutilizada tantas veces como sea necesario. La principal ventaja de esto es que una vez que se ha creado la imagen, se pueden generar múltiples contenedores a partir de ella. Si se realiza un cambio en un contenedor que resulta en un fallo del código, simplemente se elimina ese contenedor y se crea uno nuevo desde la imagen original, garantizando que el entorno inicial se mantenga intacto.

El uso de Docker para gestionar el entorno de trabajo en proyectos de simulación, como los realizados con Giulia, ofrece múltiples ventajas:

- **Consistencia del Entorno**: Docker asegura que todos los desarrolladores y usuarios del proyecto trabajen con el mismo entorno, evitando problemas derivados de las diferencias en configuraciones locales.
- **Reproducibilidad**: Gracias a la naturaleza de las imágenes Docker, se puede garantizar que las simulaciones y resultados sean reproducibles en cualquier momento, incluso en diferentes máquinas.
- **Portabilidad**: Los contenedores Docker pueden ejecutarse en cualquier lugar donde Docker esté instalado, ya sea en una computadora personal, un servidor o en la nube. Esto facilita la transición entre diferentes entornos de desarrollo, prueba y producción.

- **Aislamiento**: Cada contenedor opera de manera independiente, aislado del sistema anfitrión y de otros contenedores. Esto minimiza conflictos de dependencias y asegura un entorno limpio para cada instancia del proyecto.
- **Facilidad de Mantenimiento**: Actualizar o modificar el entorno es tan sencillo como editar el `Dockerfile` y reconstruir la imagen. Esto permite una gestión centralizada y controlada de las dependencias y configuraciones del proyecto.

En resumen, utilizar Docker para gestionar el entorno de trabajo con Giulia proporciona una solución robusta y eficiente para mantener la consistencia, portabilidad y reproducibilidad de los resultados de las simulaciones.

3.4. Optimizaciones sobre Giulia

Dentro del RL, la rapidez de nuestro entorno es crítica a la hora de entrenar un agente. Para que estos agentes converjan y obtengan buenos resultados, es necesario realizar del orden de miles de iteraciones. El simulador, Giulia, debe ser por tanto extremadamente eficiente. Sin embargo, la versión inicial de Giulia no habia sido optimizada y permitía mejoras, 100 iteraciones podían consumir 185.18 segundos.

Para poder comparar, a pesar de que Giulia tiene implementados un gran número de modelos estándar de comunicaciones inalámbricas del 3GPP, utilizaremos el modelo ITU-R M.2135-1 [44]. Este modelo es usado como referencia ya que no es el modelo más complejo, pero tampoco el que menos cálculos requiere, permitiendo evaluar cómo se comportan las optimizaciones en un modelo de complejidad media. En concreto, este modelo tiene 570 UEs y 57 estaciones base de tecnología 4G.

Además, para las comparaciones utilizaremos el tiempo en el que tardan en hacerse 100 ejecuciones. Una simple ejecución implica que el modelo realice los cálculos explicados en la sección 3.2, con toda la complejidad que ello conlleva, incluyendo una gran cantidad de UEs, estaciones base, matrices de números complejos aleatorios de gran tamaño, etc.

Las optimizaciones realizadas siguen tres principales vertientes:

- ***Plotting* ineficiente:** El código original incluía una gran cantidad de *plotting*, lo cual ralentizaba la ejecución. Se eliminó todo el *plotting* innecesario para poder comparar la velocidad de ejecución de los modelos, ya que este *plotting* solo es necesario cuando se visualizan los resultados uno a uno, no cuando se realizan múltiples ejecuciones.
- **Funciones vectorizadas:** En Python, es más eficiente utilizar funciones vectorizadas en lugar de aplicar funciones a cada elemento individualmente. Es decir, en lugar de recorrer todos los elementos de un vector en un bucle y aplicar una función a cada uno, es más eficiente aplicar la función directamente al vector. Esto se debe a que las funciones vectorizadas están implementadas en C, lo que las hace mucho más rápidas que un bucle en Python. Aprovechamos también el *broadcasting* y el *indexing* [45].
- **Utilización de GPU:** Para la implementación en GPU se utilizó la librería de Python, *Pytorch* [46]. Pytorch es una librería de código abierto desarrollada por Meta que facilita la

realización de cálculos en paralelo en la GPU. La GPU es una unidad de procesamiento que se utiliza para acelerar los cálculos, especialmente en tareas que requieren una gran cantidad de cálculos matriciales.

Gracias a todas estas optimizaciones, se logró reducir el tiempo de ejecución por cada 100 iteraciones de 185.18 segundos a 22.41 segundos, lo que representa una reducción del 87.9 % en el tiempo de ejecución.

3.4.1. Plotting Ineficiente

Como se explicó previamente, el *plotting* es una operación que consume mucho tiempo en entornos de programación. Si se realizan 100 iteraciones, después de cada iteración es necesario detener los cálculos, recopilar los resultados y mostrarlos en pantalla, lo cual consume tiempo, especialmente cuando se necesitan iteraciones lo más rápidas posibles. Por tanto, una primera estrategia de optimización fue eliminar los *plots* innecesarios, dejando solo el resultado final para comparar la velocidad de ejecución de los modelos.

Solo con esta modificación el tiempo de ejecución se redujo de 185.18 segundos a 108.017 segundos en el modelo que estamos comparando. En la Tabla 3.3, se puede observar cual es la eficiencia de nuestro modelo, ITU-R M.2135-1, así como de otros implementados en Giulia, tras dicha modificación.

Modelo	Tiempo Total
3GPP TR 38.901 UMa lsc	719.801
ITU-R M.2135 UMa Umi colocated multilayer	246.922
ITU-R M.2135 UMa multilayer	214.340
3GPP TR 36.777 UMi AV	129.000
3GPP TR 36.777 UMa AV	122.614
3GPP TR 38.901 UMi lsc	118.566
ITU-R M.2135 UMa	108.017
ITU-R M.2135 UMi	96.860
3GPP TR 36.814 Case-1	38.608
3GPP TR 36.814 Case-1.omni	22.520
3GPP TR 38.901 UMa lsc single bs	9.936
3GPP TR 36.814 Case-1 single bs	5.585

Tabla 3.3: Resultados después de *plotting* ineficiente.

3.4.2. Funciones Vectorizadas

Un claro ejemplo de cómo los bucles de Python pueden crecer en complejidad y llegar a no ser eficientes se muestra en la Figura 3.6.

En este cálculo se están determinando las distancias entre los UEs y las estaciones base, así como los ángulos que forman, para calcular posteriormente la ganancia de la antena según el diagrama de radiación, por tanto se recorren cada una de las posiciones de la matriz.

```
index_3d_min = np.argmin(distance_3d_partial_results, axis=0)
distance_3d_results = np.array([[distance_3d_partial_results[index_3d_min[i,j],i,j] for
↪  j in range(0, np.size(index_3d_min, 1))] for i in range(0, np.size(index_3d_min,
↪  0))])

index_2d_min = np.argmin(distance_2d_partial_results, axis=0)
distance_2d_results = np.array([[distance_2d_partial_results[index_2d_min[i,j],i,j] for
↪  j in range(0, np.size(index_2d_min, 1))] for i in range(0, np.size(index_2d_min,
↪  0))])

azimuth_results = np.array([[azimuth_partial_results[index_3d_min[i,j],i,j] for j in
↪  range(0, np.size(index_3d_min, 1))] for i in range(0, np.size(index_3d_min, 0))])
zenith_results = np.array([[zenith_partial_results[index_3d_min[i,j],i,j] for j in
↪  range(0, np.size(index_3d_min, 1))] for i in range(0, np.size(index_3d_min, 0))])
```

Figura 3.6: Código ejemplo bucles for en *list comprehension*.

Para mejorar esta implementación se utilizó un estilo de funciones vectorizadas y el indexamiento directo de vectores, resultando en el código visto en la Figura 3.7.

Las funciones vectorizadas permiten realizar operaciones sobre arrays enteros de manera eficiente, utilizando optimizaciones de bajo nivel que aprovechan el hardware subyacente. Por ejemplo, NumPy proporciona funciones que realizan operaciones como suma, multiplicación y funciones matemáticas avanzadas directamente sobre arrays.

El indexamiento directo de vectores permite acceder y manipular elementos específicos de un array mediante el uso de índices, que pueden ser enteros, slices o booleanos. Esto resulta en una mayor eficiencia en comparación con los bucles for tradicionales. En un bucle for, el intérprete de Python tiene que gestionar la iteración y acceder a cada elemento uno por uno, lo cual puede ser lento debido a la sobrecarga de la gestión del bucle y la interpretación de cada instrucción. En cambio, el indexamiento directo aprovecha la implementación optimizada en C de NumPy, permitiendo operaciones masivas sobre arrays sin la necesidad de bucles explícitos, reduciendo así la sobrecarga y acelerando el procesamiento.

```
index_3d_min = np.argmin(distance_3d_partial_results, axis=0)
index_2d_min = np.argmin(distance_2d_partial_results, axis=0)

# Creating arrays of indices for gathering the results efficiently
i_indices, j_indices = np.indices(index_3d_min.shape)

# Efficiently gather the 3D distance results
distance_3d_results = distance_3d_partial_results[index_3d_min, i_indices, j_indices]

# Efficiently gather the 2D distance results using the index_2d_min
distance_2d_results = distance_2d_partial_results[index_2d_min, i_indices, j_indices]

# Efficiently gather the azimuth and zenith results
azimuth_results = azimuth_partial_results[index_3d_min, i_indices, j_indices]
zenith_results = zenith_partial_results[index_3d_min, i_indices, j_indices]
```

Figura 3.7: Mejora por indexamiento directo de vectores.

Esto permitió una implementación más directa del código sin tener que iterar a través de todas las posiciones de las matrices, siendo mucho más eficiente.

Estos cambios también se aplicaron a la forma en que la posición de los UEs se implementa en el simulador, estandarizada en [44]. El código inicialmente utilizado se puede ver en la Figura 3.8.

Es un conjunto de operaciones denso en el que hay que manejar bien las dimensiones. Por cada iteración del bucle se están haciendo multiplicaciones de matrices. En las que además, dependiendo del modelo de entorno que se esté utilizando las matrices varían de tamaño. Así que se aprovechó el comando de Numpy llamado einsum [47], que permite hacer una gran cantidad de operaciones siguiendo la notación de Einstein para operaciones matriciales [48]. Estas modificaciones se pueden ver en la Figura 3.9.

De esta manera se consigue evitar los bucles for, y realizar un aplicación directa de funciones vectorizadas, además de la gran utilidad que tiene el comando Einsum, con el cual hemos realizado multiplicaciones de matrices con tamaños inconsistentes.

3.4.3. Implementación GPU

Como se mencionó anteriormente, la GPU es fundamental para acelerar las operaciones matriciales. En el simulador, se maneja una gran cantidad de matrices debido al elevado número de UEs y estaciones base, así como las conexiones entre ellos.

En la Figura 3.10, se muestra cómo la variable aleatoria que forma parte de la computación del *fast fading*, que refleja pequeños cambios en la señal debido a fenómenos como el multicamino, se calculaba utilizando Numpy, el cual realiza todos sus cálculos a través de la CPU.

Este proceso se maneja de manera más eficiente con la librería Torch, que permite especificar un *device* donde se realizarán los cálculos. En nuestro caso, utilizar la GPU es lo más conveniente; sin embargo, para los equipos sin GPU, esta asignación se realiza automáticamente mediante el código en la Figura 3.11.

Después de incluir estas líneas en el *main* del proyecto, se implementó el cálculo de *fast fading* con Torch, como se puede observar en la Figura 3.12.

Implementaciones similares se realizaron a lo largo del código en funciones que manejaban matrices muy grandes y que se utilizaban en gran parte del proyecto, como la optimización mostrada en la Figura 3.13.

La realización de estos cambios supuso una complejidad adicional más allá de la simplemente observada en las líneas de código. Al tratarse de un proyecto complejo, fue necesario cambiar la estructura del simulador y adaptar estos nuevos cambios sin tener que realizar un *refactoring* completo. Es decir, todas las partes del código donde se utilizaban estas funciones tuvieron que ser modificadas para adaptarse a la librería Torch. Además, se llevó a cabo un análisis detallado para determinar si los cuellos de botella del simulador podían ser solucionados mediante una simple implementación en GPU o si había otros factores subyacentes. Este fue el caso del código mostrado en la Figura 3.14.

Esta función, aunque está claramente estructurada y no parece presentar problemas de eficiencia a primera vista, genera un cuello de botella debido a cómo los operadores lógicos de Numpy operan sobre vectores y matrices.

```
antenna_element_GCS_position_m =
↪  np.zeros((np.size(antenna_element_LCS_position_m,0),3), dtype=np.single)
        # Process node by node as they have different rotations
        values, counts = np.unique(antenna_to_node_mapping, return_counts=True)
        for node_index in values:
            # Get azimuth rotation
            alpha_rad = np.radians(-alpha_bearing_deg[node_index])
            # Get zenith rotaton
            beta_rad = np.radians(-beta_downtilt_deg[node_index]+90)
            # Get slant rotation - Usually MIMO panels do not have a slant rotation
            gamma_rad = np.radians(-gamma_slant_deg[node_index])
            # Precompute some products
            coscos = np.cos(alpha_rad)*np.cos(beta_rad)
            cossin = np.cos(alpha_rad)*np.sin(beta_rad)
            sincos = np.sin(alpha_rad)*np.cos(beta_rad)
            sinsin = np.sin(alpha_rad)*np.sin(beta_rad)
            # Compute rotation matrix - See equation 7.1-5 of TR 38.901 V17.0.0
            R_inv = np.array([[ coscos
            ↪  , sincos                                                    ,
            ↪  -1*np.sin(beta_rad)],
                              [ cossin*np.sin(gamma_rad) -
                              ↪  np.sin(alpha_rad)*np.cos(gamma_rad),
                              ↪  sinsin*np.sin(gamma_rad) +
                              ↪  np.cos(alpha_rad)*np.cos(gamma_rad),
                              ↪  np.cos(beta_rad)*np.sin(gamma_rad) ],
                              [ cossin*np.cos(gamma_rad) +
                              ↪  np.sin(alpha_rad)*np.sin(gamma_rad),
                              ↪  sinsin*np.cos(gamma_rad) -
                              ↪  np.cos(alpha_rad)*np.sin(gamma_rad),
                              ↪  np.cos(beta_rad)*np.cos(gamma_rad) ]])
            # Get antennas of the node
            mask_antennas_of_selected_cells = antenna_to_node_mapping == node_index
            antenna_element_LCS_position_m_filtered =
            ↪  antenna_element_LCS_position_m[mask_antennas_of_selected_cells]
            # Compute LCS to GCS rotation for all antennas of the node
            antenna_element_GCS_position_m_filtered = np.array([ np.matmul(R_inv,
            ↪  np.transpose((antenna_element_LCS_position_m_filtered[i,:]))) for i in
            ↪  range(0,np.size(antenna_element_LCS_position_m_filtered,0)) ])
            antenna_element_GCS_position_m[mask_antennas_of_selected_cells] =
            ↪  antenna_element_GCS_position_m_filtered
        #Store calculations in array
        node_positions_mat_m = np.repeat(node_positions_m, counts, axis=0)
        antenna_element_GCS_position_m += node_positions_mat_m
        return antenna_element_GCS_position_m
```

Figura 3.8: Código transformación de posición LCS a GCS en bucle for.

```
        antenna_element_GCS_position_m =
        ↪  np.zeros((np.size(antenna_element_LCS_position_m,0),3), dtype=np.single)
        # Count the number of antennas per node
        _ , counts = np.unique(antenna_to_node_mapping, return_counts=True)
        # Get azimuth rotation
        alpha_rad_vector = np.radians(-alpha_bearing_deg[antenna_to_node_mapping])
        # Get zenith rotaton
        beta_rad_vector = np.radians(-beta_downtilt_deg[antenna_to_node_mapping]+90)
        # Get slant rotation - Usually MIMO panels do not have a slant rotation
        gamma_rad_vector = np.radians(-gamma_slant_deg[antenna_to_node_mapping])
        coscos_vector = np.cos(alpha_rad_vector)*np.cos(beta_rad_vector)
        cossin_vector = np.cos(alpha_rad_vector)*np.sin(beta_rad_vector)
        sincos_vector = np.sin(alpha_rad_vector)*np.cos(beta_rad_vector)
        sinsin_vector = np.sin(alpha_rad_vector)*np.sin(beta_rad_vector)
        # Compute rotation matrix - See equation 7.1-5 of TR 38.901 V17.0.0
        R_inv_vector = np.array([[ coscos_vector
        ↪  , sincos_vector                                                    ,
        ↪  -1*np.sin(beta_rad_vector)],
                            [ cossin_vector*np.sin(gamma_rad_vector) -
                            ↪  np.sin(alpha_rad_vector)*np.cos(gamma_rad_vector),
                            ↪  sinsin_vector*np.sin(gamma_rad_vector) +
                            ↪  np.cos(alpha_rad_vector)*np.cos(gamma_rad_vector),
                            ↪  np.cos(beta_rad_vector)*np.sin(gamma_rad_vector) ],
                            [ cossin_vector*np.cos(gamma_rad_vector) +
                            ↪  np.sin(alpha_rad_vector)*np.sin(gamma_rad_vector),
                            ↪  sinsin_vector*np.cos(gamma_rad_vector) -
                            ↪  np.cos(alpha_rad_vector)*np.sin(gamma_rad_vector),
                            ↪  np.cos(beta_rad_vector)*np.cos(gamma_rad_vector) ]])
        # Compute LCS to GCS rotation for all antennas of the nodes
        antenna_element_GCS_position_m = np.einsum('ijk, kj -> ki', R_inv_vector,
        ↪  antenna_element_LCS_position_m)
```

Figura 3.9: Código transformación de posición LCS a GCS sin bucle for.

```
def Rayleigh_fading(self, available_PRBs, number_of_RX_antennas, number_of_TX_antennas):
    rng = np.random.RandomState(self.seed+0)
    return (1/np.sqrt(2)) * (rng.standard_normal((available_PRBs, number_of_RX_antennas,
    ↪  number_of_TX_antennas))).astype(np.single)
```

Figura 3.10: Código *fast fading* de Numpy.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Figura 3.11: Código asignación automática dispositivo.

```
def Rayleigh_fading_torch(self, available_PRBs, number_of_RX_antennas,
↪  number_of_TX_antennas):
    rng = torch.Generator(device=self.device)
    a = (1/torch.sqrt(torch.tensor(2, device=self.device))) *
    ↪  (torch.cuda.FloatTensor(available_PRBs, number_of_RX_antennas,
    ↪  number_of_TX_antennas, device = self.device).normal_(generator=rng))
    #convert to cpu and numpy
    return a
```

Figura 3.12: Código *fast fading* de torch.

```
def mW_to_dBm_torch(value_mW):
    result = 10 * torch.log10(value_mW)
    return result
```

Figura 3.13: Código mW a dB de torch.

```
if isinstance(beam_activity_per_ue, np.ndarray):
    mask = np.logical_and(beams_in_same_carrier_than_server,
    ↪  PRB_ue_beam_interference_activity)
    sum_rss_mW = np.sum(tools.dBm_to_mW(np.where(mask, rsrp_prb_ue_to_cell_dBm,
    ↪  np.NINF)),axis=2)
else:
    sum_rss_mW = np.sum(tools.dBm_to_mW(np.where(beams_in_same_carrier_than_server,
    ↪  rsrp_prb_ue_to_cell_dBm, np.NINF)),axis=2)
```

Figura 3.14: Código cálculo Receive Signal Strength (RSS) utilizado en cálculo SINR.

```
if isinstance(beam_activity_per_ue, np.ndarray):
    mask = torch.logical_and(torch.tensor(beams_in_same_carrier_than_server,
    ↪  device=device), torch.tensor(PRB_ue_beam_interference_activity, device=device))
    selected_values_dBm_2 = torch.where(mask, torch.tensor(rsrp_prb_ue_to_cell_dBm,
    ↪  device=device, dtype=torch.float64), torch.tensor(np.NINF, device=device))
    rsrp_prb_ue_to_cell_mW_2 = tools.dBm_to_mW_torch(selected_values_dBm_2)
    sum_rss_mW = torch.sum(rsrp_prb_ue_to_cell_mW_2, axis=2)
    sum_rss_mW= sum_rss_mW.cpu().numpy()
else:
    selected_values_dBm = torch.where(torch.tensor(beams_in_same_carrier_than_server,
    ↪  device=device), torch.tensor(rsrp_prb_ue_to_cell_dBm, device=device), np.NINF)
    rsrp_prb_ue_to_cell_mW = tools.dBm_to_mW_torch(selected_values_dBm)
    sum_rss_mW = torch.sum(rsrp_prb_ue_to_cell_mW, axis=2)
    sum_rss_mW = sum_rss_mW.cpu().numpy()
```

Figura 3.15: Código cálculo Receive Signal Strength (RSS) utilizado en cálculo SINR usando torch.

Para llegar a esta versión final que se observa en la Figura 3.15, se evaluó si cada línea de código era más eficiente al realizarla en Python o en Torch. La versión final, que incluye la conversión de Torch a Numpy al final, resultó ser la más eficiente.

3.4.4. Comparación de Eficiencia del Simulador

Una vez implementadas las optimizaciones mediante la vectorización de funciones y el uso de GPU, se procedió a comparar la eficiencia del simulador. Para una comparación realista de lo que suponen estos cambios de GPU y vectorización, se utilizó como referencia el tiempo total consumido por modelo medido después de la eliminación del *plotting* innecesario, presentada en la Tabla 3.3. Tras aplicar todas las modificaciones, el tiempo total consumido por modelo se observa en la Tabla 3.4.

Modelo	Tiempo Total
3GPP TR 38.901 UMa lsc	63.226
ITU-R M.2135 UMa Umi colocated multilayer	39.023
ITU-R M.2135 UMa multilayer	32.528
3GPP TR 36.777 UMi AV	54.778
3GPP TR 36.777 UMa AV	51.053
3GPP TR 38.901 UMi lsc	47.532
ITU-R M.2135 UMa	24.732
ITU-R M.2135 UMi	27.092
3GPP TR 36.814 Case-1	16.666
3GPP TR 36.814 Case-1.omni	14.396
3GPP TR 38.901 UMa lsc single bs	7.456
3GPP TR 36.814 Case-1 single bs	4.689

Tabla 3.4: Resultados tras vectorización y uso de GPU.

Para evaluar cuantitativamente las mejoras, se calculó el porcentaje de tiempo ahorrado para cada modelo, reflejando la eficiencia ganada en comparación con el tiempo de ejecución original. La Tabla 3.5 muestra estos porcentajes de ahorro.

Los resultados indican una notable mejora en los tiempos de ejecución para la mayoría de los modelos. En particular, los modelos *3GPP TR 38.901 UMa lsc* e *ITU-R M.2135 UMa multilayer* muestran ahorros superiores al 90 % y 84 % respectivamente.

Es importante destacar que las mejoras no son tan evidentes en los modelos con tiempos de ejecución bajos. Estos modelos no aprovechan tanto la GPU debido a la menor complejidad de las multiplicaciones matriciales o a que los bucles *for* no son lo suficientemente grandes como para que la implementación en C de Numpy demuestre su eficiencia. Estos resultados subrayan la importancia de realizar un análisis detallado para identificar las áreas donde las optimizaciones tendrán el mayor impacto.

En conclusión, las optimizaciones aplicadas han resultado en una mejora significativa en la eficiencia del simulador para la mayoría de los modelos, demostrando la efectividad de la vectorización y el uso de GPU en la reducción del tiempo de ejecución.

Modelo	Porcentaje de Tiempo Ahorrado
3GPP TR 38.901 lsc UMa fr1 Umi C band plus fr3	58.17 %
3GPP TR 38.901 UMa lsc	91.21 %
ITU-R M.2135 UMa Umi colocated multilayer	84.20 %
ITU-R M.2135 UMa multilayer	84.82 %
3GPP TR 36.777 UMi AV	57.53 %
3GPP TR 36.777 UMa AV	58.36 %
3GPP TR 38.901 UMi lsc	59.90 %
ITU-R M.2135 UMa	77.11 %
ITU-R M.2135 UMi	72.02 %
3GPP TR 36.814 Case-1	56.82 %
3GPP TR 36.814 Case-1.omni	36.05 %
3GPP TR 38.901 UMa lsc single bs	24.95 %
3GPP TR 36.814 Case-1 single bs	16.06 %

Tabla 3.5: Porcentaje de tiempo ahorrado tras optimización.

Capítulo 4

Metodología

4.1. Diseño del Problema de RL

Para hacer una primera aproximación al problema, se decidió utilizar un entorno con baja complejidad, con el objetivo de poder observar si utilizar DRL es una solución viable.

Por tanto, definimos el número de UEs, U, igual a 10, y los inicializamos en un área cuadrada de $10m^2$. Estos representan por ejemplo un equipo de rescate moviéndose todos juntos en un escenario de emergencia. Por otra parte, el número de drones, D, utilizados es 1.

4.1.1. Espacio de Estados

Como se explicó en la sección 2.1.3, el espacio de estados no es lo mismo que el espacio de observaciones. En nuestro problema, y al disponer y entender completamente el entorno, podríamos definir los estados como la posición de los UEs, la distancia que tiene el agente a ellos y donde se encuentra este. Pero esto no es lo que estamos buscando, ya que nuestro objetivo es que este agente aprenda de la forma más parecida a lo necesario en un entorno real. Por tanto, este tipo de información no nos conviene. El espacio final de observaciones viene definido por:

$$\text{Pos} = [\rho^{\text{D}}_{1,t}, \rho^{\text{D}}_{1,t-m}, \rho^{\text{D}}_{1,t-M}] \tag{4.1}$$

$$\bar{\boldsymbol{\gamma}}_u = [\bar{\gamma}_{u,t}, \bar{\gamma}_{u,t-m}, \bar{\gamma}_{u,t-M}] \tag{4.2}$$

$$\mu_\theta = [\mu_{\theta,t}, \mu_{\theta,t-m}, \mu_{\theta,t-M}] \tag{4.3}$$

$$\sigma_\theta = [\sigma_{\theta,t}, \sigma_{\theta,t-m}, \sigma_{\theta,t-M}] \tag{4.4}$$

$$\text{OBS} = [\text{Pos}, \bar{\boldsymbol{\gamma}}_u, \mu_\theta, \sigma_\theta] \tag{4.5}$$

donde Pos es el vector de posiciones de los drones, $\bar{\boldsymbol{\gamma}}_u$ representa el vector de la relación señal a interferencia más ruido, μ_θ y σ_θ son la media y la desviación estándar del ángulo con el que se recibe la señal, medidas tomadas con un punto de referencia desde el este, como se observa en la Figura 4.1, y OBS es el vector de observaciones que está conformado por Pos, $\bar{\boldsymbol{\gamma}}_u$, μ_θ y σ_θ. Además, disponemos de una memoria de tamaño, M, que almacena las últimas M observaciones.

El objetivo de utilizar estas observaciones es que el dron use la información que puede inferir a través de su estación base, para tener una idea de la posición de los UEs sin directamente dársela como estado.

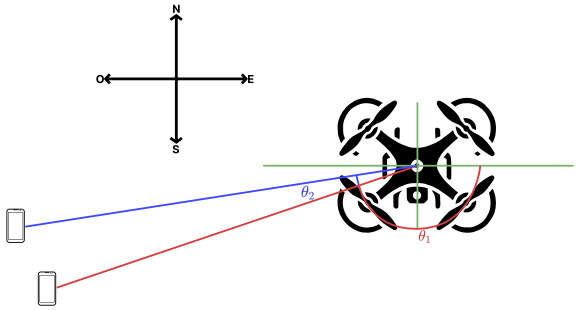


Figura 4.1: Ejemplo cálculo ángulos.

4.1.2. Espacio de Acciones

Dependiendo del modelo utilizado, que se explican en la sección 4.2, el espacio de acciones será discreto o continuo.

En el caso de un espacio de acciones discreto, el agente dispone de un conjunto finito de estas, definido por el conjunto $\mathcal{A} = \{\uparrow, \downarrow, \rightarrow, \leftarrow, \cdot\}$, donde cada acción corresponde a un movimiento en una de las siguientes direcciones: norte ($\uparrow$), sur ($\downarrow$), este ($\rightarrow$), oeste ($\leftarrow$), o permanecer en la misma posición ($\cdot$). Además, la magnitud del movimiento es constante, denotada por d, es decir, el agente se desplaza una distancia fija de d metros en una sola dirección.

En el caso de un espacio de acciones continuo el agente tiene dos variables a elegir, la dirección y la magnitud del movimiento. La dirección se define por el ángulo θ_d y la magnitud por la distancia d. Por lo tanto, el espacio de acciones es $\mathcal{A} = \{(\theta, d)\}$, donde $\theta \in [-180, 180)$ y $d \in [0, d_{\text{máx}}]$. Siendo $d_{\text{máx}}$, la distancia máxima que puede recorrer el dron en una acción. El ángulo, θ_d, está definido con respecto al este.

4.1.3. Recompensa

La elección de que recompensa elegir es de vital importancia dentro del DRL, ya que es lo que va a determinar el comportamiento del agente. En este caso, hemos utilizado como recompensa la ecuación 1.12. Sin embargo, con esto no era suficiente, ya que al estar los UEs tan concentrados, hacía que las posiciones cercanas a este clúster fueran ya de por si rentables para el agente, por tanto, no le importaba estar en la mejor posición, sino que iba saltando entre diferentes posiciones cercanas. Para solucionar esto, primero se realizó una normalización de la recompensa, y después se ajustó la recompensa para los valores más altos utilizando la siguiente ecuación:

$$\text{reward} = 2\left(\frac{\text{reward} - 4}{8{,}104 - 4}\right) - 1 \tag{4.6}$$

Si la recompensa resultante es mayor a 1.47, entonces se asigna un valor fijo de 10 a la recompensa para incentivar fuertemente estas posiciones óptimas. Los valores de la normalización, la recompensa, y la posterior recompensa fija se han obtenido mediante prueba y error, y se ha comprobado que son los que mejor resultado dan.

4.2. Algoritmos de RL para Resolver El Problema

En esta sección vamos a explicar los fundamentos teóricos de los algoritmos implementados a lo largo de nuestro proyecto.

4.2.1. Deep Q Networks

Estas redes trajeron consigo una revolución al campo del aprendizaje reforzado, ya que fue el primer trabajo de investigación que consiguió demostrar la capacidad de las redes neuronales en entornos complejos como los videojuegos de la Atari 2600 [29].

4.2.1.1. Naturaleza

En este trabajo, la red implementada se trataba de una red neuronal convolucional, que se encargaba de aproximar la función de valor de las tuplas estado-acción, $Q(s, a)$, que es la función que nos indica cuánto de buena es una acción en un estado concreto. Un concepto muy similar a lo que sucede en el algoritmo de Q-learning 1 , introducido en la sección 2.3.1, en este caso implementado con redes neuronales.

Esta red neuronal convolucional se puede intercambiar por cualquier otro aproximador de funciones, como una red neuronal densa, lo que permite adaptarlo a cualquier tipo de tarea.

El algoritmo en el que se basa el DQN es en el de Q-learning, pero con unas ciertas diferencias a la hora de implementarlo.

4.2.1.2. Función de Pérdida

Lo primero es la manera de implementar la pérdida, ya que ahora no será una tabla en la que se almacenen los valores $Q(s, a)$, sino que será una red neuronal la que se encargue de aproximar estos valores. De manera que la pérdida se calcula como:

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s, a; \theta_i\right)\right)^2\right], \tag{4.7}$$

donde $y_i = \mathbb{E}_{s'\sim\varepsilon}\left[r + \gamma \operatorname{máx}_{a'} Q(s', a'; \theta_{i-1})|s, a\right]$ es el objetivo para la iteración i, y $\rho(s, a)$ es una distribución de probabilidad sobre las secuencias, s y las acciones, a, a la que nos referimos como la distribución de comportamiento.

Los parámetros de la iteración anterior, θ_{i-1}, se mantienen fijos al optimizar la función de pérdida, $L_i(\theta_i)$, la cual no deja de ser una pérdida cuadrática como la presentada en la ecuación 2.36 entre el valor predicho por la red y el valor que debería tener.

4.2.1.3. Red Objetivo

En la función de pérdida presentada en la ecuación 4.7, no solo se consigue optimizar la red de forma que en cada iteración consigue predecir mejor los valores de la recompensa, sino que además se introduce el concepto de una red objetivo o copia. Esta red sirve principalmente para mantener

la estabilidad en el entrenamiento. De manera que θ_i es la red donde se van haciendo las actualizaciones en cada iteración y θ_{i-1} es la red objetivo que se va actualizando cada más iteraciones.

4.2.1.4. Gradiente

El gradiente de dicha pérdida se quedaría como:

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_{i-1}\right) - Q\left(s, a; \theta_i\right)\right) \nabla_{\theta_i} Q\left(s, a; \theta_i\right)\right]. \tag{4.8}$$

4.2.1.5. Deadly Triad

Teóricamente este algoritmo no está asegurado a converger, ya que cumple la *deadly triad* [49], al mezclar aproximadores de funciones no lineales, con un algoritmo de optimización basado en diferencias temporales y ser *off-policy*. En la práctica se ha demostrado que con ciertas técnicas, como el uso de un *replay buffer* y *target networks*, el algoritmo converge y es capaz de aprender a resolver tareas complejas.

4.2.1.6. Replay Buffer

Se trata de una técnica que se utiliza para mejorar la eficiencia del entrenamiento. Consiste en almacenar las transiciones que se van produciendo en el entorno en un buffer. De este buffer se muestrean las transiciones de manera uniforme para poder entrenar al agente. Este buffer tiene una capacidad finita, por lo que cuando se llena, se van eliminando las transiciones más antiguas.

4.2.1.7. Exploración versus Explotación

El algoritmo DQN utiliza una estrategia, ϵ-greedy expresado en la ecuación 2.22, para balancear la exploración y la explotación, donde un mayor ϵ hace que el agente tienda a elegir una acción aleatoria antes que la acción que le produce más recompensa. Inicialmente, ϵ es alto para fomentar la exploración, y disminuye gradualmente para favorecer la explotación de las políticas aprendidas.

El algoritmo de la DQN se puede observar en el bloque 6.

De manera intuitiva, ahora mismo la representación de los valores, Q, para cada tupla de estado-acción, pasaría de tener una visualización como en la Figura 2.8, a como la vista en la Figura 4.2.

En esta figura se quiere dar a entender que, gracias al uso de las redes neuronales, ya no es necesario construir una tabla explícita, sino que se puede estimar de manera precisa cuál es la recompensa esperada de cada tupla estado-acción. La capacidad que tienen las redes neuronales de extraer características permite comprender las tuplas de manera conjunta y realizar agrupaciones de datos complejos, como se observa en la imagen.

Algorithm 6 DQN

Inicializar la memoria de reproducción $\mathcal{D}$ con capacidad N
Inicializar la función de valor de acción Q con pesos aleatorios
for episodio = 1, M **do**
  Inicializar la secuencia $s_1 = \{x_1\}$.
  for $t = 1, T$ **do**
    Con probabilidad ϵ seleccionar una acción aleatoria a_t
    de lo contrario seleccionar $a_t = \text{máx}_a\, Q^*(s_t, a; \theta)$
    Ejecutar la acción a_t en el emulador y observar la recompensa r_t e imagen x_{t+1}
    Establecer $s_{t+1} = s_t, a_t, x_{t+1}$
    Almacenar la transición (s_t, a_t, r_t, s_{t+1}) en $\mathcal{D}$
    Muestrear un minibatch aleatorio de transiciones (s_j, a_j, r_j, s_{j+1}) de $\mathcal{D}$
    Establecer $y_j = \begin{cases} r_j & \text{si } s_{j+1} \text{ es terminal} \\ r_j + \gamma\, \text{máx}_{a'}\, Q(s_{j+1}, a'; \theta) & \text{si } s_{j+1} \text{ no es terminal} \end{cases}$
    Realizar un paso de descenso de gradiente en $(y_j - Q(s_j, a_j; \theta))^2$ según la Ecuación 4.8
  end for
end for

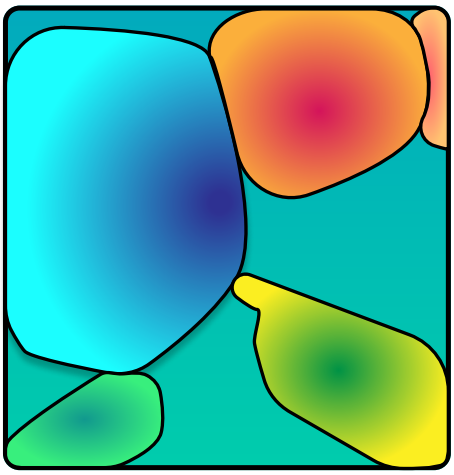


Figura 4.2: Visualización tabla Q DQN.

4.2.2. Trust Region Policy Optimization y Proximal Policy Optimization

Estos dos algoritmos difieren mucho de los que había en DRL, ya que en lugar de aproximar funciones de valor, y a raíz de esto, elegir la acción más probable, se aproxima directamente la política.

4.2.2.1. Naturaleza

Estos algoritmos están basados en los *Policy Gradients* y la ecuación base 2.26 que estos presentan, por tanto son *on-policy*, y *model-free*. En la literatura no encontramos un consenso sobre si estos métodos son actor-crítico o basados en política. Ambos heredan la estructura del algoritmo REINFORCE presentado en la sección 2.4.2.2, sin embargo añaden una red neuronal de la función

de valor para hacer la optimización. Por tanto, los consideraremos métodos actor-crítico.

En lugar de usar directamente la función, R_t, se emplea la ventaja, $\hat{A}_t$, definida en la siguiente ecuación:

$$\hat{A}_t = R_t - V(s_t), \tag{4.9}$$

donde $V(s_t)$ es la función de valor del estado s_t. Esta ventaja indica cuánto mejor es una acción en comparación con la media de las acciones posibles en un estado concreto. La función, $V(s_t)$, puede estimarse directamente con una red neuronal o indirectamente mediante métodos iterativos de la función de valor. Es esto lo que le da a TRPO y PPO un punto de vista de actor-crítico: estimar la política, π, y la función de valor, $V(s)$.

4.2.2.2. Beneficios

Buscan mejorar varios problemas asociados con los *Policy Gradients*. Uno de los principales desafíos es la difícil elección de parámetros, como la tasa de aprendizaje. Además, los cambios significativos en la política pueden causar divergencias en el entrenamiento, lo que impide que el agente aprenda adecuadamente. PPO y TRPO mitigan este problema limitando la magnitud del cambio en la política, asegurando actualizaciones más graduales y estables.

Otro problema es la eficiencia en términos de muestras. Al ser métodos *on-policy*, TRPO y PPO no pueden reutilizar las muestras de entrenamiento, lo que incurre una mayor complejidad comparado con métodos *off-policy*. Necesitan generar nuevas muestras para cada actualización de la política, haciendo el proceso de entrenamiento más costoso en términos de uso de datos.

4.2.2.3. Trust Region Policy Optimization

El TRPO, basa la función a optimizar en [35], definiendo el objetivo de la siguiente forma:

$$\begin{aligned} &\max_{\theta} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t \right] \\ &\text{subject to } \hat{\mathbb{E}}_t \left[\mathrm{KL}\left[\pi_{\theta_{\text{old}}}\left(\cdot \mid s_t\right), \pi_\theta\left(\cdot \mid s_t\right) \right] \right] \leq \delta. \end{aligned} \tag{4.10}$$

La intuición detrás de esta ecuación es que, la nueva distribución de pesos, θ, no se aleje demasiado de la distribución anterior, θ_{old}, de manera que se garantiza que las actualizaciones de la política sean suaves y no provoquen divergencias. Con la función objetivo 4.10, se busca que la política de los nuevos pesos maximice las acciones que tienen ventaja positiva, y que la divergencia de Kullback-Leibler entre las dos políticas no sea mayor que un valor, δ. La divergencia de Kullback-Leibler es una medida de la diferencia entre dos distribuciones de probabilidad. Además, dentro se implementa el *importance sampling*, representado como la fracción entre la política actual, π_θ, y la anterior, $\pi_{\theta_{\text{old}}}$, técnica utilizada para poder estimar la ventaja de la nueva política utilizando las muestras de una política anterior. El problema que tiene TRPO es que es necesario calcular el operador de Hessiano, lo cual es computacionalmente costoso. Por tanto, se propone una versión simplificada de este algoritmo, que es el PPO.

4.2.2.4. Proximal Policy Optimiaztion

En lugar de darle importancia a la diferencia entre distribuciones con la divergencia Kullback-Leibler, partimos de la siguiente:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right], \tag{4.11}$$

donde definimos $r_t(\theta)$ como el ratio de las políticas, que es la fracción de probabilidad de la política actual y la anterior. Por tanto, se nos queda la siguiente función de pérdida:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\text{mín} \left(r_t(\theta) \hat{A}_t, \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \hat{A}_t \right) \right], \tag{4.12}$$

donde ϵ es un hiperparámetro que controla la magnitud del cambio en la política. Si el ratio de las políticas es mayor que $1 + \epsilon$, se recorta a $1 + \epsilon$, y si es menor que $1 - \epsilon$, se recorta a $1 - \epsilon$. De esta manera, se garantiza que la política no cambie demasiado en cada iteración, evitando divergencias en el entrenamiento.

4.2.2.5. Generalized Advantage Estimation

La Estimación de la Ventaja Generalizada (GAE) [50] es una técnica utilizada en PPO para evaluar mejor qué tan buena es una acción en un estado. En lugar de considerar solo las recompensas inmediatas, GAE utiliza un factor de descuento, λ, para combinar recompensas futuras y actuales de manera más sofisticada. A diferencia del descuento exponencial tradicional utilizado por el factor, γ, GAE aplica un promedio ponderado exponencialmente. Esto suaviza las fluctuaciones, y proporciona una estimación más precisa y estable de la ventaja, ayudando a PPO a aprender mejores políticas de manera más eficiente. Al ajustar el factor, λ, podemos controlar cuánto queremos confiar en las recompensas a largo plazo frente a las inmediatas, logrando un equilibrio que mejora la calidad del aprendizaje del agente.

El algoritmo de PPO se puede observar en el bloque 7.

Algorithm 7 PPO, Estilo Actor-Crítico

for episodio = 1, 2, … **do**
  for actor = 1, 2, …, N **do**
    Ejecutar la política $\pi_{\theta_{\text{old}}}$ en el entorno durante T pasos de tiempo
    Calcular las estimaciones de ventaja $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimizar el objetivo L^{CPI} con respecto a θ, con K épocas y tamaño de *minibatch* $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Un ejemplo para una intuición de como funciona este algoritmo se observa en la Figura 4.3, donde podemos ver un ejemplo en el que un agente debe subir una montaña, y se observa como sigue tres principales caminos en diferentes episodios.

Se puede observar como la política, π, va mejorando a lo largo de los episodios, pero sin realizar cambios bruscos. La flecha roja indica una primera inicialización del agente donde la solución no

es la óptima. La flecha amarilla es un siguiente episodio donde el agente ha aprendido a subir un poco la montaña, y así obtener mejores recompensas. La flecha verde, y última, es a la que converge el algoritmo, siendo así la política óptima.

De esta figura podemos extraer como el PPO, en lugar de poder aprender directamente aquella política verde, es más conservador y va escalando poco a poco la montaña. Esto, que a priori no parece eficiente, es necesario ya que cambios drásticos en la distribución de pesos de la política pueden llevar al agente a no converger.

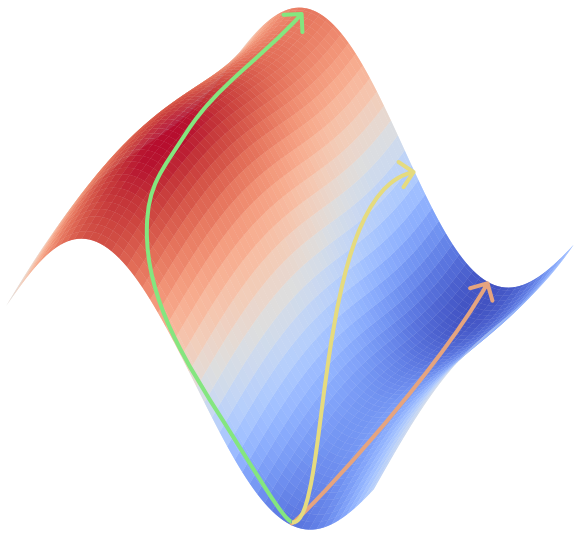


Figura 4.3: Ejemplo PPO.

4.3. Implementación

Para la implementación de los algoritmos, hemos utilizado la librería de aprendizaje por refuerzo pytorchrl [51], una herramienta de código abierto que incluye los algoritmos de aprendizaje por refuerzo más comunes. Sin embargo, debido a su simplicidad, el algoritmo DQN se ha implementado desde cero. En contraposición, hemos decidido implementar PPO utilizando una librería externa debido a su mayor complejidad y necesidad de precisión. Además, esta elección facilita la futura incorporación de otros algoritmos.

En la Figura 4.4 se muestra cómo hemos integrado Giulia en la librería torchrl. Los elementos en verde representan las partes relacionadas con Giulia, mientras que los elementos en rojo corresponden a la librería torchrl. Esta librería permite diseñar un entorno con el cual el agente interactúa, y hemos integrado Giulia en esta parte del código.

Para la comunicación entre todos los módulos, utilizamos tensordict, una librería desarrollada por el mismo autor de pytorchrl. Tensordict permite la comunicación mediante diccionarios de tensores, ofreciendo una gran versatilidad para aplicar funciones a los tensores y trabajar con ellos en la GPU.

Además, hemos diseñado un diagrama de flujo que se aplica a ambos casos de uso. En la Figura 4.5, se presenta el flujo de trabajo que seguimos para entrenar con el algoritmo PPO utilizando la librería

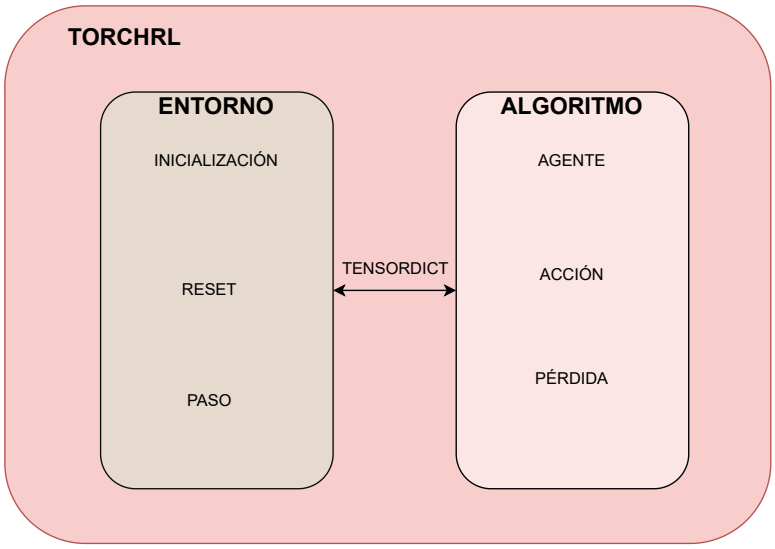


Figura 4.4: Sistema de entorno y agente.

pytorchrl.

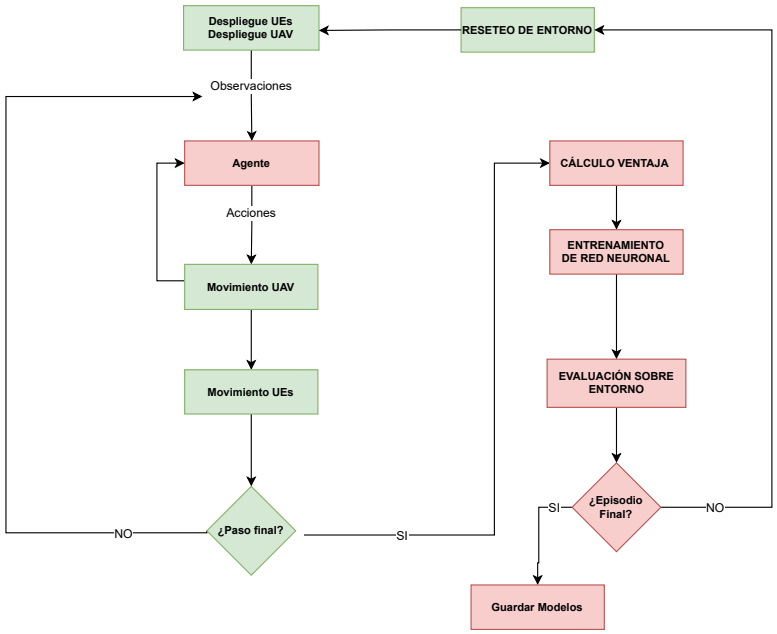


Figura 4.5: Diagrama de flujo del entrenamiento PPO con torchrl.

4.3.1. Detalles de Implementación

En esta sección, explicaremos algunos detalles avanzados sobre la implementación del algoritmo PPO.

4.3.1.1. Selección de Acciones

Debido a la naturaleza del algoritmo PPO, es necesario definir un método más complejo para seleccionar acciones en un espacio continuo. La red neuronal genera una distribución de probabilidad como salida, y se selecciona un valor de esta distribución.

La distribución de probabilidad generada es una distribución normal, caracterizada por una media y una desviación estándar. Por lo tanto, la red neuronal debe devolver la media y la desviación estándar para cada dimensión de la acción. En este caso, dado que las acciones son el ángulo y la velocidad, la red neuronal devolverá cuatro valores.

Durante el entrenamiento, para fomentar la exploración, se seleccionan acciones muestreadas de la distribución normal basada en la media y la desviación estándar. En cambio, durante la inferencia, se selecciona la acción con la mayor probabilidad, que corresponde a la media de la distribución.

4.3.1.2. Entropía para La Selección de Acciones

Para promover la exploración de estados y acciones en PPO, se introduce un nuevo hiperparámetro llamado *entropy coefficient*. Este coeficiente incrementa la desviación estándar al decidir las acciones, haciendo que el agente, durante el entrenamiento, tienda a tomar acciones más diversas y alejadas de la media de la distribución.

4.4. Casos de Uso

En esta sección presentamos los diferentes casos de uso en los que los agentes van a ser entrenados. Se distingue entre movimiento de UEs e inicialización de los agentes.

4.4.1. Movimiento UEs

En esta subsección explicamos los diferentes tipos de movimientos que van a seguir los UEs en nuestro entorno. El entorno en el que se pueden desplazar está delimitado en un rectángulo de 300m x 300m.

4.4.1.1. UEs Estáticos

Como mencionamos anteriormente, inicializamos 10 UEs distribuidos uniformemente en un área de 10m x 10m. Esta región de inicialización se extiende desde x = 0 hasta x = 10 y desde y = 0 hasta y = 10. En este escenario, los UEs permanecen en sus posiciones iniciales y no se mueven.

4.4.1.2. UEs Dinámicos

Una vez realizado el estudio sobre los UEs estáticos, es interesante avanzar un poco más en el objetivo del trabajo, estos patrones se acercan más a los que se pueden observar en la realidad. Se plantean dos nuevos movimientos para los UEs:

- **Movimiento lineal**: En este caso el clúster de UES comienza centrado en (0,0), avanza hacia el punto (300,0). Cambia de sentido hacia la izquierda hasta llegar a (-300,0) y, por último, vuelve a la posición (0,0).
- **Movimiento circular**: En este caso los UEs siguen un movimiento circular con el centro en (0,0) y un radio de 200m. En este caso, el clúster de UEs está inicializado en (0,-200),

El agente debe ser capaz de seguir el movimiento lineal y el circular.

4.4.2. Inicialización Agentes

Dentro de los escenarios mostrados previamente, tenemos que seleccionar tambien donde inicializamos el agente. Esta decisión va a ser de vital importancia a la hora de determinar si el agente aprende o no.

4.4.2.1. Inicialización Estática

Durante el entrenamiento y la evaluación el agente se mantiene en la misma posición. Este tipo de inicialización se utiliza como una primera aproximación al problema, para comprender si es capaz de resolver una tarea más sencilla.

4.4.2.2. Inicialización Aleatoria

Con el objetivo de saber si estos algoritmos son capaces de generalizar. Durante el entrenamiento se inicializa el agente en posiciones aleatorias de dentro del rectángulo sobre el que trabajamos. Se entrena durante los episodios que sea necesario y, después, se realiza una evaluación. Esta evaluación por norma general será fija, es decir, a pesar de estar entrenando en posiciones aleatorias se comprueba el rendimiento del agente siempre desde la misma posición o conjunto de posiciones. Por ejemplo, realizamos 100 episodios con esta inicialización aleatoria. Posteriormente ponemos al modelo en modo evaluación, con el objetivo de no contaminar los datos de entrenamiento, y vemos si sabe llegar al clúster de UEs si lo ponemos en la posición (0,100). Si este episodio es uno de los primeros que necesita el modelo para aprender, lo común es que el agente no sepa llegar a los UEs. Una vez se vayan avanzando en los episodios deberíamos observar como en estas evaluaciones el agente si es capaz de llegar. De esta manera se comprueban si el agente aprende a medida que avanzan los episodios.

4.5. Parámetros e Hiperparámetros

En esta sección vamos a introducir los parámetros utilizados para el simulador y los hiperparámetros con los que hemos implementado los algoritmos. Los parámetros son esenciales para definir las condiciones del entorno de simulación, mientras que los hiperparámetros son cruciales para el ajuste y el rendimiento de los algoritmos de aprendizaje reforzado.

4.5.1. Parámetros

Los parámetros del simulador se definen en la tabla 4.1. Estos incluyen el número de UEs y drones, el modelo de referencia, el tipo de entorno, y la duración de la simulación.

Parámetro	**Valor**
Número de UEs	10
Número de drones	1
Modelo	3GPP TR 36.814
Tipo de Entorno	Rectangular
Longitud Temporal Simulación	128

Tabla 4.1: Parámetros del simulador.

4.5.2. Hiperparámetros

Debido a las diferencias de los algoritmos, para un correcto funcionamiento, es necesario implementar diferentes hiperparámetros en cada uno de ellos. A continuación, se detallan los hiperparámetros utilizados para los algoritmos DQN y PPO en las tablas 4.2 y 4.3, respectivamente. Los hiperparametros óptimos han sido seleccionados a partir de [29] y [36] para DQN y PPO respectivamente.Además de una validación empírica basada en un grid-search de múltiples valore.

4.5.2.1. DQN

La tabla 4.2 muestra los hiperparámetros utilizados para el algoritmo DQN. Estos incluyen detalles sobre la memoria, las capas ocultas, las neuronas por capa, la función de activación, la tasa de aprendizaje, y otros parámetros específicos de la técnica ϵ-greedy.

4.5.2.2. PPO

La tabla 4.3 muestra los hiperparámetros utilizados para el algoritmo PPO. incluyen detalles sobre la memoria, las capas ocultas, las neuronas por capa, la función de activación, la tasa de aprendizaje, el tamaño de batch y minibatch, las épocas de entrenamiento, y otros parámetros específicos del método PPO como ϵ de recorte y entropía.

Hiperparámetro	**Valor**
Tamaño de Memoria	4
Capas Ocultas	2
Número de neuronas por capa	256
Función de Activación	ReLU
Tasa de Aprendizaje	1e-4
Tamaño de Batch	128
Tasa de ϵ-greedy inicial	0.9
Tasa de ϵ-greedy final	0.05
Decaimiento por episodio de tasa de ϵ-greedy	$8{,}5e^{-5}$
Factor de descuento, γ	0.99

Tabla 4.2: Hiperparámetros del algoritmo DQN.

Hiperparámetro	**Valor**
Tamaño de Memoria	4
Capas Ocultas	3
Número de neuronas por capa	128
Función de Activación	ReLU
Tasa de Aprendizaje	1e-4
Tamaño de Batch	128
Tamaño de minibatch	64
Épocas de entrenamiento	15
Factor de descuento, γ	0.99
ϵ de recorte	0.2
Entropía de ϵ	0.02
Factor GAE, λ	0.95

Tabla 4.3: Hiperparámetros del algoritmo PPO.

Capítulo 5

Resultados

5.1. User Equipments Estáticos

En esta sección vamos a comparar los resultados obtenidos con DQN y PPO, en el caso en el que los UEs no se desplazan. Esto se hace con el objetivo de hacer un cribado para decidir que modelo nos interesa para los siguientes casos de uso.

5.1.1. Inicialización Estática

En la primera fila de imágenes de la Figura 5.1 se muestran los resultados de entrenamiento en el entorno explicado previamente, pero con el agente siempre inicializado en (-100,0), tanto en el entrenamiento como en la evaluación. Como hemos introducido en la sección 4.4.2, esto se hace con el objetivo de saber si nuestro agente puede o no resolver este problema.

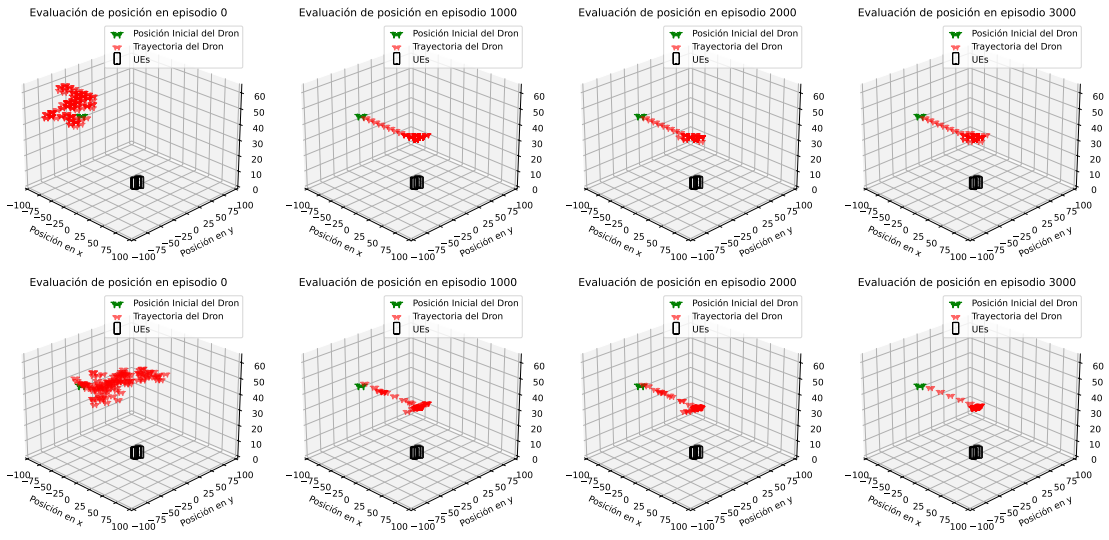


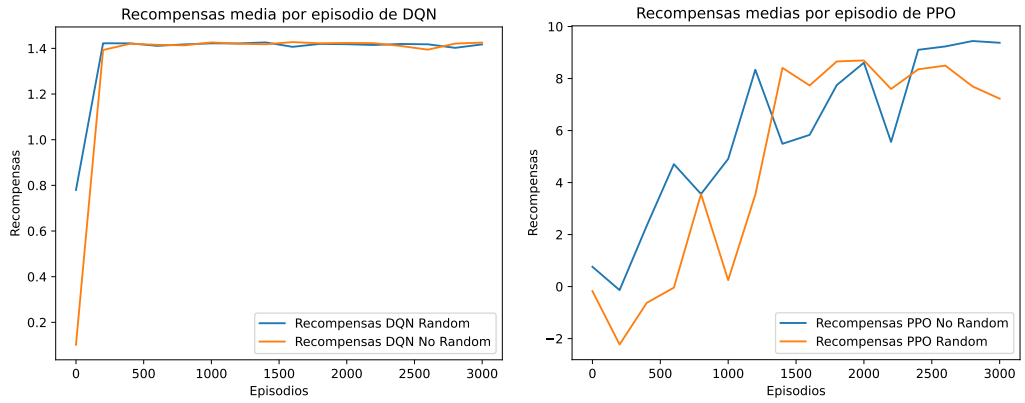
Figura 5.1: Comparación de trayectorias PPO y DQN.

A simple vista se observa que ambos modelos, DQN y PPO, convergen de una manera rápida, a los

1000 episodios ya tienen un conocimiento de donde se encuentran los UEs. El modelo PPO muestra una mayor precisión a la hora de mantenerse estático encima de los usuarios, principalmente por su naturaleza basada en un espacio de acciones continuo. En DQN, si el agente toma alguna acción incorrecta, se estará desplazando 10 metros. Sin embargo, si en PPO se toma una decisión incorrecta es más probable que la acción tomada sea menor a esos 10 metros. Las gráficas de recompensa para ambos algoritmos se presentan en la Figura 5.2.

5.1.2. Inicialización Aleatoria

Además, también llevamos a cabo esta comparación en el caso de inicialización aleatoria, donde el aprendizaje es más complejo. Como se observa en la Figura 5.2, no hay una gran diferencia en los resultados entre entrenar con aleatoriedad o no, lo que indica que el agente aprende y sabe llegar al centro desde cualquier posición. Por tanto, es capaz con los parámetros de entrada del modelo, que no incluyen las posiciones de los usuarios, de tener una representación de donde se encuentran los usuarios.



(a) Recompensa media por episodio DQN. **(b) Recompensa media por episodio PPO.**

Figura 5.2: Comparación de recompensas entre DQN (fila de arriba) y PPO (fila de abajo).

Un aspecto de particular importancia son los valores de recompensas que se aprecian para los modelos DQN y PPO. En el caso de DQN la recompensa media máxima tiene un valor de 1.4, mientras que en PPO el máximo de recompensa tiene un valor de aproximadamente 9. Volvemos a lo mencionado previamente, esto se debe a la naturaleza del propio DQN, al estar incializándose en (-100,0), solo tiene una resolución de movimiento, d. Por tanto no puede llegar a converger al punto donde se consigue la mayor recompensa, en este caso esa posición es la cercana a (5,5).

5.2. UEs en Movimiento

Como se observa en la sección anterior, el algoritmo DQN tiene un carácter más rígido, principalmente heredado por la discretización a la hora de seleccionar las acciones, por lo que para una mayor adaptación a los patrones de movimiento del dron a los de los UEs he decidido utilizar PPO en casuísticas más complejas.

5.2.1. Movimiento Lineal

En la Figura 5.3, se puede observar el ejemplo actual. En este caso, los UEs se desplazan de derecha a izquierda inicializados en la posición (0,0), llegando al extremo derecho (300,0) luego al izquierdo (-300,0) y volviendo al centro (0,0), como se explicó en la subsección 4.4.1.1. Además, los UEs están inicializados en un clúster de 10x10.

5.2.1.1. Inicialización Estática

Como se ha realizado en el ejemplo anterior, primero se realizan experimentos con el agente inicializado en la misma posición en entrenamiento y en evaluación. En la Figura 5.3, se puede observar el caso en el que el agente se inicializa en (-100,0). La flecha azul indica el movimiento que tiene el conjunto de los usuarios.

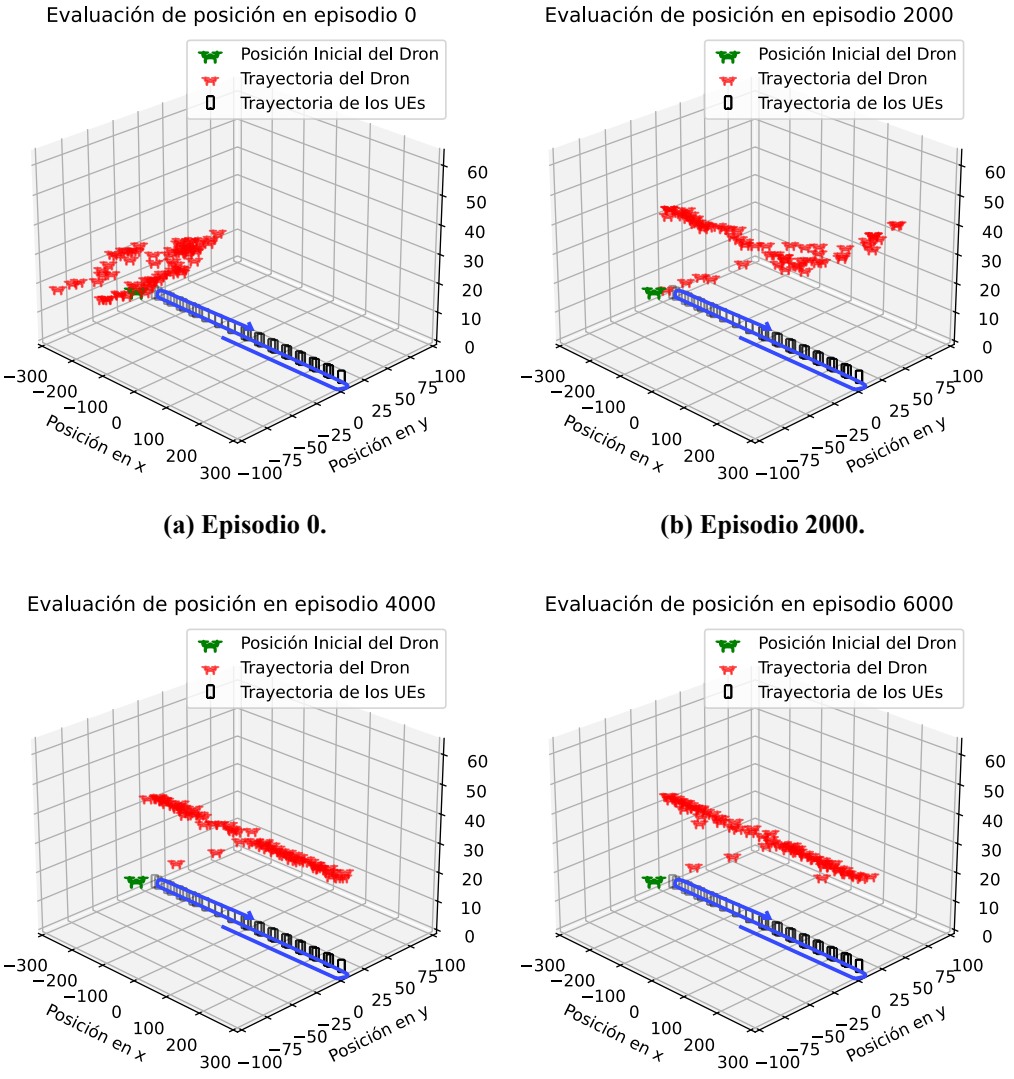
El agente muestra un claro aprendizaje conforme avanzan los episodios, siendo capaz de posicionarse exactamente encima de los UEs. Con 2000 episodios, el dron todavía no es capaz de entender que los UEs se mueven de derecha a izquierda. La situación cambia con 4000 episodios donde el dron sigue la misma trayectoria que los UEs.

En la Figura 5.4, se muestran las gráficas correspondientes con la distancia media a los UEs (medida únicamente con respecto a los ejes x e y, ya que el z no tiene movimiento), además de la recompensa media por episodio que tiene el agente. Este resultado nos ayuda a entender que el dron llega a converger a una solución independientemente de la posición de inicialización. De esta forma el dron es capaz de encontrar a los UEs y seguirles desde cualquier posición inicial.

Se observa como la distancia media tiene una variación muy grande hasta que llega al episodio 4000, en el cual ya converge a una política buena. Si miramos la gráfica de la recompensa media obtenida en el episodio 4000, todavía tenemos una política que puede mejorar. Esto se debe a que la posición óptima está extremadamente cerca de los UEs, por tanto el agente puede seguir optimizando las decisiones a tomar para que así se encuentre en todo momento exactamente encima de ellos.

5.2.1.2. Inicialización Aleatoria

Al igual que se realizó en la experimentación de los UEs estáticos para comprobar el aprendizaje del agente. En este caso también se han implementado simulaciones para determinar la capacidad del dron para encontrar soluciones óptimas iniciándolo desde posiciones aleatorias. Estos resultados se observan en la Figura 5.5. Analizado dichos resultados, se puede observar que gracias a la inicialización aleatoria durante el entrenamiento, el agente aprende a resolver todas las posiciones de la misma manera. Esto es un avance, ya que no necesitamos un entrenamiento especifico para cada posición de salida del dron. En consecuencia, independientemente de la inicialización, el dron aprende el movimiento de los usuarios alcanzando en todo momento la máxima recompensa. Este hallazgo difiere respecto a los resultados obtenidos con inicialización estática, donde la dependencia de dicha inicialización es excesivamente elevada en lo concerniente a la recompensa media obtenida. Por tanto, concluimos que el agente sabe aproximarse y seguir a los usuarios desde cualquier punto del escenario.



(a) Episodio 0.

(b) Episodio 2000.

(c) Episodio 4000.

(d) Episodio 6000.

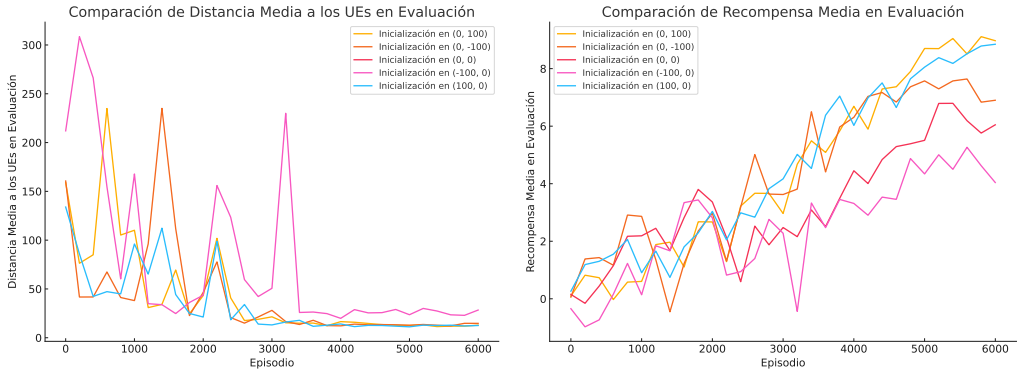
Figura 5.3: Visualización de episodios con posición de inicialización (0, -100).

5.2.2. Movimiento Circular

Después de obtener los resultados del movimiento lineal, se decide realizar el circular. Al suponer este un mayor reto para los UEs, debido a que necesitan realizar acciones más dispares que en el ejemplo anterior. Es decir, tienen que ir cambiando la trayectoria poco a poco para poder seguir el movmiento circular. Sin embargo, cuando es lineal puede mantener la acción de seguir recto hasta que llega al extremo.

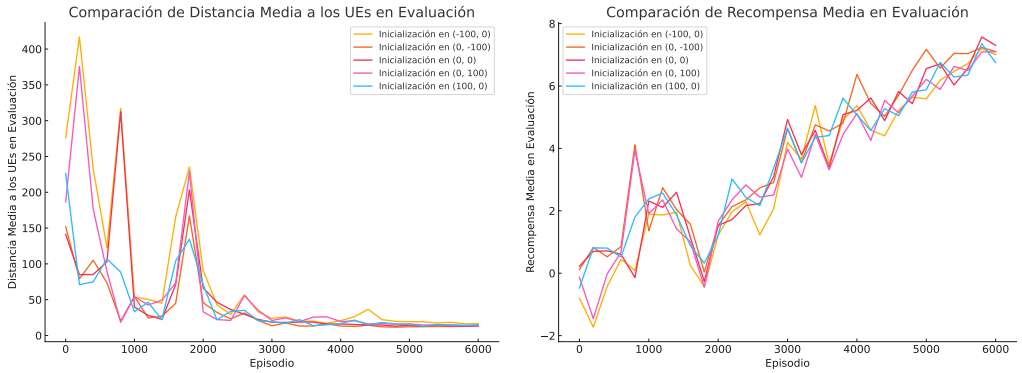
5.2.2.1. Inicialización Estática

En la Figura 5.6, se muestra como el agente es capaz de seguir a los UEs cuando estos tienen un movimiento circular. Indicar que el agente siempre parte de la posición inicial (0,0). Se observa



(a) Comparación de Distancia Media a los UEs en Evaluación. **(b) Comparación de Recompensa Media en Evaluación.**

Figura 5.4: Comparación de métricas en diferentes inicializaciones durante entrenamiento aleatorio.



(a) Comparación de Distancia Media a los UEs en Evaluación. **(b) Comparación de Recompensa Media en Evaluación.**

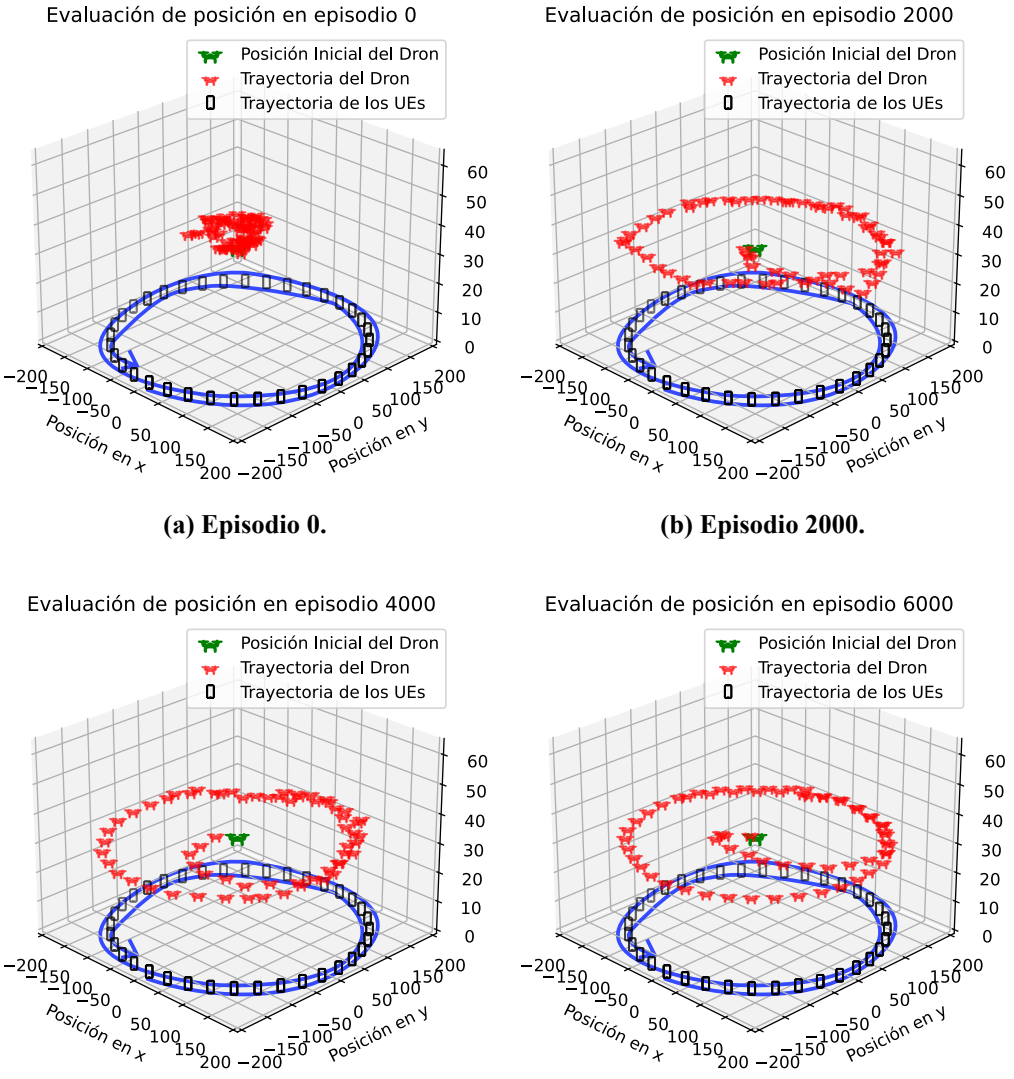
Figura 5.5: Comparación de métricas en diferentes inicializaciones

como al episodio 2000 ya es capaz de seguir a los UEs, el resto de episodios hasta el final, como pasaba anteriormente, los utiliza para ir refinando las acciones que tiene que decidir.

Una representación más detallada de la distancia y la recompensa media se encuentra en la Figura 5.7, donde se observa lo comentado previamente. En el episodio 2000 el agente ya ha aprendido a realizar un seguimiento de los usuarios, pero como sucedía en el caso de movimento lineal, esta política todavía se puede mejorar para acercarse a la óptima.

5.2.2.2. Inicialización Aleatoria

A continuación pasamos a hacer el experimento en el que el agente se inicializa de manera aleatoria. Sin embargo, para evaluar, comparamos que pasaría si posicionamos al agente en diferentes posiciones. En la Figura 5.7, se observan dichos valores además de compararse con el de la inicialización estática.



(a) Episodio 0.

(b) Episodio 2000.

(c) Episodio 4000.

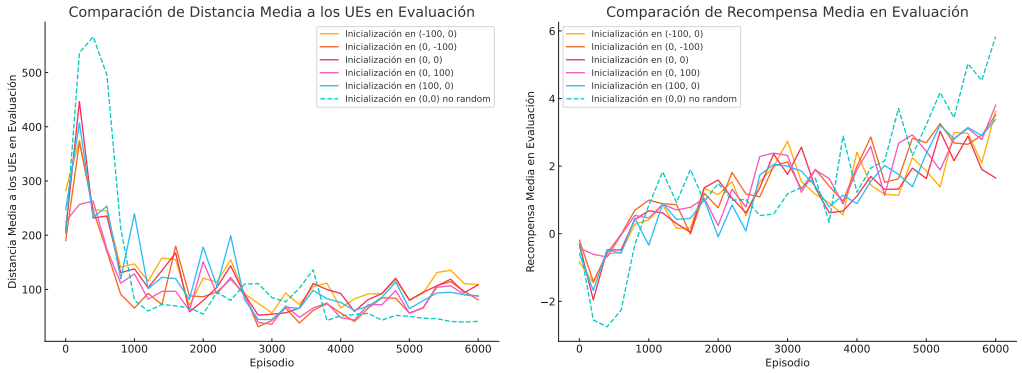
(d) Episodio 6000.

Figura 5.6: Visualización de episodios con movimiento circular con inicialización en (0,0).

Tras observar los resultados, se concluye que para poder realizar un buen bucle de entrenamiento, y que el agente pueda aprender a llegar desde todas las posiciones, con la misma rapidez con la que lo hace en inicialización estática, se debería entrenar durante más episodios.

5.3. Experimentos de Ablación

El ajuste de los parámetros es una es una ardua tarea que requiere una gran cantidad de experimentos. Los parámetros implementados en las tablas presentadas anteriormente, Tablas 4.2 y 4.3, son los definitivos que obtuvimos después de realizar múltiples entrenamientos. Esta sección explica cómo afectan los diferentes parámetros a la convergencia del algoritmo PPO.



(a) Comparación de Distancia Media a los UEs en Evaluación.

(b) Comparación de Recompensa Media en Evaluación.

Figura 5.7: Comparación de métricas en diferentes inicializaciones, incluyendo inicialización en (0,0) no random.

5.3.1. Número de Capas y Neuronas

La configuración óptima, presentada en la Tabla 4.3, consiste en 3 capas ocultas con 128 neuronas por capa. Se ha observado que, con 2 capas ocultas y 64 neuronas por capa, el agente no puede seguir a los UEs debido a su limitada capacidad de aprendizaje. Por otro lado, con 4 capas ocultas y 256 neuronas por capa, aunque el agente puede seguir a los UEs, el tiempo de convergencia es mayor.

5.3.2. Tasa de Aprendizaje y Número de episodios

La tasa de aprendizaje es un parámetro crucial en el entrenamiento de una red neuronal. Con una tasa de aprendizaje de 1e-3, los cambios en la política son demasiado bruscos, impidiendo que el agente siga a los UEs. Contrariamente, con una tasa de aprendizaje de 1e-5, los cambios son demasiado pequeños y el agente tampoco puede seguir a los UEs. Una buena tasa de aprendizaje es 3e-4. Con respecto al número de episodios tenemos que un mayor número de episodios permite al agente resolver mejor el problema. Sin embargo, un número excesivo de episodios incrementa considerablemente el tiempo de entrenamiento. Se ha comprobado que, con 6000 episodios en el caso de los UEs móviles, el agente puede seguir a los UEs en casi todos los casos.

5.3.3. Entropía de Epsilon

Este hiperparámetro controla la entropía de la política y por ende, la exploración del agente. Un valor de 0.02 logra un buen equilibrio entre exploración y explotación. Valores como 0.1 o 0.001, hacen que el agente no pueda seguir a los UEs debido a una exploración excesiva o insuficiente, respectivamente.

Capítulo 6

Conclusiones

Esta trabajo ha abordado un problema relevante en el campo de las telecomunicaciones: el servicio en situaciones de emergencia y accidentes, como los desastres naturales. Utilizando un dron capaz de adaptarse de manera dinámica y eficiente a los movimientos cambiantes de los usuarios y a las condiciones del entorno.

Mediante un enfoque matemático, se ha definido el problema de optimización necesario para conseguir el objetivo: asegurar que los UEs tienen las mejores prestaciones. Es importante señalar que en la definición se han utilizando parámetros de entrada medibles por la estación base montada en el mismo. A diferencia de la literatura existente, donde se utilizan parámetros no medibles como la localización exacta de los usuarios. este trabajo se basa en datos realistas y accesibles (SINR, ángulo de llegada de la señal y posición actual del dron), pero también altamente variables y difíciles de predecir, justificando la elección del RL para la solución propuesta.

En esta investigación se ha descrito el estado del arte del RL, y se ha demostrado la potencia del DRL cuando se aplica al ámbito de las telecomunicaciones. En concreto, se ha desarrollado un agente capaz de aprender diferentes patrones de movimiento de los UEs, utilizando el algoritmo PPO, considerado uno de los más avanzados en la actualidad.

En más detalle, en este trabajo, se ha definido rigurosamente el problema de DRL-PPO, describiendo estados, acciones y la función de recompensa. Se ha trabajado intensamente en la adaptación del simulador Giulia, utilizado como entorno para aumentar su velocidad de ejecución y permitir su integración con la librería de torchrl. Se han comparado los resultados de PPO con la DQN, y se ha demostrado que PPO converge hacia mejores soluciones debido a su capacidad de tomar acciones en el espacio continuo. Se ha demostrado también cómo el algoritmo no solo es capaz de converger cuando se entrena y se evalúa en el mismo escenario y con la misma inicialización, sino que también es capaz de funcionar entrenando con posiciones iniciales del dron aleatorias.

Los resultados de este trabajo han demostrando cómo el dron puede proporcionar un servicio excelente a UEs en una posición fija, representando, por ejemplo, a un grupo de rescate en una zona montañosa sin cobertura. Asimismo, se ha demostrado la capacidad de los agentes para funcionar en situaciones con UEs en movimiento, como en un terremoto, donde las comunicaciones para los bomberos son esenciales y el dron debe seguirlos para ofrecer la mejor cobertura posible.

Estos escenarios, aunque puedan parecer alejados de la realidad, reflejan la necesidad de comunicaciones perfectas en emergencias para que las fuerzas de seguridad actúen de manera eficiente. En

resumen, este trabajo ha demostrado que el uso de drones optimizados mediante algoritmos avanzados de DRL puede mejorar significativamente las telecomunicaciones en situaciones críticas, aportando una solución innovadora y realista a los desafíos actuales en el campo.

Como futuras líneas de investigación en este trabajo, podríamos considerar el aumento de la variedad de movimientos sobre los que entrenamos a los agentes. Con el objetivo de constatar que estos algoritmos convergen independientemente del patrón de movimiento de los UEs. Además, se podrían realizar entrenamientos mezclando diferentes tipos de movimiento, comprobando así que también son capaces de seguir a UEs en escenarios mucho más complejos. Otro enfoque sería estudiar la generalización de estos algoritmos, por ejemplo, entrenando a los agentes en un caso en el que los UEs se desplazan horizontalmente y comprobar su desempeño si se cambia el movimiento a vertical. Para aumentar el nivel de complejidad de la red y asemejarnos más a entornos reales, se podría incrementar el número de clústers de usuarios y drones.

Bibliografía

[1] Richard S Sutton y Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Harrison Kurunathan et al. "Machine learning-aided operations and communications of unmanned aerial vehicles: A contemporary survey". En: *IEEE Communications Surveys & Tutorials* (2023).

[3] Sarah M Hamylton et al. "Evaluating techniques for mapping island vegetation from unmanned aerial vehicle (UAV) images: Pixel classification, visual interpretation and machine learning approaches". En: *International Journal of Applied Earth Observation and Geoinformation* 89 (2020), pág. 102085.

[4] Qianqian Zhang, Aidin Ferdowsi y Walid Saad. "Distributed generative adversarial networks for mmWave channel modeling in wireless UAV networks". En: *ICC 2021-IEEE International Conference on Communications*. IEEE. 2021, págs. 1-6.

[5] Dinh Thai Hoang et al. *Deep Reinforcement Learning for Wireless Communications and Networking: Theory, Applications and Implementation*. John Wiley & Sons, 2023.

[6] Faris B Mismar, Brian L Evans y Ahmed Alkhateeb. "Deep reinforcement learning for 5G networks: Joint beamforming, power control, and interference coordination". En: *IEEE Transactions on Communications* 68.3 (2019), págs. 1581-1592.

[7] Faroq Al-Tam, Noélia Correia y Jonathan Rodriguez. "Learn to schedule (LEASCH): A deep reinforcement learning approach for radio resource scheduling in the 5G MAC layer". En: *IEEE Access* 8 (2020), págs. 108088-108101.

[8] Giorgio Stampa et al. "A deep-reinforcement learning approach for software-defined networking routing optimization". En: *arXiv preprint arXiv:1709.07080* (2017).

[9] Ying Liu, Junjie Yan y Xiaohui Zhao. "Deep reinforcement learning based latency minimization for mobile edge computing with virtualization in maritime UAV communication network". En: *IEEE Transactions on Vehicular Technology* 71.4 (2022), págs. 4225-4236.

[10] Chi Harold Liu et al. "Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach". En: *IEEE Journal on Selected Areas in Communications* 36.9 (2018), págs. 2059-2070.

[11] Ruijin Ding, Feifei Gao y Xuemin Sherman Shen. "3D UAV trajectory design and frequency band allocation for energy-efficient and fair communication: A deep reinforcement learning approach". En: *IEEE Transactions on Wireless Communications* 19.12 (2020), págs. 7796-7809.

[12] Rasmus V Rasmussen y Michael A Trick. “Round robin scheduling–a survey”. En: *European Journal of Operational Research* 188.3 (2008), págs. 617-636.

[13] Ronald A Howard. “Dynamic programming and markov processes.” En: (1960).

[14] Matthijs TJ Spaan. “Partially observable Markov decision processes”. En: *Reinforcement learning: State-of-the-art*. Springer, 2012, págs. 387-414.

[15] Richard Bellman. “Dynamic programming”. En: *science* 153.3731 (1966), págs. 34-37.

[16] Xu Wang et al. “Deep reinforcement learning: A survey”. En: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[17] Thomas M Moerland et al. “Model-based reinforcement learning: A survey”. En: *Foundations and Trends® in Machine Learning* 16.1 (2023), págs. 1-118.

[18] Sinan Çalışır y Meltem Kurt Pehlivanoğlu. “Model-free reinforcement learning algorithms: A survey”. En: *2019 27th signal processing and communications applications conference (SIU)*. IEEE. 2019, págs. 1-4.

[19] Christopher JCH Watkins y Peter Dayan. “Q-learning”. En: *Machine learning* 8 (1992), págs. 279-292.

[20] Melanie Coggan. “Exploration and exploitation in reinforcement learning”. En: *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University* (2004).

[21] Richard S Sutton. “Learning to predict by the methods of temporal differences”. En: *Machine learning* 3 (1988), págs. 9-44.

[22] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. En: *Advances in neural information processing systems* 12 (1999).

[23] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. En: *Machine learning* 8 (1992), págs. 229-256.

[24] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” En: *Psychological review* 65.6 (1958), pág. 386.

[25] Andrea Apicella et al. “A survey on modern trainable activation functions”. En: *Neural Networks* 138 (2021), págs. 14-32.

[26] David E Rumelhart, Geoffrey E Hinton y Ronald J Williams. “Learning Internal Representations by Error Propagation, Parallel Distributed Processing, Explorations in the Microstructure of Cognition, ed. DE Rumelhart and J. McClelland. Vol. 1. 1986”. En: *Biometrika* 71 (1986), págs. 599-607.

[27] Wayne Xin Zhao et al. “A survey of large language models”. En: *arXiv preprint arXiv:2303.18223* (2023).

[28] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. En: *nature* 529.7587 (2016), págs. 484-489.

[29] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. En: *arXiv preprint arXiv:1312.5602* (2013).

[30] Hado Van Hasselt, Arthur Guez y David Silver. “Deep reinforcement learning with double q-learning”. En: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 30. 1. 2016.

[31] Ziyu Wang et al. “Dueling network architectures for deep reinforcement learning”. En: *International conference on machine learning*. PMLR. 2016, págs. 1995-2003.

[32] Tom Schaul et al. “Prioritized experience replay”. En: *arXiv preprint arXiv:1511.05952* (2015).

[33] Timothy P Lillicrap et al. “Continuous control with deep reinforcement learning”. En: *arXiv preprint arXiv:1509.02971* (2015).

[34] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. En: *International conference on machine learning*. PMLR. 2016, págs. 1928-1937.

[35] John Schulman et al. “Trust region policy optimization”. En: *International conference on machine learning*. PMLR. 2015, págs. 1889-1897.

[36] John Schulman et al. “Proximal policy optimization algorithms”. En: *arXiv preprint arXiv:1707.06347* (2017).

[37] Georg Fischer, Florian Pivit y Werner Wiesbeck. “EISL, the Pendant to EIRP: A Measure for the Receive Performance of Base Stations at the Air Interface”. En: *Proc. European Microwave Week*. Milan, Italy, 2002, págs. 1-4.

[38] Holger Claussen et al. *Small cell networks: deployment, management, and optimization*. John Wiley & Sons, 2017.

[39] Xiaoli Chu et al. *Heterogeneous cellular networks: theory, simulation and deployment*. Cambridge University Press, 2013.

[40] John S Seybold. *Introduction to RF propagation*. John wiley & sons, 2005.

[41] A. Goldsmith. *Wireless Communications*. Cambridge University Press, 2005.

[42] 3GPP TR 36.814. *Evolved Universal Terrestrial Radio Access (E-UTRA); Further Advancements for E-UTRA Physical Layer Aspects*. Inf. téc. v 9.0.0.

[43] Claude Elwood Shannon. “A mathematical theory of communication”. En: *The Bell system technical journal* 27.3 (1948), págs. 379-423.

[44] M Series. “Guidelines for evaluation of radio interface technologies for IMT-Advanced”. En: *Report ITU* 638.31 (2009).

[45] NumPy Developers. *Code Explanations*. `https://numpy.org/doc/stable/dev/internals.code-explanations.html`. Accessed: 2024-06-17.

[46] PyTorch Developers. *PyTorch Documentation*. `https://pytorch.org/docs/stable/index.html`. Accessed: 2024-06-17.

[47] NumPy Developers. *Numpy Einsum*. `https://numpy.org/doc/stable/reference/generated/numpy.einsum.html`. Accessed: 2024-06-17.

[48] Alan H Barr. “The Einstein summation notation”. En: *An Introduction to Physically Based Modeling (Course Notes 19), pages E* 1 (1991), pág. 57.

[49] Hado Van Hasselt et al. “Deep reinforcement learning and the deadly triad”. En: *arXiv preprint arXiv:1812.02648* (2018).

[50] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. En: *arXiv preprint arXiv:1506.02438* (2015).

[51] Pytorch. *Pytorchrl*. `https://pytorch.org/rl/stable/index.html`. Accessed: 2024-06-17.