Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Optimización de IRS mediante Aprendizaje por Refuerzo para Redes 6G

Alumno: Pedro Cidoncha Molina

Tutores: Diego Martín de Andrés
Francisco Hernando Gallego

Fecha: 10 de julio de 2025

Agradecimientos

Quiero agradecer, en primer lugar, a mis amigos, por estar ahí siempre que los he necesitado, por todo el apoyo durante estos años de carrera. Sé que cuando los vuelva a necesitar, seguirán ahí, y por eso, más que unos amigos, me llevo una familia.

A mi hermana, por su cariño y apoyo constante. Y sobre todo a mis padres, porque ellos son quienes me han brindado esta oportunidad. Siempre lo han dado todo por mí, por ofrecerme un futuro lleno de posibilidades y por mi felicidad. Han estado a mi lado en cada paso del camino, y sé que siempre los tendré ahí, acompañándome. Este trabajo también es fruto vuestro.

Finally, I would also like to thank my tutors Diego, Fran, and especially Masoud Kaveh for all the support I have received, their patience, and for giving me this excellent opportunity.

Resumen

Este proyecto presenta el diseño y desarrollo de un sistema de optimización inteligente para redes 6G mediante técnicas de aprendizaje por refuerzo profundo, concretamente utilizando el algoritmo Deep Deterministic Policy Gradient (DDPG). El objetivo es maximizar la tasa de secreto en un sistema de comunicación asistido por una superficie reflectante inteligente (IRS), en un entorno donde el canal de transmisión no es completamente conocido, reflejando así condiciones más realistas.

A lo largo del trabajo se ha desarrollado un entorno de simulación, una arquitectura modular del agente, diversas funciones de recompensa adaptadas a la seguridad física, y una interfaz por consola que permite entrenar, evaluar y comparar modelos de manera interactiva. Además, se han propuesto criterios claros de evaluación y se ha demostrado la estabilidad y eficacia del modelo frente a métodos tradicionales como AO (Alternating Optimization) y SDR (Semidefinite Relaxation), los cuales requieren conocimiento total del canal.

Los resultados muestran que el enfoque basado en DDPG permite una mejora sustancial de la tasa de secreto, adaptándose dinámicamente a distintos escenarios sin necesidad de información completa del canal, lo que lo convierte en una solución prometedora para redes de próxima generación.

Palabras claves: Aprendizaje profundo por refuerzo, DDPG, Superficies reflectantes inteligente (IRS), Seguridad física, Redes 6G.

Abstract

This project presents the design and development of an intelligent optimisation system for 6G networks using deep reinforcement learning techniques, specifically using the Deep Deterministic Policy Gradient (DDPG) algorithm. The objective is to maximise the secrecy rate in a communication system assisted by an intelligent reflecting surface (IRS), in an environment where the transmission channel is not completely known, thus reflecting more realistic conditions.

Throughout the work, a simulation environment, a modular agent architecture, various reward functions adapted to physical security, and a console interface that allows models to be trained, evaluated, and compared interactively have been developed. In addition, clear evaluation criteria have been proposed and the stability and effectiveness of the model has been demonstrated compared to traditional methods such as AO (Alternating Optimisation) and SDR (Semidefinite Relaxation), which require full knowledge of the channel.

The results show that the DDPG-based approach allows for a substantial improvement in the secrecy rate, dynamically adapting to different scenarios without the need for complete channel information, making it a promising solution for next-generation networks.

Keywords: Deep reinforcement learning (DRL), DDPG, Intelligent reflective surfaces (IRS), Physical security, 6G networks.

Índice general

Índice de figuras

Índice de tablas

Parte I

Descripción del proyecto

Capítulo 1

Introducción

Este proyecto, desarrollado en colaboración con la **Universidad de Aalto, Finlandia** *bajo la guía de* ***Masoud Kaveh****, investiga el uso de superficies reflectantes inteligentes (IRS) para mejorar la tasa de secreto en comunicaciones inalámbricas. Para resolver el problema de optimización asociado a la configuración de las IRS, se utilizará el algoritmo* **Deep Deterministic Policy Gradient (DDPG)***, un enfoque de aprendizaje profundo por refuerzo (DRL).*

Este trabajo busca contribuir al desarrollo de soluciones avanzadas para la seguridad en redes inalámbricas, explorando el potencial de la inteligencia artificial en la optimización de sistemas de comunicación.

1.1. Planteamiento del problema (*Problem Statement*)

En las redes de comunicación inalámbricas tradicionales, garantizar la confidencialidad de la información transmitida es un desafío crítico. Normalmente, en estos sistemas, tenemos un transmisor (llamémoslo Alice, a menudo en una estación base "BS") que envía información a un receptor (que llamaremos Bob). Pero siempre hay un fisgón (al que vamos a llamar Eve) que intenta interceptar ilegalmente la información destinada a Bob.

A diferencia de las técnicas criptográficas convencionales, que dependen de claves secretas y protocolos de cifrado, la **seguridad física en la capa de comunicación** (Phisical Layer Security, PLS) busca mejorar la seguridad aprovechando las propiedades del canal inalámbrico.

En este contexto entran en juego las **Superficies Reflectantes Inteligentes** (IRS por sus siglas en inglés), una nueva tecnología prometedora que ha surgido para mejorar la tasa de secreto en las comunicaciones inalámbricas. Las IRS consisten en superficies reconfigurables con elementos pasivos que pueden modificar la fase y dirección de las ondas electromagnéticas, permitiendo así optimizar la propagación de la señal a Bob mientras se degrada la recepción en Eve. Sin embargo, encontrar la configuración óptima de los elementos del IRS para maximizar la tasa de secreto representa un problema de optimización complejo y no trivial.

Los enfoques tradicionales para resolver este problema incluyen técnicas de optimización convexa y no convexa, como la optimización alternada (AO), la relajación semidefinida (SDR), la aproximación convexa sucesiva (SCA) y la propagación fraccionaria (FP). Sin embargo, estos métodos suelen requerir modelos precisos del canal y tienen una alta complejidad computacional, lo que limita su aplicabilidad en escenarios dinámicos o con incertidumbre en la información del canal.

En este documento se abordará una alternativa más flexible y adaptable, el uso de **aprendizaje profundo por refuerzo** (DRL, Deep Reinforcement Learning), que permite aprender políticas óptimas para la configuración de superficies IRS sin necesidad de modelos explícitos del canal. En particular, el algoritmo **Deep Deterministic Policy Gradient** (DDPG) ha demostrado ser una técnica efectiva para problemas de optimización en entornos continuos y de alta dimensionalidad, lo que lo hace ideal para este caso en particular.

Por lo tanto, en este documento se afrontará la solución del problema de la mejora de la tasa de secreto desde el punto de vista del DRL con el algoritmo DDPG para la optimización dinámica de las IRS. El objetivo es evaluar su desempeño y analizar su viabilidad en diferentes escenarios de comunicación.

1.2. Objetivos

El objetivo principal de este proyecto es el desarrollo, comprensión e implementación de un modelo de inteligencia artificial de aprendizaje profundo con refuerzo basado en DDPG para la optimización de la configuración de una superficie IRS con el fin de maximizar la tasa de secreto en un sistema de comunicación inalámbrica.

Para lograr esto, estableceremos unos objetivos más pequeños y específicos:

- Definir y modelar el problema de optimización de la tasa de secreto, considerando la situación antes planteada, es decir, teniendo en cuenta la presencia de un atacante pasivo Eve y el impacto de la configuración del IRS.
- Diseñar e implementar un modelo basado en DDPG que permita aprender políticas óptimas de reflexión del IRS sin necesidad de un modelo explícito del canal.
- Evaluzar el desempeño del modelo mediante simulaciones, comparándolo con enfoques convencionales de optimización convexa y no convexa.
- Analizar la viabilidad del modelo en distintos escenarios, considerando distintas variaciones.

1.3. Condicionantes

En el desarrollo de este proyecto nos tendremos que adaptar a una serie de condicionantes que pueden influir en la implementación y evaluación del modelo:

1. **Disponibilidad de simulaciones:** El modelo debe ser evaluado en entornos de simulación debido a la falta de acceso a hardware real con IRS. Por tanto, aunque siempre se intentará hacer de la forma más fiel posible a la realidad, la calidad de los resultados, dependerá en cierto modo de la fidelidad del modelo de canal utilizado.
2. **Tiempo de desarrollo:** No disponemos de tiempo indefinido para la elaboración del proyecto, debemos completarlo en un marco temporal definido. Esto limita la exploración de múltiples variantes de algoritmos y configuraciones.
3. **Precisión del modelo de canal:** aunque en escenarios reales la información de los canales entre Alice, Bob y Eve puede estar sujeta a incertidumbre, en este proyecto asumiremos que la información de estos canaes es acesible en la simulación,

4. **Fiabilidad de los resultados:** La variabilidad inherente a los algoritmos de DRL puede hacer que los resultados dependan de la inicialización y de los hiperparámetros, por lo que será necesario realizar múltiples ejecuciones para obtener conclusiones sólidas, y esto va de la mano con el siguiente condicionante.

5. **Limitaciones computacionales:** El entrenamiento de algoritmos de aprendizaje profundo por refuerzo (DRL) requiere un alto poder de cómputo. Por tanto tendremos que adaptar la complejidad del modelo y la cantidad de iteraciones de entrenamiento al hardware del que dispongamos.

1.4. Alcance del proyecto

Este proyecto se limita al diseño y validación, **en simulación**, de un agente DDPG que ajuste dinámicamente una superficie reflectante inteligente (IRS) para maximizar la tasa de secreto en un enlace transmisor-receptor (Alice-Bob) frente a un oyente pasivo (Eve). No contempla sin embargo ninguna integración harware ni pruebas sobre bancos de ensayo físicos.

El trabajo abarcará los siguientes aspectos:

1. Modelado matemático
   - Formulación de la tasa de secreto con enlaces directos y reflejados.
   - Definición del espacio de estados (canales, potencias, configuraciones IRS) y acciones (fases de los elementos IRS).
2. Entorno de simulación en Python
   - Generación de canales inalámbricos (Rayleigh/Rician) y ruido térmico.
   - Implementación modular con *NumPy* pra la física del canal y *TensorFlow/Keras* para el DRL.
3. Desarrollo del agente DDPG
   - Actor-crítico con redes neuronales profundas.
   - Replay buffer, target networks y políticas de exploración adecuadas a un espacio de acción continuo (Fases).
   - Desarrollo de una función de recompensa adecuada basada en la tasa de secreto instantánea.
4. Campaña de experimentos
   - Entrenamiento y validación bajo distintos números de elementos IRS, potencias de transmisión y posiciones de Eve.
   - Análisis de sensibilidad a la estimación imperfecta del canal
5. Benchmarking
   - Implementación de almentos métodos clásicos como Alternating Optimization (AO) y SDR.

   - Comparación en términos de tasa de secreto promedio, convergencia y coste computacional.

6. Documentación y entregables
   - Código comentado y reproducible (repositorio Git)
   - Informe técnico con gráficos, discousión de resultados y limitaciones
   - Presentación final con conclusiones y lineas futuras.

Fuera del alcace de este proyecto están:

- Diseño o frabricación de hardware IRS.
- Medidas en entornos reales OTA (Over-the-Air)
- Protocolos de cifrado o capas superiores (capa MAC/Aplicación)

1.4.1. Entregables principales

1. Repositorio GitHub con entorno de simulación, agente DDPG y sripts de entrenamiento/prueba al igual que una pequeña interfaz para su uso en consola. También contendrá implementaciones de los métodos de referencia.
2. Informe técnico con descripción del modelo, elección de hiperparámetros y metodología experimiental, además de resultados comparativos y análisis críticos.

1.5. Estructura de la memoria

La presente memoria se estructura en distintos capítulos que reflejan las fases de desarrollo del proyecto, desde su planteamiento teórico hasta su implementación práctica y evaluación experimental. A continuación, se describe brevemente el contenido de cada uno de ellos:

- **Capítulo 1 - Introducción:** Presenta el contexto del proyecto, sus objetivos principales, la motivación detrás del estudio y la colaboración con la Universidad de Aalto, así como una introducción al problema de seguridad en comunicaciones inalámbricas y la propuesta de utilizar IRS con aprendizaje por refuerzo.
- **Capítulo 2 - Planificación:** Se detalla la planificación temporal y técnica llevada a cabo a lo largo del proyecto, incluyendo fases de desarrollo, cronograma, herramientas utilizadas, y la gestión del versionado mediante GitHub.
- **Capítulo 3 - Diseño del entorno de simulación:** Recoge el marco teórico necesario para entender el proyecto. Se describen los fundamentos de la seguridad física en redes, el funcionamiento de las superficies reflectantes inteligentes (IRS), y las bases del aprendizaje por refuerzo profundo, incluyendo el algoritmo DDPG.
- **Capítulo 4 - Desarrollo de la solución:** Es el núcleo del trabajo. Aquí se describe con detalle la implementación del entorno de simulación, el diseño del agente de aprendizaje, las funciones de recompensa desarrolladas, el proceso de entrenamiento y evaluación, la interfaz interactiva por consola, y la evolución del modelo a través de distintas versiones.

- **Capítulo 5 - Conclusiones y validación del modelo:** Se presentan las conclusiones finales del proyecto, así como los criterios utilizados para validar los entrenamientos. Además, se incluyen comparativas con otros métodos clásicos como AO y SDR, y se discuten las ventajas del enfoque propuesto, junto con posibles líneas de trabajo futuro.

Capítulo 2

Planificación

2.1. Metodología

El desarrollo de este trabajo combina simultáneamente la formulación de hipótesis científicas y la implementación de soluciones basadas en algoritmos de aprendizaje profundo por refuerzo (DRL), en particular el agente DDPG. Este tipo de tareas implica una evolución continua del modelo a través de fases de diseño, experimentación, análisis y validación. Dado el carácter iterativo e incierto de este proceso, se adopta una **metodología híbrida ágil–investigación**, especialmente adaptada a la naturaleza exploratoria y técnica del proyecto.

2.1.1. Marco metodológico y enfoque iterativo

La metodología seleccionada se fundamenta en ciclos de desarrollo cortos, de entre una y dos semanas, cada uno con objetivos claros y entregables concretos. Esta estructura permite evaluar de forma temprana y continua las decisiones de diseño, detectar fallos conceptuales o técnicos, y adaptar el rumbo en función de los resultados obtenidos.

Cada iteración sigue la siguiente estructura:

1. **Definición del objetivo:** Se establece una meta clara y medible para el ciclo, alineada con hipótesis previas o necesidades técnicas (por ejemplo, probar una nueva formulación de la recompensa o aumentar el número de elementos IRS).

2. **Desarrollo técnico:** Se codifican las modificaciones necesarias en el entorno, el agente o los hiperparámetros, utilizando TensorFlow/Keras, NumPy y Jupyter como apoyo.

3. **Entrenamiento y validación:** El agente es entrenado, y se registran métricas como tasa de secreto, convergencia, estabilidad de la política, entre otras.

4. **Documentación y decisión:** Cada iteración genera un entregable técnico (código, e informe detallado) y una reflexión final que guía si se continúa, se pivota o se refina el enfoque.

Este esquema permite responder de forma ágil a preguntas de investigación, evaluar alternativas de forma empírica, y mantener la coherencia y trazabilidad del avance técnico-científico.

2.1.2. Tipos de entregables y criterios de evaluación

Los entregables generados al cierre de cada ciclo constituyen unidades funcionales que pueden reutilizarse o evolucionar en ciclos posteriores. Estos incluyen:

- **Código funcional versionado y probado**
- **Documento de ejecución de entrenamiento** con gráficas y tablas de rendimiento
- **Informe técnico breve** con resultados cuantitativos y análisis
- **Informe comparativo** (benchmarking contra métodos AO, SDR, etc)
- **Repositorio versionado** con commit etiquetado y resumen de la iteración

Los criterios de éxito por iteración incluyen:

- Cumplimiento del objetivo propuesto
- Mejora o comportamiento razonable de la métrica clave
- Calidad y reutilizabilidad del entregable generado
- Registro y justificación de decisiones para futuras iteraciones

2.1.3. Gestión del conocimiento y toma de decisiones

El seguimiento de todos los entrenamientos se ha realizado mediante un documento centralizado en el que para cada ejecución del agente DDPG se registraban:

- **Nombre del entrenamiento.** Un nombre significativo que comenzaba con la palabra 'Model', después se indicaba la versión del código tras un guión bajo ('_V2'), la tanda de ejecución de ese modelo tras un guión ('-3'), y finalmente partes descriptivas específicas del entrenamiento, por ejemplo, si tomamos el entrenamiento *Model_ V5-3_ L3-3000*, este nombre indica:
  - El entrenamiento corresponde a la versión 5 del código.
  - El entrenamiento corresponde a la tercera tanda de ejecución y ajuste de ese código.
  - *'L3'* indica que el parámetro λ de la recompensa toma el valor de 3.
  - Este entrenamiento se ha hecho con 3000 episodios de entrenamiento.
- **Características del experimento:** Se anotaban parámetros significativos de ese experimento, como son, la recompensa usada, el número de episodios que se van a usar, la tasa de ε-decay que vamos a tomar, el tipo de exploración, etc.
- **Gráficas del entrenamiento:** Gráficas que indicaban la evolución de:
  - El actor_loss
  - El critic_loss
  - La recompensa
  - La tasa de secreto

durante el entrenamiento, además, para recompensas más complejas que se iban adaptando dinámicamente durante el entrenamiento también se recogían gráficas que mostraban la evolución de los parámetros de recompensa durante el proceso de entrenamiento. Además al final de las pruebas se impementó una gráfica a modo de mapa de calor para determinar la posición y fase de los elementos de la IRS al igual que de las antenas de la base tras la optimización.

- **Análisis de gráficas y pruebas:** Se analizaban detenidamente las gráficas del experimento al igual que una serie de prueba a las que se sometía, comparando objetivos como la convergencia, la comparación con parámetros anteriores, la mejora de la tasa de secreto, etc.

- **Conclusión del experimento:** Párrafo final expresando si el entrenamiento había cumplido con los criterios de mejora, los futuros caminos que se podrían tomar a raíz de ese entrenamiento y el descarte o aprobación de ciertos parámetros

Este documento sirve como memoria viva de cada iteración: facilita la trazabilidad de cambios, permite recuperar rápidamente configuraciones exitosas y apoya la toma de decisiones basadas en evidencia empírica.

2.1.4. Beneficios del enfoque adoptado

A lo largo del desarrollo, esta metodología ha demostrado su utilidad para gestionar la complejidad técnica y la incertidumbre científica del proyecto. Entre los principales beneficios destacan:

- **Incrementalidad:** Cada bloque del sistema (simulador, agente, entorno, recompensas) fue desarrollado y validado por separado antes de ser integrado.

- **Flexibilidad ante hallazgos:** Se permitió la rápida adaptación ante comportamientos inesperados, como el sobreajuste o efectos secundarios no deseados (como penalizaciones excesivas a la señal en Bob).

- **Menor retrabajo:** La estructura modular y el enfoque iterativo minimizaron la necesidad de rehacer código o experimentos desde cero.

- **Transparencia y trazabilidad:** La estructura modular y el enfoque iterativo minimizaron la necesidad de rehacer código o experimentos desde cero.

2.1.5. Síntesis metodológica

En resumen, la combinación de un enfoque iterativo, orientado a objetivos y enriquecido con herramientas de seguimiento y análisis, permite el desarrollo de un modelo DDPG robusto, flexible y científicamente validado. La metodología híbrida agil-investigación no solo permite guiar el desarrollo técnico, sino que también permite estructurar la generación del conocimiento, el análisis de resultados y la toma de decisiones informadas

2.2. Planificación temporal

El desarrollo de este trabajo se ha estructurado mediante un enfoque modular, iterativo y basado en ciclos cortos, siguiendo una metodología híbrida de investigación y desarrollo ágil. Esta planificación temporal tiene como objetivo organizar las actividades clave del proyecto, establecer un marco de trabajo realista y escalable, y asegurar la coherencia entre el modelo teórico, la implementación técnica y la validación empírica del sistema propuesto.

Cada fase del proyecto se ha planificado como un bloque funcional autocontenible, con entregables concretos, métricas de éxito asociadas y criterios de avance. Esta organización modular permite gestionar la complejidad inherente al diseño de agentes de aprendizaje profundo por refuerzo (DRL) en entornos de comunicaciones inteligentes, con requisitos tanto de rigor científico como de solidez en la implementación.

2.2.1. Estructura modular del proyecto

El proyecto se ha dividido en los siguientes 8 módulos funcionales:

Módulo	**Iteraciones (aprox.)**	**Entregables**	**Métrica clave**
Modelado matemático	1-2	Formulación de la tasa de secreto	Coherencia formal
Simulador del entorno	2	Funciones para simulación del canal y de todos los elementos relevantes del entorno	Fiabilidad del entorno
Arquitectura DDPG	3-4	Implementación del modelo actor-crítico	Entrenabilidad
Función de recompensa	2	Formulación e implementación de una función de recompensa basada en la tasa de secreto	Estabilidad de señal
Entrenamientos iniciales	2	Agente con rendimiento superior al aleatorio	Tasa de secreto media
Comparación con AO/SDR	2	Informe de benchmarking	Delta en rendimiento
Análisis de escenarios	2	Pruebas con distintas configuraciones	Robustez
Documentación final	Contínuo	Código y documentación reproducible	Completitud

Tabla 2.1: Estructura modular del proyecto

Cada uno de estos bloques está diseñado para avanzar de forma incrementas, permitiendo que

nuevos descubrimientos y mejoras puedan integrarse sin necesidad de rediseñar completamente el sistema.

2.2.2. Fases del desarrollo

Fase 1 - Fundamentos teóricos y modelado formal

Duración estimada: 2 semanas.

Actividades: En esta fase del desarrollo se espera hacer una revisión bibliográfica de los métodos de DRL aplicados a las comunicaciones inalámbricas, además de conseguir un modelado matemático fiel a la realidad de aspectos como el canal, IRS e interferencias de Eve, consiguiendo así una definición formal del problema a resolver, la tasa de secreto a mejorar y las condiciones límite.

Criterios de cierre: Obtención de unas fórmulas coherentes con la literatura, y validadas por revisión cruzada con una serie de papers base.

Fase 2 - Diseño del simulador del entorno

Duración estimada: 2 semanas.

Actividades: Se espera implementar un módulo capaz de simular con cierta calidad el entorno de ejecución real, implementando generadores Rayleigh con variación paramétrica para la simulación del canal, y conseguir así escenarios reproducibles con semilla fija y trazabilidad.

Criterios de cierre: Se espera obtener un módulo NumPy con funciones generadoras y un Jupyter Notebook de pruebas estadísticas.

Fase 3 - Arquitectura del agente DDPG

Duración estimada: 4 semanas.

Actividades: Implementación del agente con redes actor y crítico, además de la integración del *Replay Buffer* y *target networks*. Se espera obtener una cierta estabilidad inicial con una recompensa genérica (tomaremos la tasa de secreto sin transformar al inicio).

Criterios de cierre: Verificación de entrenabilidad básica en entorno simple y reducido.

Fase 4 - Diseño y testeo de funciones de recompensa

Duración estimada: 2 semanas.

Actividades: Formulación e implementación de diferentes funciones de recompensas con base en la tasa de secreto. Priorizar las penalizaciones por falta de convergencia o por comportamiento errático del agente en el entrenamiento. Y finalmente, una vez formuladas e implementadas las funciones de recompensa, pruebas con diferentes valores de parámetros, tolerancias y shaping progresivo.

Criterios de cierre: Estabilidad de la señal y mejora consistente durante el entrenamiento.

Fase 5 - Entrenamientos iniciales y validación empírica

Duración estimada: 2 semanas.

Actividades: Primeros entrenamientos completos con baseline aleatorios registrando las pérdidas, la evolución de la recompensa y la tasa de secreto durante el proceso. Hacer una validación estadística de mejora con respecto al azar.

Criterios de cierre: Documento con informes, gráficas de evolución y análisis de la convergencia y estabilidad.

Fase 6 - Comparación con métodos clásicos (AO/SDR)

Duración estimada: 2 semanas.

Actividades: Lectura de papers sobre la implementación de estos algoritmos para la resolución del problema y posible implementación base de ambos algoritmos para hacer una evaluación comparativa de la tasa de secreto en igualdad de condiciones. Realización de benchmark con distintos valores de N (elementos de IRS) y ruido.

Criterios de cierre: Delta de rendimiento frente a métodos deterministas.

Fase 7 - Análisis de escenarios y robustez

Duración estimada: 2 semanas.

Actividades: Modificación de parámetros como la posición de Eve y el número de elementos IRS, analizando posteriormente el comportamiento del agente mediante distintas métricas, al igual que la visualización con mapas de calor de la distribución final de fases.

Criterios de cierre: Documento comparativo e informe de robustez y generalización.

Fase 8 - Documentación y reproducibilidad

Duración estimada: Contínuo.

Actividades: Realización de una documentación mínima por iteración (Markdown, commits), al igual que la realización de un código comentado y modularizado y un registro de configuraciones en una tabla centralizada.

Criterios de cierre: El cierre de esta fase supondrá la finalización del proyecto.

2.2.3. Diagrama de Gantt

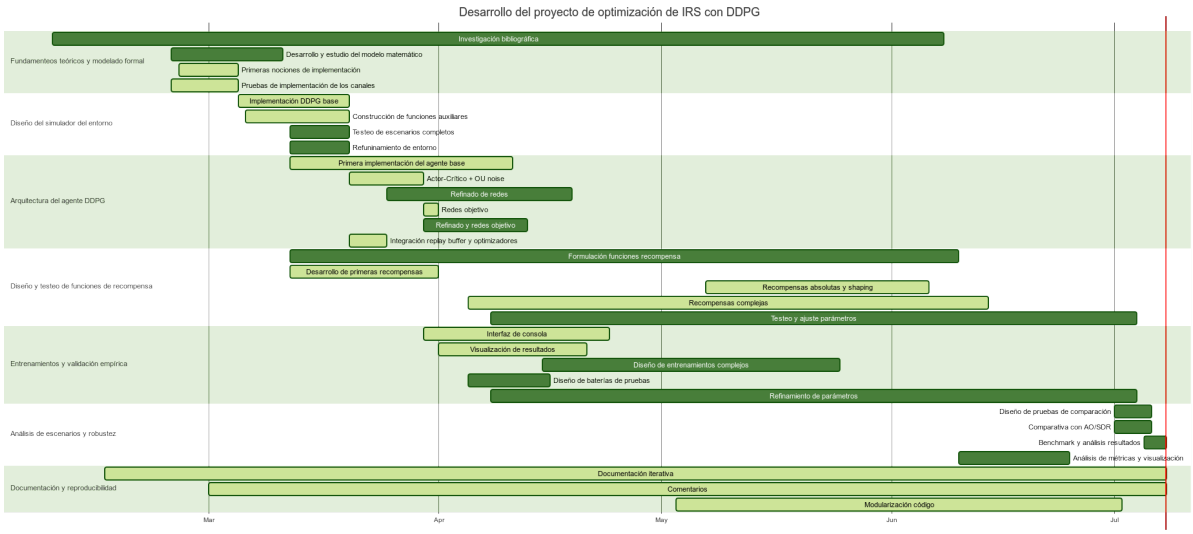

Figura 2.1: Diagrama de Gantt de planificación temporal

2.3. Presupuestos

El desarrollo de este proyecto ha implicado una combinación de trabajo técnico especializado, uso intensivo de recursos computacionales y herramientas software específicas para aprendizaje profundo por refuerzo. A continuación, se detallan los costes estimados asociados a cada categoría relevante.

2.3.1. Recursos humanos

El principal coste del proyecto reside en el tiempo dedicado por mí, el autor, como parte del programa formativo del grado. Para estimar de forma más precisa el esfuerzo técnico y el coste asociado al tiempo invertido, se ha empleado el método **Wideband Delphi**.

Este método consiste en la consulta iterativa y anónima a expertos para obtener estimaciones consensuadas y reducir sesgos. En este caso, se realizaron varias rondas de estimación junto con mis tutores y el experto Masoud Kaveh, quienes aportaron su experiencia en proyectos similares.

Tras varias iteraciones, se consensuaron las siguientes estimaciones de horas para cada actividad del proyecto, tomando un coste hora simbólico de 20 euros/hora, habitual en consultoría técnica junior en proyectos de investigación aplicada.

Actividad	**Horas estimadas**	**Subtotal**
Estudio y revisión bibliográfica	40 h	800 euros
Desarrollo del simulador del entorno	30 h	600 euros
Implementación del agente DDPG	70 h	1400 euros

Actividad	Horas estimadas	Subtotal
Diseño y prueba de recompensas	45 h	900 euros
Entrenamiento y ajuste de modelos	50 h	1000 euros
Análisis de resultados y testing	30 h	600 euros
Redacción y documentación	50 h	1000 euros
Total estimado	**305 h**	**6100 euros**

Tabla 2.2: Presupuesto de recursos humanos basado en método Wideband Delphi

2.3.2. Recursos hardware

Durante el desarrollo de este proyecto se ha hecho uso de dos recursos hardware principalmente:

Ordenador personal

Uso: Este se ha usado en las primeras fases del desarrollo, cuando todavía no tenía disponibilidad de la máquina virtual asignada por la universidad, para entrenar las primeras versiones del modelo. Y durante todo el proyecto para programación local, preprocesado de datos, redacción de documentos y validación rápida de pruebas.

Características: CPU Intel i5, 12GB RAM, GPU NVIDIA GTX 1050 (4GB)

Coste imputado: No he considerado depreciación ni coste de amortización directa por ser un equipo personal ya disponible. Se estima un uso parcial del 30 % del tiempo del proyecto.

Máquina virtual del departamento de informática

Uso: Este equipo se ha usado como motor principal de entrenamiento intensivo de los modelos DRL.

Características: CPU Intel Xeon E5-2630 v4 @ 2.2GHz, 9,4 GB RAM, Sin GPU

Coste real: No me ha supuesto costes ya que la universidad me la ha facilitado; sin embargo, se estima que el coste en el mercado de una máquina virtual con características similares a esta podría rondar entre 0,3 y 1 euro/hora de uso intensivo.

Coste estimado: Se estima que una estimación del coste del uso de este recurso en el mercado es de entre 500 y 800 euros, sin embargo, no lo voy a incluir en el coste real ya que es un recurso proporcionado por la universidad.

2.3.3. Recursos software

Durante este proyecto se ha utilizado únicamente software libre y de código abierto, por lo que no se han generado costes por licencias. Cabe destacar que la elección de herramientas abiertas ha permitido mantener el presupuesto a cero en esta categoría, pero siempre sin sacrificar funcionalidad ni calidad.

2.3.4. Conclusión

Finalmente, como resumen, podemos ver que los costes reales de este proyecto son la suma de los anteriores:

- Coste de recursos humanos: 6100 euros
- Coste de recursos hardware: 0 euros (recurso cedido)
- Coste de recursos software: 0 euros
- **Coste final:** 6100 euros

Concluimos entonces que el proyecto ha sido desarrollado con una alta eficiencia económica, optimizando el uso de recursos institucionales y software libre. El único coste relevante es el asociado al trabajo del autor, cuyo valor estimado se corresponde con una dedicación intensiva y especializada a lo largo de varias semanas.

Gracias a la infraestructura proporcionada por la universidad, ha sido posible la ejecución de modelos complejos de aprendizaje profundo sin recurrir a soluciones comerciales, lo que reduce significativamente la barrera de entrada para futuras investigaciones similares.

2.4. Gestión de riesgos

En proyectos de investigación con fuerte componente técnico, como el presente trabajo centrado en la aplicación de Deep Reinforcement Learning para la optimización de la tasa de secreto en redes IRS, resulta fundamental anticipar los posibles riesgos que puedan afectar al cumplimiento de los objetivos, la planificación temporal o la calidad de los resultados obtenidos. En esta sección se identifican los principales riesgos asociados, se analiza su impacto potencial y se describen las medidas preventivas y de contingencia adoptadas.

2.4.1. Identificación de riesgos

Para la identificación de los riesgos relacionados con este proyecto se han tenido en cuenta enfoques basados en la experiencia personal y enfoques cualitativos basados en la revisión de documentos de proyectos sobre técnicas y modelos de DRL y ajuste de IRS. Los principales riesgos encontrados son los siguientes:

- **Riesgo 1 - No convergencia del modelo DDPG.** Este es uno de los riesgo técnico más relevantes, asociado a la alta sensibilidad del aprendizaje profundo por refuerzo, a la configuración de hiperparámetros, al diseño del modelo, a la implementación y modelado del entorno y a la mala elección de funciones de recopensa.
- **Riesgo 2 - Limitaciones computacionales.** Para ahorrar tiempo se entrenan varios modelos a la vez con diferentes configuraciones de hiperparámetros o función de recompensa, puede que en algún momento la carga computacional de los entrenamientos supere la capacidad disponible en la máquina virtual cedida.
- **Riesgo 3 - Retraso en el tiempo por bloqueos en la comunicación con tutores o con la universidad de Aalto.** Aunque este proyecto lo esté desarrollando exclusivamente

el autor, en ocasiones hay dudas que ya sea por tema de nivel de experiencia o nivel de conocimiento, hay que transmitir a tutores o colaboradores, y en caso de que alguna de estas cuestiones sea urgente y haya demora en alguna de las partes, esto puede resultar en un bloqueo temporal que llevará un correspondiente retraso temporal en la planificación.

- **Riesgo 4 - Caída temporal o fallo en la máquina virtual.** En caso de fallo eléctrico en el sistema de alimentación de la universidad de Valladolid, o caída temporal por alta afluencia en las máquinas virtuales, se perderá por completo el hardware utilizado como herramienta de entrenamiento de modelos.
- **Riesgo 5 - Fallo en la planificación y falta de tiempo para llegar a la entrega final.** En caso de ser demasiado optimista con la planificación del proyecto, puede que ya no haya tiempo al final para la reestructuración de este y al final no pueda presentarse en la convocatoria en la que estaba planeada.
- **Riesgo 6 - Pérdida del repositorio o código por error humano.** En el caso de que sin querer borre el código del repositorio ya que creía que era otra cosa o por que me confundí de opción, habríamos perdido todo el código y todo el trabajo realizado con la programación del modelo y los entrenamientos y avace de este.
- **Riesgo 7 - Código demasiado complejo, desorganizado y poco eficiente.** Puede pasar que como al principio el conocimiento sobre el modelo es limitado, a la hora de programar vaya programando cosas de forma poco estructurada y que pueden dificultar el mantenimiento posterior hasta el mejor entendimiento del modelo, y al final si esto se va acumulado puede resultar en un código dificil de comprender y mantener a la par que muy poco eficiente computacionalmente.
- **Riesgo 8 - Resultados inconsistentes por errores de simulación.** Puede que errores en la construcción del entrono, los canales o los parámetros simulados lleven a inconsistencias en los resultados obtenidos en las pruebas o entrenamientos, haciendo que sea muy dificil sacar una conclusión y posiblemente que las que saquemos sean erróneas.

2.4.2. Estimación de los riesgos

Riesgo	**Evaluación cualitativa**	**Factores sobre los que impacta**
No convergencia del modelo DDPG	Alta	Calidad del resultado, tiempo y continuidad del proyecto
Limitaciones computacionales	Medio	Continuidad del proyecto
Retraso en el tiempo por bloqueos en la comunicación con tutores o con la universidad de Aalto	Media	Continuidad del proyecto y tiempo
caída temporal o fallo en la máquina virtual	Medio	Continuidad del proyecto

Riesgo	Evaluación cualitativa	Factores sobre los que impacta
Fallo en la planificación y falta de tiempo para llegar a la entrega final.	Alta	Tiempo y calidad del resultado
Pérdida del repositorio o código por error humano	Alta	Tiempo, continuidad del proyecto y calidad del resultado
Código demasiado complejo, desorganizado y poco eficiente	Media	Tiempo, calidad del resultado
Resultados inconsistentes por errores de simulación	Alta	Tiempo, continuidad del proyecto y calidad del resultado

Tabla 2.3: Estimación de riesgos

2.4.3. Matriz de Probabilidad x Impacto

A continuación vamos a ver el nivel de riesgo real usando una matriz de probabilidad x impacto. Los niveles se han definico según la siguiente escala cualitativa:

Impacto\Probabilidad	Baja	Media	Alta
Alta	Moderado	Alto	Crítico
Media	Bajo	Moderado	Alto
Baja	Bajo	Bajo	Moderado

Tabla 2.4: Escala cualitativa

Y finalmente mapeando los riesgos nos queda la siguiente tabla:

Riesgo	Probabilidad	Impacto	Nivel de riesgo
No convergencia del modelo DDPG	Alta	Alto	Crítico
Limitaciones computacionales	Bajo	Medio	Bajo
Retraso en el tiempo por bloqueos en la comunicación con tutores o con la universidad de Aalto	Media	Medio	Moderado
Caída temporal o fallo en la máquina virtual	Media	Bajo	Bajo
Fallo en la planificación y falta de tiempo para llegar a la entrega final.	Media	Alto	Alto

Riesgo	Probabilidad	Impacto	Nivel de riesgo
Pérdida del repositorio o código por error humano.	Baja	Alta	Moderado
Código demasiado complejo, desorganizado y poco eficiente	Media	Medio	Moderado
Resultados inconsistentes por errores de simulación	Baja	Alta	Moderado

Tabla 2.5: Matriz de probabilidad x impacto

2.4.4. Plan de contingencia

En esta sección se describirá de forma breve los formas de reaccionar o prevenir los riesgos identificados:

1. **No convergencia del modelo DDPG.**
   - Prevención: Ajustando progresivamente los parámetros en función de los análisis de los entrenamientos realizados e intentando buscar siempre la mejora. También probando con arquitecturas simplificadas y con un diseño modular de recompensas.
   - Contingencia: Intentando volver a ajustar los parámetros para conseguir la convergencia o volviendo a una versión anterior en la que todavía no se ha descontrolado, y en caso de que no se haya conseguido todavía una convergencia, realizando una análisis exhaustivo y migrando poco a poco a una arquitectura más simple en la que esté garantizada la convergencia.
2. **Limitaciones computacionales.**
   - Prevención: Limitando los entrenamientos a 9 modelos a la vez (número que ha demostrado ser estable en pruebas previas con la máquina virtual disponible) e intentando hacerlo siempre en horarios de menor ajetreo, como por ejemplo horarios nocturnos.
   - Contingencia: Reiniciar la máquina virtual y hablar con el técnico encargado de ella.
3. **Retraso en el tiempo por bloqueos en la comunicación con tutores o con la universidad de Aalto.**
   - Prevención: Planificar los correos y comunicados con margen temporal. Establecer feedback semanal.
   - Contingencia: Buscar tareas que se puedan realizar de forma paralela al bloqueo para avanzar con el proyecto por diversos frentes.
4. **Caída temporal o fallo en la máquina virtual.**

- Prevención: Limitar el entrenamiento a 9 modelos a la vez, que la máquina soporta bien.
- Contingencia: Hablar con el técnico y, en caso de que el fallo sea prolongado, preparar un pequeño entorno en mi ordenador personal para la ejecución y entrenamiento de algunos modelos.

5. **Fallo en la planificación y falta de tiempo para llegar a la entrega final.**
   - Prevención: Dedicarle tiempo a la planificación siendo realista e informándome de lo que se suele tardar en hacer cada parte del proyecto, tanto en el desarrollo como en la investigación.
   - Contingencia: No agobiarme y planificar —esta vez bien— el final del proyecto para la siguiente entrega, o intentar apurarme para conseguir entregarlo en la primera convocatoria.
6. **Pérdida del repositorio o código por error humano.**
   - Prevención: Realizar copias locales del repositorio y tener una copia en la máquina virtual, otra en el repositorio remoto y otra de forma local en mi ordenador, de forma habitual.
   - Contingencia: Intentar reconstruir el código a partir de una versión anterior guardada por correo, localmente o en la máquina virtual.
7. **Código demasiado complejo, desorganizado y poco eficiente.**
   - Prevención: Mantener buenos hábitos de programación, como la estructuración clara del código, la adición de comentarios y la modularización. También dedicar tiempo a optimizarlo una vez finalizado el desarrollo de cada parte.
   - Contingencia: Dedicar un par de días completos a la reestructuración del código para hacerlo más legible. Supone una pérdida de tiempo a corto plazo, pero es una ganancia a futuro.
8. **Resultados inconsistentes por errores de simulación.**
   - Prevención: Realizar tests continuos y validación cruzada con valores analíticos para evitar la inconsistencia de los datos. Si empiezan a no cuadrar, investigar el motivo e intentar evitarlo.
   - Contingencia: Revisión global del modelo y, en caso necesario, reestructuración a partir de la versión anterior más coherente.

2.5. Balance temporal y económico

En cuanto a la planificación temporal del proyecto, si bien la estimación inicial de las duraciones para cada fase fue realizada con base en un análisis riguroso y la experiencia de expertos mediante el método Wideband Delphi, la ejecución práctica ha presentado algunas desviaciones.

En particular, algunas fases de desarrollo, especialmente las relacionadas con el diseño y testeo de funciones de recompensa, así como los entrenamientos iniciales, se extendieron más allá

del tiempo previsto debido a la complejidad inherente y a la necesidad de iteraciones adicionales para conseguir estabilidad y rendimiento satisfactorio. Por otro lado, la fase de documentación y redacción se completó con mayor rapidez de la esperada, gracias a una metodología continua de documentación incremental y control de versiones, lo que permitió mantener la información actualizada y evitar cuellos de botella al final del proyecto.

En términos económicos, dado que el coste asociado es directamente proporcional al tiempo invertido y tomando un coste simbólico de 20 euros/hora, el impacto de estas desviaciones temporales se refleja en un coste final ligeramente superior al inicialmente presupuestado. Sin embargo, la planificación económica se mantuvo dentro de márgenes aceptables para un proyecto de fin de grado con las características aquí descritas.

Este balance pone de manifiesto la importancia de la flexibilidad y adaptación durante la ejecución, especialmente en proyectos de investigación y desarrollo que involucran componentes experimentales y metodologías iterativas.

Capítulo 3

Antecedentes

3.1. Entorno de negocio

3.1.1. Importancia de la seguridad en redes inalámbricas

En un mundo cada vez más interconectado, las redes inalámbricas desempeñan un papel crucial en sectores como las telecomunicaciones, la industria 4.0, las ciudades inteligentes y el Internet de las cosas (IoT). Sin embargo, la naturaleza abierta del medio inalámbrico convierte la seguridad de la información transmitida en un desafío crítico.

Las técnicas de cifrado convencionales siguen siendo eficaces, pero su dependencia de claves secretas las hace vulnerables ante el avance de la computación cuántica o ataques sofisticados. Ante esta situación, ha ganado protagonismo la **seguridad en la capa física (Physical Layer Security, PLS)**, un enfoque que explota las propiedades inherentes del canal inalámbrico para reforzar la confidencialidad sin requerir intercambios de claves.

Dentro de esta tendencia, las **Superficies Reflectantes Inteligentes (Intelligent Reflecting Surfaces, IRS)** han emergido como una tecnología prometedora para mejorar la seguridad física en redes inalámbricas [4]. Trabajos recientes han explorado el uso de IRS en escenarios vehiculares [5] y en sistemas con UAVs y comunicaciones masivas [8], demostrando la versatilidad y eficacia de esta tecnología. Su capacidad para modificar dinámicamente la propagación de las señales permite mejorar la recepción en el receptor legítimo y degradar la señal en receptores no autorizados, lo que representa una valiosa herramienta para la mejora de la seguridad.

3.1.2. Relevancia de las IRS en la industria

Debido a la capacidad para mejorar la cobertura, eficiencia energética y seguridad de las redes inalámbricas, las IRS han despertado el interés de grandes empresas de telecomunicaciones y organismos internacionales [9, 2, 13]. Algunas como Ericsson, Nokia, Huawei y Qualcomm están invirtiendo activamente en investigación sobre la aplicación de las IRS en redes 5G avanzadas y futuras redes 6G.

Desde el punto de vista comercial, las IRS tienen aplicaciones transversales en diversos sectores, entre los que destacan:

- **Operadores de telecomunicaciones:** Permiten optimizar la cobertura y mejorar la seguridad sin necesidad de desplegar infraestructura adicional [9, 12].

- **Empresas de IoT y automoción:** Aseguran la transmisión fiable y segura en redes de sensores y vehículos autónomos.
- **Defensa y ciberseguridad:** Ofrecen un refuerzo adicional en entornos donde las comunicaciones seguras son críticas.
- **Ámbito militar:** Las IRS pueden desempeñar un papel clave en el ámbito militar, al permitir la creación de canales seguros de comunicación [1] en entornos hostiles, asegurando que solo el receptor legítimo pueda decodificar la señal.
- **Banca y secto financiero:** Fortalecen la seguridad en la transmisión de datos sensibles entre servidores y terminales de cliente.

A pesar de su potencial, las IRS se encuentran aún en **fase de desarrollo experimental**. No obstante, diversas señales indican un fuerte crecimiento del mercado en la próxima década. Entre las principales tendencias destacan:

- **Crecimiento de la inversión en redes 6G:** Las IRS son vistas como un componente esencial de la infraestructura de redes 6G, cuyo despliegue se espera a partir de 2030.
- **Aumento de patentes y colaboraciones:** Empresas como Huawei, Samsung y ZTE han registrado múltiples patentes relacionadas con IRS en los últimos años.
- **Interés en la integración con IA:** Se están explorando enfoques basados en aprendizaje profundo por refuerzo (DRL) para la configuración autónoma de IRS [7, 13], en línea con el enfoque de este proyecto.

3.1.3. Tendencia hacia soluciones basadas en inteligencia artificial

El uso de **inteligencia artificial (IA)** en el ámbito de la seguridad en redes inalámbricas ya no es una opción futura, sino una necesidad estratégica, dada la creciente complejidad, densidad y dinamismo de los entornos de comunicación modernos.

Entre las ventajas clave del uso de la IA en este contexto se encuentran:

- Detección de patrones de ataque y adaptación dinámica de las políticas de transmisión en tiempo real.
- Adaptación rápida a la variabilidad del canal, mejorando la eficiencia en tiempo real.
- Optimización autónoma de recursos, sin intervención humana directa.
- Capacidad de aprender las características del canal y utilizarlas para reforzar la seguridad, sin necesidad de compartir claves.

En particular, el uso de **aprendizaje profundo por refuerzo (Deep Reinforcement Learning, DRL)** para configurar de manera óptima las IRS representa un avance significativo frente a los métodos clásicos basados en optimización matemática explícita.

Varias compañías están apostando por redes auto-gestionadas y auto-seguras en sistemas 5G/6G, en las cuales la inteligencia artificial es un componente esencial. Este proyecto, por tanto, se alinea con las tendencias más vanguardistas en la evolución hacia sistemas inteligentes,

seguros y adaptativos, haciendo así que tenga una aplicación directa en el futuro de las telecomunicaciones.

En este contexto, el presente trabajo, desarrollado en colaboración con la *Universidad de Aalto, Finlandia*, busca contribuir a esta línea de investigación al explorar el uso de DDPG para optimizar IRS en sistemas seguros de comunicación. Los resultados obtenidos podrán servir como referencia para futuras aplicaciones comerciales, industriales y militares, consolidando la relevancia de las IRS y la inteligencia artificial en la evolución de las redes inalámbricas del futuro.

3.2. Estado del arte

3.2.1. Introducción a la seguridad en la capa física (PLS)

En los sistemas de comunicación tradicionales, la seguridad se basa principalmente en técnicas de criptografía implementadas en las capas superiores del modelo de red. Sin embargo, estas técnicas, que dependen de claves secretas y protocolos de cifrado, pueden ser vulnerables a ataques computacionales más avanzados. Es en este contexto donde surge la **seguridad en la capa física (PLS)** como alternativa prometedora, ya que aprovecha las características inherentes del canal de comunicación para proteger la información.

Entre las estrategias más destacadas para mejorar la PLS se encuentran el **ruido artificial (AN)** y el **cooperative jamming**, que buscan interferir a los espías mientras el receptor legítimo recibe la señal deseada. No obstante, estas soluciones suelen implicar un aumento en el consumo energético o la necesidad de hardware adicional, lo que limita su aplicación práctica.

Con la evolución de los metamateriales, ha surgido una nueva tecnología, las **Intelligent Reflecting Surfaces (IRS)**, en español *Superficies Reflectantes Inteligentes*. Las IRS son superficies compuestas por elementos pasivos capaces de ajustar dinámicamente el desfase de las señales reflejadas, sin procesamiento activo de radiofrecuencia. Esta capacidad permite redirigir las señales para reforzar la comunicación con Bob, y al mismo tiempo, degradar la calidad de la señal interceptada por Eve. Lo más destacable es que las IRS ofrecen una solución eficiente en costes y consumo de energía, allanando el camino hacia sistemas más seguros y sostenibles.

3.2.2. Estado actual de las IRS para PLS

En la literatura reciente, las IRS han demostrado ser una herramienta eficaz para aumentar la seguridad en la capa física. Diversos estudios han explorado algoritmos avanzados para maximizar la **Tasa de secreto (SR)**, optimizando tanto la transmisión activa desde la estación base, como los ajustes pasivos en la IRS [12, 7].

Importantes contribuciones en el estudio y optimización de metasuperficies inteligentes para comunicaciones inalámbricas han sido realizadas por Diego Martín de Andrés, Francisco Hernando y Masoud Kaveh. Sus trabajos destacan por la implementación de métodos avanzados que mejoran la eficiencia y adaptabilidad de las superficies reflectantes, consolidando las bases para aplicaciones prácticas en sistemas de seguridad física [1].

Algunos de los enfoques que se han solido tomar para resolver este problema incluyen técnicas de optimización como las siguientes:

- **Block Coordinate Descent (BCD)**: Esta estrategia consiste en optimizar los desfases de la IRS de forma secuencial, ajustando uno a uno. Ofrece buenos resultados, sin embargo

la convergencia puede ser lenta y subóptima en configuraciones grandes.

- **Transformación Charnes-Cooper + Semidefinite Relaxation (SDR)**: esta técnica reformaula el problema para hacerlo más manejable y relajarlo a una forma semidefinida. El inconveniente principal es la alta carga computacional, lo que dificulta su implementación práctica en tiempo real.
- **Alternating Optimization (AO)**: Este es un enfoque más flexible que descompone el problema en dos partes: la optimización de la formación de haces (beamforming) en la estación base y los ajustes de fase en la IRS. AO mejora la convergencia, pero requiere un conocimiento previo del modelo del canal, lo que puede ser una limitación en entornos cambiantes.
- **Fractional Programming (FP) + Manifold Optimization (MO)**: Este es un método más reciente que transforma el problema original en una forma más simple y lo resuelve iterativamente mediante optimización en variedades (manifolds). Los resultados muestran una convergencia rápida y alto rendimineto, acercándose al límite teórico dela tasa de secreto.

Las IRS han demostrado ser una tecnología revolucionaria para la seguridad en la capa física, aunque los métodos actuales aún enfrentan desafíos en términos de complejidad computacional y adaptabilidad a escenarios dinámicos.

Ante estas limitaciones, se propone abordar el problema de optimización de la tasa de secreto mediante **aprendizaje profundo por refuerzo (DRL)**, concretamente con el algoritmo **DDPG (Deep Deterministic Policy Gradient)**. Este enfoque nos permitirá resolver este problema en condiciones cambiantes y continuas.

3.2.3. Introducción a DDPG en el contexto de IRS

DDPG es un algoritmo de aprendizaje profundo por refuerzo basado en actor-crítico. Este algoritmo está diseñado para manejar problemas con acciones en un espacio continuo, como es el caso de nuestro problema.

Hemos elegido este enfoque ya que para resolver el problema de maximización de la tasa de secreto en un sistema con IRS, debemos elegir un enfoque que pueda manejar la naturaleza continua y dinámica del problema. Este algoritmo es particularmente adecuado porque:

- Este algoritmo maneja acciones continuas, lo que es clave para optimizar los ángulos de fase de la IRS
- Aprende de la experiencia permitiendo adaptarse a cambios en los canales sin necesidad de recalcular todo desde cero.
- Optimiza decisiones a largo plazo, mejorando la tasa de secreto en entornos dinámicos.

3.2.4. Comparación entre DDPG y AO

Tanto la optimización alterna (AO) como el aprendizaje profundo por refuerzo basado en DDPG son enfoques poderosos para resolver problemas de optimización en comunicaciones inalámbricas y redes inteligentes. Sin embargo cada uno tiene sus puntos fuertes y débiles dependiendo de los problemas y recursos disponibles.

Optimización alterna (AO)

Esta es una técnica iterativa que descompone un problema complejo en subproblemas más manejables, resolviéndolos de manera alternada. En el contexto de las IRS, AO separa la optimización en dos bloques:

- Primero ajusta la formación de haces en la estación base mientras mantiene los desfases de la IRS fijos.
- Y después optimiza los desfases de la IRS manteniendo la formación de haces constante.

Este proceso se repite hasta que la solución converge. Aunque AO es más eficientes que métodos tradicionales como SDR, depende de conocer el modelo de canal y puede estancarse en mínimos locales, lo que limita su rendimiento en entornos dinámicos o desconocidos. En resumen las ventajas y desventajas de este algoritmo son las siguientes:

- **Ventajas:**
  - Menor complejidad computacional al optimizar una variable a la vez. Esto simplifica problemas grandes
  - Convergencia rápida en muchos casos, ya que suele alcanzar soluciones aceptables en pocas iteraciones.
  - Mayor interpretabilidad debido a que su funcionamiento es matemáticamente comprensible y puede analizarse teóricamente.
- **Desventajas:**
  - Puede quedar atrapado en soluciones subóptimas en el contexto de óptimos locales
  - Es dificil de escalar ya que en problemas altamente dinámicos y de gran dimensión, su desempeño puede verse limitado.

DDPG

Este algorimto está diseñado para manejar acciones continuas, lo que es ideal para este problema. Para una visión más clara se presentan las siguientes ventajas y desventajas del mismo.

- **Ventajas:**
  - Puede adaptarse a entornos dinámicos y cambios en la red sin necesidad de reoptimización manual, por lo que puede considerarse de aprendizaje adaptativo.
  - No requiere modelado exacto del canal, a diferencia de AO, puede aprender estrategias sin depender de modelos matemáticos exactos.
  - Proporciona mejores soluciones globales ya que puede encontrar soluciones cercanas al óptimo global en problemas complejos y no convexos.
- **Desventajas:**
  - El proceso de entrenamiento puede ser lento y costoso en términos de datos y cómputo.

- Tiene mayor complejidad computacional, comparando con AS, puede requerir recursos computacionales elevados (como GPUs), especialmente en entornos con alta dimensionalidad o en simulaciones complejas.
- Es menos interpretable, ya que no siempre es fácil entender por qué el modelo toma ciertas decisiones.

Conclusión

Tomando tódas las diferencias y haciendo una tabla de comparación directa nos queda la siguiente

Característica	**Optimización Alterna (AO)**	**DRL con DDPG**
Complejidad computacional	Baja	Alta
Velocidad de convergencia	Rápida	Lenta (necesita entrenamiento)
Capacidad de adaptación	Baja (no se ajusta a cambios dinámicos)	Alta (aprende de la experiencia)
Escalabilidad a grandes problemas	Limitada	Buena para problemas grandes y dinámicos
Garantía de solución óptima	Puede quedar en óptimos locales	Tiende a encontrar soluciones mejores, pero sin garantía de óptimo global
Dependencia del modelo matemático	Alta (requiere formulaciones matemáticas)	Baja (aprende sin modelos exactos)

Tabla 3.1: Comparación entre Optimización Alterna y DRL con DDPG

Si nos encontramos ante un problema estático y bien modelado, entonces AO es la mejor opción ya que es rápido y eficiente en problemas donde los parámetros no cambian con frecuecia y se tiene un modelo matemático claro.

Mientras que si el problema es dinámico y complejo, entonces la solución pasa por DDPG ya que, si las condiciones de la red cambian constantemente (usuarios en movimiento, interferencia variable, etc) DRL con DDPG puede aprender y adaptarse.

Dado que en la vida real los canales de comunicación son dinámicos y difíciles de modelar con precisión, la mejor opción para enfocar este problema es DRL con DDPG en lugar de AO. Las razones finales son:

- Los canales camiban en tiempo real, y un modelo de DRL puede aprender a adaptarse din necesidad de reoptimizar desde cero en cada cambio.
- Las IRS tienen elementos pasivos (no procesan señales), por lo que su configuración óptima no es trivial. DRL puede encontrar patrones en la optimización de fase sin depender de ecuaciones exactas.

- AO se queda atrapado en óptimos locales, mientras que DDPG puede explorar mejores configuraciones a lo largo del tiempo.
- Sabemos además que el problema es no convexo debido a la optimización conjunta de w y Φ lo que hace que AO no sea ideal en este caso.

3.2.5. Comparación entre DDPG y otros algoritmos de aprendizaje por refuerzo (RL)

En este apartado se presenta una comparación entre DDPG y otros algorimos clave de RL, que han demostrado ser una herramienta poderosa para resolver problemas de optimización dinámica en entoros inciertos, como es el caso del ajuste de fases en las IRS. Nos centraremos en analizar sus diferencias en términos de tipo de acción, ventajas y limitaciones, con el fin de justificar la elección de DDPG para este proyecto.

Algoritmo	**Tipo de Acción**	**Ventaja Principal**	**Desventaja**
DQN	Discretas	Eficiente y estable	No funciona bien con acciones continuas
DDPG	Continuas	Maneja acciones continuas y tiene buena convergencia	Necesita mucha exploración
PPO	Discretas o continuas	Más establa que DDPG en entornos ruidosos	Menos eficiente en problemas deterministas
SAC	Continuas	Mejor exploración que DDPG	Más costoso computacionalmente

Tabla 3.2: Comparación entre DDPG y otros algoritmos de RL

Para resolver el problema de maximización de la tasa de secreto en un sistema con una IRS, debemos elegir un enfoque que pueda manejar la naturaleza continua y dinámica del problema, esto descarta DQN. Además el problema de optimización es determinista, ya que dado un canal hay una configuración óptima, lo que hace que DDPG sea más eficiente que SAC o PPO.

3.2.6. Resumen de elección de DDPG

Método	**SI / NO**	**¿Por qué?**
Optimización Alterna (AO)	NO	Se queda atascado en soluciones locales y no se adapta a cambios dinámicos en el canal
DQN (Deep Q-Networks)	NO	Solo maneja acciones discretas pero la fase de la IRS es continua (entre 0 y 2π)

Método	SI / NO	¿Por qué?
PPO (Proximal Policy Optimizacion)	NO	Aunque admite acciones continuas, está optimizado para entornos ruidosos o estocásticos, lo que lo hace menos eficiente para problemas deterministas como el ajuste preciso de fase en IRS.
SAC (oft Actor-Critic)	NO	Diseñado para entornos estocásticos; en problemas deterministas como el nuestro, su capacidad de exploración no se traduce en un mejor rendimiento, y su coste computacional es elevado, lo que no es viable con nuestros recursos.
DDPG (Deep Deterministic Policy Gradient)	SI	Majena acciones continuas, optimiza determinísticamente y aprende una política directamente.

Tabla 3.3: Resumen de algoritmos

3.3. Fundamentos teóricos

Antes de abordar el desarrollo de la solución propuesta, es fundamental establecer una base sólida sobre los conceptos teóricos que sustentan este trabajo. Esta sección presenta los principios clave relacionados con la inteligencia artificial y el aprendizaje por refuerzo profundo, así como los fundamentos de las superficies reflectantes inteligentes (IRS) y la comunicación segura en la capa física. A través de esta revisión se proporciona el marco conceptual necesario para comprender la formulación del problema y la aplicación del algoritmo DDPG en el contexto de redes inalámbricas seguras potenciadas por IRS.

3.3.1. Inteligencia Artificial (IA)

La **Inteligencia Artificial (IA)** es un campo de la informática que estudia cómo dotar a las máquinas de la capacidad de realizar tareas que, tradicionalmente, requieren inteligencia humana. Estas tareas incluyen el razonamiento, el aprendizaje, la planificación, el reconocimiento de patrones, la toma de decisiones o la comprensión del lenguaje natural.

Desde sus inicios en los años 50, la IA ha evolucionado desde sistemas expertos basados en reglas hacia enfoques más avanzados como el *Machine Learning* (aprendizaje automático) y el *Deep Learning* (aprendizaje profundo). Esta evolución ha sido impulsada por la disponibilidad de grandes volúmenes de datos, el avance del hardware y el desarrollo de algoritmos más eficientes.

Existen diferentes ramas dentro de la IA:

- **IA simbólica:** basada en lógica y reglas explícitas.
- **IA conexionista:** basada en redes neuronales, más parecida al funcionamiento del cerebro humano.
- **IA evolutiva:** inspirada en procesos biológicos como la selección natural.

Aplicación al proyecto

En el contexto de este proyecto, la IA se utiliza para diseñar agentes inteligentes capaces de aprender a actuar en entornos dinámicos con el objetivo de maximizar una recompensa, mediante la técnica conocida como *Reinforcement Learning* (aprendizaje por refuerzo). Esta técnica representa una intersección entre la IA y la teoría del control y tiene especial utilidad en problemas donde no existe una supervisión directa, sino que el agente debe descubrir qué acciones son óptimas a través de la interacción con el entorno.

La elección de técnicas de IA, y más concretamente de aprendizaje por refuerzo profundo (Deep Reinforcement Learning), se justifica por la naturaleza compleja y no lineal del problema abordado: la optimización de la tasa de secreto en redes IRS. Esta tarea requiere que el sistema aprenda estrategias óptimas a partir de interacciones simuladas en un entorno altamente dinámico.

3.3.2. Machine Learning (Aprendizaje automático)

El **aprendizaje automático (Machine Learning, ML)** es una subdisciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos y técnicas que permiten a los sistemas aprender automáticamente a partir de datos, sin estar explícitamente programados para cada tarea concreta. En lugar de seguir reglas rígidas predefinidas, un modelo de ML extrae patrones o relaciones presentes en los datos para generalizar y realizar predicciones o tomar decisiones sobre nuevas entradas.

La idea central del aprendizaje automático es mejorar el rendimiento de un sistema en una tarea específica a medida que aumenta la experiencia (es decir, los datos disponibles). Esta experiencia se traduce habitualmente en un conjunto de datos de entrenamiento sobre los cuales el modelo ajusta sus parámetros internos.

Los tipos principales de Machine Learning son:

- **Aprendizaje supervisado:** El modelo se entrena a partir de un conjunto de datos etiquetado, es decir, en el que cada entrada está asociada a una salida deseada. El objetivo es aprender una función que, dadas nuevas entradas, prediga su correspondiente salida. Ejemplos típicos incluyen la regresión y la clasificación.
- **Aprendizaje no supervisado:** El modelo trabaja con datos no etiquetados, tratando de descubrir estructuras ocultas o agrupaciones en los datos. Este enfoque se utiliza comúnmente en tareas como la segmentación, la reducción de dimensionalidad o la detección de anomalías.
- **Aprendizaje por refuerzo (RL):** El sistema aprende a tomar decisiones mediante la interacción con un entorno. No se proporcionan etiquetas explícitas, sino señales de refuerzo (recompensas o penalizaciones) que guían el aprendizaje. Este enfoque se basa en la maximización acumulativa de la recompensa y se detalla más adelante, ya que constituye el núcleo del presente trabajo.

Aplicación al proyecto

En este proyecto el Machine Learning actúa como base para técnicas más avanzadas, como el aprendizaje profundo (*Deep Learning*) y el aprendizaje por refuerzo profundo (*Deep Reinforcement Learning*), utilizadas en este proyecto. En concreto, la naturaleza compleja y dinámica

del entorno de comunicación considerado hace inviable un diseño basado en reglas o una solución supervisada tradicional. Por ello, se recurre a enfoques de ML más flexibles y adaptativos, capaces de aprender políticas óptimas a través de la experiencia directa.

3.3.3. Redes neuronales artificiales

Las **redes neuronales artificiales** (**ANN**, por sus siglas en inglés) son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por nodos interconectados, denominados *neuronas artificiales*, organizados en capas. Estas neuronas transforman las entradas mediante funciones matemáticas y las propagan a través de la red con el objetivo de aproximar funciones complejas, detectar patrones y aprender representaciones jerárquicas de los datos.

Cada conexión entre neuronas tiene un peso asociado que determina la intensidad de la señal que se transmite. Durante el proceso de entrenamiento, estos pesos se ajustan con el fin de minimizar un error (en aprendizaje supervisado) o maximizar una recompensa (en aprendizaje por refuerzo), dependiendo del tipo de aprendizaje.

El perceptrón: la unidad básica

La neurona artificial más simple y fundamental es el **perceptrón**, propuesto por *Frank Rosenblatt* en 1958. Se trata de un modelo binario de clasificación que recibe un vector de entrada $\mathbf{x} = [x_1, \dots x_n]$, lo pondera con una serie de pesos $\mathbf{w} = [w_1, \dots, w_n]$, y aplica una función de activación para determinar su salida:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

donde

- b es un término de sesgo (*bias*)
- f es típicamente una función escalón (step) o sigmoide
- y es la salida del perceptrón, generalmente 0 o 1

Este modelo, aunque simple, sienta las bases de estructuras más complejas como las redes multicapa, donde las salidas de varias neuronas se combinan y retroalimentan.

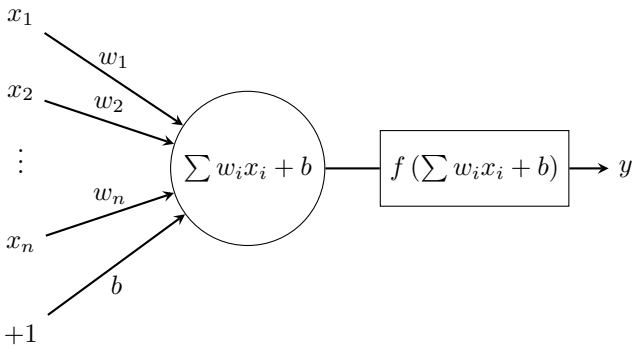

Figura 3.1: Perceptrón

Estructura general de una red neuronal

Una red neuronal típica está formada por:

- Una **capa de entrada**, que recibe los datos del problema.
- Una o más **capas ocultas**, responsables del procesamiento intermedio de la información (donde ocurre el aprendizaje).
- Una **capa de salida**, que produce la predicción o acción final.

En redes profundas, cada capa aprende representaciones cada vez más abstractas de los datos, lo que permite afrontar tareas altamente no lineales y complejas.

Aplicación al proyecto

En el presente trabajo, centrado en la aplicación del *Deep Reinforcement Learning* a la optimización de la tasa de secreto en redes IRS, las redes neuronales juegan un papel clave como función de aproximación dentro del algoritmo DDPG (*Deep Deterministic Policy Gradient*). Este algoritmo utiliza dos redes neuronales principales:

- Una **red del actor**, que representa la política del agente y se encarga de seleccionar acciones (valores continuos) dadas las observaciones del entorno.
- Una **red del crítico**, que evalúa la calidad de las acciones tomadas por el actor, estimando el valor de la función Q para cada par estado-acción.

Además, para estabilizar el aprendizaje, se utilizan versiones "congeladas" de ambas redes, conocidas como *target networks*.

Estas redes objetivo se actualizan lentamente a lo largo del tiempo mediante una interpolación con las redes principales, lo que permite suavizar las actualizaciones y mejorar la estabilidad del entrenamiento. Este enfoque es esencial para garantizar una convergencia más robusta en entornos complejos y no lineales como el considerado en este proyecto.

3.3.4. Deep Learning (Aprendizaje profundo)

El **aprendizaje profundo (Deep Learning)** es una subdisciplina del Machine Learning que utiliza redes neuronales artificiales con múltiples capas ocultas para modelar relaciones complejas y altamente no lineales entre datos. A diferencia de los modelos más tradicionales, las redes profundas son capaces de aprender representaciones jerárquicas de la información, lo que les permite destacar en tareas como visión por computador, procesamiento de lenguaje natural, y, más recientemente, en sistemas de toma de decisiones complejas como el aprendizaje por refuerzo.

Las arquitecturas de aprendizaje profundo suelen incluir:

- **Redes neuronales densas (fully connected)**, en las que cada neurona está conectada con todas las de la capa anterior y posterior.
- **Redes convolucionales (CNN)**, muy útiles para procesar datos estructurados espacialmente como imágenes.
- **Redes recurrentes (RNN, LSTM)**, orientadas a secuencias temporales.

- **Redes actor-crítico y arquitecturas personalizadas**, comunes en entornos de decisión y control como el aprendizaje por refuerzo profundo.

Una de las grandes ventajas del Deep Learning es su capacidad para **extraer automáticamente** características relevantes a partir de datos crudos, sin requerir ingeniería manual de características. No obstante, esta capacidad requiere grandes volúmenes de datos y recursos computacionales significativos para entrenar modelos con millones de parámetros.

Deep Reinforcement Learning (DRL)

Cuando el *Deep Learning* se combina con técnicas de *Reinforcement Learning*, da lugar al llamado **Deep Reinforcement Learning (DRL)**, una rama especialmente poderosa que ha permitido alcanzar resultados impresionantes en entornos de alta complejidad, como videojuegos, robótica, o, como en este proyecto, optimización de redes de comunicaciones.

En DRL, las redes profundas se utilizan para aproximar funciones de valor, modelar políticas, o predecir recompensas esperadas. Sin embargo, a diferencia del aprendizaje supervisado, en donde el objetivo está claramente definido por un conjunto de etiquetas, en el aprendizaje por refuerzo el agente debe aprender a través de la interacción con el entorno.

La importancia de la función de recompesa

En este contexto, la **función de recompensa** cobra un papel absolutamente central. Se trata de la señal que guía al agente hacia comportamientos deseables, premiando las acciones correctas y penalizando las perjudiciales. A través de esta señal escalar, el agente evalúa indirectamente la utilidad de sus decisiones en cada estado del entorno.

Un mal diseño de la función de recompensa puede provocar:

- Aprendizaje ineficiente o extremadamente lento.
- Comportamientos indeseados (el agente optimiza la métrica equivocada).
- Convergencia a políticas subóptimas o incluso divergencia del entrenamiento.

Por tanto, una de las tareas más críticas en proyectos de Deep Reinforcement Learning es el diseño cuidadoso de esta función, equilibrando las múltiples metas del sistema y asegurando que el agente reciba retroalimentación suficiente y coherente.

En este trabajo, el diseño y afinado de la función de recompensa ha sido un proceso iterativo y experimental, ya que de su correcta formulación depende en gran medida que el agente logre optimizar la tasa de secreto sin comprometer la estabilidad del entorno de entrenamiento ni generar comportamientos imprevisibles o ineficientes.

3.3.5. Arquitectura de Superficies Reflectantes Inteligentes (IRS)

Las **Superficies Reflectantes Inteligentes** (*Intelligent Reflecting Surfaces*, **IRS**) representan una de las tecnologías emergentes más prometedoras para la próxima generación de redes inalámbricas (6G). Estas superficies permiten controlar de forma inteligente la propagación de las ondas electromagnéticas en el entorno, reconfigurando dinámicamente el canal de comunicación con el objetivo de mejorar la calidad del enlace, la eficiencia espectral y la seguridad de la transmisión.

Una IRS se compone típicamente de un conjunto de elementos pasivos reflectantes, distribuidos en una matriz o panel plano, que pueden ajustar la fase, y en algunos casos también la amplitud, de la señal reflejada. A diferencia de técnicas tradicionales basadas en relés activos o antenas MIMO con procesamiento digital complejo, las IRS ofrecen una **solución energéticamente eficiente y de bajo coste**, ya que los elementos son pasivos y no generan ni amplifican señales.

Modelo básico de funcionamiento

La IRS se sitúa estratégicamente entre el transmisor y el receptor, o bien entre el transmisor y posibles interferencias o intrusos. Mediante el ajuste de las fases de reflexión de sus elementos, la IRS puede:

- **Mejorar la ganancia del canal directo** entre emisor y receptor, al reforzar constructivamente la señal.
- **Anular o debilitar canales indeseados**, como aquellos dirigidos hacia receptores no autorizados.
- **Crear entornos de propagación favorables**, incluso en presencia de obstáculos o en entornos de línea de visión obstruida (NLoS)

El canal equivalente resultante de la interacción entre el canal directo y las reflexiones de la IRS es altamente no lineal y dependiente de múltiples variables, lo que convierte su optimización en una tarea compleja.

Control de la IRS mediante programación digital

Cada elemento reflectante puede ser controlado digitalemente mediante una placa controladora, que le asigna un coeficiente de reflexión (complejo) determinado. Este coeficiente define principalmetnte un **desfase** que la IRS introduce a la onda incidente. Si toddos los elementos se configuran adecuadamente, pueden lograr una suma constructurva de las señales reflejadas en el receptor deseado, o una interferencia destructiva en los receptores no deseados.

En la práctica, los coeficientes de fase disponibles suelen estar cuantizados, es decir, limitados a un número finito de valores posibles, lo que añade restricciones adicionales al problema de optimización.

Aplicación a la seguridad física del canal

Uno de los campos de aplicación más intersantes de las IRS (que es la aplicación en la que nos centraremos en este proyecto), es la mejora de la **seguridad física del canal**. Frente a métodos tradicionales basados en criptografía, la seguridad física busca garantizar confidencialidad desde el propio nivel físico del canal, aprovechando la aleatoriedad inherente a las condiciones del entorno.

Gracias a su capacidad de modificar el entorno de propagación, una IRS puede ser utilizada para **maximizar la tasa de secreto**, redirigiendo las señales hacia sus receptores legítimos y reduciéndo las percibidas por receptores espía.

Aplicación al proyecto

En este trabajo, se utiliza una arquitectura IRS en combinación con un agente de apredizaje profundo por refuezo, que ajusta dinámicamente los coeficientes de fase con el objetivo de maximizar la tasa de secreto anteriormente mencionada y que se explicará más a fondo en la siguiente sección, adaptándose a las condiciones del entorno y sin necesidad de una supervisión explícita.

3.3.6. Tasa de secreto (SR)

La **Tasa de Secreto** (Secrecy Rate, SR) es una métrica fundamental en comunicaciones seguras a nivel de la capa física (*physical layer security*), cuya finalidad es cuantificar cuánta información puede transmitirse de forma segura desde un transmisor legítimo (Alice) hacia un receptor autorizado (Bob), en presencia de un receptor no autorizado o espía (Eve).

Matemáticamente, la tasa de secreto se define como la direrencia entre la capacidad del canal legítimo y la capacidad del canal del espía. Su expresión es la siguiente

$$SR = [\log_2(1 + \mathrm{SNR}_{Bob}) - \log_2(1 + \mathrm{SNR}_{Eve})]^+ \tag{3.1}$$

donde la notación $[x]^+ = \max(0, x)$ y SNR_{Bob} y SNR_{Eve}. son las relaciones señal-ruido (*Signal-to-Noise Ratio*) correspondientes a Bob y Eve, respectivamente. En caso de que el canal de Eve sea mejor que el de Bob, la tasa de secreto se anula, ya que no es posible garantizar confidencialidad en ese caso.

Aplicación en este proyecto: Aprendizaje por refuerzo y optimización de la tasa de secreto

En este proyecto, se aborda la tarea de optimizar la tasa de secreto mediante el uso de un agente basado en *Deep Reinforcement Learning*, concretamente mediante el algoritmo DDPG (que veremos con detalle en secciones posteriores). El agente observa el estado del entorno y aprende a seleccionar las configuraciones de fase de la IRS que maximizan la tasa de secreto como función de recompensa.

Este enfoque permite una adaptación dinámica y no supervisada a las condiciones cambiantes del canal, evitando la necesidad de modelos exactos o de información completa sobre el sistema, y aprovechando el aprendizaje a partir de la interacción con el entorno.

3.3.7. Formulación matemática del problema

Consideramos un sistema de comunicaciones MISO (Multiple-Input, Single-Output) compuesto por una estación base (BS) equipada con múltiples antenas transmisoras, un único usuario legítimo (Bob) con una sola antena receptora, una superficie inteligente reconfigurable (IRS) con N elementos reflectantes pasivos, y un espía pasivo (Eve) que intenta interceptar la señal sin interferir activamente en el canal.

El objetivo es diseñar tanto el vector de *beamforming* w en la estación base como los coeficientes de fase de los elementos de la IRS, $\Phi = [e^{j\psi_1}, \ldots, e^{j\psi_N}]$, de forma que se maximice la tasa de secreto del sistema.

El problema de optimización se plantea como:

$$\max_{w,\phi} SR(w, \Phi) \quad \text{sujeto a: } \|w\|^2 \leqslant P_{\max},\ \psi_i \in [0, 2\pi] \tag{3.2}$$

Donde:

- $w \in \mathbb{C}^M$ es el vector de *beamforming* de la estación base, con M antenas transmisoras.
- $\Phi \in \mathbb{C}^N$ representa los coeficientes de reflexión de fase de los N elementos de la IRS, con la restricción de $\psi_i \in [0, 2\pi]$.
- $P_{\max}$ es la potencia máxima de transmisión disponible en la estación base.
- $SR(w, \Phi)$ es la tasa de secreto, definida anteriormente en la ecuación (3.1)

Para simular las condiciones del entorno inalámbrico, se adopta un modelo de desvanecimiento plano Rayleigh para los distintos canales involucrados. La simulación se realiza generando matrices o vectores complejos cuya magnitud sigue una distribución Rayleigh y cuya fase es uniforme, representando un entorno con múltiples trayectorias aleatorias sin línea de vista dominante. Los distintos canales que entrarán en juego son los siguientes:

- $H_{TI} \in \mathbb{C}^{N \times M}$ será el canal entre la base y la IRS
- $h_{TB} \in \mathbb{C}^{M \times 1}$ será el canal entre la base y Bob
- $h_{TE} \in \mathbb{C}^{M \times 1}$ será el canal entre la base y Eve
- $h_{IB} \in \mathbb{C}^{N \times 1}$ será el canal entre la IRS y Bob
- $h_{IE} \in \mathbb{C}^{N \times 1}$ será el canal entre la IRS y Eve

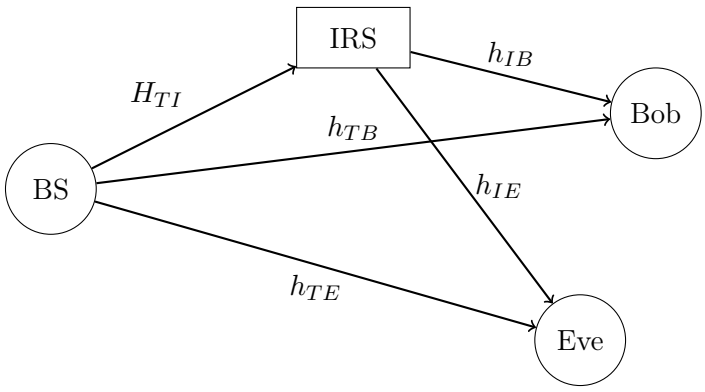

Figura 3.2: Modelo del sistema MISO con IRS, usuario legítimo (Bob) y espía (Eve).

Las señales recibidas en Bob y en Eve se calcularán de la siguiente forma:

$$\begin{aligned} \text{SNR}_{Bob} &= (h_{IB}^H \, \Phi \, H_{TI} + h_{TB}^H) w + n_B \\ \text{SNR}_{Eve} &= (h_{IE}^H \, \Phi \, H_{TI} + h_{TE}^H) w + n_E \end{aligned} \tag{3.3}$$

donde n_B y n_E son ruidos aditivos, que se generarán como variables gausianas complejas con media 0 y varianza σ_B^2 y σ_E^2 respectivamente.

Este problema es altamente no convexo por las siguientes razones:

- La función objetivo $SR(w, \Phi)$ depende de forma no lineal de las variables de diseño, a través de las expresiones de SNR_{Bob} y SNR_{Eve} que involucran múltiples trayectorias (directas e indirectas vía IRS).
- La tasa de secreto se expresa como una diferencia de logaritmos, lo que genera una función objetivo que no es ni cóncava ni convexa.
- Existen productos bilineales y cuadráticos complejos entre w y Φ, lo que introduce un fuerte acoplamiento no lineal entre las variables.
- Las trayectorias múltiples inducen interferencia acoplada entre componentes directas e indirectas, lo que complica la estructura de la solución y da lugar a múltiples óptimos locales.

Como resultado, resolver este problema mediante técnicas tradicionales como programación convexa o relajaciones semidefinidas puede ser computacionalmente costoso o poco eficaz, especialmente en entornos dinámicos donde los canales varían frecuentemente.

Dado el carácter complejo, dinámico y parcialmente observable del entorno, se propone abordar este problema mediante un agente de **Aprendizaje por Refuerzo Profundo** (DRL). Este enfoque permite al agente:

- Aprender políticas de optimización directamente a partir de la interacción con el entorno.
- Adaptarse a cambios en los canales de propagación.
- Maximizar la tasa de secreto de forma continua sin requerir modelos explícitos del sistema.

En concreto, en este proyecto se emplea el algoritmo DDPG, basado en el uso de redes neuronales continuas y diferenciables que permiten tratar directamente variables reales y complejas como w y Φ.

3.3.8. Fundamentos de DDPG (Deep Deterministic Policy Gradient)

El algoritmo **Deep Deterministic Policy Gradient (DDPG)** es un método de aprendizaje profundo por refuerzo ideal para este problema por su capacidad de manejar acciones contínuas y adaptarse a entornos no estacionarios, como el ajuste de fases de una IRS.

Se basa en una arquitectura *actor-crítico* y extiende el algoritmo DPG (Deterministic Policy Gradient) con técnicas de estabilización propias del aprendizaje profundo, como redes objetivo y experiencia repetida (*experience replay*).

Durante el desarrollo de esta sección, la notación que se usará será la siguiente:

- s, denotará el estado en el que nos encontramos, es decir, la información que se le pasa al modelo y de la que deducirá que acción realizar.
- a, denotará la acción que elegirá el modelo para el estado s.
- r, denotará la recompensa que obtendrá el modelo al aplicar la acción a al estado s.
- Siempre que queramos referirnos, a un determinado estado, acción o recompensa y al siguiente, se utilizarán los subíndices de la siguiente manera, s_t para el estado actual, y s_{t+1} para el estado siguiente.

Arquitectura Actor-Crítico

DDPG utiliza dos redes principales:

- **Actor (μ):** Una red neuronal que representa la política determinista $\mu(s|\theta^\mu)$. Esta red recibe un estado s como entrada y devuelve una acción $a = \mu(s)$ que maximiza el valor esperado a largo plazo.
- **Crítico (Q):** Una red neuronal que estima la función de valor acción $Q(s, a|\theta^Q)$, proporcionando una evaluación del beneficio esperado al ejecutar la acción a en el estado s.

Durante el entrenamiento, las redes se actualizan mediante gradientes calculados sobre minilotes extraídos de una memoria de experiencias. El crítico se entrena minimizando el error de Bellman:

$$L(\theta^Q) = \mathbb{E}_{s,a,r,s'}\left[\left(Q(s, a|\theta^Q) - y\right)^2\right],$$

donde:

$$y = r + \gamma Q'(s', \mu'(s'|\theta^{\mu'})|\theta^{Q'}),$$

Las variables con tilde (Q', μ') corresponden a las redes objetivo suavizadas, que se actualizan lentamente para mejorar la estabilidad del entrenamiento, la r corresponde a la recompensa obtenida al aplicar la acción al estado en cuestión, y el parámetro $\gamma \in (0, 1)$ es el factor de descuento, que le indica al modelo como de *previsor a futuro* tiene que ser (cuanto más cercano a 1, más se tiene en cuenta el futuro).

El actor se actualiza maximizando la salida del crítico respecto a la política:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_s\left[\nabla_a Q(s, a|\theta^Q)|_{a=\mu(s)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)\right].$$

se usa el gradiente de la red del crítico para actualizar los pesos del actor en la dirección que más beneficie al modelo.

Redes objetivo (Target Networks)

Las redes objetivo son copias lentas y estables de las redes actor y crítico originales. Su función es proporcionar objetivos estables durante el entrenamiento, evitando oscilaciones debidas a actualizaciones simultáneas de la política y la estimación del valor.

Denotaremos como μ' y Q' a las redes objetivos, con parámetros $\theta^{\mu'}$ y $\theta^{Q'}$. Estas redes se actualizan de forma suavizada a partir de las redes principales:

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}, \quad \theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}, \tag{3.4}$$

donde $0 < \tau \ll 1$ es una constante de suavizado (por ejemplo, en nuestro caso se usará $\tau = 0{,}001$).

El uso de estas redes en las construcción del modelo trae distintas ventajas:

- **Estabilidad:** Proporcionan una referencia casi constante al optimizar la red crítico, reduciendo inestabilidades numéricas.
- **Convergencia más suave:** Evitan que pequeños errores en la estimación del valor se amplifiquen al retroalimentarse en pasos sucesivos.

- **Mejor control de retropropagación:** Al mantener la política objetivo fija a corto plazo, se reduce la varianza de los gradientes.

Sin las redes objetivo, el entrenamiento de DDPG se vuelve inestable y puede diverger, especialmente en entornos complejos como es el nuestro, en el que simulamos la realidad de los canales de comunicación inalámbrica.

Memoria de experiencias (Replay Buffer)

Uno de los componentes clave en DDPG es el uso de una memoria de experiencias, también llamada *replay buffer*. Este mecanismo consiste en almacenar "experiencias" de la forma (s_t, a_t, r_t, s_{t+1}) durante la interacción del agente con el entorno.

Durante el entrenamiento, en lugar de usar únicamente la experiencia más reciente, se seleccionan mini-lotes aleatorios de esta memoria para actualizar las redes. Esto introduce las siguientes ventajas:

- **Romper la correlación temporal:** En tareas secuenciales, las experiencias consecutivas están altamente correladas. El muestreo aleatorio permite entrenar con datos más independientes y variados.
- **Reutilización de datos:** Permite reutilizar experiencias pasadas, mejorando la eficiencia sample-based y reduciendo la varianza del entrenamiento.
- **Estabilidad del aprendizaje:** Contribuye a una dinámica de entrenamiento más estable y menos propensa a oscilaciones.

En este proyecto, el replay buffer almacenará episodios de la forma que se ha descrito anteriormente

$$(s_t, a_t, r_t, s_{t+1}).$$

Exploración en entornos continuos

En las implementaciones estándar del modelo DDPG se introduce como técnica de exploración la addición de ruido del tipo Ornstein-Uhlenbeck (OU), debido a que este algoritmo utiliza una política determinista y este tipo de ruido es el adecuado para sistemas físicos contínuos.

El ruido OU modela un proceso estocástico con memoria (ruido correlacionado temporalmente), lo que resulta útil para evitar fluctuaciones bruscas en acciones consecutivas. Su dinámica se define por la siguiente ecuación diferencial estocástica:

$$dx_t = \theta(\mu - x_t)dt + \sigma dW_t$$

Para su implementación práctica en algoritmos numéricos como DDPG, esta expresión se discretiza utilizando el método de Euler-Maruyama, resultando en la siguiente forma:

$$x_{t+1} = x_t + \theta(\mu - x_t)\,\Delta t + \sigma\sqrt{\Delta t}\,\mathcal{N}(0,1)$$

donde:

- θ regula la velocidad de retorno hacia el valor medio μ.

- σ controla la magnitud del ruido.
- Δt es el paso de tiempo en la simulación.
- $\mathcal{N}(0, 1)$ es una variable aleatoria Gaussiana estándar.

Este tipo de ruido es particularmente apropiado para entornos físicos donde se espera una evolución suave y dependiente del tiempo (por ejemplo, ajustes de fase o amplitud en señales de radio).

Sin embargo, para este proyecto se ha decidido reforzar esa exploración con la técnica de exploración ε-greedy, comúnmente utilizada en algoritmos de aprendizaje por refuerzo discretos, pero adaptada aquí al caso continuo. Esta técnica consiste en lo siguiente:

- Con probabilidad ε, se selecciona una acción aleatoria dentro del espacio de acciones permitidas.
- Con probabilidad $1 - \varepsilon$, se ejecuta la acción propuesta por la política con el plus de exploración del ruido OU (es decir, la acción generada por la red actor, perturbada por el ruido OU).

Véase el fragmento de código con la implementación combinada

```
def act(self, state, train = True): #Epsilon-greedy + ruido OU
    #Whe are going to use a mix between epsilon-greedy and OU noise
    #Firstly, we predict the action
    action = self.actor_model.predict(state.reshape(1,-1), verbose = 0)[0].reshape(-1,1)

    if train and np.random.rand() <= self.epsilon:
        action = np.random.uniform(-1, 1, (self.action_size,1)) #Exploration with epsilon-greedy
    elif train:
        action += self.noise.sample().reshape(-1,1) #Add the OU noise during the explotation

    #Clip to have the action between [-1, 1]
    action = np.clip(action, -1, 1)
    return action
```

Esta combinación permite al agente explorar tanto de manera dirigida (mediante ruido suave) como de forma más amplia y aleatoria (mediante ε-greedy), incrementando así la cobertura del espacio de acciones durante el entrenamiento.

Importancia de la función de recompensa

La función de recompensa es el elemento central que guía el aprendizaje del agente. Su diseño debe reflejar de forma precisa el objetivo final del sistema, en este caso, maximizar la tasa de secreto [10].

Una recompensa mal definida puede inducir al agente a aprender políticas ineficaces o incluso perjudiciales para el rendimiento del sistema.

La elaboración de una buena función de recompensa es una de las piezas clave, y una de las pártes más delicadas a la par que complicada, para el desarrollo de un modelo de aprendizaje por refuerzo. Para conseguir una buena función de recompensa hay que tener en cuenta algunos aspectos.

- **Escalado:** Si la recompensa tiene una magnitud demasiado pequeña o grande, puede dificultar el aprendizaje. Normalizarla puede ser útil.
- **Diferenciabilidad:** Aunque DDPG puede trabajar con funciones no diferenciables respecto al entorno, una recompensa suave y continua facilita la propagación de gradientes.
- **Diseño específico:** En este caso, se puede usar directamente el valor de $SR(w, \Phi)$ como recompensa, o funciones auxiliares que penalicen tasas negativas, desbalance entre SNRs, o violaciones de restricciones de potencia. Para este proyecto en cuestión, se partió de la base de la tasa de secreto como función de recompensa, y se evolucionó teniendo en cuenta todos los factores que estaban incluidos en el problema y el objetivo específico del modelo a desarrollar.

Por tanto, el comportamiento del agente está estrechamente ligado al diseño de esta función, y experimentar con distintas formulaciones puede ser crucial para lograr un rendimiento óptimo.

3.4. Fundamentos técnicos

Para el desarrollo, entrenamiento y evaluación del agente DDPG, se ha utilizado una infraestructura distribuida basada en una máquina virtual (VM) cedida por la Universidad de Valladolid. Esta elección se debe a que los procesos de entrenamiento profundo pueden ser intensivos en recursos computacionales, especialmente en memoria y CPU, aunque en este caso no se ha requerido aceleración por GPU [3].

3.4.1. Entorno de desarrollo

La máquina virtual fue configurada con un entorno de Python donde se instalaron todas las dependencias necesarias. Para facilitar el flujo de trabajo y la gestión del código, se creó un repositorio en GitHub donde se almacenaron los distintos scripts y modelos entrenados. Este repositorio estaba sincronizado con la máquina virtual, permitiendo mantener un historial de versiones, control de cambios y colaboración eficiente.

El desarrollo del código se realizó desde un ordenador local utilizando **Visual Studio Code** conectado a la máquina virtual a través de `SSH`. Esta configuración permitió programar y ejecutar remotamente sobre la VM, pero con la comodidad de una interfaz gráfica local. Además, como la máquina virtual no contaba con entorno gráfico, todas las gráficas generadas por los entrenamientos se visualizaron desde el entorno local o desde el repositorio.

3.4.2. Librerías y herramientas utilizadas

Durante el desarrollo se emplearon diversas librerías de Python, entre las que destacan:

- **NumPy**: para llevar a cabo todas las operaciones necesarias para modelar el problema ya que todos los datos estaban distribuidos como vectores (como el vector de beamforming w) y matrices (como las distintas matrices de los canales).
- **TensorFlow**: como framework principal para construir y entrenar las redes neuronales del agente DDPG (actor y crítico), además del uso de optimizadores (Adam) y reguladores (L2) [6].

- **Math** y **random**: para las funciones matemáticas básicas que se requerían para modelar el problema, al igual que para la generación del ruido, los canales, etc.
- **Matplotlib** y **Seaborn**: para la generación de gráficos con información sobre los entrenamientos, como la evolución de la recompensa, el progreso de la tasa de secreto, los distintos valores de los parámetros de la recompensa en función del tiempo, etc, y además, para realizar representaciones visuales mediante mapas de calor del estado final de la IRS tras la actuación del modelo.
- **os**, **time**, **psutil**: para operaciones del sistema, control de tiempos de ejecución y monitorización del uso de recursos (memoria y CPU), durante los entrenamientos.

Este conjunto de herramientas permitió implementar, entrenar y analizar el comportamiento del modelo de forma eficiente, manteniendo la modularidad y escalabilidad del proyecto.

Parte II

Desarrollo de la solución

Capítulo 4

Desarrollo de la solución

4.1. Construcción del software

4.1.1. Construcción del agente

En este apartado se detalla el diseño e implementación del agente de aprendizaje por refuerzo profundo utilizado en el proyecto, basado en el algoritmo Deep Deterministic Policy Gradient (DDPG). Este agente ha sido programado desde cero utilizando NumPy y TensorFlow y se compone de dos redes neuronales principales (actor y crítico), sus respectivas redes objetivo (target networks), y una estructura de memoria para el aprendizaje a partir de la experiencia (replay buffer).

Arquitectura Actor-Crítico

El agente ha sido diseñado para operar en un espacio de estados y acciones continuo. Las dimensiones del estado (`state_size`) y de la acción (`action_size`) están definidas en función de los vectores del sistema de comunicaciones:

- **Estado:** Formado por las partes real e imaginarias del vector de beamforming w, el vector de fase ϕ, y el vector de canal efectivo CB, que es el que le pasa la información del canal al agente, y está formado por componentes reales e imaginarias y tiene la misma dimensión de w.

```
self.state_size = 2*self.dim_w + self.dim_phi + 2*self.dim_w
```

- **Acción:** Compuesta por valores que modifican w y ϕ, por lo tanto el tamaño de la acción tiene que ser, los componentes reales de w, los componentes imaginarios de w y el vector ϕ.

```
self.action_size = 2*self.dim_w + self.dim_phi
```

La acción a su vez va a encontrarse dentro de un rango normalizado $[-1, 1]$.

El agente está compuesto por dos redes neuronales, que son:

- **Red actor:** Se encarga de seleccionar las acciones óptimas dado un estado, es por eso que su dimensión de entrada es la dimensión del estado. Consta de varias capas densas de 256, 128, 128 y 64 neuronas respectivamente, como función de activación, las capas ocultas tienen la función `swish`, ya que buscamos que fluidez en la salida y buena propagación del gradiente, mientras que la capa de salida tiene la función `tanh`, con la que buscamos escalar la acción al rango deseado. Para evitar sobreajuste, utilizamos regularización `L2`, con parámetro `1e-5`. Hemos elegido el optimizador `Adam`, de `keras`, y para protegernos contra gradientes explosivos utilizamos `clipnorm=1.0` (técnica de *gradient clipping*).

- **Red crítico:** Estima lo buena o mala que es una acción dado un estado. Su entrada, a diferencia de la red actor, es la concatenación del estado y la acción. Tiene la misma configuración que la red actor, excepto la funciones de activación. Para las capas ocultas, la función de activación que se usa es la función `ReLU`, que mejora la convergencia y la estabilidad, es buena para críticos que buscan salidas continuas, como en nuestro caso.

Para escoger las funciones de activación se estudiaron diversas configuraciones, combinando funciones como `ReLU`, `Leaky ReLU`, `ELU` y `Swish` de distintas formas entre el actor y el crítico. Tras analizar los resultados, se concluyó que la mejor configuración fue la representada en la figura como *Model_ E*.

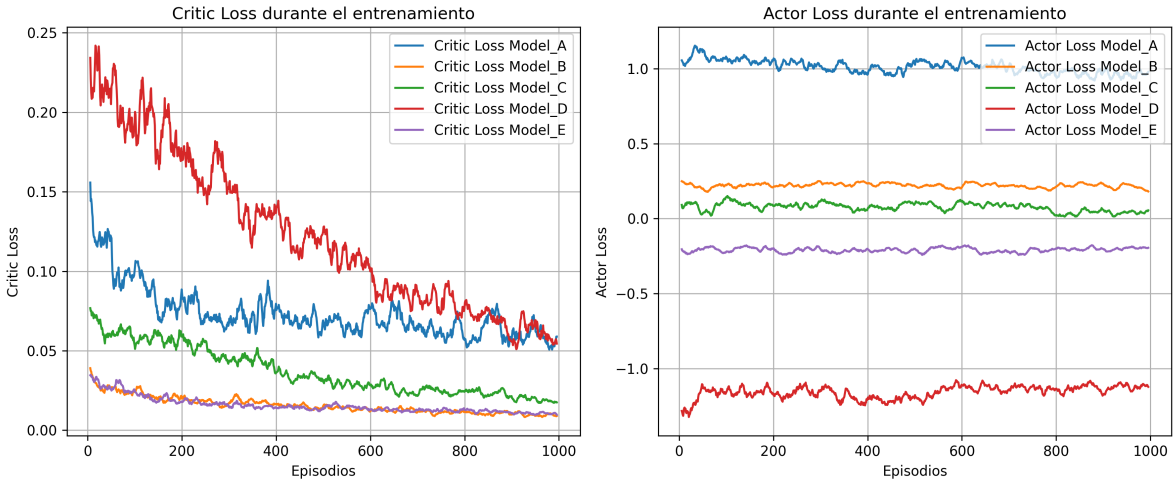

Figura 4.1: Comparación de funciones de activación

Las tasas de aprendizaje del actor y del crítico son independientes, siendo la del actor más baja que la del crítico.

```
self.critic_learning_rate = 1e-5 #Learning rate for the critic_model (0.001)
self.actor_learning_rate = 5e-6 #Learning rate for the actor_model (0.0005)
```

Estas tasas tienen niveles bajos para mayor estabilidad en el entrenamiento, sobre todo para evitar que el critico y el actor hagan saltos bruscos y para aprovechar la precisión de las actualizaciones, especialmente cuando ya está cerca de un óptimo. Además, teniendo en cuenta la configuración de las redes, es la opción que más sentido tiene.

Redes objetivo

Con el objetivo de mejorar la estabilidad del aprendizaje, se implementan redes objetivo para el actor y el crítico. Estas redes se actualizan suavemente tras cada iteración utilizando el parámetro τ

```
self.tau = 0.001 #Used to update target networks
```

y siguiendo la ecuación

$$\theta_{\text{target}} \leftarrow \tau\theta + (1-\tau)\theta_{\text{target}}$$

Inicialmente, estas redes se crean como clones de las redes originales

```
self.actor_target = clone_model(self.actor_model)
self.critic_target = clone_model(self.critic_model)
```

y se actualizan mediante la siguiente función

```
def update_target_networks(self):
    actor_weights = np.array(self.actor_model.get_weights(), dtype = object)
    actor_target_weights = np.array(self.actor_target.get_weights(), dtype = object)
    critic_weights = np.array(self.critic_model.get_weights(), dtype = object)
    critic_target_weights = np.array(self.critic_target.get_weights(), dtype = object)

    new_actor_weights = self.tau * actor_weights
                        + (1 - self.tau) * actor_target_weights
    new_critic_weights = self.tau * critic_weights
                        + (1 - self.tau) * critic_target_weights

    self.actor_target.set_weights(new_actor_weights)
    self.critic_target.set_weights(new_critic_weights)
```

Técnicas de exploración

Para fomentar la exploración eficiente en espacios continuos, se implementa ruido de tipo Ornstein-Uhlenbeck, que genera trayectorias suaves y correlacionadas temporalmente, adecuadas para tareas de control. Además, se aplica una estrategia epsilon-greedy para combinar exploración aleatoria con aprendizaje guiado por las redes.

El ruido de Ornstein-Uhlenbeck (ya definido antes en los fundamentos teóricos en la sección 3.3.8) se ha implementado de la siguiente forma:

```
class OrnsteinUhlenbeckNoise:
    def __init__(self, size, mu=0.0, sigma=0.2, theta=0.15, dt=1e-2,
                           decay_rate=0.995, sigma_end=0.05):
        self.mu = mu
        self.sigma = sigma #Intensidad del ruido
        self.theta = theta #Controla que tan rápido el ruido regresa a la media
        self.dt = dt
        self.size = size
        self.state = np.ones(self.size) * self.mu

        #Vamos a utilizar una estrategia decay para que sigma vaya disminuyendo
        self.sigma_end = sigma_end #Sigma final del que no bajará más
        self.decay_rate = decay_rate  #Mide cuanto de rápido se va apagando el ruido

    def reset(self):
        self.state = np.ones(self.size) * self.mu
        self.sigma = max(self.sigma_end, self.sigma * self.decay_rate)
```

```
def sample(self):
    x = self.state
    dx = self.theta * (self.mu - x) * self.dt
            + self.sigma * np.sqrt(self.dt) * np.random.randn(self.size)
    self.state = x + dx
    return self.state
```

Al igual que la definición, ya se explicó en la sección 3.3.8 la combinación de ambas técnicas con una muestra del código.

Para asegurar una transición gradual de la exploración a la explotación, el parámetro ε se reduce exponencialmente a través de un factor de reducción '*decay*'. Esto se realiza de la siguiente forma, en cada episodio del entrenamiento:

- Si $\varepsilon < \varepsilon_{\text{mín}}$ entonces $\varepsilon = \varepsilon * \varepsilon$-decay.
- En caso contrario, se mantiene el ε actual.

El parámetro de reducción ε-decay se calcula para que se llegue a $\varepsilon_{\text{mín}}$ en el 70 % del entrenamiento

```
def calculate_epsilon_decay(self, episodios):
   self.epsilon_decay = (self.epsilon_min/self.epsilon) ** (1/(0.7*episodios))
   return self.epsilon_decay
```

La evolución del parámetro se puede apreciar en las siguientes gráficas

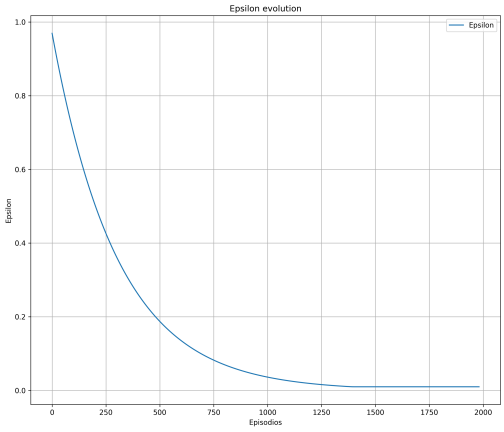

Figura 4.2: Evolución de ε durante el entrenamiento

Replay Buffer

El agente almacena sus experiencias pasadas en una memoria circular de tamaño limitado (2000 experiencias), estas experiencias se guardan de la forma (`state`, `action`, `reward`, `next_state`).

Esta memoria se emplea para entrenar al modelo mediante la selección aleatoria de minibatches, lo que evita la correlación entre muestras consecutivas y mejora la eficiencia del aprendizaje.

Entrenamiento del agente

El entrenamiento del agente se lleva a cabo en tres fases:

- **Actualización del crítico:** a partir del valor estimado del `critic_target`, se ajusta la predicción del `critic_model` minimizando el error cuadrático medio (MSE). Además, para mejorar la estabilidada del gradiente, el valor objetivo se ajusta en el intervalo $[-10, 10]$.
- **Actualización del actor:** esta actualización se hace mediante `tf.GradientTape`, se calcula el gradiente de la función de pérdida del actor.
- **Actualización de las redes objetivo:** finalmente, actualizamos las redes objetivo

Guardado y carga del agente

El agente está preparado para guardar y cargar tanto los modelos entrenados como la memoria de experiencias en archivos `.keras` y `.pkl` respectivamente, lo cual permite, reentrenar o comparar entrenamientos en diferentes versiones del algoritmo.

Flujo del agente

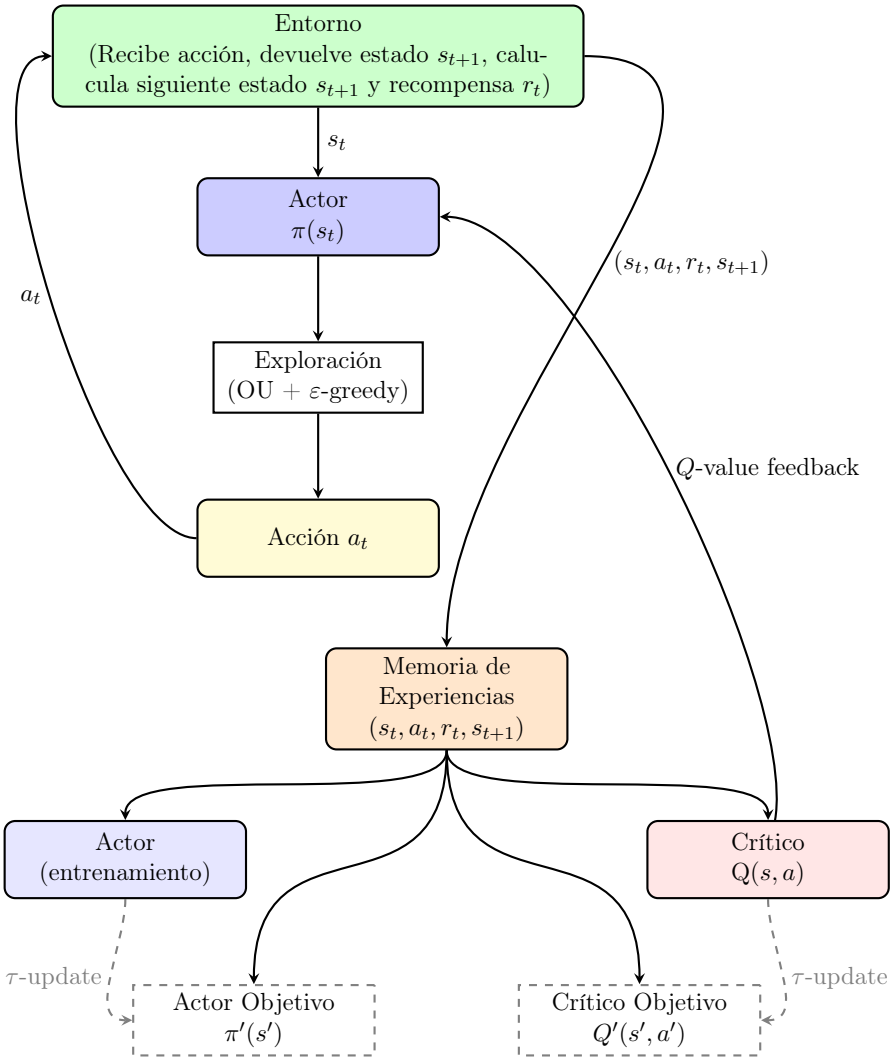

Figura 4.3: Flujo del agente

4.1.2. Entorno simulado

Esta sección describe el diseño del entorno que se ha usado para entrenar y evaluar el rendimiento de los agentes descritos anteriormente en escenarios de comunicación segura mediante superficies reflectantes inteligentes (IRS). En entorno ha sido implementado en Python y simula con detalle los efectos de los canales inalámbricos, el desvanecimiento Rayleigh, el ruido aditivo complejo y la variabilidad estocástica, ofreciento una aproximación fiel de escenarios reales para la toma de decisiones y optimización dinámica.

Descripción del entorno IRS

El entorno simula un sistema MISO (Multiple-Input Single-Output), con un transmisor con múltiples antenas, un receptor legítimo (Bob), un espía (Eve), y una IRS con múltiples elementos

pasivos.

Cuando se crea, el entorno recibe un vector con 11 posiciones, donde cada posición indica un parámetro del sistema:

Posición	Parámetro
0	**Número de antenas en la base**
1	**Número de elementos de la IRS**
2	**Potencia máxima para la transmisión en la base**
3	**Tasa de pérdida de señal en la transmisión**
4	**Varianza en el ruido de Bob**
5	**Varianza en el ruido de Eve**
6	**Distancia entre la base y la IRS**
7	**Distancia entre la base y Bob**
8	**Distancia entre la base y Eve**
9	**Distancia entre la IRS y Bob**
10	**Distancia entre la IRS y Eve**

Tabla 4.1: Parámetros del entorno

El entorno será el que se encargue de modular los canales, el de calcular el siguiente estado en función de una acción dada, de calcular la recompensa que incentivará a aprender al agente, y en resumen de todos los cálculos que tienen que ver con la trasmisión y que no impliquen el cálculo de la acción correspondiente a un estado.

El estado del entorno se representará como composición de la parte real e imaginaria del vector w, las fases ϕ, y el vector de canal efectivo hacia Bob (dividido también en parte real e imaginaria).

```
np.concatenate([self.w.real, self.w.imag, self.phi, eff_B.real, eff_B.imag])
```

Modelado de los canales, fading y tasa de secreto

Los canales se modelan con desvanecimiento Rayleigh, ajustado mediante pérdida de trayectoria de acuerdo a la distancia

```
def rayleigh_fading(self, rows, cols, d):
    channel = np.sqrt(d**-self.chi) * (np.random.randn(rows, cols)
                  + 1j * np.random.randn(rows, cols)) / np.sqrt(2)
    return channel
```

La matriz diagonal de la IRS, se construye mediante el vector de fases ϕ, de la siguiente manera

$$\Phi = diag(e^{j\psi_1}, \dots e^{j\psi_N}) \qquad \text{donde } \psi_i = \phi[i-1]$$

finalmente, la función para el cálculo de la matriz es

```
def calculate_Phi(self, phi):
    return np.diag(np.exp(1j*phi.flatten()))
```

Finalmente, para la respresentación de la tasa de secreto en el entorno, el cálculo se sigue tal y como se explica en (3.1), reflejado mediante las siguientes funciones

```
#Cálculo de la señal en Bob
def calculate_CB(self, Phi, w):
    CB = (np.linalg.norm((np.conj(self.h_IB).T @ Phi @ self.H_TI + np.conj(self.h_TB).T) @ w)**2)
            / self.sigma_B
    return CB

#Cálculo de la señal en Eve
def calculate_CE(self, Phi, w):
    CE = (np.linalg.norm((np.conj(self.h_IE).T @ Phi @ self.H_TI + np.conj(self.h_TE).T) @ w)**2)
            / self.sigma_E
    return CE

#Cálculo de la tasa de secreto
def secret_rate(self, phi, w):
    Phi = self.calculate_Phi(phi)
    CB = self.calculate_CB(Phi, w)
    CE = self.calculate_CE(Phi, w)
    sr = math.log2(1+CB) - math.log2(1+CE)
    return max(0, sr)
```

Ruido y modelado estocástico

El entorno simula la transimisión de una señal mediante un conjunto de símbolos QAM aleatorios (`tx_symbols`) para simular condiciones reales de la señal y calcular las relaciones señal-ruido (SNR).

Tanto Bob como Eve reciben una versión ruidosa de la señal transmitida con un ruido gaussiano cmplejo aditivo, que viene dado por

$$n_B = \mathcal{CN}(0, \sigma_B^2) \qquad \text{y} \qquad n_E = \mathcal{CN}(0, \sigma_E^2)$$

La señal recibida, es la que viene dada en cada caso por las ecuaciones (3.3), implementadas de la siguiente forma

```
def calculate_signal_Bob(self, tx_symbols, Phi, w):
    #Calculamos el ruido en Bob
    nb = (np.random.randn(len(tx_symbols)) + 1j * np.random.randn(len(tx_symbols))) * self.sigma_B
    #Calculamos la señal de la transimión en Bob
    rx_Bob = (np.conj(self.h_IB).T @ Phi @ self.H_TI + np.conj(self.h_TB).T) @ w * tx_symbols + nb
    return rx_Bob

def calculate_signal_Eve(self, tx_symbols, Phi, w):
    #Calculamos el ruido en Eve
    ne =  (np.random.randn(len(tx_symbols)) + 1j * np.random.randn(len(tx_symbols))) * self.sigma_E
    #Calculamos la señal de la transimión
    rx_Eve = (np.conj(self.h_IE).T @ Phi @ self.H_TI + np.conj(self.h_TE).T) @ w * tx_symbols + ne
    return rx_Eve
```

Dinámica del entorno (episodios y pasos)

Cada episodio simula una transmisión bajo una realización fija o variable de los canales. Un episodio puede durar un número indefinico de pasos, ya que no definimos ninguna condición de finalización explícita. En cada paso, el agente aplica una acción sobre el estado actual, y el entorno devuelve un nuevo estado y una recompensa basada en el cambio de desempeño del sistema.

Como en la vida real, los canales varían en el tiempo, a la vez que vamos a tener múltiples estados de w y ϕ, se contempla la opción de reiniciar de forma aleatoria tanto los canales como las variables w y ϕ, cada vez que se reinicia el entorno.

Funciones de recompensa

En aprendizaje por refuerzo, una de las partes más importantes a la par que difíciles y sutiles, es el desarrollo de una buena función de recompensa que sirva para guiar el entrenamiento del agente. En nuestro caso, la construcción de la función de recompensa ha pasado por varias fases, partiendo desde una recompensa simple, dada por la diferencia de tasa de secreto, evolucionando hasta una recompensa compleja adecuada para el problema real y adaptándose de forma absoluta al problema. A continuación se resumen brevemente las distintas recompensas del modelo.

- **Recompensa 1 (Diferencia de tasa de secreto):** Esta es una recompensa inicial, de las etapas más tempranas del proyecto que permitía a grosso modo dirigir el modelo hacia donde queríamos, sin enfocarnos mucho en el detalle. Consistía en efectuar la diferencia entre la tasa de secreto anterior, y la calculada tras aplicar la acción. Debido a que esta diferencia podía ser pequeña en cuanto a magnitud, se escala multiplicándola por 10.

```
def calculate_reward_1(self, new_secret_rate):
    if self.prev_secret_rate is None:
        #Si no hay ninuna tasa anterior, se devolverá 0
        self.prev_secret_rate = new_secret_rate
        return 0

    reward = new_secret_rate - self.prev_secret_rate
    self.prev_secret_rate = new_secret_rate
    return 10*reward
```

- **Recompensa 2 (Penalización estática de la señal de Eve):** Esta recompensa es una pequeña evolución de la anterior. En esta se desglosa el cálculo de la tasa de secreto en las señales de Bob (CB) y Eve (CE), y se penaliza el aumento de la señal de Eve con un parámetro estático, que tras muchas pruebas se fijó en 9. De todas formas, este parámetro se puede modificar en la consola cuando se eligen los parámetros de entrenamiento del modelo.

  Como esta recompensa puede dar valores pequeños en magnitud, se escala la recompensa multiplicándola por 100

```
def calculate_reward_2(self, phi, w):
    Phi = self.calculate_Phi(phi)

    #Calculamos los valores tras aplicar la acción
    new_CB = self.calculate_CB(Phi, w)
    new_CE = self.calculate_CE(Phi, w)
    new_secret_rate = self.secret_rate(phi, w)
```

```
        #Si no están los valores previos seteados, entonces se establecerán y el resultado será 0
        if self.prev_CB is None:
            self.prev_CB = new_CB
        if self.prev_CE is None:
            self.prev_CE = new_CE
        if self.prev_secret_rate is None:
            self.prev_secret_rate = new_secret_rate

        #Si el usuario no ha introducido valores de parámetros, entonces se establecerán
        #los predeterminados
        if(self.alpha is None):
            alpha = 9 # Factor de penalización de CE
        else:
            alpha = self.alpha
        if(self.beta is None):
            beta = 5 # Factor de addición de SR
        else:
            beta = self.beta

        #Cálculo de la recompensa
        reward_1 = new_CB - self.prev_CB
        reward_2 = self.prev_CE - new_CE
        reward_3 = new_secret_rate - self.prev_secret_rate
        reward = reward_1 + (alpha)*reward_2+(beta)*reward_3
        reward = 100*reward

        self.prev_CB = new_CB
        self.prev_CE = new_CE
        self.prev_secret_rate = new_secret_rate

        return reward, alpha, beta
```

- **Recompensa 3 (Penalización dinámica de CE):** Esta recompensa surge como solución a un problema que tenía la recompensa anterior, y es que había veces que el agente aprendía a reducir la señal de Eve reduciendo toda la señal, y es por eso que premiaba mucho más la pérdida de señal de Eve que la evolución de la tasa de secreto. Como solución se consideró hacer que el parámetro α fuera dinámico, y en caso de que la tasa de secreto bajara mucho, este parámetro se reduciría para que el incremento de la tasa de secreto vuelva a tomar el protagonismo, y así evitar que el agente aprenda a reducir toda la señal.

  Además de esto, también se normalizan las diferencias de CE, CB y SR, para que todo esté en la misma magnitud y no haya problemas de escalado.

```
    def calculate_reward_3(self, phi, w):
        Phi = self.calculate_Phi(phi)

        #Calculamos los valores tras aplicar la acción
        new_CB = self.calculate_CB(Phi, w)
        new_CE = self.calculate_CE(Phi, w)
        new_secret_rate = self.secret_rate(phi, w)

        #Si no están los valores previos seteados, entonces se establecerán
        #y el resultado será 0
        if self.prev_CB is None:
            self.prev_CB = new_CB
        if self.prev_CE is None:
            self.prev_CE = new_CE
        if self.prev_secret_rate is None:
            self.prev_secret_rate = new_secret_rate

        #Calculamos las diferencias normalizadas, y para suavizar los valores
```

```
        #usamos la tangente hiperbólica
        delta_CB = np.tanh((new_CB - self.prev_CB) / (abs(self.prev_CB) + 1e-6))*2
        delta_CE = np.tanh((new_CE - self.prev_CE) / (abs(self.prev_CE) + 1e-6))*2
        delta_SR = np.tanh((new_secret_rate - self.prev_secret_rate)
                    / (abs(self.prev_secret_rate) + 1e-6))*2

        #Calculamos el parámetro dinámico alpha
        alpha = 1 + max(0, delta_CE*5) #Penaliza más cuando CE mejora rápido

        #El valor de beta puede ser introducido por el usuario, en caso contrario se
        #establecerá por defecto a 2
        if(self.beta is None):
            beta = 2 # SR delta factor
        else:
            beta = self.beta

        #Cálculo de la recompensa
        reward = delta_CB - alpha*delta_CE + beta*delta_SR

        self.prev_CB = new_CB
        self.prev_CE = new_CE
        self.prev_secret_rate = new_secret_rate

        return reward, alpha, beta
```

- **Recompensa 3 versión absoluta:** Es una modificación de la recompensa anterior para que se tenga en cuenta solo el caso en el que nos encontramos, que no se tengan en cuenta métricas anteriores, en este caso, el parámetro α será más grande cuanto más se acerque CE a CB.

```
def calculate_reward_absolute_3(self, phi, w):
    Phi = self.calculate_Phi(phi)

    #Calculamos las métricas con las que vamos a trabajar
    new_CB = self.calculate_CB(Phi, w)
    new_CE = self.calculate_CE(Phi, w)
    new_SR = self.secret_rate(phi, w)

    #Calculamos el logaritmo de las señales de Eve y Bob
    CB_log = math.log2(1+new_CB)
    CE_log = math.log2(1+new_CE)

    #Calculamos el factor dinámico
    alpha = 1 + np.tanh(CE_log / (CB_log + 1e-6)) * 2  # Penaliza más si CE se acerca a CB

    #El valor de beta puede ser introducido por el usuario, en caso contrario se
    #establecerá por defecto a 2
    if(self.beta is None):
        beta = 2 # SR delta factor
    else:
        beta = self.beta

    #Calculamos la recompensa
    reward = CB_log - alpha*CE_log + beta*new_SR

    return reward, alpha, beta
```

- **Recompensa 4 (Recompensa compleja):** Esta recompensa se hizo tras un análisis intensivo de las métricas de éxito del modelo. Se elaboró como representación más fiel del caso real, teniendo en cuenta las señales recibidas por Bob y Eve, la tasa de secreto, y los ratios señales-ruido de ambos. Además, los parámetros de penalización de CE y de refuerzo

de la SR se calculan de manera dinámica para evitar que la tasa de secreto caiga (es la recompensa más completa).

```
def calculate_reward_4(self, phi, w, tx_symbols):
    Phi = self.calculate_Phi(phi)

    #Calculamos las métricas con las que vamos a trabajar
    new_CB = self.calculate_CB(Phi, w)
    new_CE = self.calculate_CE(Phi, w)
    new_secret_rate = self.secret_rate(phi, w)

    #Si no están los valores previos seteados, entonces se establecerán
    #y el resultado será 0
    if self.prev_CB is None:
        self.prev_CB = new_CB
    if self.prev_CE is None:
        self.prev_CE = new_CE
    if self.prev_secret_rate is None:
        self.prev_secret_rate = new_secret_rate

    #Calculamos las relaciones señal-ruido
    rx_Bob = self.calculate_signal_Bob(tx_symbols, Phi, w)
    rx_Eve = self.calculate_signal_Eve(tx_symbols, Phi, w)
    snr_bob = self.calculate_snr(np.mean(np.abs(rx_Bob) ** 2), self.sigma_B)
    snr_eve = self.calculate_snr(np.mean(np.abs(rx_Eve) ** 2), self.sigma_E)

    #Calculo de la recompensa
    if(self.prev_CB == 0 and new_CB != 0):
        rew1 = 1
    elif(self.prev_CB == 0 and new_CB == 0):
        rew1 = 0
    else:
        rew1 = np.clip((new_CB - self.prev_CB) / (abs(self.prev_CB)), -1,1)

    if(self.prev_CE == 0 and new_CE != 0):
        rew2 = 1
    elif(self.prev_CE == 0 and new_CE == 0):
        rew2 = 0
    else:
        rew2 = np.clip((self.prev_CE - new_CE) / (abs(self.prev_CE)), -1,1)

    rew3 = np.tanh(snr_bob-snr_eve) #Para que esté entre (-1 y 1)
    rew4 = new_secret_rate - self.prev_secret_rate

    #Calculamos los factores dinámicos para la penalización
    beta = 4 - min(0, rew2*2)
    alpha = 2 - min(0, rew4*2)

    reward = rew1 + beta*rew2 + rew3 + alpha*rew4

    return reward, alpha, beta
```

- **Recompensa 4 versión absoluta:** Al igual que en la recompensa 3, esta es una modificación de la recompensa 4 para que se tenga en cuenta solo el caso en el que nos encontramos, que no se tengan en cuenta las métricas anteriores, al igual que en la recompensa 3, el parámetro α será más grande cuanto más se acerque CE a CB.

```
def calculate_reward_absolute_4(self, phi, w, tx_symbols):
    Phi = self.calculate_Phi(phi)

    #Calculamos las métricas con las que vamos a trabajar
    new_CB = self.calculate_CB(Phi, w)
    new_CE = self.calculate_CE(Phi, w)
```

```
    new_secret_rate = self.secret_rate(phi, w)

    if self.prev_secret_rate is None:
        #If there isnt any previous SR
        self.prev_secret_rate = new_secret_rate

    #Calculamos las relaciones señal-ruido
    rx_Bob = self.calculate_signal_Bob(tx_symbols, Phi, w)
    rx_Eve = self.calculate_signal_Eve(tx_symbols, Phi, w)
    snr_bob = self.calculate_snr(np.mean(np.abs(rx_Bob) ** 2), self.sigma_B)
    snr_eve = self.calculate_snr(np.mean(np.abs(rx_Eve) ** 2), self.sigma_E)

    #Calculamos la recompensa
    rew1 = math.log2(1+new_CB)
    rew2 = math.log2(1+new_CE)
    rew3 = np.tanh(snr_bob-snr_eve) #Para que esté entre (-1 y 1)
    rew4 = new_secret_rate - self.prev_secret_rate

    #Calculamos los factores dinámicos para la penalización
    alpha = 1 + np.tanh(rew2 / (rew1 + 1e-6)) * 2
    beta = 2 - min(0, rew4*2)

    reward = rew1 + alpha*rew2 + rew3 + beta*rew4

    return reward, alpha, beta
```

- **Recompensa 5 (Penalización absoluta de CE):** Con esta recompensa lo que pretendemos es volver al estado inicial, buscar la mejora de la tasa de secreto pero esta vez con penalización de la señal CE de forma absoluta, solo teniendo en cuenta el estado en el que nos encontramos, este parámetro α puede ser seleccionado por el usuario, pero en caso de que no se seleccione ninguno, se elegirá el valor de 2 (da buenos resultados).

```
def calculate_reward_5(self, phi, w):
    Phi = self.calculate_Phi(phi)
    if(self.alpha is None):
        alpha = 2 # Tiene que ser > 1 (2 es un buen parámetro)
    else:
        alpha = self.alpha

    CB = self.calculate_CB(Phi, w)
    CE = self.calculate_CE(Phi, w)

    reward = math.log2(1+CB) - alpha*math.log2(1+CE)
    return reward, alpha
```

- **Recompensa 6 (Penalización absoluta de CE con SP):** Esta recompensa es una mejora de la anterior ya que se aplica una técnica denominada *shaping progresivo*. Esta técnica consiste en hacer que el modelo reciba al principio una recompensa por ir acercándose al objetivo y que poco a poco se vaya especificando. Aplicado a nuestro caso consiste en, primero mejorar simplemente CB, pero poco a poco ir aumentando el parámetro de penalización de CE para que vaya aprendiendo a penalizarlo a la vez que mejora CB. El parámetro alpha llega a su último valor `alpha_end` en el último episodio, y empieza a incrementarse en el 20 % del entrenamiento.

```
def calculate_reward_6(self, phi, w, episode, total_episodes):
    Phi = self.calculate_Phi(phi)

    #Calculamos las señales
    CB = self.calculate_CB(Phi, w)
    CE = self.calculate_CE(Phi, w)
```

```
alpha_end = 2  #El alpha final va a ser 2
alpha_start = 0.5 #El valor inicial va a ser de 0.5

#Aplicamos el shaping progresivo
if episode < 0.2 * total_episodes:
    alpha = alpha_start
else:
    progress = (episode - 0.2*total_episodes) / (0.8*total_episodes)
    alpha = alpha_start +  progress * (alpha_end - alpha_start)

reward = math.log2(1+CB) - alpha*math.log2(1+CE)
return reward, alpha
```

4.1.3. Entrenamiento del modelo

Pipeline de entrenamiento

El proceso de entrenamiento del modelo se organiza en una función principal encargada de coordinar cada etapa del ciclo de aprendizaje. Dicha función recibe como argumentos el entorno (`env`), el agente (`Anthena`), y los parámetros, de los cuales se utilizarán el número de antenas de la base (`M`) y el número de elementos de la IRS (`N`). El entrenamiento se lleva a cabo en episodios, donde en cada uno de ellos el agente interactúa con el entorno durante un número definido de pasos. A continuación se describen las etapas principales del pipeline:

1. **Configuración inicial:** En esta etapa se solicita al usuario el nombre que va a tener el modelo (con este nombre se guardarán las gráficas, los archivos del modelo entrenado y el `.csv` con los datos del entrenamiento, el número total de episodios de entrenamiento, la cantidad de pasos por episodio y la frecuencia con la que se reinician los canales del entorno.

   También se permite la elección interactiva de la función de recompensa y si fuera necesario, el valor de los parámetros α y β.

2. **Inicialización del agente y logs:** Se ajusta el ε-decay del agente, para controlar la exploración-explotación durante el entrenamiento, y se inicializan las estructuras para registrar recompensas, métricas internas, y recursos consumidos durante el proceso.

3. **Bucle de entrenamiento:** En cada episodio, el entorno se reinicia aleatoriamente según una lógica condicional basada en la frecuencia de reinicio establecida. Se ejecutan los pasos del episodio, donde el agente toma decisiones mediante su política actual, y actualiza su memoria con distintas experiencias.

   Una vez finalizado el episodio, se realiza una fase de aprendizaje a partir del *replay buffer*, actualizando las redes del actor y del crítico y las redes objetivo. Cuando termina esta fase de entrenamiento se almacenan métricas como la recompensa media, evolución de ε, tasa de secreto media, pérdidas del actor y del crítico, y el uso de recursos.

4. **Elaboración de gráficas y guardado de datos:** Al finalizar el bucle de entrenamiento, se elaboran las gráficas necesarias para ver y analizar el entrenamiento, se guardan los parámetro del entrenamiento en un archivo `.csv` y por último se guardan los datos de las redes neuronales al igual que el *replay buffer* por si se quiere proseguir con el entrenamiento de este modelo o hacer pruebas a posteriori con él.

Nota: En caso de que se quiera entrenar un modelo en diferentes etapas, con recompensas distintas en cada etapa se puede hacer reentrenando el modelo, sin embargo, también se puede sustituir la parte de reinicio del entorno por el siguiente fragmento, que se adaptaría a las pruebas que se quisieran hacer:

```
if (entrenamiento < 300): #Hasta el episodio 299
    state = env.reset(w=True, phi=True, Hs=False)
    env.choosen_reward = 6
elif (entrenamiento < 1000): #Desde el 300 hasta el 999
    state = env.reset(w=True, phi=True, Hs=(entrenamiento % 5 == 0))
    env.choosen_reward = 3
else: #Desde el episodio 1000 hasta el final
    state = env.reset(w=True, phi=True, Hs=True)
    env.choosen_reward = 4
```

o en caso de que en vez de especificar el número de episodios exactos se quiere especificar el porcentaje del entrenamiento:

```
if (entrenamiento < 0.2*episodios): #Durante el primer 20%
    state = env.reset(w=True, phi=True, Hs=False)
    env.choosen_reward = 6
elif (entrenamiento < 0.5*episodios): #Durante el siguiente 30%
    state = env.reset(w=True, phi=True, Hs=(entrenamiento % 10 == 0))
    env.choosen_reward = 3
elif (entrenamiento < 0.8*episodios): #Durante el siguiente 30%
    state = env.reset(w=True, phi=True, Hs=(entrenamiento % 5 == 0))
    env.choosen_reward = 4.5
else: #Del 80% al final
    state = env.reset(w=True, phi=True, Hs=True)
    env.choosen_reward = 4
```

Parámetros usados

Los principales parámetros configurables durante el entrenamiento son:

- **Número de episodios:** Define cuántas veces el agente interactuará con el entorno completo (cuantas veces se entrenará)
- **Pasos por episodio:** Controla la longitud máxima de cada episodio (Lo que aporta el episodio a la memoria de entrenamientos)
- **Frecuencia de reinicio de canal:** Permite elegir el número de veces que el entorno se entrena con el mismo canal, conseguir una buena frecuencia de reinicio de canal es clave para estabilizar el entrenamiento del agente.
- **Recompensa escogida:** Se selecciona al inicio del entrenamiento y permite escoger la función de recompensa, de entre las antes mencionadas, que guiará al agente durante el proceso.

En un futuro se plantea que en cada entrenamiento se puedan elegir los parámetros del entorno, como la distancia de la base a los distintos elementos, la tasa de ruido del canal, los elementos de la IRS, el número de antenas de la base... Sin embargo, actualmente no es necesario tanto nivel de detalle.

Visualización y registros

Durante el entrenamiento se realiza un registro detallado de métricas clave, almacenadas en un diccionario `episode_logs`. Estos registros se utilizan para generar visualizaciones interactivas entre las que destacan:

- Evolución de la recompensa media
- Pérdidas del actor y del crítico
- Variación de epsilon durante el entrenamiento
- Evolución de los componentes de la recompensa
- Tasa de secreto media por episodio

Además, el almacenamiento de estos datos permite la representación de múltiples modelos en un mismo gráfico, favoreciendo así la comparación entre distintos parámetros de aprendizaje.

A la hora de representar los gráficos, se ajustan dinámicamente según la duración del entrenamiento, utilizando una ventana móvil adecuada al número de episodios, esto es para que no se vea tan '*apelotonado*'.

Uso de recursos y ejecución remota

Durante el entrenamiento, se realiza un seguimiento del uso de recursos computacionales mediante la librería psutil, registrando:

- Porcentaje de CPU usado
- Uso de memoria RAM
- Tiempo empleado en cada episodio

Esto permite analizar el rendimiento del sistema durante el entrenamiento y facilita la comparación entre ejecuciones.

La estructura del código y los registros generados facilitan la ejecución remota del entrenamiento o en entornos distribuidos, al centralizar las métricas, usar archivos para el guardado de estado y mantener independencia modular entre agente, entorno y controlador de entrenamiento. Esta estructura se ha elegido así ya que en un futuro se plantea un modelo en el que de forma distribuida se vayan entrenando dos agentes, uno que se especialice en las acciones para el vector de beamforming w y otro que se especialice para el vector de fases de la IRS, ϕ.

4.1.4. Versionado y evolución del modelo

Historial de versiones

El desarrollo del modelo ha seguido un enfoque iterativo y meticulosamente versionado y documentado, manteniendo un control exhaustivo mediante Github y un documento de evolución. A continuación se resume la evolución de las versiones principales:

- **Versión 1:** Fue el inicio del proyecto, un modelo inicial y muy primitivo que contaba con las herramientas básicas para el entendimiento de un modelo DDPG. Fue completamente desarrollado en Jupyter Notebooks, ya que esta primera versión sirvió de base para asentar el entorno, para realizar los primeros experimentos (todo con restricciones simuladas no reales), y se pudo obtener una pequeña versión funcional pero muy básica.
- **Versión 2:** Esta versión se usó para implementar todo lo relacionado con las redes objetivo, ruido, y carácterísticas complejas que tiene un modelo más avanzado de DDPG. Además fue el comienzo de la investigación para técnicas más avanzadas. Sobre este modelo se hicieron muchos cambios de golpe y se volvió altamente inestable, tuvo que ser descartado y se volvió a la versión inicial.
- **Versión 3:** También desarrollada en Jupyter Notebooks. Sobre esta versión se implementaron poco a poco algunos de los cambios vistos en la versión 2, sin embargo se fueron implementando poco a poco y estabilizandose uno a uno, logrando así un modelo estable para canales estáticos y parámetros simulados.

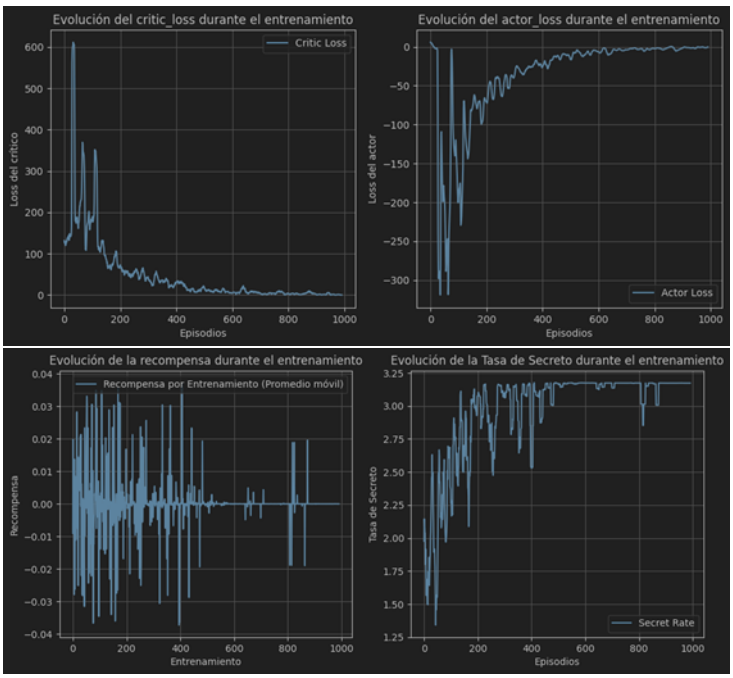

Figura 4.4: Entrenamiento exitoso Versión 3

- **Versión 4:** En esta versión fue cuando se me cedió la máquina virtual, es por esto que supuso una transición de Jupyter Notebooks a un único script en Python. Se mejoró la estructura del código y se introdujeron las primeras versiones de las redes objetivo completamente funcionales, al igual que el ruido de OU, se hicieron también recompensas más elaboradas (recompensas 2 y 3) y se consiguió un entorno más controlado.
- **Versión 5:** Se refactorizó el código en módulos separados: agente, entorno, modelos auxiliares, interfaz de consola y display de gráficas. Esto permitió una mayor flexibilidad a la hora de realizar pruebas y entrenamientos personalizados, así como una mejor trazabilidad,

mantenimiento y reutilización del código. Se siguieron desarrollando las recompensas más complejas (recompensa 4 y retoques en la recompensa 3). Con este modelo se consiguió comprender un problema que llevábamos arrastrando desde el principio, este era que el modelo no recibía información útil del canal.

- **Versión 6:** Modularización completa del código. En esta versión, la arquitectura del proyecto permite una separación total del entorno, agente, entrenamiento, pruebas, visualización y configuración. Está preparada para soportar entrenamientos distribuidos en el futuro (por ejemplo, separación entre dos agentes, uno que optimice w y otro que optimice ϕ).

  En esta versión del modelo también se han desarrollado las recompensas absolutas (recompensas 3-absoluta, 4-absoluta, 5 y 6), y se han aplicado técnicas específicas orientadas a la estabilidad del modelo y a menor sobreajuste, para así conseguir mejores resultados, algunas de estas técnicas son la regularización L2, el shaping progresivo, o el clipping en el gradiente.

  También se ha conseguido una mejora notable en la interfaz de consola y monitorización del entrenamiento, permitiendo ver los recursos que consume y solapar gráficas para la comparación de entrenamientos.

Cada versión ha implicado cambios en arquitectura, recompensas, hiperparámetros, estructura del agente y estrategias de exploración. Todos los cambios están documentados en un documento de seguimiento al igual que controlados y sincronizados en GitHub.

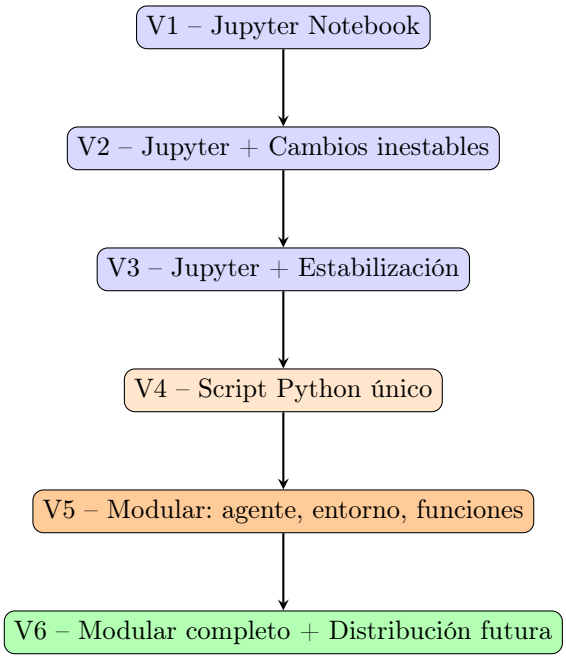

Figura 4.5: Evolución arquitectónica del modelo a través de las versiones

Esta evolución refleja no solo una mejora técnica progresiva, sino también una maduración en el enfoque de diseño, desde una experimentación exploratoria hasta una solución robusta, escalable y preparada para escenarios distribuidos y multiagente.

Problemas encontrados y soluciones

Durante el desarrollo del modelo se han identificado diversos problemas técnicos y conceptuales que se han ido resolviendo progresivamente:

- **Exploración ineficaz:** La exploración aleatoria inicial (ε-greedy puro) resultó poco eficiente. Se introdujo por esto ruido de Ornstein-Uhlenbeck (OU) para suavizar la exploración, pero resultó ser demasiado suave y no se conseguía suficiente exploración. Es por esto que se optó por una estrategia mixta, en la que se combinaban ambas exploraciones, estrategia que dió muy buenos resultados de exploración, y permitió un mayor control sobre esta.

- **Recompensas poco informativas:** Como ya se ha mencionado anteriormente, la elaboración de una buena función de recompensa es crucial para el desarrollo de un modelo de aprendizaje por refuerzo, es por esto que las primeras versiones de las funciones de recompensa eran muy primitivas y proporcionaban señales débiles que no incentivaban bien al modelo a aprender. Para solucionar esto se escalaron las recompensas, se normalizaron para que las magnitudes no variaran fuertemente y se formularon nuevas que incluían penalización a Eve, y métricas derivadas de la tasa de secreto como el ratio señal-ruido (SNR).

- **Inestabilidad numérica en el crítico:** Cuando el modelo se empezó a volver cada vez más complicado, el `critic_loss` explotaba debido a desbalanceos. Esto era imperativo corregirlo ya que la función del crítico era muy importante en el modelo, por lo que su inestabilidad suponía la inestabilidad del aprendizaje global. Para corregirlo, se ajustó la arquitectura de las redes neuronales, se normalizaron las entradas, se ajustó el parámetro de actualización de las redes objetivo y se usó regularizadores tanto en el actor como en el crítico. Con estas medidas se consiguió una alta estabilidad en el aprendizaje incluso con cambio de canal constante.

- **Sobreajuste o estancamiento:** En múltiples ocasiones el modelo aprendía políticas subóptimas o se quedaba estancado. Esto es debido a que en algunos casos, las recompensas penalizaban tanto CE que se conseguía más recompensa bajando la tasa de secreto, ya que se reducía la señal en general. Para mitigar esto se introdujo una exploración controlada y un minucioso análisis de todos los parámetros de las recompensas para encontrar el parámetro adecuado.

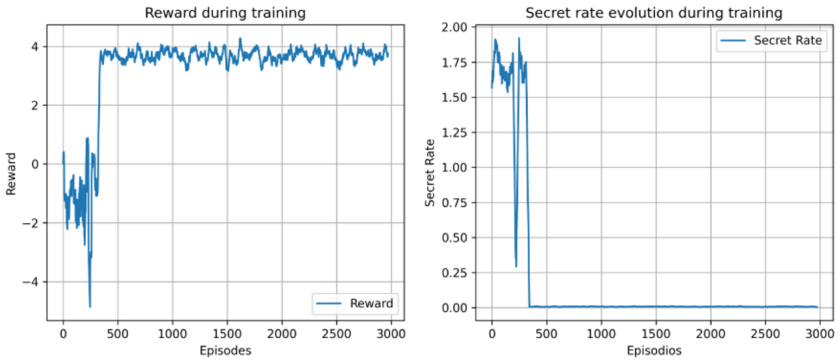

Figura 4.6: Modelo sobreajustando a política subóptima

- **No llegaba información útil del canal al modelo:** Este fue uno de los problemas más graves que me encontré en el desarrollo del modelo, y es que el modelo aprendía a aumentar la tasa de secreto, pero de forma general, ya que no le llegaba información útil del canal para que se pudiera adaptar bien.

  Para solucionar este problema se podrían haber pasado parámetros como son las matrices de los canales, y hubiera estado solucionado, sin embargo, esto no era realista, ya que en un entorno real no se puede saber el estado exacto del canal. Tuve que realizar una investigación más a fondo de que métricas se podían obtener en ámbitos reales, y la elegida fué la señal recibida por el receptor legítimo, esto ayuda al modelo a conocer el entorno en el que está trabajando, y poder adaptarse así a las condiciones. Está fue una de las razones por las que nació la versión 6.

Estos no fueron los únicos problemas surgidos a lo largo del desarrollo, sin embargo si que son los más importantes y los que más afectaron a la elaboración del modelo y al camino que tomó.

Comparación entre versiones

A lo largo del proceso, se han ido comparando sistemáticamente las versiones del modelo mediante:

- Gráficas como las de recompensa y tasa de secreto por episodio.
- Evolución del critic y actor loss, tanto por episodio, como a nivel general.
- Distintas baterías de pruebas a posteriori con distintos canales, vectores y condiciones iniciales.
- Parámetros de consumo de recursos para ver el rendimiento.

Los resultados muestran una mejora progresiva en la capacidad del modelo para generalizar y maximizar la tasa de secreto:

Versión	Mejora de tasa de secreto	Estabilidad	Modularidad	Exploración
V1	Baja	Baja	Nula	Aleatoria
V2	Inestable incluso para entornos sin cambio de canal	Baja	Nula	Aleatoria
V3	Alta en entornos sin cambio de canal	Media	Baja	Aleatoria
V4	Variable	Media-Alta	Media	Mixta
V5	Media con buen tuning	Alta	Alta	Mixta
V6	Media-Alta mantenida	Alta	Muy alta	Mixta adaptativa

Tabla 4.2: Comparación entre versiones

En particular, la combinación de recompensas dinámicas, ruido OU, y canal cambiante en cada episodio desde los últimos entrenamientos de la versión 4, ha permitido que los modelos alcancen buenos valores de tasa de secreto en diversos entornos.

Selección de la versión final

Como versión final del proyecto se ha seleccionado la versión 6, debido a las siguientes razones:

- Permite un entrenamiento robusto, incluso con canales cambiantes en cada episodio (aunque para más estabilidad al principio se suele poner cambio cada 5 episodios).
- Incluye exploración mixta, recompensas complejas y adaptativas.
- Ha mostrado resultados consistentes en la mayoría de pruebas a posteriori.
- Su arquitectura modular facilita el mantenimiento, reentrenamiento, validación y futuras extensiones (como entrenamientos distribuidos).
- Es completamente trazable y replicable, con control de versiones detallado en GitHub.

4.1.5. Aplicación interactiva

Diseño de la interfaz por consola

Dado que el desarrollo y entrenamiento del modelo se realiza en una máquina virtual sin entorno gráfico ni acceso a GPU, se ha diseñado una interfaz interactiva por consola completamente funcional. Esta interfaz permite operar con el modelo sin necesidad de modificar directamente el código fuente, haciendo uso de menús y comandos estructurados que guían al usuario en todas

las etapas del ciclo de vida del modelo: configuración, entrenamiento, evaluación y análisis de resultados.

El diseño está centrado en la usabilidad por terminal, empleando colores, formatos y elementos visuales como tablas o barras de progreso para mejorar la experiencia del usuario. Todas las interacciones clave, como la elección de parámetros, selección de modelos, visualización de gráficas o acceso a la arquitectura de las redes neuronales entrenadas, están accesibles desde la interfaz.

Al ejecutar el programa por primera vez se desplegará un primer menú

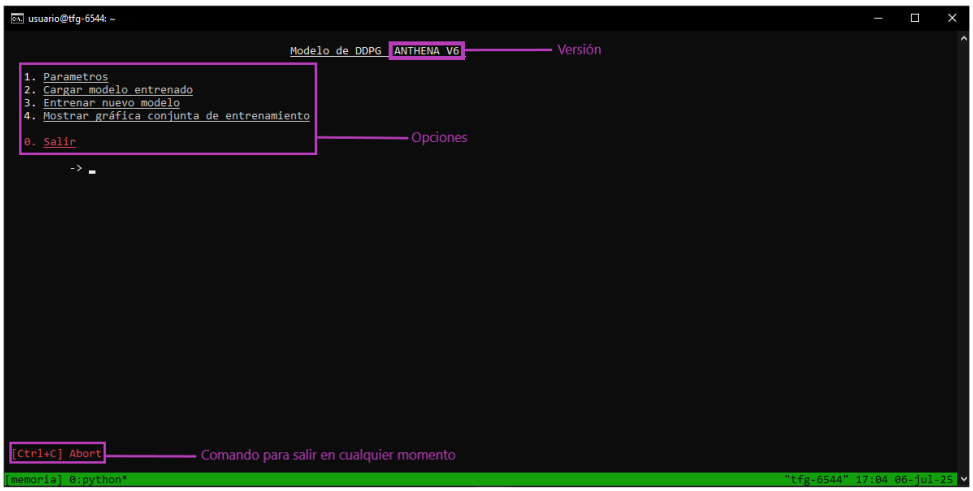

Figura 4.7: Menú principal

1. **Parámetros.** Al introducir la primera opción, se muestra un menú con todos los parámetros de emulación del entorno cargados.

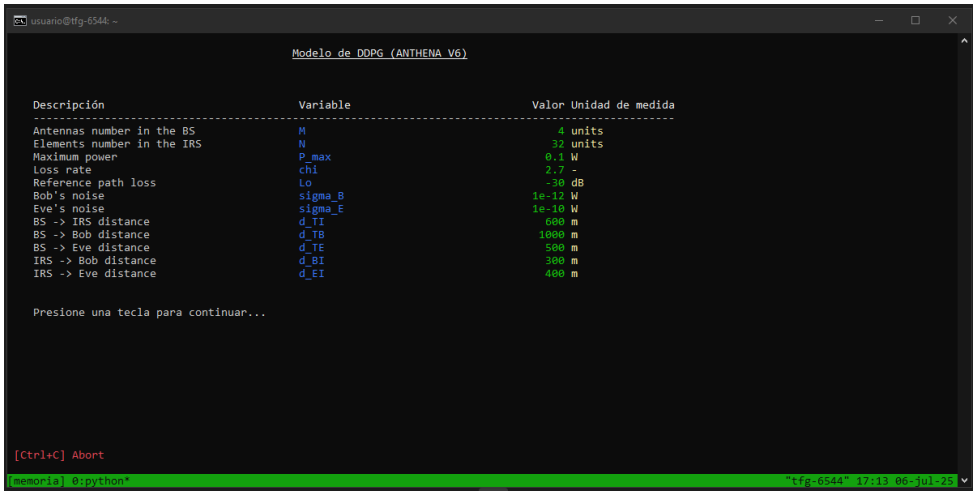

Figura 4.8: Menú parámetros

2. **Cargar modelo entrenado.** Al introducir esta opción, se desplegará otro submenú con todos los modelos disponibles para esa versión, donde se podrá introducir el nombre de uno de estos modelos. A continuación, se podrá elegir que hacer con este modelo, si

entrenarlo otra vez, mostrar sus gráficas, la estructura de redes neuronales que utiliza o realizar pruebas.

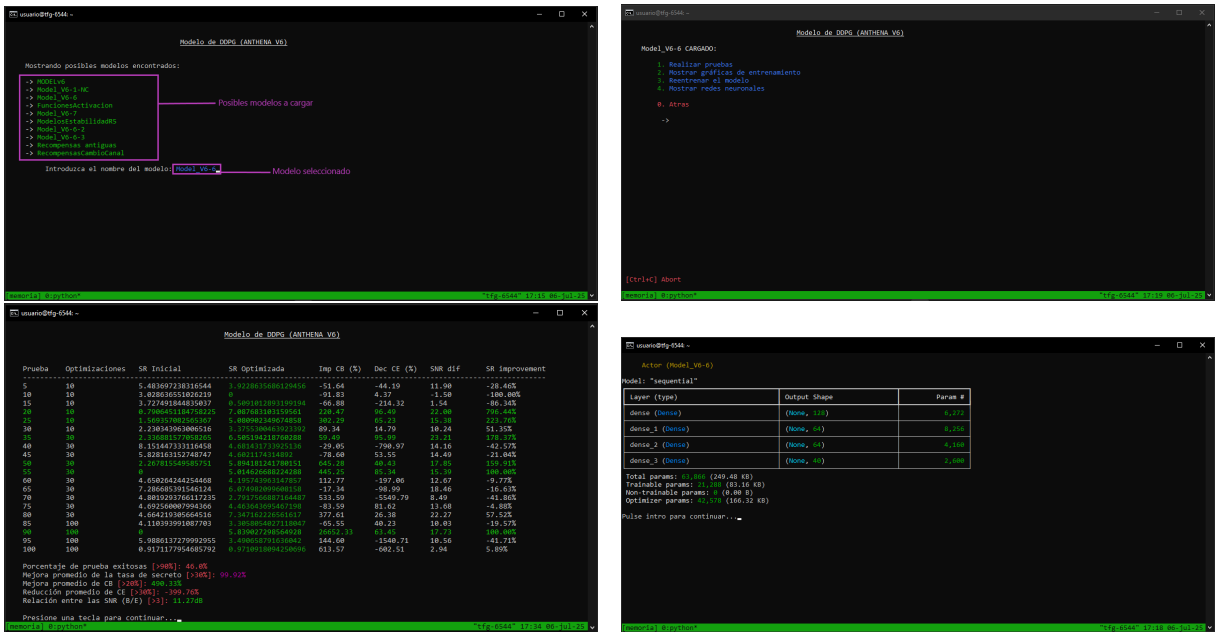

Figura 4.9: Carga de modelo entrenado

3. **Entrenar nuevo modelo.** Al seleccionar esta opción, se desplegará otro submenú en el que se permitirá introducir los parámetros generales del entrenamiento, así como la selección de una recompensa y los parámetros específicos de la recompensa en función de la escogida.

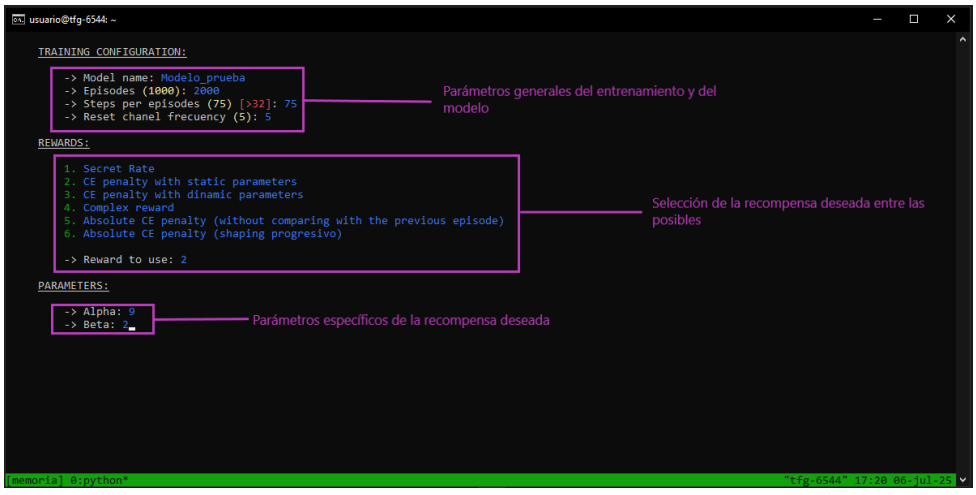

Figura 4.10: Menú de configuración de entrenamiento

Al terminar de introducir estos datos, durante el entrenamiento se mostrará una pantalla en la que se podrán ver los últimos 18 entrenamientos, con una breve muestra de los datos

principales en cada uno de ellos. Para ver el progreso del modelo también se podrá visualizar el nombre del modelo que se está entrenando y una barra de progreso con el porcentaje a la derecha.

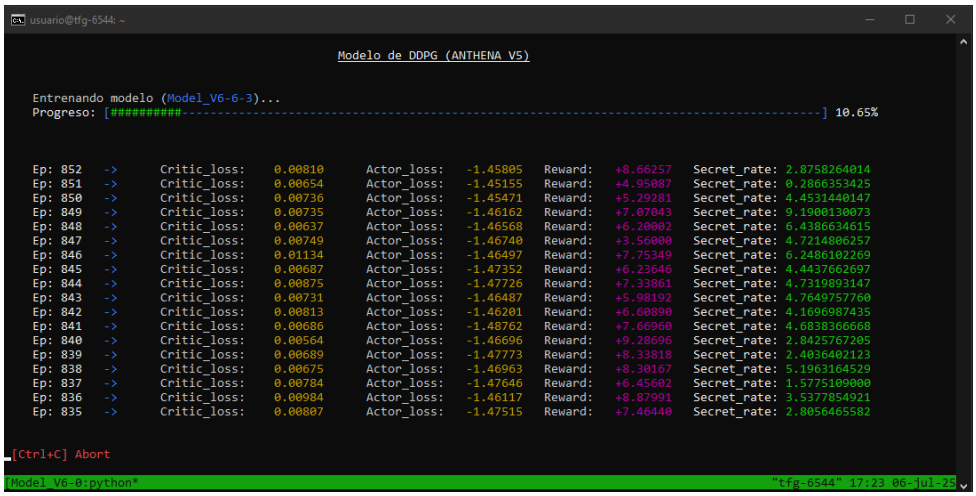

Figura 4.11: Menú entrenamiento

4. **Mostrar gráfica conjunta de entrenamiento.** Al seleccionar esta opción, se mostrará una pantalla en la que será posible elegir entre distintos modelos de la versión en la que nos encontremos para representar los datos del los distintos entrenamientos en una sola gráfica. (Esta opción solo está disponible a partir de la versión 6).

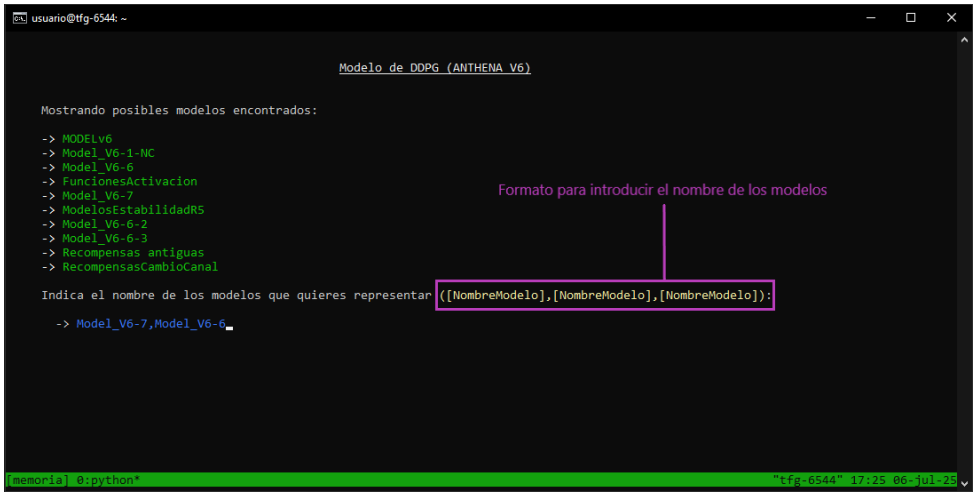

Figura 4.12: Menú gráficas conjuntas

Funcionamiento y comandos personalizados

Debido a las limitaciones del entorno (sin GPU, sin GUI y en entorno remoto), se ha incorporado el uso de tmux para gestionar sesiones persistentes y ejecutar procesos de entrenamiento en segundo plano sin perder el control del entorno de trabajo.

Para su incorporación, se creó un script en el directorio raíz de la máquina virtual, denominado `iniciar_sesion.sh` que contenía órdenes para que tras su ejecución, se creara una sesión (o si existía, que se abriera la sesión correspondiente), este script hacía lo siguiente:

- Comprobaba si existía una sesión de tmux con el nombre deseado
- Si no existía, creaba una nueva sesión de forma que:
  - Se activaba el entorno virtual de Python
  - Se redirigía a la carpeta donde se estaba trabajando según la versión correspondiente
  - Se limpiaba la consola para una vista más clara
- Finalmente, conectaba directamente al usuario con dicha sesión

Para ejecutar el archivo de forma más cómoda, se elaboró un comando personalizado

```
alias iniciar_sesion = "source ~/iniciar_sesion.sh"
```

Este flujo de trabajo permitía, con un único comando, posicionarse en el entorno de trabajo adecuado, listo para entrenar o evaluar modelos sin intervención manual adicional y ahorrar así poco a poco un tiempo muy valioso.

Entrenamiento y evaluación desde la aplicación

Desde la interfaz interactiva se puede:

- Configurar el entrenamiento de nuevos modelos desde cero, seleccionando todos los parámetros relevantes y recompensas mediante inputs guiados.
- Cargar modelos entrenados previamente, permitiendo realizar:
  - Evaluación a través de baterías de pruebas
  - Elaboración de gráficas de entrenamiento que se guardarán en el repositorio.
  - Reentrenamiento a partir de un modelo base, permitiendo así la combinación de diversas recompensas pudiendo así elaborar conjuntos de entrenamientos más complejos.
  - Inspección de la arquitectura de las redes actor y crítico
- Comparación de resultados entre varios modelos mediante la elaboración de gráficas combinadas.

Además, durante el entrenamiento se mostrará información detallada por progreso del entrenamiento, mediante métricas clave por episodio, y mediante la evaluación periódica sin exploración para validar el aprendizaje real de la política.

El sistema también proporciona grán flexibilidad posibilitando la opción de modificar parámetros importantes en el entrenamiento como lo son α, β.

Cabe destacar que toda la interfaz está diseñada de una forma modular que permite la integración sencilla de cualquier funcionalidad adicional.

Capítulo 5

Conclusiones y validación del modelo

5.1. Evaluación de entrenamientos y criterios de éxito

Durante el desarrollo del modelo se han realizado entrenamientos monitorizados y evaluaciones periódicas. Esto ha permitido verificar que el agente no obtiene buenos resultados por azar, sino que realmente aprende políticas válidas.

Para considerar un entrenamiento como exitoso, se han establecido varios criterios de éxito, estos son:

- **Convergencia estable.** Una condición importante es que la evolución del `actor_loss` y del `critic_loss` sea buena, es decir, convergente y sin irregularidades.
- **Incremento sostenido de la recompensa media.** Señal de que la política está mejorando de forma continua.
- **Mejora constante de la tasa de secreto.**
- **Reducción de la señal de Eve (CE).** Sin comprometer excesivamente la señal de Bob (CB), aunque el caso ideal sería conseguir a la vez mejorar la señal de Bob.
- **Relación SNR Bob/Eve.** La relación señal ruido entre Bob y Eve tiene que ser mayor a 3dB.
- **Porcentaje de mejora de tasa de secreto.** El porcentaje de pruebas en las que se ha mejorado la tasa de secreto tiene que ser mayor a un 90 %.

Todas estas métricas son válidas, pero una muy importante es la de los recursos, sin embargo, esta desde el primer momento se cumplió ya que para la aplicación de este modelo, solo hace falta un entrenamiento exitoso. Una vez implementado, este modelo irá aprendiendo conforme va actuando.

Una vez visto las métricas de éxito vamos a ver los resultados obtenidos en función de estas. Inicialmente, vamos a ver la convergencia en las funciones de pérdida del actor y el crítico.

En entornos en los que no hay variación de canal se consiguió una buena convergencia en fases muy tempranas del proyecto.

Sin embargo, una vez se empezó a cambiar el canal se fue desestabilizando progresivamente, llegando a un punto en el que era muy difícil conseguir estabilidad, sin embargo, mediante distintos métodos de refinado anteriormente explicados, se consiguió que al final, incluso cambiando el canal a cada episodio, estas métricas se estabilizaran de una forma excelente.

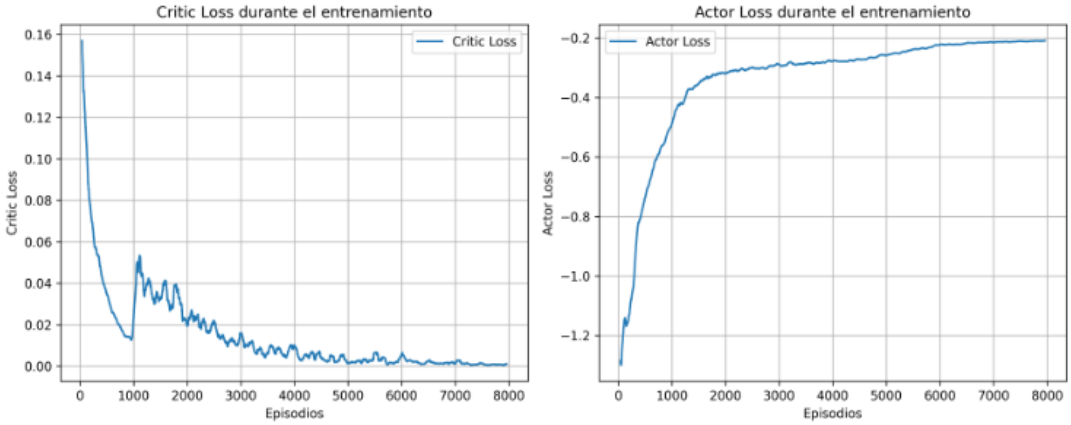

Figura 5.1: Actor_loss y Critic_loss con cambio constante de canal

En cuando al incremento de la función de recompensa, en la fase de desarrollo de cada una de ellas era muy difícil obtener un comportamiento estable, pero una vez vistas cuales eran las óptimas, y refinándolas poco a poco, se consiguió incluso con una simulación de entorno muy fiel a la realidad, una alta estabilidad y mejora constante.

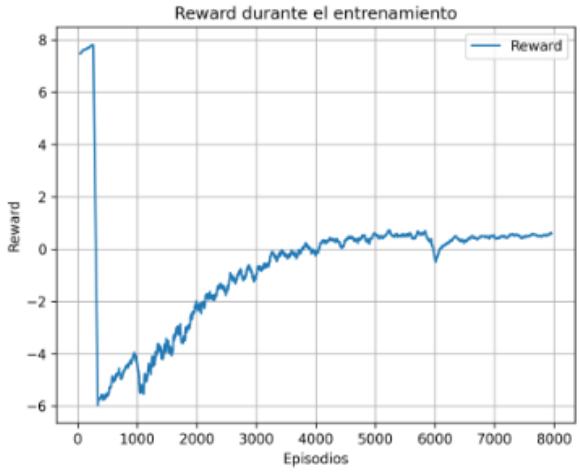

Figura 5.2: Recompensa en entorno real

Esto muestra como el modelo es consistente y aprende una política correcta, de forma estable, para la mejora de la recompensa. Como resultado tenemos que el modelo es fuerte y funcional incluso en entornos altamente realistas en los que las condiciones del canal están en constante cambio. Por lo que la construcción del modelo es un éxito y por lo tanto todo dependería de la construcción de la función de recompensa adecuada.

Además, se ha observado cómo, con un método de entrenamiento que combina varias recompensas, se consigue mejorar la tasa de secreto mejorando la señal de Bob y a la vez empeorando la de Eve, en entornos en los que el canal es constante. Y esto se ha conseguido en tan solo unos pocos episodios.

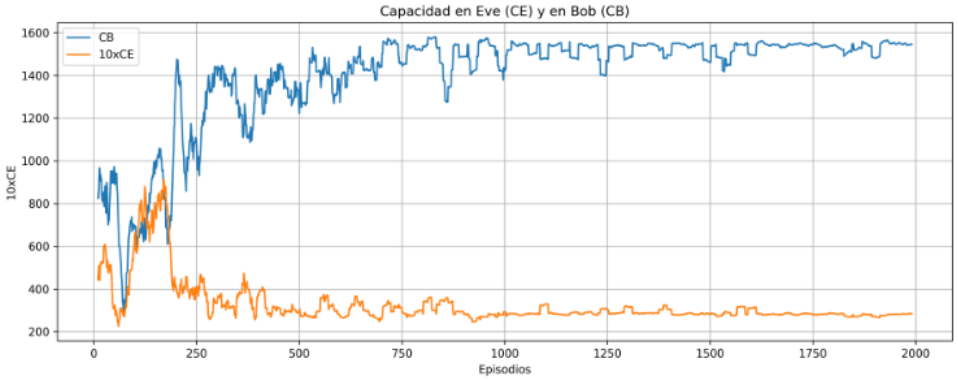

Figura 5.3: Señales en Bob y Eve en entorno de canal constante

5.2. Comparación con métodos tradicionales

Para validar la eficacia del modelo, se ha comparado con dos técnicas clásicas, que son **AO** (Alternating Optimization) y **SDR** (Semidefinite Relaxation).

Dado que las técnicas AO y SDR requieren conocimiento completo del estado del canal, lo cual no es asumible en entornos reales, se ha optado por realizar una comparación empírica indirecta, replicando las condiciones de simulación utilizadas en el artículo [11]. Para ello, se han extraído los valores de interés para las pruebas y se han generado nuevas curvas con el modelo DDPG entrenado bajo las mismas condiciones. Esto permite contrastar el rendimiento en cuanto a la tasa de secreto alcanzada por el modelo de aprendizaje frente a los métodos tradicionales.

Se han generado dos gráficas comparativas, una que muestra la tasa de secreto generada por los distintos métodos en función de la distancia de la base a Bob, y otra que nos muestra la tasa de secreto en función de los distintos elementos de la IRS.

Además de mostrar solamente la comparación con AO y SDR, se van a incluir los datos de la tasa de secreto obtenida de forma aleatoria y de la tasa de secreto teórica si no hubiera IRS.

Cabe indicar que los modelos DDPG usados para las extracciones de datos para las gráficas han sido entrenados en un entorno con variaciones leves del canal, lo que permite centrarse en la comparación de su capacidad de aprendizaje sin información perfecta del canal, frente a algoritmos que sí dependen de esta.

Las gráficas comparativas son las siguientes.

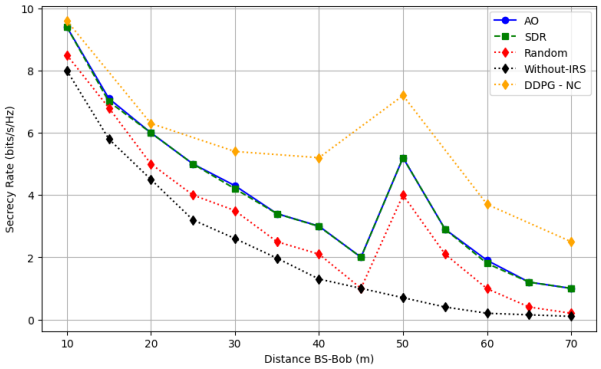

Figura 5.4: Comparativa según la distancia desde la base a Bob

El modelo propuesto basado en DDPG demuestra un rendimiento claramente superior en términos de tasa de secreto, en comparación con los algoritmos AO y SDR, incluso sin acceso a la información exacta del canal, solo teniendo en cuenta condiciones y métricas del canal que son factibles de conseguir en la realidad.

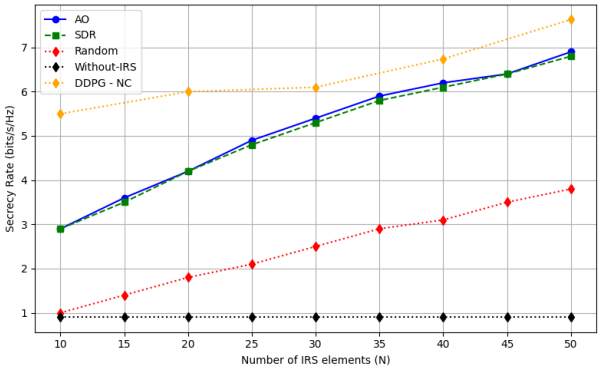

Figura 5.5: Comparativa según el número de elementos de la IRS

Esta segunda evaluación compara el rendimiento del modelo frente al número de elementos de la IRS, observándose que el modelo DDPG obtiene mejores tasas de secreto para todos los valores de N. Este comportamiento demuestra que el modelo no solo es robusto frente a condiciones de canal cambiantes, sino también altamente escalable, siendo capaz de adaptarse mejor al número de elementos IRS para maximizar la tasa de secreto, sin necesidad de información adicional.

Los algoritmos AO y SDR han sido considerados métodos de referencia en literatura, pero su aplicación está limitada por suposiciones poco realistas ya que requieren conocimiento perfecto del canal, es por esto que el modelo DDPG rompe esta barrera, logrando resultados similares o mejores sin esta limitación, esto representa una aportación clara y valiosa a este campo.

5.3. Conclusiones finales

Este trabajo ha demostrado que los algoritmos de aprendizaje por refuerzo profundo, concretamente DDPG, son una herramienta eficaz para abordar problemas complejos de optimización

en redes 6G, incluso en escenarios donde el canal de transmisión no es completamente conocido. Esta capacidad de operar sin información completa del canal representa una mejora significativa respecto a enfoques anteriores, que requieren dicho conocimiento para funcionar correctamente.

El modelo desarrollado ha mostrado un comportamiento sólido y robusto tanto en entornos estáticos como dinámicos, manteniendo la estabilidad de aprendizaje y una buena eficiencia computacional. Esto lo convierte en una solución viable y realista frente a otros algoritmos como SDR o AO, que, aunque potentes, dependen de información que no siempre está disponible en aplicaciones prácticas.

En definitiva, este proyecto sienta las bases para el desarrollo de arquitecturas adaptativas e inteligentes en redes 6G, abriendo la puerta a su aplicación en entornos distribuidos y dinámicos, donde la capacidad de aprendizaje autónomo representa una ventaja decisiva.

5.4. Trabajos futuros

El modelo actual es una primera versión que ha conseguido dejar claro que es una muy buena opción para el problema que estamos trantando, sin embargo, no deja de ser una primera versión, por lo que abre múltiples líneas de mejora y expansión:

- **Funciones de recompensa:** Seguir con el estudio y la elaboración de una función de recompensa que consiga un mejor comportamiento y enfoque más al modelo hacia el resultado final deseado.
- **Sistema multiagente:** Para futuras versiones, es una muy buena opción la implementación de un sistema multiagente donde un agente esté preparado para la optimización de w y otro para la optimización de ϕ de forma distribuida.
- **Entorno GPU:** Actualmente se está trabajando con una máquina virtual sin GPU, pero se puede contemplar la ampliación del sistema a entornos con GPU para poder así realizar entrenamientos más complejos.
- **Otras técnicas:** Este algoritmo ha sido el que mejor se ha adaptado al problema de forma teórica, sin embargo, puede que haya otros que nos den también buenos resultados, como por ejemplo el PPO. Para un futuro sería bueno implementar alguno de estos métodos para ver su comportamiento.
- **Acercarnos a la realidad:** Todos estos entrenamientos y datos se han extraido haciendo entrenamientos en entornos simulados, algunos muy cerca de condiciones reales, pero siguien siendo simulados. En un futuro convendría estudiar el comportamiento del algoritmo en un entonro totalmente realista, o muy fiel a esta.
- **Incluir otras métricas:** A raíz del punto anterior, también hay que tener en cuenta cuando se aplique a entornos reales otro tipo de métricas adicionales como son el consumo energético o la capacidad total de red, por lo que en versiones avanzadas del proyecto sería recomendable incluirlas.
- **Autimatización de pruebas:** Para obtener resultados más generalizables se puede construir una estructura de automatización de pruebas y validaciones cruzadas.

Bibliografía

[1] Diego Martín de Andrés, Francisco Hernando y Masoud Kaveh. «A Novel Approach to the Optimization of Metasurfaces for Wireless Communications». En: *IEEE Transactions on Antennas and Propagation* 65.12 (2017), págs. 6483-6494. DOI: 10.1109/TAP.2017.2755563.

[2] Ertugrul Basar. «Wireless Communications Through Reconfigurable Intelligent Surfaces». En: *IEEE Access* 7 (2019), págs. 116753-116773. DOI: 10.1109/ACCESS.2019.2935192.

[3] Jie Chen, Ying-Chang Liang, Yong Pei y Huaiyu Guo. «Intelligent reflecting surface: A programmable wireless environment for physical layer security». En: *IEEE Access* 7 (2019), págs. 116753-116762. DOI: 10.1109/ACCESS.2019.2934506.

[4] Kexin Feng, Xiaofeng Li, Yujie Han, Shi Jin y Yifan Chen. «Physical layer security enhancement exploiting intelligent reflecting surface». En: *IEEE Wireless Communications Letters* 9.9 (2020), págs. 1687-1691. DOI: 10.1109/LWC.2020.2997948.

[5] Farshad Reza Ghadi, Mahsa Kaveh, K. K. Wong y Diego Martín de Andrés. «Physical layer security performance of cooperative dual RIS-aided V2V communications». En: *IEEE Sensors Journal* (2023). Online available: https://discovery.ucl.ac.uk/id/eprint/10199664/. DOI: 10.1109/JSEN.2023.3245678.

[6] Rui Lin, Zhiguo Han, Xiaojun Wang y Yingying Cai. «Deep reinforcement learning for physical layer security enhancement in energy harvesting based cognitive radio networks». En: *Sensors* 23.2 (2023), pág. 807. DOI: 10.3390/s23020807.

[7] Xiaoming Liu y Rui Zhang. «Physical Layer Security in IRS-assisted Wireless Networks: A Reinforcement Learning Approach». En: *IEEE Wireless Communications Letters* 9.11 (2020), págs. 1905-1909. DOI: 10.1109/LWC.2020.3017541.

[8] Do Thanh Toan, Pham Thanh Phong, Daniel B. Da Costa y Marco D. Renzo. «Physical layer security for IRS-UAV-assisted cell-free massive MIMO systems». En: *IEEE Access* 12 (2024), págs. 4871-4885. DOI: 10.1109/ACCESS.2024.3456789.

[9] Qingqing Wu y Rui Zhang. «Intelligent Reflecting Surface Enhanced Wireless Network: Joint Active and Passive Beamforming Design». En: *IEEE Transactions on Wireless Communications* 18.11 (2019), págs. 5394-5409. DOI: 10.1109/TWC.2019.2936025.

[10] Haijun Yang, Zhi Xiong, Jie Zhao, Dusit Niyato, Lin Xiao y Qi Wu. «Deep reinforcement learning-based intelligent reflecting surface for secure wireless communications». En: *IEEE Transactions on Wireless Communications* 20.1 (2021), págs. 741-754. DOI: 10.1109/TWC.2020.3038347.

[11] Xianghao Yu, Deli Xu, Robert Schober y Derrick Wing Kwan Ng. «IRS-assisted wireless security enhancement: Joint beamforming and jamming optimization». En: *IEEE Transactions on Communications* 68.3 (2020), págs. 1735-1750. DOI: 10.1109/TCOMM.2019.2962553.

[12] Shengjie Zhang, Xijun Wang, Xianfu Chen y Jun Zhang. «Intelligent Reflecting Surface-Aided Physical Layer Security: Joint Active and Passive Beamforming Design». En: *IEEE Transactions on Vehicular Technology* 69.10 (2020), págs. 12101-12105. DOI: 10.1109/TVT.2020.3012615.

[13] Ning Zhao, Yuhao Zhang, Jiayi Zhang y Ying-Chang Liang. «Artificial Intelligence Empowered Intelligent Reflecting Surface Aided Wireless Communications». En: *IEEE Wireless Communications* 28.6 (2021), págs. 118-125. DOI: 10.1109/MWC.001.2100099.