Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN

INGENIERÍA DE LAS TELECOMUNICACIONES.

TRABAJO FIN DE MÁSTER

BEAMFORMING ADAPTATIVO BASADO EN DEEP REINFORCEMENT LEARNING PARA COMUNICACIONES IBFD (IN-BAND FULL-DUPLEX).

Estudiante: *Chuga, Perugachi, Daniel.*
Directores: *Angueira, Buceta, Pablo; Montalban, Sanchez, Jon.*
Departamento: *Departamento de Ingeniería de Comunicaciones*
Curso académico: *2021-2022.*

Bilbao, 11 de Septiembre de 2022.

Resumen

Español:
El continuo crecimiento´de contenidos a través de los actuales sistemas de radiodifusión, hacen necesaria una evolución hacia nuevas tecnologías que puedan cubrir las futuras necesidades. En el panorama de la televisión digital, el comité ATSC 3.0, propone una nueva arquitectura, IDL/ITCN, que permita realizar la convergencia hacia lo que denominan como la próxima generación de televisión digital. Sin embargo, estas nuevas tecnologías incorporan nuevos retos, como la gestión de una gran cantidad de señales interferentes. Dentro de este contexto, este proyecto tiene como objetivo establecer unas bases iniciales hacia lo que derivaría en una investigación mayor, la cual pueda facilitar la gestión de las señales de interferencia dentro de estos nuevos escenarios. Para ello, se propone una solución que combina las actuales técnicas para la gestión de interferencias, con algoritmos de machine learning. De esta forma se pretende obtener una solución más eficiente que la conseguida con los actuales sistemas.

Palabras clave: Agente, entorno, Deep Q-Network, IDL, ITCN, Beamforming, IBFD.

Inglés:
The continuous growth in content delivery through the current broadcasting systems makes necessary the evolution towards new technologies that can address future needs. In terms of digital television, the ATSC 3.0 committee proposes a new architecture, IDL/ITCN, to enable convergence towards the next generation of digital television. However, these new technologies incorporate new challenges, such as managing a large number of interfering signals. In this context, this project aims to establish the initial basis for further research to facilitate the management of interference signals within these new scenarios. For this purpose, we proposed a solution that combines current interference management techniques with machine learning algorithms. In this way, it is intended to obtain a more optimal solution than the one achieved with the traditional systems.

Keywords: Agent, enviroment, Deep Q-Network, IDL, ITCN, Beamforming, IBFD.

Euskera:
Egungo irrati-difusio sistemen bidez pairatu den eduki-hornikuntzaren gorakadak teknologia berrietaranzko bilakaera bat eskatzen du, egoera berri honek sortu dituen beharrei erantzun ahal izateko. Telebista digitalaren alorrean, ATSC 3.0 batzordeak arkitektura berri bat proposatzen du hurrengo belaunaldiarekiko konbergentzia gauzatu ahal izateko, IDL/ITCN bezala ezagutzen dena. Hala ere, teknologia berri horiek erronka berriak eskatzen dituzte, hala nola interferentzia-seinale askoren kudeaketa. Testuinguru horren barruan, proiektu honen helburua hasierako oinarriak ezartzea da, gerora, ikerketa handiago bat ekarriko lukeena egoera berri horien barruan interferentzia-seinaleen kudeaketa errazteko. Horretarako, interferentziak kudeatzeko metodo tradizionalak eta machine learning algoritmoak konbinatu nahi dira, egungo sistemekin lortutakoa baino irtenbide hobea lortzea ahalbidetuko dutenak.

Gako-Hitzak: Agente, ingurunea, Deep Q-Network, IDL, ITCN, Beamforming, IBFD.

Índice general

Siglas

3GPP 3D Generation Partnership Project. 28

AI Artificial Intelligence. 2

AoA Angle Of Attack. 29

ATSC Advance Television Systems Committee. 1

CNN Convolutional Neural Network. 8

CSI Channel State Information. 8

DQN Deep Q-Network. 6

IBFD In Band Full-Duplex. 1

IDE Integrated Development Environment. 11

IDL In-Band distribution Link. 1

ITCN Inter-Towers Communications Network. 1

LDM Layered Division Multiplexing. 1

LTE Lont Term Evolution. 16

MADDPG Multi-Agent Deep Deterministic Policy Gradient. 8

MIMO Multiple-Input Multiple-Output. 7

MISO Multiple-Input Single-Output. 1, 6

MRT Maximum Ratio Transmissio. 6

MSE Mean Squared Error. 34

RL Reinforcement Learning. 24

ULA Uniform Linear Array. 29

WLAN Wireless Local Area Network. 16

WMMSE Weighted Minimum Mean Square Error. 7

ZF Zeroforcing. 6

Índice de figuras

Índice de cuadros

1 Introducción

En la última década, el suministro de contenidos multimedia ha evolucionado considerablemente en cuanto a calidad de vídeo (4K/8K), interactividad y experiencia de usuario. Con el fin de satisfacer esta creciente demanda de calidad y disponibilidad de servicio, la próxima generación de televisión digital, también conocida como ATSC 3.0, ha desarrollado nuevas técnicas que pretenden ofrecer servicios como alta definición a los receptores fijos, o servicios robustos de calidad HD a receptores móviles. Para ello, en primer lugar, es necesario evolucionar la red existente, que consta de un único transmisor de alta potencia (HPHT), a redes de múltiples transmisores de baja potencia que trabajen sobre una única frecuencia.

Hoy en día, se pueden encontrar varias propuestas relacionadas con el estándar ATSC 3.0, que pretenden llevar a cabo la convergencia hacia la nueva generación de televisión digital. Un ejemplo se puede ver en el artículo [1], donde se propone una nueva forma de transmitir el flujo de contenidos audiovisuales, o como ellos lo denominan, backhaul. En relación con este artículo, en el artículo [2], proponen el uso de la tecnología Layerd Division Multiplexing (LDM) como mecanismo de transporte para la entrega del backhaul como parte de la Enhance Layer (EL). A esta aplicación se la conoce como In-Band distribution Link (IDL), que más tarde, en el artículo [3], se presentó como una arquitectura de sistemas IDL, y se evaluó bajo diferentes canales de propagación. Recientemente, en el artículo [4], se ha propuesto una mejora a los enlaces IDL, basado en una red de comunicaciones entre torres (ITCN). En este caso, las comunicaciones entre transmisores se producen en ambas direcciones, lo que implica una comunicación full-duplex (IBFD) entre nodos de difusión. Analizando toda esta información, se puede concluir que tanto el modelo IDL como la red ITCN, pueden proporcionar grandes ventajas en los sistemas broadcast, como la reducción de costes de mantenimiento o el uso eficiente de los recursos del espectro. De esta forma, estas nuevas tecnologías, se convierten en una gran propuesta para llevar a cabo la convergencia hacia la nueva generación de televisión digital. Sin embargo, la realización de un sistema full-duplex en banda, sigue siendo un reto debido a la gran gestión de interferencias que se debe realizar. En los escenarios propuestos por la red ITCN, debido al uso de comunicaciones IBDF, existe una señal 'loopback' de realimentación, que se acopla a la antena receptora con un nivel de potencia superior a la señal deseada. Esto hace necesaria la introducción de mecanismos capaces de cancelar los altos niveles de las señales de interferencia.

Actualmente, las técnicas de cancelación de interferencia, combinan el procesamiento de señales por hardware y software, basado en la estimación de canales inalámbricos.

Sin embargo, estos métodos se basan en modelos analíticos que pueden resultar complejos en cuanto a su resolución, o demasiado complicados para llevarse a cabo en aplicaciones en tiempo real. En este contexto, la combinación de técnicas tradicionales con soluciones emergentes de Inteligencia Artificial (AI), como las redes neuronales, podrían mejorar significativamente la cancelación de las señales de interferencia, ya que son capaces de enfrentarse a imperfecciones de canal desde una perspectiva que no implica enfoques matemáticos. Este tipo de algoritmos, simplemente requieren de una fase de entrenamiento donde aprenden a cumplir un objetivo, y así posteriormente poder ser utilizados, para la resolución de los escenarios para los cuales han sido entrenados, sin la necesidad de resolver operaciones constantemente. Esto implicaría una reducción del cálculo computacional, haciendo que los futuros sistemas de aplicación broadcast, lleguen a ser viables en la realidad.

2 Objetivos

2.1 Objetivo global

Dentro de este contexto, este proyecto tiene como objetivo principal el estudio, diseño y validación de un modelo de Beamforming adaptativo basado en el algoritmo de Reinforcement Learning, Deep Q-Network. Con este modelo se pretende abrir un camino hacia una solución, que permita una mejor gestión de las señales de interferencia, dentro de los nuevos escenarios propuestos para la próxima generación de televisión digital. Como ya se ha mencionado anteriormente, estos nuevos escenarios hacen uso de comunicaciones IBFD (In-Band Full Duplex), las cuales hacen que los transmisores operen bajo la misma frecuencia, tanto para el sistema de recepción de la señal, como para el sistema de transmisión de la señal a su zona de servicio. En estos casos, al no haber un aislamiento perfecto entre transmisor y receptor, existe una componente proveniente del transmisor, que se acopla al receptor actuando como una señal de interferencia. Esto hace que sea necesaria la introducción de una cadena de técnicas de cancelación, que minimicen y cancelen dichos acoplamientos.

En el artículo [5], se definen tres etapas de cancelación, de las cuales, la técnica de Beamforming es definida como una primera fase que permita facilitar la cancelación de las otras dos etapas restantes. Sin embargo, este tipo de soluciones pueden no ser suficiente para abordar las nuevas tecnologías emergentes como IDL/ITCN, debido a las grandes señales de interferencia que se deben gestionar. Por lo tanto, este TFM pretende analizar la idoneidad del uso de un algoritmo de Reinforcement Learning, concretamente el algoritmo Deep Q-Network, como una posible solución al problema.

2.2 Objetivos específicos

Con el fin de cumplir el objetivo principal, se establecieron los siguientes objetivos específicos:

1. Crear un escenario de auto-interferencia y cancelación que pueda incluir técnicas de Beamforming.
2. Conocer/disponer de diferentes técnicas de inteligencia artificial para la solución de sistemas de Beamforming adaptativos.
3. Disponer de un entorno y de herramientas de análisis/prueba que permitan evaluar el rendimiento de configuraciones de Beamforming, basadas en el algoritmo Deep Q-Network.

3 Beneficios

Este proyecto tiene como objetivo definir los pasos iniciales que permitan el desarrollo de una investigación aun mayor, cuyos beneficios tanto técnicos, como económicos y sociales, pueden ser de gran importancia dentro de la próxima generación de televisión digital. A continuación, se describen algunos de los beneficios que se podrían obtener a través de esta línea de investigación.

3.1 Beneficios técnicos.

El hecho de usar mecanismos de inteligencia artificial para sustituir tareas que se realizan con métodos tradicionales, puede proporcionar unos beneficios técnicos que se pueden traducir en:

- Automatización de los procesos, haciendo que los modelos aprendan a desarrollar tareas de forma autónoma. En este caso, se trata de hacer que un modelo entrenado, aprenda a ajustar el diagrama de radiación con el fin de maximizar la señal deseada, y reducir la señal de interferencia. Hoy en día, aún es posible encontrar sistemas donde este tipo de tareas se realiza de forma manual, bajo la supervisión de un ser humano.
- La automatización de estos procesos derivan en una reducción del error humano.
- Mayor precisión. Este tipo de modelos, pueden proporcionar una mayor precisión en comparación con la que se obtendría si la tarea fuese realizada por un ser humano.
- Reducción del coste computacional. Este tipo de modelos, una vez entrenados, pueden llegar a analizar los datos en tiempo real, reduciendo los tiempos de producción.

3.2 Beneficios económicos.

La eficiencia que se lograría en la gestión de recursos, explicada en el apartado anterior, se podría traducir directamente en beneficios económicos. La reducción del coste computacional, conlleva una reducción en el tiempo de ejecución, además de una mayor eficiencia energética, la cual es directamente proporcional a una reducción de costes. Por otra parte, el hecho de poder automatizar una tarea que, a su vez,

implique una reducción de tiempo, se puede traducir directamente como un ahorro económico tanto en recursos humanos, como en tiempo empleado para elaborar otras tareas.

3.3 Beneficios sociales.

Una mayor eficiencia en el uso de los recursos disponibles, tiene un impacto directo en el flujo de datos que se puede transmitir. Cuanta más capacidad se dispone para la transmisión de datos, mayor es el número de servicios que se pueden ofrecer, permitiendo así, poder mejorar la experiencia de usuario. Esto permitiría aumentar la calidad de los videos transmitidos, adaptándolos a las futuras necesidades requeridas en lo que será la nueva generación de televisión digital.

4 Metodología

A lo largo de este apartado, se hará un resumen del procedimiento llevado a cabo para lograr conseguir los objetivos marcados en el apartado 2. Para ello, se hará uso del siguiente esquema, donde en cada uno de los apartados, se explicara resumidamente los pasos que se han llevado a cabo.

1. Elaboración del estado del arte.
2. Estudio del problema de auto-interferencia en los escenarios IDL/ITCN.
3. Definición del entorno de simulación: Matlab y Quadriga.
4. Estudio y diseño del algoritmo Deep Q-Network.
5. Implementación del modelo Beamforming basado en el algoritmo DQN.
6. Simulaciones del modelo DQN.

4.1 Estado del arte

En primer lugar, se realizó una búsqueda en cuanto a modelos de Beamforming basados en algoritmos de machine learning. En esta búsqueda, se encontraron varios proyectos, donde la gran mayoría son elaborados dentro de escenarios 5G. Es muy común el uso de técnicas de Beamforming dentro de este tipo de escenarios, debido a la topología y arquitectura mallada por las que están compuestos, haciendo necesaria la gestión de un gran número de interferencias. Los proyectos que se han tenido en cuenta a la hora de realizar este proyecto, son los siguientes:

4.1.1 Beamforming basado en aprendizaje automático para canales MISO entre dos usuarios

Este primer ejemplo, citado en el artículo [6], propone un diseño de Beamforming basado en redes neuronales profundas, dentro de un escenario compuesto por dos usuarios con canales de interferencia MISO. En este caso, los usuarios aprenden a seleccionar entre dos de los esquemas más populares para realizar Beamforming: Maximum Ratio Transmission (MRT) y zeroforcing (ZF). En cada momento, el modelo entrenado, le recomienda al usuario que esquema utilizar en función de los valores de la potencia de transmisión y los vectores de canal. Para lograr este objetivo, se entrena al modelo haciendo uso de un Dataset con diferentes datos etiquetados en

función de la potencia de entrada y los vectores de canal. De esta forma, en función de estos parámetros de entrada, el modelo aprende a realizar una clasificación correcta.

4.1.2 Fast Beamforming basado en aprendizaje no supervisado para un enlace descendente MIMO.

En este segundo ejemplo [7], se propone un método de diseño Beamforming basado en un aprendizaje no supervisado que da una solución más óptima al clásico algoritmo WMMSE. Los algoritmos clásicos, como WMMSE, son soluciones óptimas pero de una alta complejidad computacional. Con el fin de reducir esta complejidad, se propone un modelo basado en redes neuronales profundas, que ofrece un servicio real-time solo con operaciones simples de la red neuronal. A diferencia que en el caso anterior, el entrenamiento se basa en un método de extremo a extremo sin muestras etiquetadas, evitando así, el complejo proceso de obtención de un dataset previo. Además, se hace uso de un algoritmo "pruning"para reducir la complejidad computacional y el volumen de la DNN, haciéndolo más adecuado para dispositivos de baja capacidad computacional.

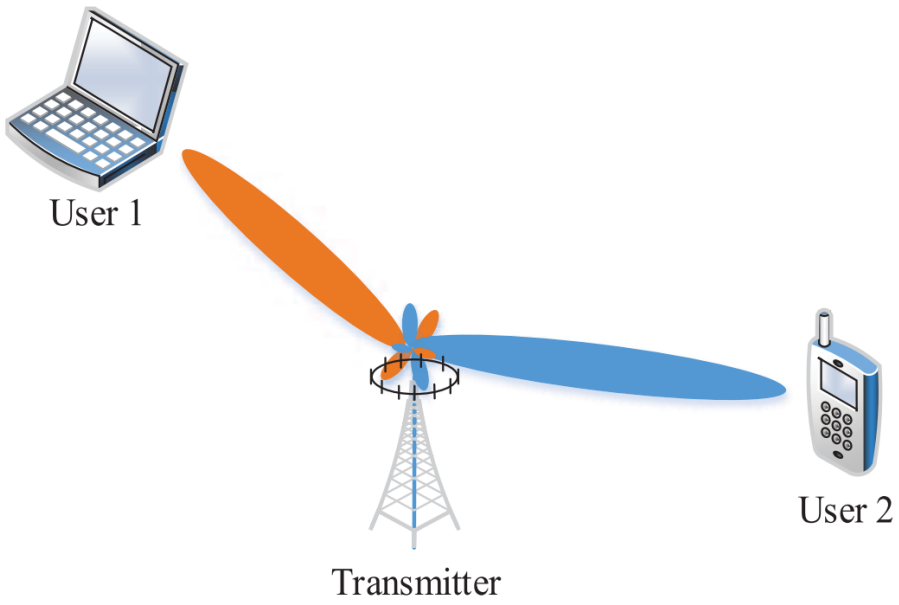
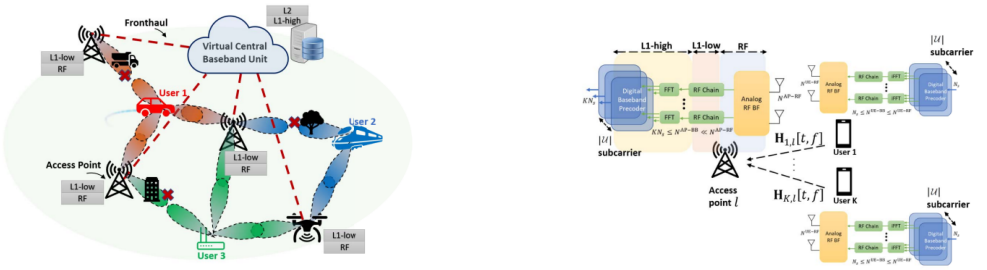


Figura 1 Escenario de la comunicacion Downlink. Imagen tomada del articulo: [8]

4.1.3 Fast Beamforming con Reinforcement learning en comunicaciones MIMO

Este trabajo, citado en el artículo [9], se centra en la realización de un modelo Beamforming para comunicaciones mmWave (en onda milimétrica) basadas en comunicaciones MIMO. Su objetivo principal consiste en proporcionar un servicio continuo en

los dispositivos de alta movilidad, donde existe una gran presencia de interferencias y atenuación en la señal. En este caso, la creación del modelo se hace a través del algoritmo Deep Reinforcement Learning. Al igual que en el caso anterior, este es un algoritmo que no requiere de datos previamente etiquetados. Durante el entrenamiento, el modelo se aprende únicamente con las observaciones y recompensas obtenidas en cada instante de tiempo. Las recompensas, son valores que determina como de buena es una observación en función del objetivo final.



(a) Disponibilidad de la red con el modelo Beamforming

(b) Diagrama de bloques del sistema

Figura 2 Esquemas del proyecto realizado [9]

4.1.4 Beamforming robusto usando Multi-Agent Deep Reinforcement Learning

Finalmente, en el trabajo [8] se implementa un modelo Beamforming dentro de un escenario multicelular MISO en presencia de una información de estado (CSI) imperfecta. En este caso, el aprendizaje está basado en el algoritmo MADDPG (Multi-Agent Deep Deterministic Policy Gradient), donde cada estación base actúa como un agente independiente, y cada uno dispone de una red neuronal DDPG con una estructura CNN (Convolutional Neural Network). El objetivo consiste en aprender a elegir la dirección correcta del beam, y a asignar la potencia de múltiples usuarios a través de un protocolo de intercambio de información limitado, con el único fin de proporcionar robustez frente a errores del CSI.

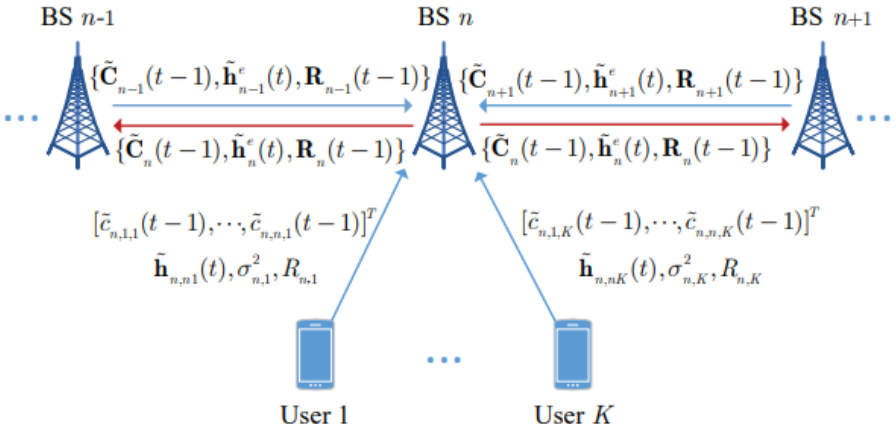


Figura 3 Esquemas del proyecto realizado [8]

Una vez analizado estos ejemplos, se puede intuir que, la gran mayoría de modelos de beamforming basados en algoritmos de machine learning, no superan los resultados de los clásicos algoritmos de iteración como WMMSE, simplemente los igualan. Sin embargo, presentan una mayor eficiencia en cuanto a cálculo computacional, haciendo que sean de gran utilidad para servicios en tiempo real como los nuevos estándares de televisión.

4.2 Estudio del problema de auto-interferencia en los escenarios IDL/ITCN

Tras realizar un estudio sobre las diversas alternativas de algoritmos de inteligencia artificial existentes para la creación de modelos Beamforming, se dio paso al estudio del problema al que se pretende dar solución. En este caso, como ya se ha mencionado en otros apartados, se trata del problema de auto-interferencia en los escenarios IDL/ITCN, debido al uso de comunicaciones IBFD.

En primer lugar, fue necesario entender el escenario IDL/ITCN en el que se va a trabajar, con el fin de tener en cuenta todos los elementos necesarios, para su posterior simulación en un entorno de desarrollo. Para ello se tuvo en cuenta los artículos [3] y [4], entre otros, donde se describen las nuevas propuesta tecnológicas (los enlaces IDL y la arquitectura ITCN), que pretenden solventar las futuras necesidades en cuanto a calidad y capacidad, dentro de la nueva generación de televisión digital. Así mismo, otros artículos, como [5] o [10], fueron de gran ayuda para entender los beneficios de aplicar algoritmos de machine learning dentro de los escenarios IDL/ITCN.

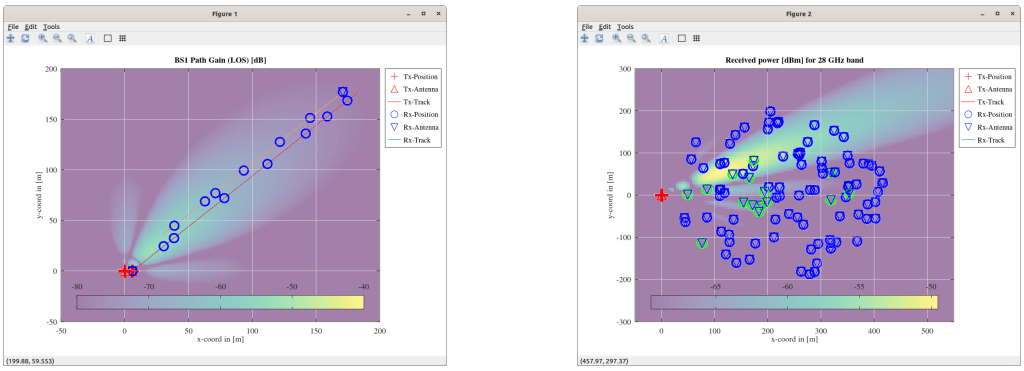
Finalmente, en esta misma sección, también se estudió el funcionamiento teórico de la técnica de Beamforming, que se pretende utilizar para cancelar la auto-interferencia

en los escenarios IDL/ITCN. La página [11], fue útil para la compresión teórica de esta técnica, y así poder ejecutarla dentro de un entorno de desarrollo.

4.3 Definición del entorno de simulación: Matlab y Quadriga.

Una vez entendido los elementos que componen los escenarios IDL/ITCN, se dio paso a su simulación dentro de un entorno de desarrollo. Este entorno debe tener la capacidad de simular un escenario donde se dé el problema de auto-interferencia. Entre los múltiples proyectos encontrados a la hora de realizar el estado del arte, se encontró el repositorio [12].

Este repositorio contiene un proyecto que desarrolla un modelo beamforming, basado en un algoritmo de Reinforcement Learning, para escenarios 5G. En este caso, el escenario simulado es más complejo que el que se pretende llevar a cabo, ya que en él, se tienen en cuenta la gestión de múltiples interferencias provenientes de diferentes usuarios en movimiento. Aun así, es una buena referencia como punto de partida. Según la definición de este proyecto, el escenario de simulación se puede ejecutar tanto en Matlab, como en Octave. Ambas plataformas hacen uso de los recursos proporcionados por el simulador de canales, Quadriga. Este generador es muy conocido por los resultados realistas que proporciona a la hora de simular modelos de canal.



(a) Ejemplo de multi-frecuencia Quadriga. (b) Ejemplo de movilidad Quadriga.

Figura 4 Ejemplos de diferentes escenarios en Quadriga

Con el fin de poder diseñar y simular el escenario deseado, ha sido necesario aprender a manejar los recursos proporcionados por Quadriga. En su página oficial [13], viene recogida toda la información sobre las funciones y escenarios que se pueden simular, además de varios ejemplos ya realizados, que permiten estudiar su funcionamiento. Tras realizar estos ejemplos, fue posible elaborar un escenario que suministre los datos necesarios para alimentar el entrenamiento del algoritmo de Deep Q-Network. Este entrenamiento se lleva a cabo en otro editor conocido como **Spyder**, el cual trabaja

sobre el lenguaje Python. Este IDE es comúnmente utilizado para el desarrollo de modelos basados en algoritmos de machine learning.

4.4 Estudio y diseño del algoritmo Deep Q-Network

Antes de empezar a programar dentro de Spyder, fue necesario estudiar y entender el funcionamiento de los algoritmos de Reinforcement Learning. En particular, los de la familia Q-Learning, a la cual pertenece el algoritmo Deep Q-Network [14] que se va a utilizar en este proyecto. La familia de algoritmos Q-Learning, centra su aprendizaje en el intercambio de información entre un agente y un entorno. En estos escenarios, el entorno es el encargado de proporcionarle la información necesaria al agente, para que este aprenda a moverse dentro de él. Ambos actores han sido creados dentro de Spyder, haciendo uso de la librería TensorFlow, la cual viene explicada en su página web [15] a través de diversa documentación. Además de la documentación oficial, ofrece información detallada sobre los diferentes tipos de algoritmos de machine learning con los que se puede trabajar, los de Reinforcement Learning entre ellos, y varios tutoriales para su implementación. La realización de estos tutoriales y pruebas varias, han sido útiles para conocer los diversos parámetros e hiperparámetros, necesarios de gestionar para entrenar este tipo de algoritmos.

Los parámetros e hiperparámetros, son los conjuntos de valores que se deben ajusta a la hora de crear el modelo para un problema dado. En este caso, es de interés conocer los hiperparámetros que se deben manejar, ya que a diferencia de los parámetros de un modelo, estos son los valores de las configuraciones utilizadas durante el proceso de entrenamiento. Estos valores no se obtienen de los datos, y a priori no se puede conocer su valor óptimo con exactitud. Por lo que generalmente se tiende a usar reglas genéricas, valores para los cuales ya se ha comprobado su eficacia, o simplemente se busca la mejor opción mediante prueba y error.

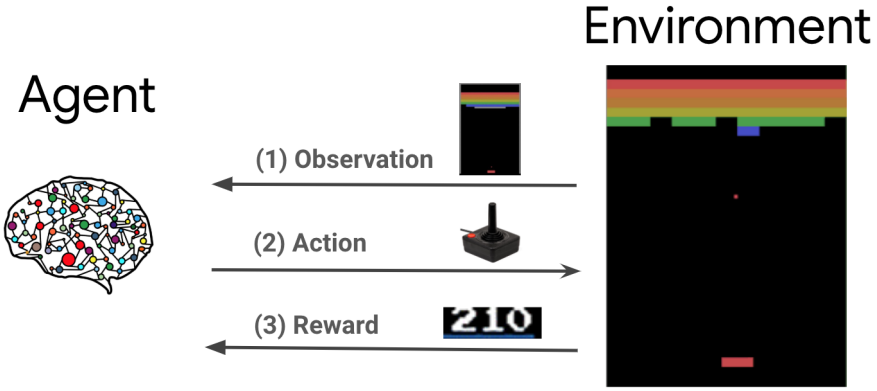


Figura 5 Intercambio de información entre agente y entorno. Imagen tomada de la página [15].

4.5 Implementación del modelo Beamforming basado en el algoritmo DQN

Tras haber estudiado el escenario de auto-interferencia IDL/ITCN que se debe simular, y los diversos algoritmos de Reinforcement Learning con los que se puede construir un modelo Beamforming, se dio paso a la combinación de ambos conceptos para la construcción de la solución propuesta. Por una parte, haciendo uso de Octave y Quadriga, se elaboró un escenario compuesto por 2 transmisores, uno para la señal deseada y otro para la señal de interferencia, además de un receptor para captar ambas señales. Con el escenario montado, se realizaron varias simulaciones en las que los transmisores iban cambiando su posición. Este movimiento de los transmisores, permitió obtener una mayor variedad en los canales simulados, y de esta forma, una mayor cantidad de datos para el entrenamiento del algoritmo. Esta información es almacenada para su posterior uso en la creación del entorno de entrenamiento del algoritmo dentro de Spyder.

Durante el entrenamiento, el entorno hace uso de los datos almacenados para proporcionar al receptor, que este caso actúa como agente, la información sobre el estado del canal. Con esta información, el agente debe ir aprendiendo a lo largo del entrenamiento a seleccionar, entre un abanico de posibles diagramas de radiación, un diagrama que le permita realizar una mayor cancelación de la señal de interferencia.

4.6 Simulaciones del modelo DQN

Finalmente, con todos los módulos montados y comunicándose entre sí, se realizan varias pruebas con el fin de ajustar los valores de los hiperparámetros. Esta es una tarea que conlleva mucho tiempo, debido a los múltiples hiperparámetros que se deben ajustar, y al tiempo de duración de cada entrenamiento. En cada prueba, se entrena y evalúa al modelo con los mismos datos. Una vez que se consigue una alta puntación de predicción para los datos de entrenamiento, el modelo es evaluado con otros datos diferentes. De esta forma, se comprueba si su conocimiento es extrapolable a otros escenarios similares para los cuales no ha sido entrenado.

Tras explicar de forma resumida lo realizado a lo largo del proyecto, se da paso a explicar de una manera más detallada ciertos puntos de interés, que son importantes a la hora de entender los pasos realizados.

5 Análisis del problema de Auto-interferencia en IDL/ITCN

Tal y como se ha mencionado en el apartado de 4, con el fin de comprender el problema de auto-interferencia al que se pretende dar solución, fue necesario estudiar los escenarios IDL/ITCN. En este apartado, se explicará con más detalle los conceptos que se describen en los artículos [3] y [4], donde se explican las propuestas de los enlaces IDL, y la red de comunicación ITCN. Además, se explicará en que consiste la técnica de Beamforming, y como se pretende aplicar esta técnica dentro de los escenarios IDL/ITCN.

5.1 Descripción general del problema de auto-interferencia

Los autores del artículo [3], explican como el enlace de distribución IDL pretende transmitir las señales de backhaul (datos de distribución) y de control en la misma banda de frecuencias que el servicio de televisión. De esta forma, las torres de transmisión que proporcionan cobertura de TV, se comunicarían de forma inalámbrica utilizando el mismo espectro que el contenido de difusión transmitido. Las señales de backhaul y de control se enviarían de una torre a la siguiente, donde cada torre actuaria como un nodo In-Band Full-Duplex (IBFD), haciendo que transmitan y reciban simultáneamente señales dentro del mismo canal de radiofrecuencia.

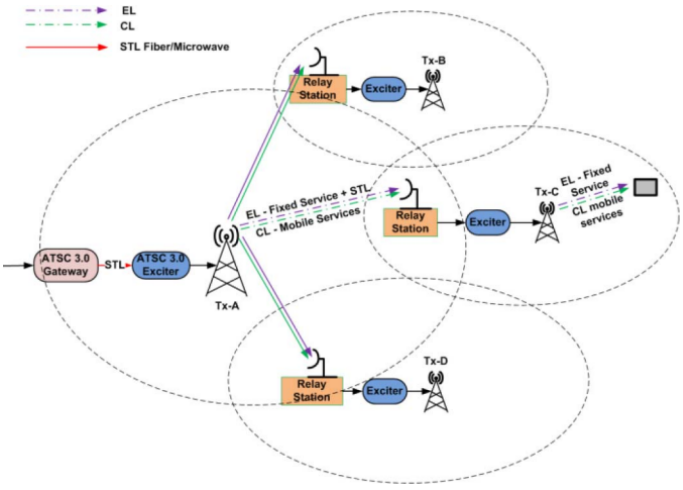


Figura 6 ATSC 3.0 SFN con distribución inalámbrica en banda. Imagen tomada del artículo: [3]

Posteriormente, los mismos autores en el artículo [4], propusieron la red de comunicación entre torres ITCN como una versión mejorada de los enlaces IDL. La propuesta

realizada consiste en un esquema de comunicación bidireccional, donde la comunicación entre torres no se realiza en cascada, sino que las torres podrán responder a la información recibida utilizando una red mallada, donde todos los nodos están conectados entre sí. Uno de los principales inconvenientes de este modelo es la elevada señal de interferencia, la cual es común en los sistemas IBFD. Lo que implica la necesidad de mecanismos de cancelación de la señal de alta eficiencia, sin embargo, los mecanismos actuales pueden no ser lo suficientemente útiles, debido a que se requiere una cancelación superior a los 60 dB para que se pueda desarrollar correctamente esta idea. Además, la etapa de cancelación de alto rendimiento requerida conlleva otros problemas como la limitación del rango dinámico.

La siguiente figura muestra una vista esquemática de una comunicación dentro de una red ITCN/IDL, donde se ve reflejado el problema de auto-interferencia.

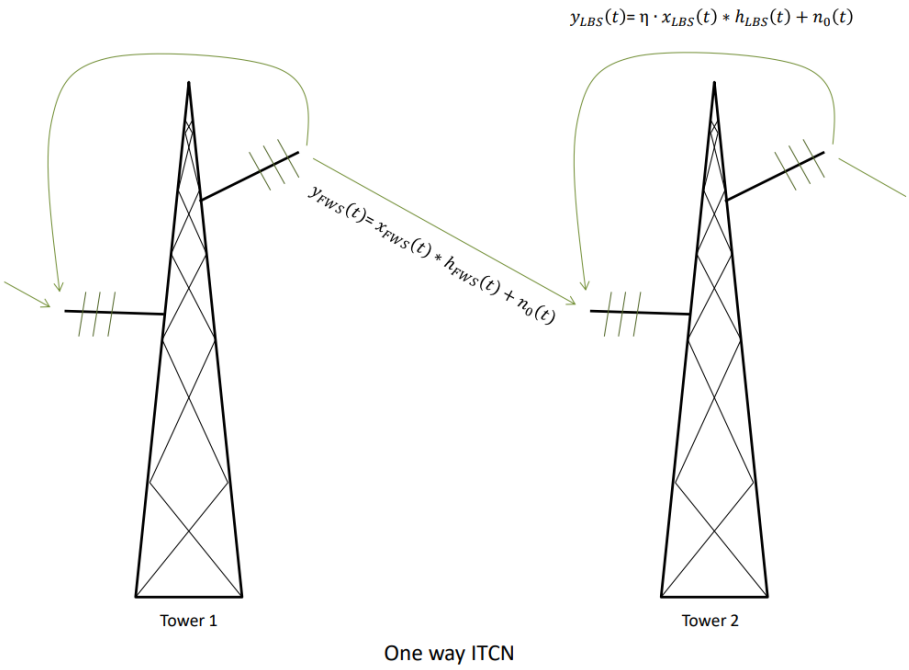


Figura 7 Esquemas de una conexión unidireccional ITCN/IDL. Imagen tomada del artículo [10]

En la figura 7, se puede ver como la señal que recibe la segunda torre viene reflejada por la siguiente ecuación:

$$y(t) = x_{fws}(t) * h_{fws}(t) + x_{lbs}(t) * h_{lbs}(t) + n_0(t) \tag{5.1}$$

donde el segundo factor de la ecuación: $x_{lbs}(t) * h_{lbs}(t)$ hace referencia a la señal de auto-interferencia que se pretende reducir.

Por lo tanto, es necesario buscar nuevas alternativas que solvente el problema de auto-interferencia y faciliten la convergencia hacia la próxima generación de televisión digital. Un ejemplo en cuanto a soluciones, son las 3 etapas propuestas en el artículo [10], que pretenden reducir la componente de auto-interferencia en los entornos IDL/ICTN. Las etapas mencionadas son:

1. **Beamforming**: Esta técnica permite obtener una mayor calidad de señal, reduciendo el ratio entre la señal deseada y la interferencia. Para ello, controla la directividad de la antena del receptor, que se adaptará para minimizar la componente de bucle invertido y maximizar la recepción deseada. Actualmente, se utilizan técnicas iterativas.

2. **Cancelación analógica**: Este módulo pretende proporcionar una importante reducción del ruido y las interferencias, reduciendo la necesidad de rango dinámico en el módulo de cancelación digital.

3. **Cancelación digital**: Esta etapa proporcionará la mayor parte de la ganancia de cancelación.

Estas 3 etapas se pueden ver reflejadas en la siguiente imagen, sacada del artículo [5].

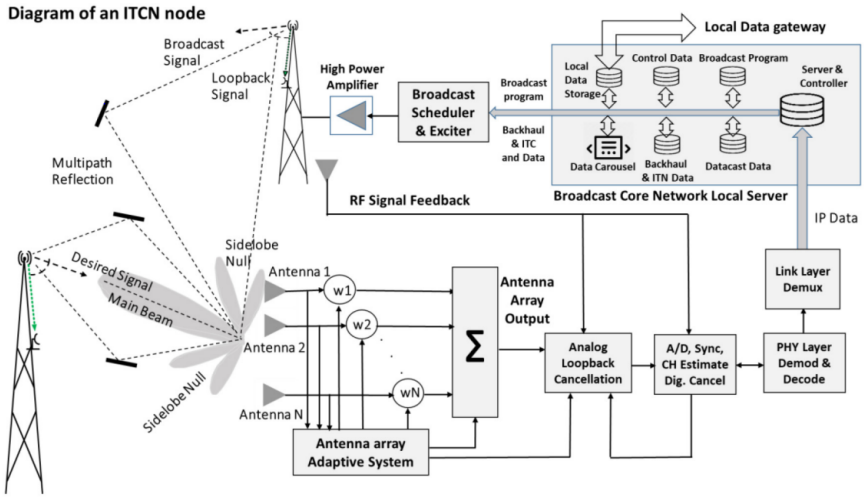


Figura 8 Diagrama de bloques de un nodo de red ITCN en banda. Imagen tomada del artículo [5]

La figura 8, representa el esquema de un nodo ITCN donde se aplican las 3 etapas de cancelación. La primera de ellas hace referencia a un Beamforming adaptativo, que será el que se va a tener en cuenta dentro de este proyecto.

5.2 Beamforming

La técnica de Beamforming es muy utilizada dentro de los sistemas de comunicaciones inalámbricos (como 5G, LTE y WLAN), para mejora la relación señal-ruido de las señales recibidas. Esta técnica, por una parte, permite enfocar las señales transmitidas hacia ubicaciones específicas, por otra parte, en recepción, permite enfocar el diagrama de radiación hacia las señales deseadas. En la siguiente imagen, recogida del enlace [16], se aprecia como se puede adaptar el diagrama de radiación de una antena haciendo uso de la técnica Beamforming.

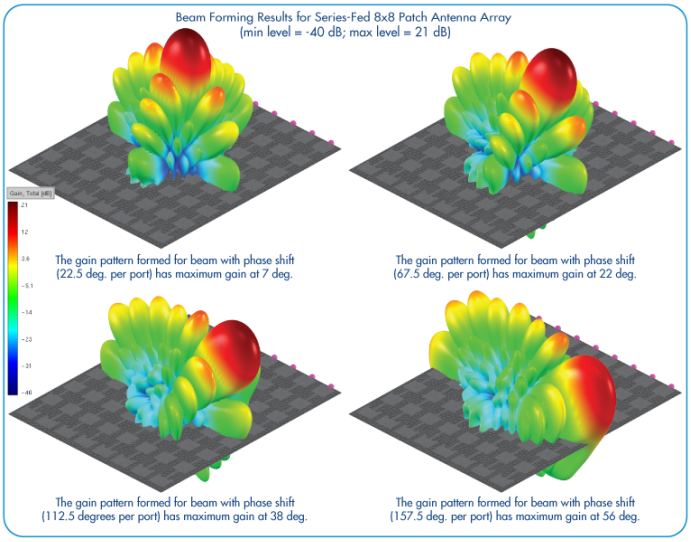


Figura 9 Ejemplo de Beamforming. Imagen tomada del enlace [16]

La propuesta realizada en la figura 9, se trata de un Beamforming adaptativo, que permite manipular el conjunto de antenas, con el fin de modificar la directividad de las mismas. Al cambiar el diagrama de radiación de forma adaptativa, es posible adaptar y maximizar la ganancia del beam principal en la dirección de la señal deseada en cada momento, obteniendo así un mayor rendimiento. Además, esta técnica permite utilizar los nulos del diagrama de radiación para reducir la distorsión del multitrayecto, y lo que es más interesante, para reducir la señal de auto-interferencia y sus reflexiones por multitrayecto. En la figura 10, se puede ver un ejemplo compuesto por dos transmisores, uno que proporciona la señal deseada y otro la señal de interferencia, y un receptor que capta ambas señales. En este ejemplo se adapta el diagrama de radiación del receptor, para que el beam principal apunte hacia el transmisor 2 de la señal deseada, y a su vez, que la señal procedente del transmisor 1, pueda incidir sobre algún nulo del receptor. En cada instante de tiempo se puede ir adaptando los diagramas en función de las necesidades.

Los métodos clásicos de optimización de Beamforming suelen basarse en métodos iterativos, los cuales han demostrado tener un gran rendimiento. Sin embargo, im-

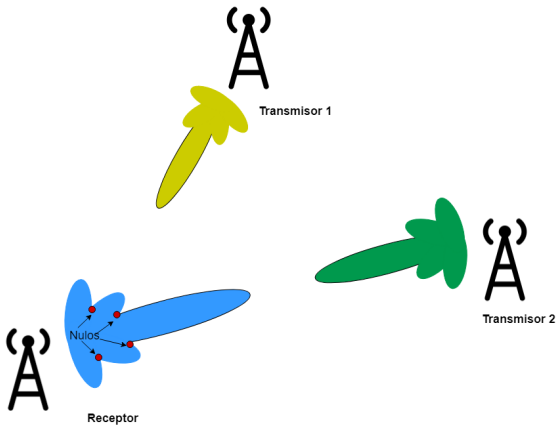


Figura 10 Beamforming en el receptor para captar la señal del Transmisor 2.

plican operaciones complejas, que suponen un gran inconveniente cuando se trata de aplicaciones en tiempo real. Tal y como se ha descrito en el apartado 4.1, es posible optimizar el diagrama de radiación haciendo uso de algoritmos de machine learning, como las redes neuronales, que controlen la amplitud y fase de cada uno de los elementos de la antena w_1, w_2, ..., w_N. En el artículo [7], ya se demostró que el uso de estos algoritmos proporcionan un rendimiento similar a las soluciones iterativas, mostrando una menor complejidad.

Esto abre la posibilidad a que, estos algoritmos, puedan ser utilizados dentro de los entornos IDL/ITCN para reducir la señal de auto-interferencia, obteniendo un escenario como el siguiente:

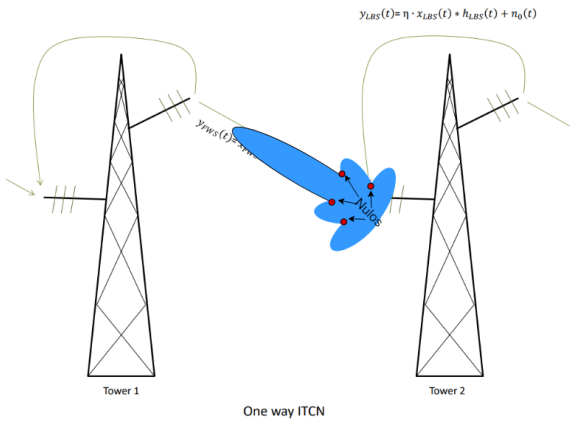


Figura 11 Beamforming en un nodo ITCN.

En el caso ideal, se obtendría el beam principal apuntando hacia la señal deseada de la torre 1, y la señal de loopback se captaría a través de los nulos, haciendo que su ganancia sea mínima.

6 Deep Reinforcement Learning

Una vez explicado el problema que se pretende resolver, se da paso a la explicación del algoritmo utilizado para crear un modelo Beamforming, que pueda reducir la señal de interferencia en un escenario IDL/ITCN. El algoritmo utilizado es Deep Q-Network, el cual pertenece al subcampo Reinforcemente learning dentro de los algoritmos de machine learning. A partir de ahora, con el fin de abreviar su escritura, se hará referencia al subcampo Reinforcement Learning a través de la sigla RL. Los subcampos que componen el área de machine learning son los siguientes:

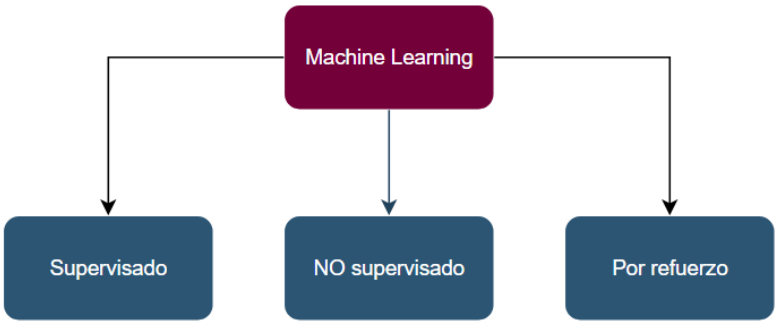


Figura 12 Subconjuntos de algoritmos de machine learning.

El subconjunto de algoritmos de aprendizaje supervisado, son aquellos donde los datos de entrada utilizados durante el entrenamiento, se encuentran previamente etiquetados. Por otra parte, los algoritmos pertenecientes al subconjunto de no supervisados, trabajan con datos no etiquetados, aprendiendo a encontrar patrones o reglas dentro del conjunto de datos de entrada. Finalmente, los algoritmos de RL, se caracterizan por su método de entrenamiento, ya que a diferencia de los otros algoritmos, en este caso, el aprendizaje reside en el intercambio de información entre un agente y un entorno.

6.1 Reinforcement Learning

Este tipo de algoritmos son frecuentemente utilizados en el sector de videojuegos o en el desarrollo de vehículos autónomos, donde el agente puede ser el vehículo y el entorno, un espacio donde se encuentra el agente. Estos algoritmos son entrenados en base a un esquema de recompensas y castigos dentro de un entorno de simulación. Dentro de este entorno, el agente debe aprender a tomar decisiones que le permitan avanzar hacia el objetivo previamente definido.

6.1.1 Componentes principales

Antes de dar paso a la explicación sobre el funcionamiento de los algoritmos RL, es necesario conocer las componentes y terminología que lo componen. Las componentes principales son:

- **Agente**: es el actor que será entrenado para aprender a tomar decisiones que le permitan solventar el problema en cuestión.
- **Entorno**: es el espacio donde se mueve el agente y con el cual interactúa, proporcionándole la información necesaria para su aprendizaje.

Por otra parte, las variables que permiten la interacción entre estas dos componentes son:

- **Estado**: es la información proporcionada por el entorno al agente, mediante la cual se le notifica el estado en el que se encuentra dentro del entorno.
- **Espacio de acciones**: son el conjunto de posibles acciones que puede ejecutar el agente dentro del entorno.
- **Reward**: es el valor que indica la idoneidad de ejecutar una acción en un estado. Estos valores son definidos en una función de recompensa, para proporcionar al agente una noción de recompensa acumulativa a lo largo de su entrenamiento.
- **Política**: son el conjunto de reglas aprendidas por parte del agente durante el entrenamiento, a través de las cuales decidirá que acción tomar en cada estado. La política mapea los estados en acciones que permitan al agente obtener el mayor reward posible a lo largo de un episodio. A veces, es muy común sustituir la palabra política por la de agente, ya que representa el cerebro del agente.

La interacción entre agente y entorno se puede visualizar en la figura 13. Esta comunicación moldea la política a lo largo del entrenamiento.

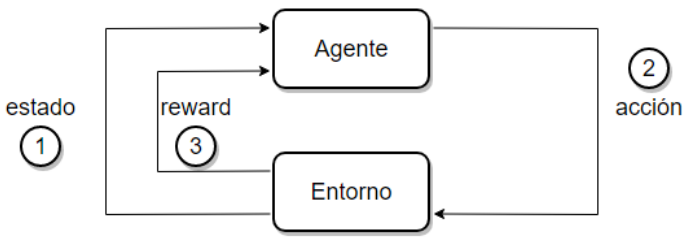


Figura 13 Esquema de los componentes de un algoritmo Reinforcement Learning.

En el primer paso, el entorno envía la información sobre el estado actual al agente. Esta información es utilizada por parte del agente para ejecutar una acción que realice un cambio de estado. Finalmente, una vez ejecutada la acción, el entorno le devuelve un valor de reward, para notificarle como de buena ha sido la ejecución de dicha acción en dicho estado. Este procedimiento se repite en cada uno de los estados por los que pasa el agente, hasta alcanzar un estado terminal que finalice el episodio. El objetivo del entrenamiento, es conseguir el máximo valor de reward en cada episodio.

6.1.2 Estrategia para la selección de acciones: Exploración vs Explotación.

Como se ha mencionado anteriormente, el agente debe aprender a tomar las decisiones correctas en cada uno de los estados por los que pasa dentro del entorno. Sin embargo, al inicio del entrenamiento, el conocimiento que tiene el agente sobre el entorno es nulo, es decir, carece de información respecto al espacio en el que se encuentra, sin saber que acciones debe ejecutar. Por lo tanto, es necesaria la existencia de una fase donde el agente se encuentre explorando el entorno y aprendiendo que acciones ejecutar. A esta fase se la conoce como **exploración**, y es donde el agente ejecuta acciones de forma aleatoria que le permiten obtener recompensas por parte del entorno, con el fin de tener cierta noción sobre lo bien o mal que lo está haciendo. Por otra parte, el agente no puede estar ejecutando acciones de forma aleatoria constantemente, sino que debe de ir explotando el conocimiento que ha ido adquiriendo. Es aquí donde surge la segunda fase conocida como **explotación**. En esta fase, el agente hace uso del conocimiento adquirido hasta ese momento, con el fin de ir avanzando por los estados que le permitan obtener una mayor recompensa. Por esto, es fundamental lograr un equilibrio entre ambas fases, donde el agente disponga del tiempo suficiente para explorar el entorno, para posteriormente poder explotar el conocimiento adquirido y alcanzar su objetivo final. Una estrategia muy conocida y utilizada para llevar a cabo estas dos fases es la de **épsilon-greedy**.

Épsilon greedy

Esta estrategia consiste en definir una variable llamada épsilon, con un valor inicial igual a 1, la cual ira decreciendo a medida que avanza el entrenamiento. Esta variable determinará la probabilidad con la que se entra en la fase de exploración o en la de explotación. En cada estado por el que pasa el agente, se toma un valor aleatorio, si este es menor que épsilon, se entra en la fase de exploración donde se tomara una acción aleatoria para ser ejecutada. Por el contrario, si el valor aleatorio es mayor que épsilon, se utiliza el conocimiento adquirido hasta ese momento, permitiendo así seleccionar la mejor acción a ejecutar en dicho estado.

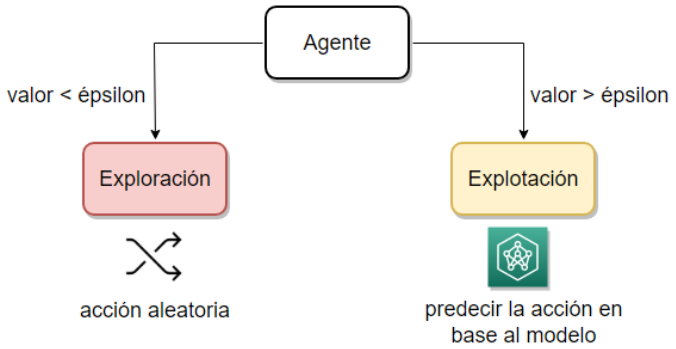


Figura 14 Estrategia epsilon-greedy

Hay muchas formas con las cuales se puede hacer decrecer la variable épsilon durante el entrenamiento. Una opción típica consiste en, multiplicar épsilon por una variable de valor inferior a la unidad, y que este vaya decreciendo exponencialmente en función del número de episodios recorridos ($epsilon * valor^N$, donde N es el número de episodios.). Otra opción más simple, consiste en restar a épsilon un valor inversamente proporcional al número de episodios ($\frac{1}{N}$, siendo N el número de episodios). En el momento de definir esta estrategia, será necesario seleccionar el valor de los hiperparámetros:

- Valor inicial de épsilon.
- Valor final de épsilon.
- Valor con el que decrece épsilon.

6.2 Tipos de algoritmos

Una vez definidos la terminología y los conceptos básicos de los algoritmos RL, en este apartado, se da paso a explicar el funcionamiento de algunos de los algoritmos más utilizados dentro del subcampo RL. Según la documentación de Open AI [17], una de las mayores compañías de investigación sobre IA, la clasificación de los algoritmos RL modernos se pueden representar a través del siguiente esquema:

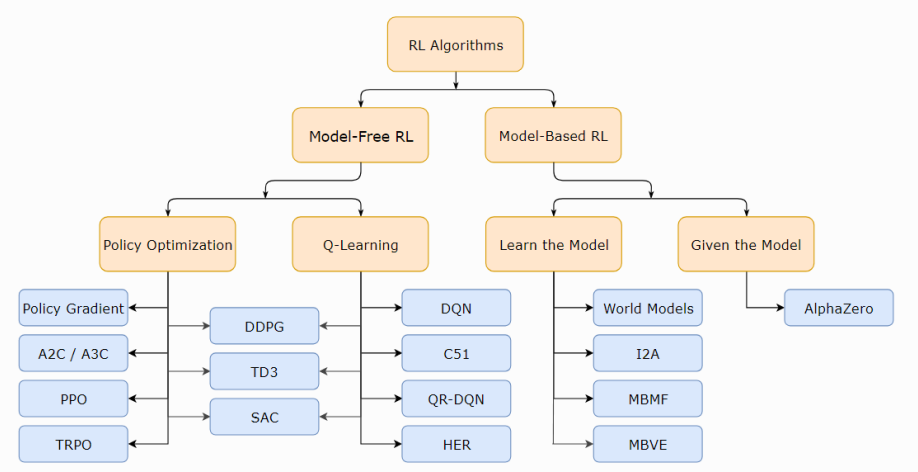


Figura 15 Clasificación de los algoritmos RL modernos según OpenAI. Imagen tomada de la publicación: [17]

Sin embargo, en esta sección, solo se explicarán los pertenecientes a la familia Q-Learning, que es de donde proviene el algoritmo Deep Q-Network, utilizado en este proyecto.

6.2.1 Q-Learning.

Los algoritmos Q-learning, son aquellos que basan su aprendizaje en la ecuación de Bellman:

$$Q(s,a) = r + \gamma * max_{a'} Q(s^{'}, a^{'}) \tag{6.1}$$

Esta ecuación calcula un valor llamado: valor-Q, para un estado $\boldsymbol{s}$ y una acción $\boldsymbol{a}$. El valor-Q, representa la 'calidad' de ejecutar dicha acción en dicho estado, y se calcula como la suma de la recompensa inmediata r, más una componente que representa la recompensa futura del siguiente estado. Este segundo factor. $max_{a'} Q(s', a')$, es el valor Q que se obtiene tras ejecutar la mejor acción posible en el siguiente estado s', y es multiplicado por un factor de descuento (γ) comprendido entre 0 y 1. El valor de γ, es un hiperpárametro que representa el peso que se le otorga a las recompensas a corto y largo plazo. En ejemplos simples de resolver, los valores Q obtenidos se pueden representar en una tabla Q:

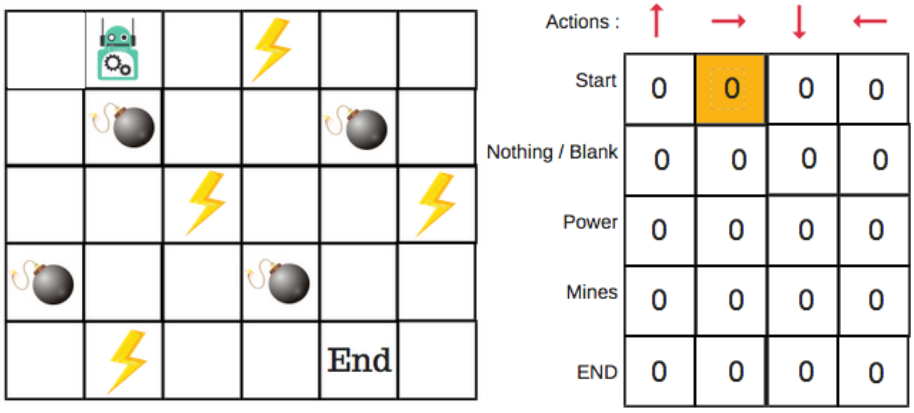


Figura 16 Ejemplo de una tabla Q en un escenario simple. Imagen tomada de la publicación: [18]

El ejemplo sencillo de la figura 16, dispone de una tabla Q, que consta de 4 acciones y 5 estados. Dentro de ella, se almacenan los valores Q que indican la idoneidad de ejecutar cada una de las acciones en cada estado. Inicialmente, tienen un valor igual a 0, y a medida que el entrenamiento avanza, estos valores se van ajustando hasta conseguir un mapa de estado-acciones que represente la política del agente. El hecho de poder representar la ecuación de Bellman 6.2.1 como una tabla o matriz de valores, hace que este tipo de escenarios sean muy sencillos de resolver. Sin embargo, en la vida real, los problemas constan de millones de estados diferentes y cientos de acciones distintas, haciendo que las tablas Q tomen grandes dimensiones, donde su uso ya no es viable. Es aquí donde surge el algoritmo Deep Q-Network o DQN, publicado en el artículo [14].

6.2.2 Deep Q-Network

El algoritmo DQN es una combinación entre el clásico algoritmo Q-Learning y las redes neuronales profundas. Es conocido que las redes neuronales profundas, son una herramienta muy útil para aproximar funciones no lineales. En el caso de DQN, permiten aproximar la función Q sin la necesidad de usar tablas para su representación, solucionando así, el problema de escalabilidad que presenta el algoritmo Q-learning. En estos casos, el estado actual proporcionado por el entorno, actúa como el input de la red, la cual genera como salidas un valor Q por cada una de las acciones posibles.

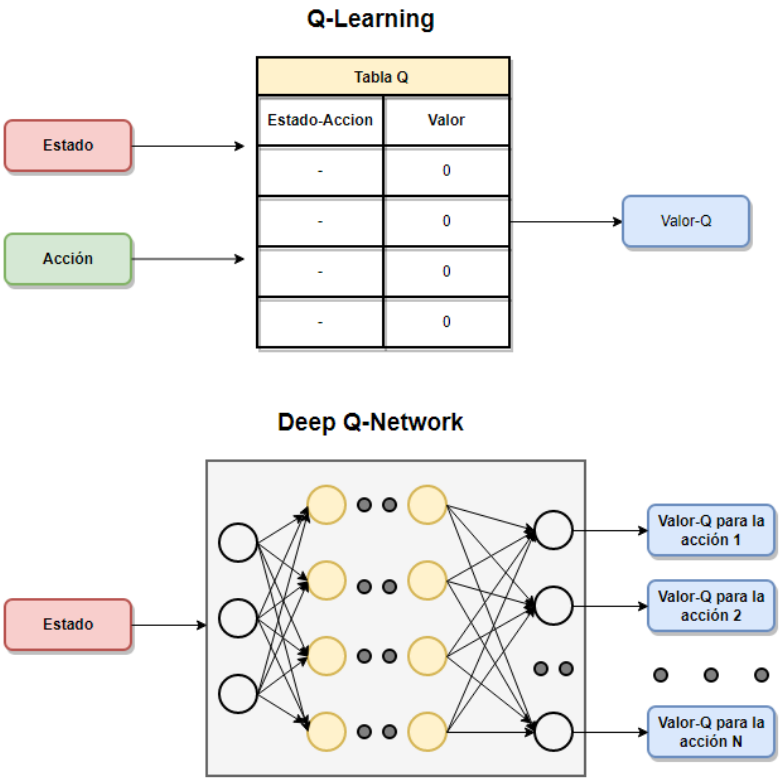


Figura 17 Diferencia entre Q-learning y DQN.

En cuanto al proceso de aprendizaje, las redes neuronales actualizan sus parámetros θ_i, minimizando el error estimado por una función de perdidas que determina el error de predicción. Esta función se calcula como la diferencia entre el valor esperado (el real) y el valor de predicción. En los algoritmos DQN, la función de perdidas se puede representar como el error cuadrático medio del valor Q objetivo y el valor Q de predicción.

$$L_i(\theta_i) = \mathbb{E}[(Q_{target} - Q_{predicted}(s, a; \theta_i))^2] \tag{6.2}$$

siendo el valor Q objetivo:

$$Q_{target} = r + \gamma * max_{a'} Q(s', a'; \theta_i') \tag{6.3}$$

Sin embargo, tal y como se menciona en el artículo [14], el uso de aproximadores de funciones no lineales, como las redes neuronales, sobre algoritmos de RL, hacen que estos sean inestables. Esta inestabilidad puede derivar en:

- Una correlación presente en la secuencia de observaciones. Al trabajar con secuencia de datos, el modelo puede aprender a seguir estas secuencias, induciendo un aprendizaje erróneo.
- Una correlación entre los valores $Q_{predicted}$ y los valores Q_{target}.

– Pequeñas actualizaciones de los valores Q, pueden cambiar significativamente la política, cambiando así, la distribución de los datos.

Para solucionar esta inestabilidad, fueron introducidos los conceptos de: Target Network y Experience Replay.

Target Network

Según la ecuación de Bellman 6.2.1, los valores $Q(s, a)$ son actualizados a través de las recompensas futuras $Q(s', a')$, donde solo existe un step de diferencia entre los estados s y s', esto hace que ambos valores sean muy similares y difíciles de diferenciar por parte de la red. Además, como la ecuación de Bellman es usada para el cálculo de los valores Q objetivo dentro de la función de pérdidas 6.2.2, hace que el aprendizaje sea inestable.

Con el fin de solventar este problema, se introduce una segunda red llamada Target Network. El objetivo de la Target Network consiste en calcular los valores $Q(s', a')$ de la ecuación de Bellman, y así, poder mejorar la estabilidad del aprendizaje, reduciendo la correlación entre valores $Q(s, a)$ y los $Q(s', a')$. Esta nueva red es una copia de la red principal y no es entrenada, sus parámetros simplemente se actualizan sincronizándolos periódicamente con los de la red principal.

Experience Replay

Inspirada en un mecanismo biológico, la técnica de experience replay, aleatoriza los datos de tal manera que elimina la correlación entre las observaciones o estados consecutivos. Si la red aprendiese únicamente a través de observaciones consecutivas, puede aparecer sesgos en estas secuencias, que impidan aprender otros caminos hacia el objetivo más eficientes. Por esto, la técnica de experience replay, permite almacenar las experiencias del agente en cada instaste de tiempo, dentro de una memoria de tamaño finito N, llamada *replay memory*. Durante el entrenamiento, se tomarán de forma aleatoria las experiencias almacenadas dentro de la replay memory para alimentar la red neuronal. De esta forma, se consigue romper con la correlación entre observaciones consecutivas. Las experiencias del agente se almacenan en la replay memory como tuplas que se define de la siguiente forma:

$$e_t = (s_t, a_t, r_t, s_{t+1}) \tag{6.4}$$

Esta tupla contiene información del estado actual s_t, la acción tomada en ese estado a_t, la recompensa obtenida tras ejecutar esa acción en dicho estado r_t, y el siguiente estado s_{t+1}. Algunos de los hiperparámetros importantes a definir en la creación de la replay memory son:

- Memory size: define el tamaño de la memoria, es decir, cuantas tuplas se pueden almacenar como máximo. Una vez se alcanza este valor, la memoria se va actualizando con nuevos datos, eliminando los anteriores.
- Batch size: define el cuantas tuplas tomaran de la memoria para entrenar la target network.

7 Diseño de la propuesta

A continuación, en los siguientes apartados, se da paso a la explicación de todo el procedimiento llevado a cabo durante la realización de este proyecto. Empezando desde el estudio y simulación del problema de auto-interferencia en los escenarios IDL/ITCN, hasta la creación y simulación del algoritmo Deep Q-Network dentro de Spyder.

7.1 Creación del escenario en Quadriga y obtención de los datos de entrenamiento.

En primer lugar, fue necesario el estudio del problema de auto-interferencia dentro de los escenarios IDL/ITCN, que tal y como se ha explicado en el apartado 4.2, se debe al uso de comunicaciones IBFD, es decir, a que sus torres de comunicaciones, transmiten y reciben simultáneamente sobre la misma frecuencia. Esto hace que, la señal transmitida desde un nodo hacia su zona de servicio, se acople en su receptor actuando como una señal de interferencia.

Una vez aclarado el escenario que se pretende simular, se da paso a su construcción. Aunque se ha mencionado que el escenario real consta de un único nodo que actúa como transmisor y receptor simultáneamente, en este caso, se ha representado el problema de auto-interferencia a través de un escenario compuesto por dos transmisores y un receptor. Este escenario, reflejado en la figura 18, ha sido simulado a través de Quadriga, y permite obtener los canales equivalentes al problema de auto-interferencia deseado.

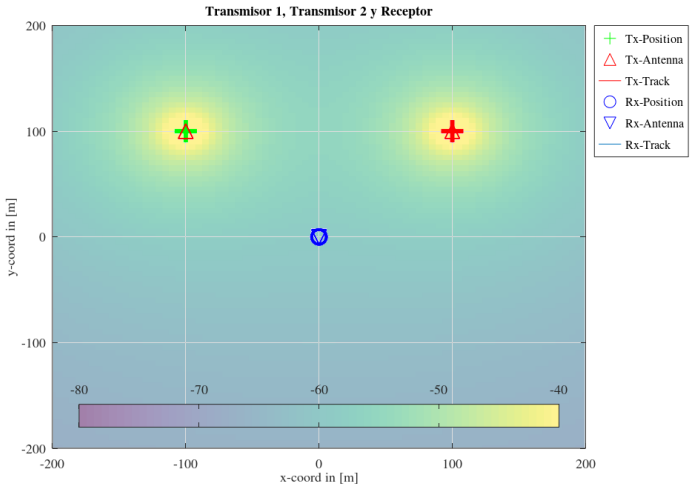
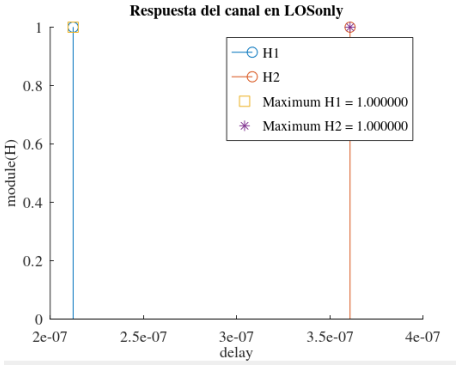


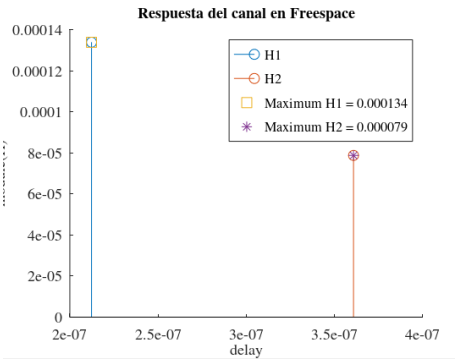
Figura 18 Escenario simulado con Quadriga.

En este caso, uno de los transmisores será utilizado para transmitir la señal deseada, y el segundo, para la señal de interferencia. Por otra parte, el receptor será el encargado de aplicar la técnica de Beamforming, que le permita maximizar la ganancia de la señal deseada, y a su vez, intentar anular, lo máximo posible, la señal de interferencia. Para la simulación, se han utilizado antenas omnidireccionales en cada uno de los transmisores, y se ha transmitido con una potencia de 0 dB. El hecho de usar estos parámetros tan generales, se debe a que el objetivo principal de esta simulación, es únicamente el de proveer diferentes muestras de los canales simulados, tanto de la señal deseada, como de la señal de interferencia.

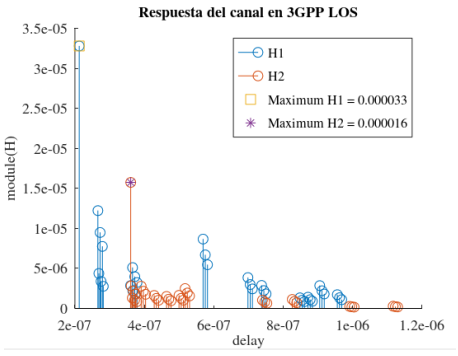
Estos canales dependerán del espacio o escenario, donde se simulen los transmisores. En este sentido, Quadriga ofrece una gran variedad de posibilidades, desde espacios simples como LOSonly, el cual contiene una visión directa, sin shadowing, ni perdidas de espacio libre, hasta escenarios más complejos como los modelos 3GPP, donde se pueden dar situaciones de multitrayecto. Algunas de las pruebas realizadas dieron los siguientes canales como resultado:



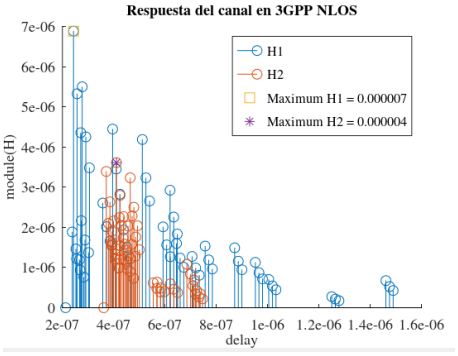
(a) Respuesta del canal en LOSonly.



(b) Respuesta del canal en FreeSpace.



(c) Respuesta del canal en 3GPP LOS.



(d) Respuesta del canal en 3GPP NLOS

Figura 19 Respuestas de canales simulados en diferentes escenarios de Quadriga.

Las simulaciones representadas en la figura 19, fueron útiles para comprender el funcionamiento de Quadriga, y comprobar que la maqueta diseñada, proporciona los resultados deseados. En cada una de las imágenes, viene representado en azul, el canal de la señal deseada H1, y en rojo, el canal de la señal de interferencia H2.

A partir de este momento, el objetivo se centró intentar anular lo máximo posible las muestras del canal de interferencia, y para que esto fuese posible, era necesario conseguir la mayor cantidad de ejemplos. Para obtener una gran variedad en los datos de entrenamiento, se realizaron varias simulaciones dentro de un escenario 3GPP con visión directa, donde los transmisores iban cambiando su posición. Las posiciones fueron seleccionadas de tal forma que permitiesen abarcar el mayor rango de casos posibles. A continuación, se muestran varios ejemplos de algunos escenarios simulados desde donde se han captado los datos.

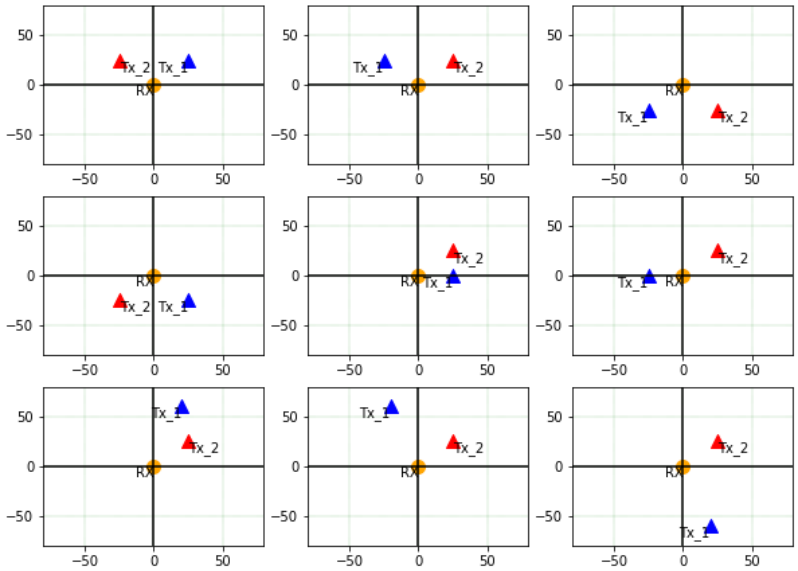


Figura 20 Posibles escenarios de entrenamiento.

En total, se han seleccionado 14 combinaciones diferentes para simular las posiciones de los transmisores, y en cada combinación, se han capturado muestras de los canales, H1 y H2, en 5 instantes de tiempo diferentes. Cuantas más muestras se tomen, el algoritmo DQN aprenderá a reconocer mejor los canales en cada combinación. Otros parámetros que se han obtenido en cada simulación son los ángulos de incidencia, AoA, y retardos de los coeficientes que componen los canales H1 y H2.

7.2 Diseño del modelo Beamforming en el receptor.

Una vez obtenidos los datos proporcionados por los transmisores, se dio paso al diseño del modelo Beamforming dentro del receptor. Para ello, fue necesario la definición de una antena, y comprender como generar un diagrama de radiación a partir de ella. En este caso, se decidió que el receptor haría uso de una configuración de antenas de tipo Uniform Linear Array (ULA). Esta configuración define un conjunto de dipolos separados entre sí, por una distancia d, la cual debe ser menor o igual a la mitad de la longitud de onda.

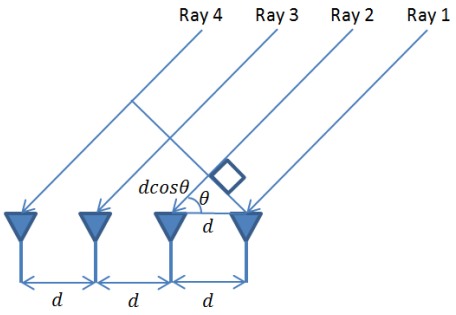


Figura 21 Onda plana que incide sobre la antena tipo ULA. Imagen tomada de [11]

La figura 21, representa la configuración de una antena tipo ULA, en la cual incide una onda plana. Cada uno de los rayos, incide con un retraso proporcional a $dcos(\theta)$. Estos retrasos son los que determinan si las señales se van a sumar de forma constructiva o destructiva. Por otra parte, si se considera cada antena del array como isotrópica, el diagrama de radiación se puede representar con la siguiente fórmula:

$$FA = e^{j\phi} \sum_{n=1}^{4} e^{-j2\pi \frac{(n-1)dcos(\theta)}{\lambda}} \tag{7.1}$$

Esta ecuación denominada como factor de array, representa el campo radiado por el array, donde el coeficiente de alimentación, $e^{j\phi}$, contiene la fase ϕ, que marca la dirección del diagrama. Aplicando este concepto dentro de Spyder, fue posible programar un 'codebook' que almacena diferentes diagramas de radiación, apuntando a distintas direcciones. Este codebook, será utilizado en la posterior fase de entrenamiento, para el aprendizaje del receptor. La siguiente figura muestra un ejemplo de 8 posibles diagramas de radiación que podría seleccionar el receptor.

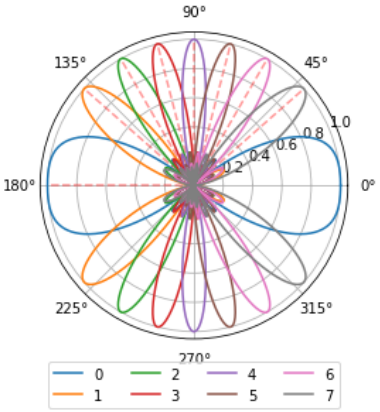


Figura 22 Posibles diagramas de radiación a seleccionar por parte del receptor.

Para la creación de estos diagramas de radiación, se ha programado un array de antenas tipo ULA, compuesto por 8 elementos, separados entre sí por una distancia igual a $\lambda/2$.

7.3 Diseño del algoritmo Deep Q-Network.

Después de haber desarrollado en escenario que proporcione los datos de entrenamiento, y haber programado un receptor que aplique la técnica de Beamforming, se dio paso al desarrollo del algoritmo Deep Q-Network que dotara de inteligencia al receptor.

En primer lugar, fue necesaria la definición de la red neuronal, que será utilizada tanto para la red principal, como para la target network. Esta red neuronal almacena la política aprendida por parte del agente a lo largo del entrenamiento, la cual le permite tomar decisiones dentro del entorno. Analógicamente, podría definirse como el cerebro del agente.

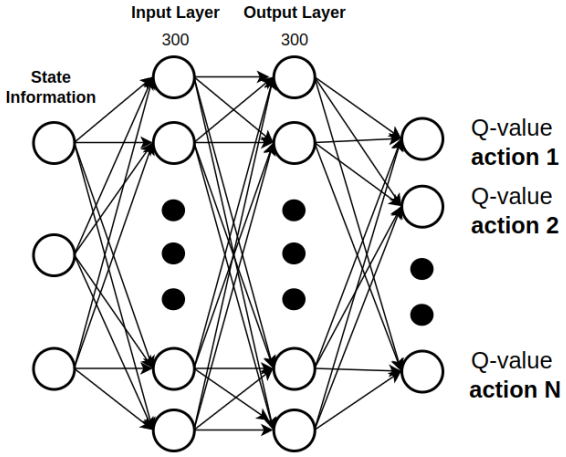


Figura 23 Red Neuronal utilizada.

En este caso, tal y como se ve en la figura 23, se ha optado por una red compuesta por 2 capas, donde cada una de ellas dispone de 300 neuronas. El número de capas y neuronas, son hiperparámetros, que se han ido modelando a lo largo de los entrenamientos, en función de los resultados obtenidos.

Por otra parte, según lo explicado en el apartado 6.1, dentro de los algoritmos Q-Learning, existen varias componentes, o actores, que interactúan entre sí a la hora de entrenar el algoritmo. Por una parte, se definen las componentes: agente y entorno, las cuales intercambian información que permita al agente llevar a cabo el aprendizaje, y por otra parte, se define la información que intercambian entre ellos: estados, acciones y recompensa. Así mismo, es necesario crear una función recompensa que determine

el comportamiento del agente dentro del entorno. Estas componentes son las que definen el escenario donde se entrenara al algoritmo.

7.3.1 Escenario: entorno, agente, estados y acciones.

Inicialmente, se plantearon dos escenarios para la resolución del problema, donde el primero de ellos se definía así:

1º Planteamiento: Generación automática del diagrama de radiación.

En este primer planteamiento se pretendía que el receptor aprendiese a generar, de una forma autónoma, el diagrama de radiación idóneo en cada situación de auto-interferencia a la que se enfrente. Para ello, las componentes principales fueron definidas como:

- **Estado**: esta variable representa la información con la que se va a alimentar a la red neuronal para ser entrenada. En este caso, se decidió utilizar como estado, una matriz compuesta por los coeficientes de los canales H1 y H2, y sus correspondientes ángulos de incidencia, AoA. Con esto se pretende que el receptor aprenda a reconocer los patrones dentro de los canales, con la ayuda de los ángulos de incidencia.
- **Agente**: el agente es el actor que almacena la política aprendida a lo largo del entrenamiento. En este caso, será el receptor quien actúe como agente, con el objetivo de aprender a generar un diagrama de radiación que le permita reducir la auto-interferencia, haciendo uso de la información proporcionada por los estados. Con esta información, para poder generar el diagrama de radiación, el agente de ser capaz de aprender a cambiar los valores de las fases ϕ_i, correspondientes a la ecuación 7.1, de a cada una de sus antenas.
- **Acciones**: para que el agente sea capaz de cambiar estas fases, debe aprender a ejecutar acciones que le permitan modificar la fase de cada una de las antenas. Para ello, se definieron 2 acciones por cada elemento del array, y una que influye en todos los elementos. En este caso, se definió una antena ULA compuesta por 4 elementos, lo que da un total de 9 posibles acciones a elegir. Las acciones que se podían tomar en cada elemento son:
    1. **subir**: aumentar el valor de la fase ϕ_i del elemento de antena correspondiente.
    2. **bajar**: disminuir el valor de la fase ϕ_i del elemento de antena correspondiente.

3. **mantener**: no realizar ningún cambio de fase.

El hecho de poder conseguir un modelo que genere automáticamente un diagrama de radiación, proporcionando los valores adecuados para las fases ϕ, era ideal para afrontar el problema de auto-interferencia. Sin embargo, tal y como se verá en el siguiente apartado 8, no se obtuvieron los resultados deseados, teniendo que descartar esta situación. Es aquí, donde fue necesaria la búsqueda de otra alternativa, que de solución al problema.

2º planteamiento: selección automática del diagrama de radiación definido dentro de un codebook.

En este caso, se desistió sobre la idea de crear el diagrama de radiación automáticamente, y se definió un codebook compuesto por 25 posibles diagramas de radiación, de los cuales, el agente, debe aprender a seleccionar el más adecuado en cada momento. Por consiguiente, los conceptos del algoritmo DQN fueron definidos de la siguiente forma:

- **Estado**: al igual que en el caso anterior, el estado sigue siendo una matriz compuesta por los coeficientes de los canales H1 y H2, sus correspondientes ángulos de incidencia AoA. Además, con el fin de orientar al agente dentro del coodebook, se incluye la posición en la que se encuentra dentro de él. La información que componen los estados, se fue determinando en base pruebas de acierto y error, observando los resultados obtenidos en cada una de ellas.

- **Agente**: en esta situación, el agente sigue siendo el receptor, pero con un objetivo diferente. Ahora el agente debe aprender a moverse dentro del codebook, seleccionando el diagrama que mejor se adapte a sus necesidades.

- **Acciones**: para que el agente pueda moverse dentro del codebook, se definieron 3 acciones:
    1. **Desplazamiento hacia la derecha**: al tomar esta acción, el agente selecciona el siguiente diagrama hacia su derecha.
    2. **Desplazamiento hacia la izquierda**: al tomar esta acción, el agente selecciona el siguiente diagrama hacia su izquierda.
    3. **Mantener posición**: se mantiene en el diagrama que tiene seleccionado.

7.3.2 Definición de la función recompensa.

Tras definir cuáles son los actores que van a participar en el entrenamiento del algoritmo Deep Q-Network, se dio paso a la definición de la función recompensa, que define el comportamiento del agente dentro del entorno. En la sección anterior, a la hora de definir al agente en cada escenario, se mencionaba la función que debía cumplir, pero no como la aprendería. Este aprendizaje viene definido a través de la función recompensa, donde se establecen las recompensas que obtendrá el agente, al pasar por cada uno de los estados posibles.

En este caso, se pretende que el modelo aprenda a seleccionar el diagrama de radiación, que le proporcione la mejor relación señal/interferencia posible en cada instante. Para ello, durante el entrenamiento, dentro de la función recompensa, se compara el canal que ha recibido el receptor a través del diagrama de radiación seleccionado, con el canal ideal que se pretende obtener. El canal ideal se representa con los valores de las muestras del canal H1, que es el que se pretende recibir, y con las muestras del canal H2 a cero. La comparación entre ambos canales se realiza a utilizando la función Mean Squared Error (MSE), la cual devuelve un valor que caracteriza la similitud entre ambos canales. Cuanto menor sea este valor, mayor será la similitud entre ambos canales, siendo 0 en el caso ideal.

En consecuencia, este valor de MSE, es utilizado para calcular la recompensa que obtiene el agente en cada uno de los estados por los que pasa, dándole así una noción sobre la idoneidad de las acciones tomadas en cada uno de ellos. Por lo tanto, la función recompensa quedaría definida como:

Algorithm 1 Definición de la función recompensa

```
procedure FUNCION RECOMPENSA(H_recibido, H_ideal)   ▷ H1 y H2 normalizados
    valor ⇐ mse(H_recibido, H_ideal)
    if valor < umbral then
        reward ⇐ reward + 50                  ▷ Recompensa de objetivo alcanzado.
        contador+ = 1
        if contador > 4 then
            reward ⇐ reward + 400     ▷ Recompensa de objetivo final cumplido.
            done ⇐ True               ▷ Para notificar que el objetivo se ha cumplido.
    else if steps > 25 then
        reward ⇐ reward − 100   ▷ Recompensa de numero de pasos alcanzados.
        done ⇐ True               ▷ Para notificar que el objetivo se ha cumplido.
    else
        reward ⇐ reward − steps ∗ 1,2                 ▷ Recompensa por avanzar.
        done ⇐ False
    return reward
```

En este algoritmo 1, vienen reflejadas las recompensas positivas y negativas, que se le otorgan al agente al pasar por los diferentes estados posibles. En cada estado, se calcula un valor MSE que será comparado con un valor umbral previamente definido. Este umbral ha sido calculado de tal forma que, represente el valor mínimo con el que se puede considerar que el objetivo ha sido alcanzado, es decir, que el canal recibido por el receptor es similar al ideal. En esta situación, si el valor de MSE es menor que el umbral, el agente recibe una recompensa positiva igual al reward acumulado hasta ese momento, más un valor igual a 50. Sin embargo, pese a haber alcanzado el objetivo, el episodio no finaliza aquí. Es necesario establecer otra norma que le indique al agente que debe mantener el diagrama de radiación fijo, mientras el canal no varíe. De esta forma, se pretende que una vez alcanzado el objetivo, el agente aprenda a mantenerlo. Para ello, se establece un contador, que le permitirá ir acumulando recompensa positiva hasta no sobrepasar un valor previamente definido. Una vez sobrepasado este valor, se da por finalizado el episodio, proporcionando una recompensa positiva igual al reward acumulado hasta ese momento, más un valor igual a 400. Esta es la recompensa más grande que puede obtener el agente, dándole a entender que es el objetivo que debe alcanzar.

Por otra parte, las recompensas negativas se obtienen en dos situaciones. La primera de ellas, se da cuando el agente realiza 25 pasos dentro del entorno, es decir, cuando ha pasado por 25 estados diferentes, o iguales. Se establece este límite de 25 pasos, debido a que el codebook está compuesto por 25 diagramas diferentes, y se entiende que tras haber recorrido estos 25 estados, el agente ha tenido la oportunidad de haber seleccionado, mínimo una vez, el diagrama más adecuado para ese episodio. En el caso de no haber alcanzado el objetivo, tras haber recorrido 25 estados, se dará por finalizado el episodio, otorgando una recompensa negativa igual al reward obtenido hasta ese momento, menos un valor igual a 100. Para generar el codebook se ha definido una antena tipo ULA, compuesta por 8 elementos, con el fin de generar unos diagramas de radiación cuya directividad sea mayor en la dirección de máxima radiación.

Finalmente, la segunda situación, donde se obtiene una recompensa negativa, sucede cuando el agente se mueve dentro del abanico de posibilidades. En cada estado por el que pasa, si no se cumple ninguna de las anteriores condiciones, el entorno devuelve al agente una recompensa igual a la recompensa acumulada hasta ese momento, menos un valor proporcional al número de estados que lleva recorridos. Con esto se pretende que el agente aprenda a cumplir su objetivo, pasando por el menor número de estados posible.

7.3.3 Definición de los hiperparámetros del entrenamiento

Antes de dar paso a la definición del algoritmo de entrenamiento, es necesario explicar algunos de los hiperparámetros que influyen en el aprendizaje del algoritmo. Algunos de estos hiperparámetros, se definieron a lo largo del apartado 6.1, y otros correspondientes al entrenamiento del algoritmo serán enumerados a continuación:

- **Epochs**: Este valor define el número total de iteraciones que se realizan sobre todos los datos de entrenamiento, determinando así, el tiempo de duración del entrenamiento. En este caso, se ha optado por un valor de **15 epochs**, el cual se ha obtenido realizando pruebas de acierto/error, que serán explicadas en el apartado de resultados.

- **Número de steps**: Representa el número total de pasos que dará el agente en una epoch, o lo que es lo mismo, el número de estados por los que pasara durante una epoch. En cada uno de ellos, se actualizan los parámetros de la red neuronal. Para obtener su valor, se tuvo en cuenta el número de escenarios donde debe ser entrenado el algoritmo. Como se obtuvieron 14 escenarios, y en cada una de ellos, se realizaron 5 capturas, se obtuvo un total de 70 escenarios posibles. Además, según lo definido en la función recompensa 1, un episodio puede durar un máximo 25 steps, con lo cual, en el peor de los casos, se necesitaría un total de 1750 steps en cada epoch, para poder recorrer todos los escenarios. Por este motivo, se optó por un valor de steps igual a 2000.

- **Lost Function**: tal y como se definió en el apartado [14], este tipo de redes neuronales, para actualizar sus parámetros internos, requieren de una función de perdidas que determine el error de predicción. En este caso, se ha optado por la función que calcula el **error cuadrático medio** para estimar la función de perdidas.

- **Optimizer**: este hiperparámetro es el encargado de optimizar la red neuronal, actualizando el valor de sus parámetros en cada una de la epochs. El algoritmo de optimización más conocido es el de '**descenso del gradiente**', y para este proyecto se ha hecho uso de **Adam**, siendo este uno de los más conocidos dentro de esta familia.

- **Learning rate**: este valor define el ritmo con el que los parámetros de una red neuronal, convergen hacia una solución correcta. Un valor muy grande podría hacer que la red nunca llegue a converger, y un valor muy pequeño, podría hacer que la red no aprenda la solución más óptima. En este caso, se

ha seleccionado, en base a pruebas de error/acierto, un valor de **0.00025**. Esto se verá más detalladamente en el siguiente apartado de resultados.

- **Target update**: este valor define cada cuantos steps se actualizara la target network. En este caso, se ha optado por un valor igual a **20**.

Por otra parte, como ya se vio en el apartado 6.1, otros hiperparámetros a tener en cuenta, son los utilizados a la hora de seleccionar la estrategia con la que el agente tomara las acciones, y los utilizados para definir la Replay Memory.

Estrategia épsilon-greedy.

Tal y como se explicó en el apartado 6.1.2, esta estrategia define, en base al decrecimiento de una variable llamada épsilon, si el agente toma las acciones de una forma aleatoria, o haciendo uso del conocimiento ya adquirido hasta ese momento. Para ello, es necesario decidir como hacer decrecer esta variable épsilon, y seleccionar el valor de los hiperparámetros que tomaran parte en este proceso.

En este caso, se ha optado por decrementar la variable épsilon teniendo en cuenta el número de epoch. Para ello se establecieron dos fases, donde la duración de cada una de ellas son los primeros hiperparámetros a definir. La primera de estas fases, tiene una duración igual al 10 % del total de epochs, y la segunda, una duración igual al 90 % del total de epochs. Con esto se pretende establecer una fase inicial, que dure al menos el 10 % del entrenamiento total, donde el agente se encuentre explorando el entorno, seleccionando acciones aleatorias. Por otra parte, en la segunda fase, se decrementa la variable épsilon de una forma proporcional al número de epochs totales, permitiendo así, aumentar la probabilidad de que el agente use el conocimiento ya adquirido, para seleccionar la mejor acción posible.

Algorithm 2 Estrategia epsilon-greedy.

procedure SELECT ACTION($epoch, start, end, epochs, epsilon$)
  $explora \Leftarrow 0{,}1 * epochs$ ▷ se define la duracion de las fases.
  $explota \Leftarrow 0{,}8 * epochs$
  if $(epoch < explora)$ **then** ▷ Fase de exploración.
    $epsilon \Leftarrow start$
  else if $(epoch < explota)$ **then** ▷ Fase de explotación.
    $epsilon \Leftarrow (start - (epoch - explora))/((explota - explora) * (start - end))$
  else
    $epsilon \Leftarrow end$
  if $(epsilon > randomValue)$ **then**
    $action \Leftarrow random(actions)$ ▷ Se toma una accion aleatoria.
  else
    $action \Leftarrow max(policy_{net}(actions))$ ▷ Se usa la red neuronal.
  return *action*

En el algoritmo 2, se definen las 2 fases que componen el proceso, y los hiperparámetros, **start y end**, que hacen referencia al valor inicial y final de la variable épsilon. Como es habitual en este tipo de estrategia, se toma un valor inicial igual a **1**, y un valor final igual a **0.05**. Es importante tener en cuenta que, este valor final, determina la probabilidad con la que el agente entrará en la fase de exploración en los últimos steps del entrenamiento. Por esto, es recomendable no establecer un valor muy pequeño, ya que a pesar de que en esta fase el agente debería ser capaz de cumplir su objetivo, aún es necesario que disponga de ciertos momentos donde pueda tomar acciones de forma aleatoria, que le permitan descubrir otros caminos más óptimos hacia su objetivo.

Replay Memory

En cuanto a la definición de la replay memory, simplemente se tomaron los valores más comunes para este tipo de entrenamiento.

- **Batch size = 64.** Esto quiere decir, que cada vez que se quiera actualizar la red principal, se tomaran 64 tuplas aleatorias de la memoria. A este conjunto de tuplas se le conoce como 'batch'.
- **Memory size = 100000**. La memoria almacenará un total de 100000 tuplas. En el momento que supere este valor, sé ira actualizando con nuevas tuplas, descartando las más antiguas.

7.3.4 Entrenamiento del modelo Beamforming basado en Deep Q-Network.

Una vez definidos los actores que toman parte en el entrenamiento, la función recompensa y los diversos hiperparámetros, se da paso a la explicación del procedimiento llevado a cabo para entrenar el modelo Beamforming. Tal y como se ha definido en el apartado anterior, el entrenamiento tiene una duración de 10 epoch, y en cada epoch, se realizan 2000 steps. Antes de empezar con el entrenamiento, es necesario inicializar todos los componentes, como las redes neuronales, el estado inicial, $\boldsymbol{S}$, o la replay memory. Una vez está todo inicializado, en cada uno de los steps se realizan los siguientes pasos:

- Seleccionar una accion $\boldsymbol{a}$, con el estado $\boldsymbol{S}$, en base a la estrategia Epsilo-greedy definida.
- Seleccionar el nuevo diagrama de radiación, $\boldsymbol{beam}$.
- Calcular el siguiente estado, $\boldsymbol{S'}$.

- Calcular el valor de MSE, para el siguiente estado, y obtener el valor de reward, $\boldsymbol{r}$.
- Almacenar la tupla (S, S', a, r) dentro de la replay memory.
- Actualizar el estado S.
- Actualizar la red neuronal principal, $policy_{net}$.
- Comprobar sí es necesario actualizar la Target network, $target_{net}$.
- Comprobar si el episodio ha finalizado.

Estos pasos se pueden ver reflejados en el siguiente algoritmo, y aunque no vienen detallados todos los pasos que realmente se realizan, es útil para entender el procedimiento que se lleva a cabo dentro del entrenamiento.

Algorithm 3 Definición de la función recompensa

```
1: procedure ENTRENAMIENTO DQN(H_1, H_2, AoA_1, AoA_2, codebook)
2:     state ⇐ preprocess(H_1, H_2, AoA_1, AoA_2, position)
3:     strategy ⇐ epsilonGreedyStrategy(start, end, epochs)
4:     policy_net, target_net, agent, memory ⇐ initializeActors(strategy, actions)
5:     for (epoch < 10) do
6:         for (steps < 1000) do
7:             action ⇐ agent.selectAction(state, policy_net, epoch)
8:             beam ⇐ changeBeam(action, codebook, position)
9:             nextState ⇐ updateState(AoA_1, AoA2, H_1, H_2, beam)
10:            MSE ⇐ meanSquaredError(nextState, H_ideal)
11:            reward, done ⇐ funcionRecompensa(MSE, step)
12:            memory.push(state, nextState, action, reward, done)
13:            state ⇐ nextState
14:            if memorySize > batchSize then
15:                states, actions, rewards, nextStates, dones ⇐ memory(batchSize)
16:                Q_prime ⇐ target_net(nextStates)
17:                Q_target ⇐ rewards + learningRate * Q_prime
18:                Q ⇐ policy_net(states)
19:                loss ⇐ meanSquaredError(Q_target, Q)
20:                loss ⇐ meanSquaredError(Q_target, Q)
21:                updateNetwork(policy_net)
22:            updateTargetNetwork(policy_net, target_net)
23:            if done = True then
24:                H_1, H_2, AoA_1, AoA_2 ⇐ updateData()
25:                state ⇐ preprocess(H_1, H_2, AoA_1, AoA_2, position)
```

En las primeras líneas, 2-4 del algoritmo 3, se inicializan todas las variables que van a ser partícipes del entrenamiento, donde las más importantes son las redes neuronales,

el estado inicial con el que dará comienzo el entrenamiento, el agente y la estrategia que va a utilizar. El estado inicial se obtiene procesando los datos correspondientes a los canales: H_1 y H_2, con el fin de obtener una matriz normalizada entre 0 y 1, que facilite el ajuste de los parámetros de las redes neuronales. Con los datos ya inicializados, se da comienzo al entrenamiento, donde en cada uno de los steps, se realizan los pasos enumerados anteriormente.

En primer lugar, se selecciona una acción utilizando la estrategia epsilon-greedy definida en la sección 14. Esta estrategia devolverá una acción seleccionada de forma aleatoria, o usando la red neuronal principal $policy_{net}$, dependiendo del valor de la epoch en la que se encuentre en ese momento. Una vez seleccionada la acción, se hace uso de ella para seleccionar el diagrama de radiación dentro del codebook previamente definido. Con este nuevo diagrama, definido como beam, se calculan los nuevos valores de los coeficientes de H_1 y H_2 haciendo uso de los ángulos de incidencia AoA. De esta forma se consigue el nuevo estado, S', que refleja la respuesta del beam seleccionado ante los canales H_1 y H_2, es decir, representa los nuevos canales recibidos por el receptor a través del beam seleccionado. Este nuevo estado, junto al canal ideal previamente definido, son utilizados para calcular el valor de MSE, que refleja la similitud entre ambos canales. El valor MSE obtenido, es proporcionado a la función recompensa, definida en 1, para calcular el valor de reward y determinar, a través de la variable done, si el episodio ha finalizado o no. Una vez que se obtienen los valores de las variables: estado, acción, siguiente estado, reward y done, son almacenadas en forma de tupla (S, S', a, r), dentro de la replay memory. Con los valores ya almacenados, se actualiza la variable estado, S, con los valores del siguiente estado, S', para que la siguiente iteración inicie con los valores actualizados.

Los siguientes pasos del algoritmo se corresponden con la actualización de las redes neuronales, $policy_{net}$ y $target_{net}$. Para actualizar la red $policy_{net}$, como se ve en la línea 14, es necesario comprobar si la memory replay contiene el número de tuplas mínimo requerido para actualizar la red. En el caso de ser así, se procede a seleccionar un conjunto de tuplas aleatorias, o batch, de la replay memory, cuyo tamaño viene definido por el hiperparámetros batchSize. Los valores de este conjunto de tuplas, son utilizados para calcular los valores Q a través de la ecuación de Bellman 6.2.1, y así, poder actualizar los parámetros de las redes neuronales calculando la función de perdidas. En este caso, la función de perdidas se calcula a través del error cuadrático medio entre el valor Q objetivo, Q_{target}, y el valor Q de predicción. Por otra parte, la target network se actualiza en función del hiperparámetro target update, cuyo valor indica cada cuanto debe ser actualizada esta red.

Finalmente, con el valor done obtenido de la función recompensa, se comprueba si ha finalizado un episodio de entrenamiento. Los episodios pueden finalizar tras haber alcanzado el objetivo, o tras haber excedido el número de step máximos. En el caso de que un episodio finalice, se actualiza el escenario cambiando los valores de los canales recibidos. Estos nuevos valores se corresponde a otro escenario, donde los transmisores tienen nuevas posiciones. Con esto se pretende que el algoritmo sea entrenado en todos los escenarios previamente obtenidos a través de Quadriga, explicada en el apartado 7.1.

Todo el procedimiento explicado hasta este momento se puede ver reflejado, de forma resumida, en la siguiente imagen:

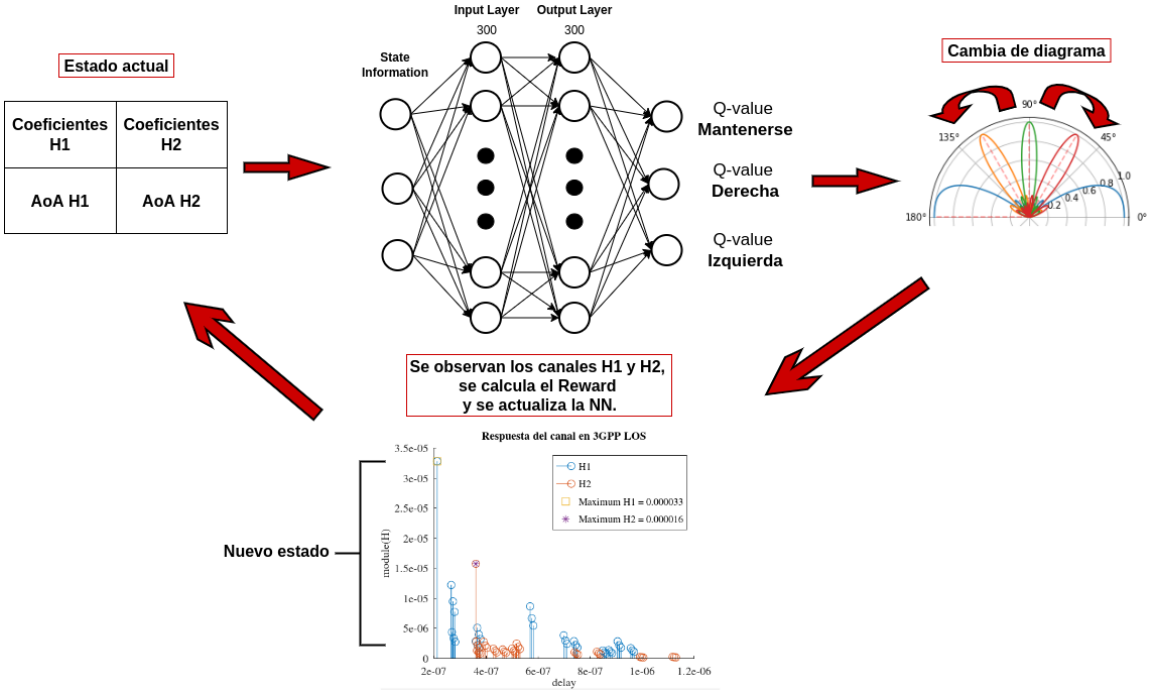


Figura 24 Esquema del entrenamiento del modelo Beamforming.

8 Resultados.

Una vez explicado el procedimiento llevado a cabo para entrenar el algortimo DQN, se da paso a explicar los resultados obtenidos durante el proceso de entrenamiento. En primer lugar, se mostrarán los resultados de las pruebas realizadas para elegir los valores de algunos hiperparámetros, y posteriormente, se explicarán los resultados obtenidos en los escenarios explicados en el apartado 7.3.1.

8.1 Resultados previos: Definición de los hiperparámetros.

El primer paso consistió en realizar varias pruebas que permitan ajustar los valores de los hiperparámetros explicados en el apartado 7.3.3. En este caso, solo se explicarán las pruebas realizadas sobre 3 hiperparámetros, con el único fin de entender como se ajustan este tipo de variables. Los hiperparámetros que se han ajustado son: capas y neuronas, learning rate y número de epoch. Las pruebas consisten en ir probando diferentes valores para cada uno de estos hiperparámetros.

8.1.1 Capas y neuronas.

A la hora de seleccionar estos valores hay que tener en cuenta que un mayor número de neuronas y capas, puede proporcionar un mejor aprendizaje, pero también requiere de mayor tiempo de procesamiento. Esto se ha podido comprobar en las siguientes simulaciones, donde se realiza un total de 10 epochs, y se evalúa el porcentaje de aciertos conseguido, al cambiar el número de neuronas, en cada una de ellas. Este porcentaje denota cuantos episodios se han realizado correctamente respecto al total de episodios ejecutados en cada epoch.

Simulación 1: 2 capas y 50 neuronas en cada capa.

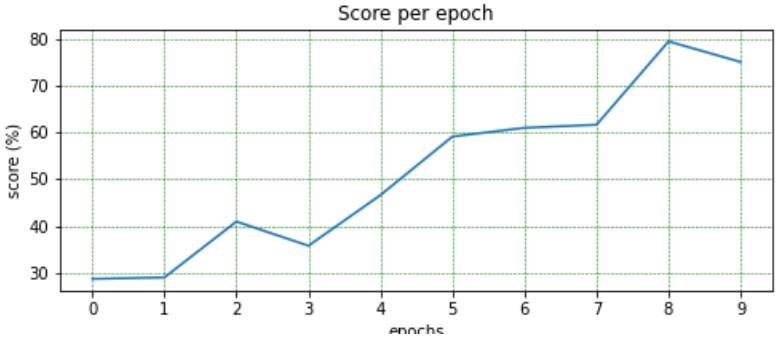


Figura 25 Entrenamiento con una red neuronal de 2 capas y 50 neuronas.

Resultados:

- Tiempo de simulación: **29 s/epoch.**
- Score máximo: **79.2 %.**

Simulación 2: 2 capas y 100 neuronas en cada capa.

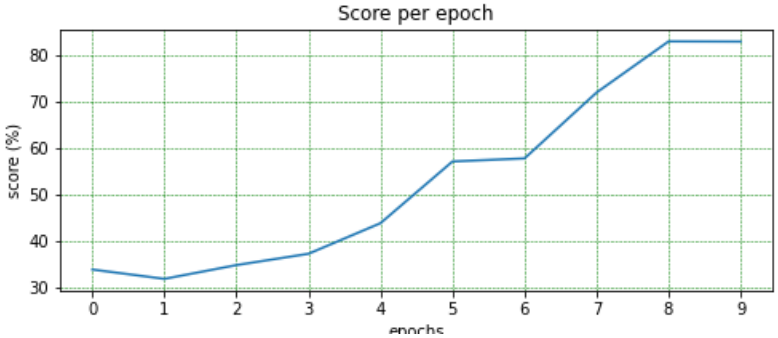


Figura 26 Entrenamiento con una red neuronal de 2 capas y 100 neuronas.

Resultados:

- Tiempo de simulación: **31 s/epoch.**
- Score máximo: **82.81 %.**

Simulación 3: 2 capas y 300 neuronas en cada capa.

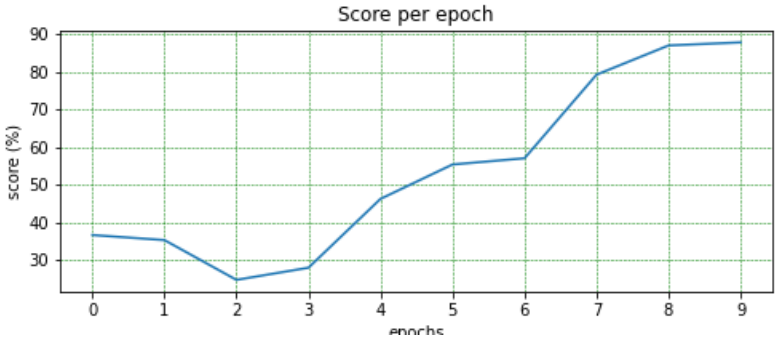


Figura 27 Entrenamiento con una red neuronal de 2 capas y 300 neuronas.

Resultados:

- Tiempo de simulación: **35 s/epoch.**
- Score máximo: **87.7 %.**

Simulación 4: 2 capas y 400 neuronas en cada capa:

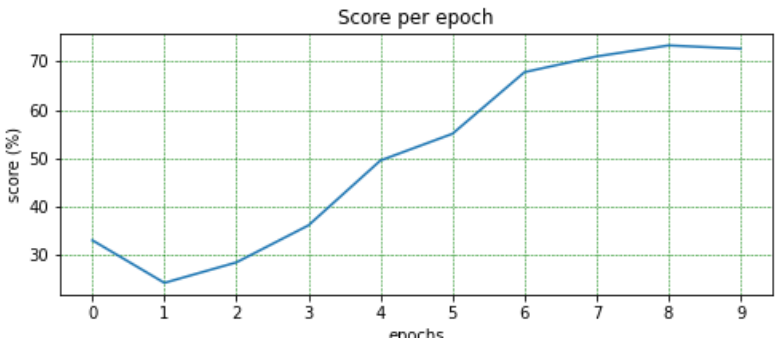


Figura 28 Entrenamiento con una red neuronal de 2 capas y 400 neuronas.

Resultados:

- Tiempo de simulación: **37 s/epoch**.
- Score máximo: **72.64 %.**

En las simulaciones vistas hasta este momento, se puede comprobar que la mejor opción se muestra en figura 27, donde la red neuronal está formada por 2 capas de 300 neuronas. A partir de este número de neuronas, tal y como se puede ver en la figura 28, el aprendizaje empieza a decaer. Para definir el número de capas, se ha tenido en cuenta el número de neuronas seleccionado (300), y se ha añadido una capa más, con este mismo número de neuronas, para comprobar que impacto tiene dentro del entrenamiento. En este caso, se observa el siguiente comportamiento:

Simulación 5: 3 capas y 300 neuronas en cada capa.

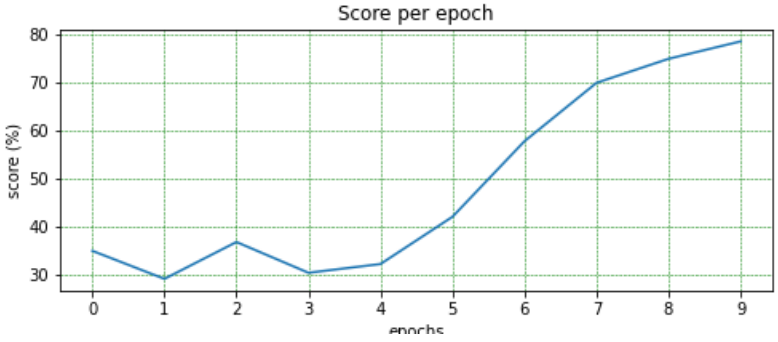


Figura 29 Entrenamiento con una red neuronal de 3 capas y 300 neuronas.

Resultados:

- Tiempo de simulación: **55 s/epoch.**
- Score máximo: **78.62 %**

Como se puede ver en la figura 29, el valor máximo obtenido en cuanto a porcentaje de acierto es de 78.62 %, siendo inferior al obtenido en la simulación 27, donde se usaron 2 capas. Además de obtener un peor resultado, tal y como era de esperar, el tiempo de simulación es muy superior al obtenido con 2 capas. Por esta razón, se decide que los valores en cuanto a número de capas y de neuronas para la red neuronal son:

- **Número de capas: 2.**
- **Número de neuronas: 300.**

8.1.2 Learning rate.

Este hiperparámetro, de una forma muy resumida, se podría considerar como la variable que determinara la velocidad de aprendizaje de nuestro algoritmo. Tal y como se mencionó en el apartado 7.3.3, un valor muy grande podría hacer que el algoritmo no llegase a converger, y un valor muy pequeño, podría hacer que la solución proporcionada no sea la más óptima. En este caso, se realizan 4 simulaciones para observar el comportamiento de este hiperparámetro, y así poder decidir que valor es el más adecuado para este proyecto.

Simulación 1: Learning Rate = 0.8.

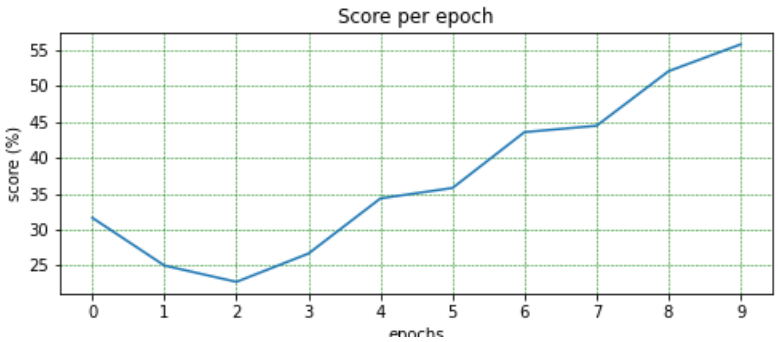


Figura 30 Entrenamiento con una LR = 0.8.

Resultados:

- Score máximo: **55.76 %**

Simulación 2: Learning Rate = 0.25.

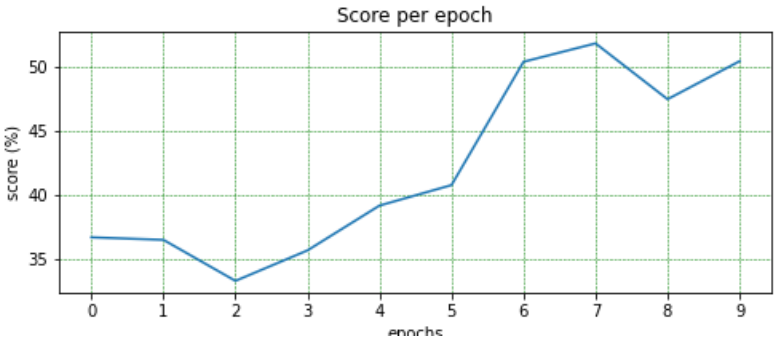


Figura 31 Entrenamiento con una LR = 0.25.

Resultados:

- Score máximo: **50.49 %**

Simulación 3: Learning Rate = 0.00025

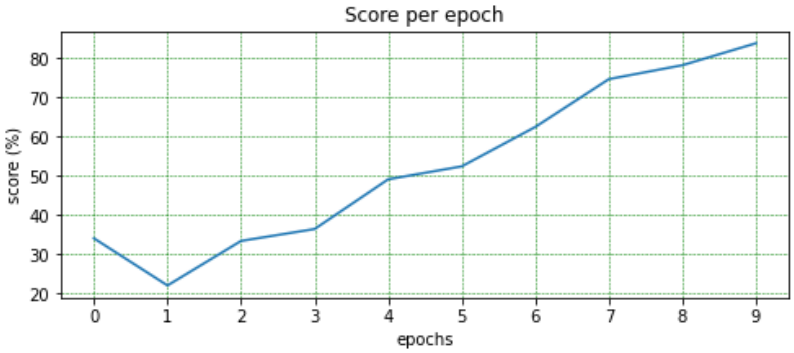


Figura 32 Entrenamiento con una LR = 0.00025.

Resultados:

- Score máximo: **85.65 %**

Simulación 4: Learning Rate = 0.000000025

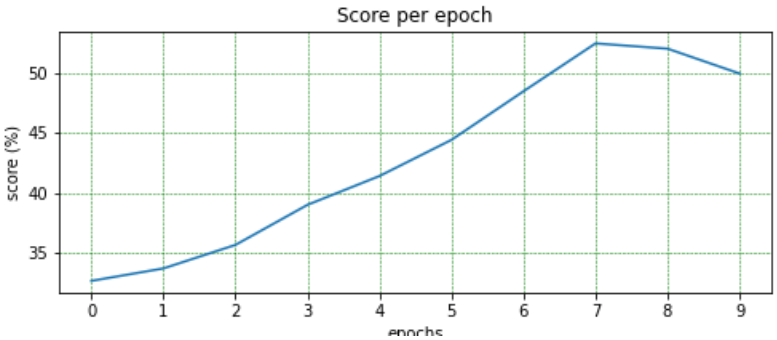


Figura 33 Entrenamiento con una LR = 0.000000025.

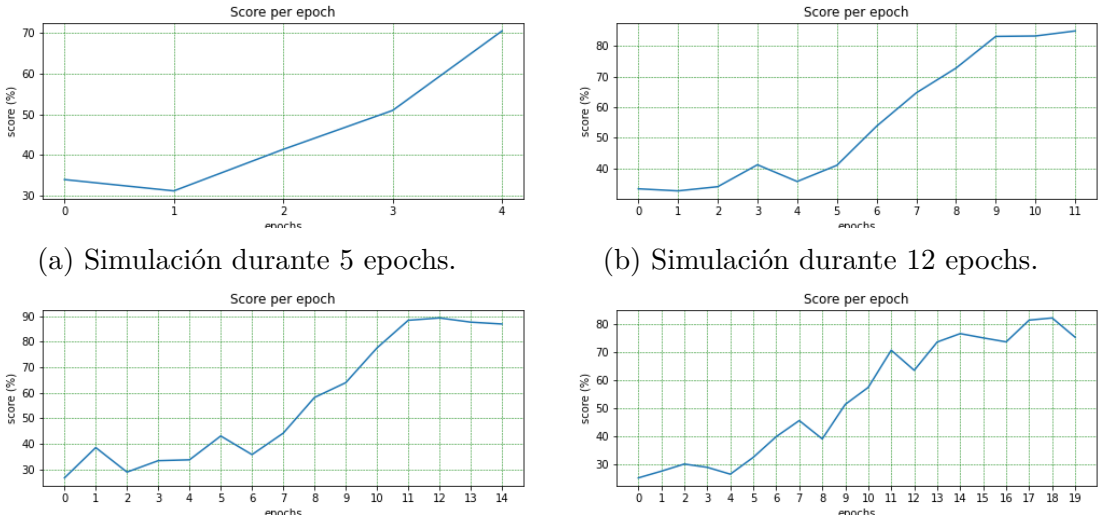
Resultados:

- Score máximo: **50.0 %**

Analizando las imágenes se comprueba como un valor alto de learning rate hace que el aprendizaje no converja hacia una solución correcta, proporcionando un porcentaje de acierto máximo cercano al 50/55 %. Así mismo, si se toma un valor muy bajo, como en la simulación 33, el aprendizaje se vuelve inestable, dando como resultado final un valor máximo de 50 %. En esta situación, tal y como se ve en la figura 32, un valor de **0.025**, para el hiperparámetro de learning rate, es el más adecuado.

8.1.3 Número de epochs.

Finalmente, se explica como se ha obtenido el valor correspondiente al hiperparámetro epochs. Este hiperparámetro marca la duración del entrenamiento, definiendo el número de iteraciones que se realizan sobre los datos de entrenamiento. Al igual que en los casos anteriores, el valor óptimo para este hiperparámetro, ha sido obtenido en base a simulaciones de prueba/error, observando el porcentaje de acierto obtenido en cada caso. En esta situación, se utiliza un gráfico con las 4 simulaciones realizadas, con el fin de observar a la evolución del porcentaje de acierto en relación con el número de epochs establecido.



(a) Simulación durante 5 epochs.

(b) Simulación durante 12 epochs.

(c) Simulación durante 15 epochs.

(d) Simulación durante 20 epochs.

Figura 34 Simulaciones realizadas para diferentes números de epochs.

Observando la figura 34, se deduce que un valor pequeño, como 5 epoch, no es suficiente para que el algoritmo aprenda a resolver todos los casos planteados. Así mismo, en la cuarta imagen 34d, se puede ver como un aumento progresivo del número de epochs puede implicar un desaprendizaje, que proporcione un porcentaje de acierto cada vez menor. Por lo tanto, el número de epoch adecuado para esta situación, debía encontrarse entre el rango de 5 y 20 epochs. Esto se comprueba en las imágenes 34b y 34c, donde se puede ver claramente, como el porcentaje de acierto alcanza unos valores superiores al 80 %, siendo el caso ideal, el simulado en la imagen 34c, done el porcentaje de acierto llega a alcanzar un **88 %**, para un total de **15 epochs**.

8.2 Resultados finales.

Finalmente, en este apartado, se da paso a los resultados finales obtenidos en los escenarios mencionados en el apartado 7.3.1. En primer lugar, se explicarán únicamente los resultados obtenidos dentro del escenario donde el receptor debía generar el diagrama de radiación de una forma autónoma, dejando las conclusiones y el porqué fue descartado para el siguiente apartado. Posteriormente, se dará paso a explicar los resultados obtenidos dentro del escenario que finalmente se consiguió resolver.

8.2.1 Primer planteamiento: generación automática del diagrama de radiación.

En este primer caso, el receptor debía ser capaz de obtener, de una forma autónoma, los valores de las fases ϕ correspondientes a cada elemento del array, que generen

un diagrama de radiación, el cual minimice la auto-interferencia existente en cada situación. A la hora de realizar los entrenamientos, se llevaron a cabo una multitud de pruebas idénticas a las explicadas en el apartado anterior, sin embargo, en este caso, no se consiguió ajustar el valor de los hiperparámetros como para obtener unos resultados coherentes. En la siguiente imagen, se muestra una simulación de 15 epochs, realizada con una red compuesta por 2 capas de 300 neuronas cada una, y un learning rate de 0.25.

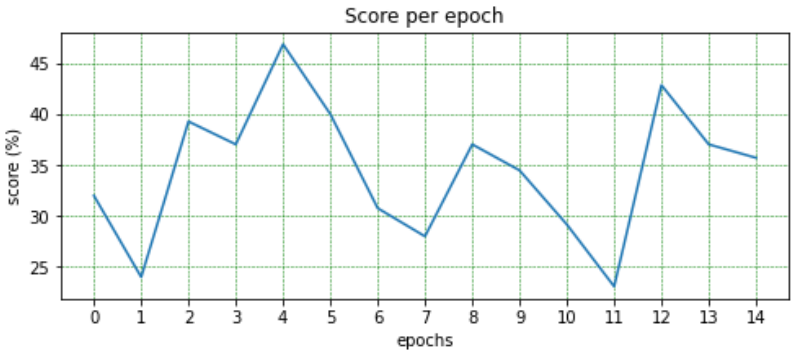


Figura 35 escenario 1.

Para esta situación, se observó que un valor de learning rate más alto proporcionaba un mejor porcentaje de acierto, aunque seguía sin ser el esperado. Una vez entrenado el modelo, fue evaluado sobre los datos con los que se entrenó, y representado a través de imágenes que proporcionen una noción más intuitiva sobre lo aprendido. Estas imágenes se ven representadas en la siguiente figura:

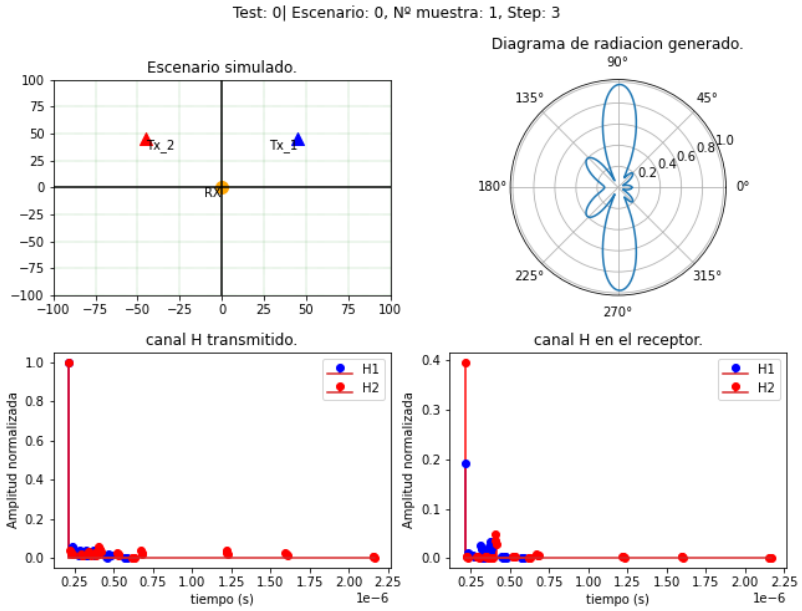


Figura 36 Escenario 1 simulado: muestra 1, step 3.

En esta figura 36, por un lado, en la parte superior de la imagen, se visualiza el escenario donde se ha evaluado el modelo y el diagrama generado automáticamente por el receptor, así mismo, en la parte inferior de la imagen, se visualizan los canales transmitidos por los transmisores, y los canales recibidos en el receptor a través del diagrama generado.

Observando estos resultados, se aprecia como el diagrama de radiación generado no es el más adecuado en esa situación, y a medida que avanzan los steps, como se puede ver en la siguiente figura 37, el resultado no mejora.

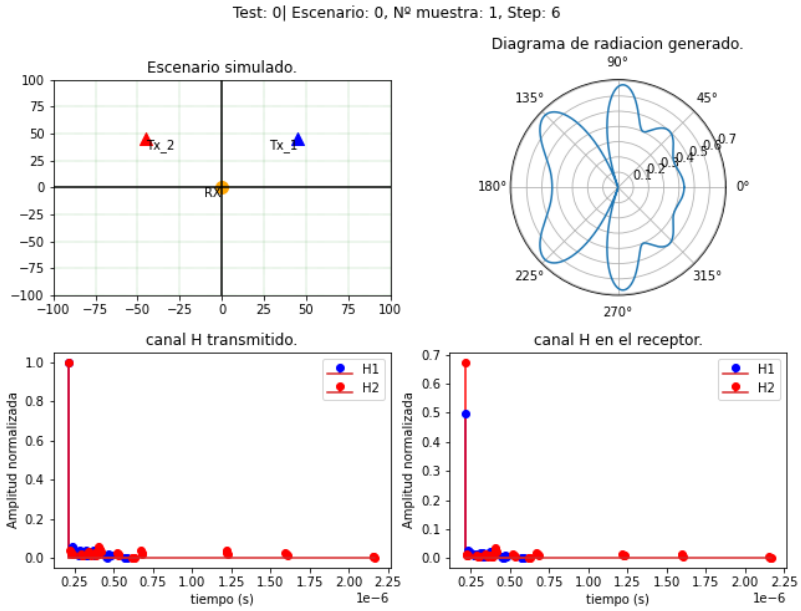


Figura 37 Escenario 1 simulado: muestra 1, step 6.

A la vista de estos resultados, el complejo aprendizaje por parte de la red y los tiempos que se requieren para ello, hizo necesaria la búsqueda de otra solución que permitiese resolver el problema de auto-interferencia.

8.2.2 Segundo planteamiento: selección automática del diagrama de radiación definido dentro de un codebook.

Una vez descartado el primer planteamiento, se decidió afrontar el problema partiendo de diagramas de radiación previamente definidos, evitando que el agente los genere, y simplemente aprenda a seleccionar el más adecuado en cada momento.

En esta situación, como ya se vio en el apartado 8.1, los hiperparámetros se pudieron ajustar correctamente hasta obtener un porcentaje de acierto superior al 87 %. Para evaluar el funcionamiento de este modelo, en primer lugar, al igual que se hizo en el escenario 1, se evaluó sobre los datos de entrenamiento, y posteriormente, se evaluó

con unos datos de entrada diferentes a los de entrenamiento. Estos nuevos datos de entrada, representan escenarios donde los transmisores se encuentran en posiciones que han sido seleccionadas aleatoriamente, para las cuales el modelo no ha sido entrenado. De esta forma se pretende comprobar el comportamiento del modelo frente a situaciones que nunca antes ha visto. Estos resultados se desarrollan en los siguientes apartados:

Evaluación con los datos de entrenamiento:

Esta primera imagen hace referencia al estado inicial con el que parte la primera prueba de evolución. En este caso, los transmisores se encuentran situados a la misma distancia en relación con el receptor, formando un ángulo de 45 grados respecto al eje horizontal. En cuanto al diagrama seleccionado, se parte desde la posición 5 dentro del codebook, dando como resultado el diagrama observado en la parte superior de la figura 38. Este diagrama inicial permite al receptor, recibir un canal que se ve reflejado en la parte derecha de la zona inferior de la figura 38.

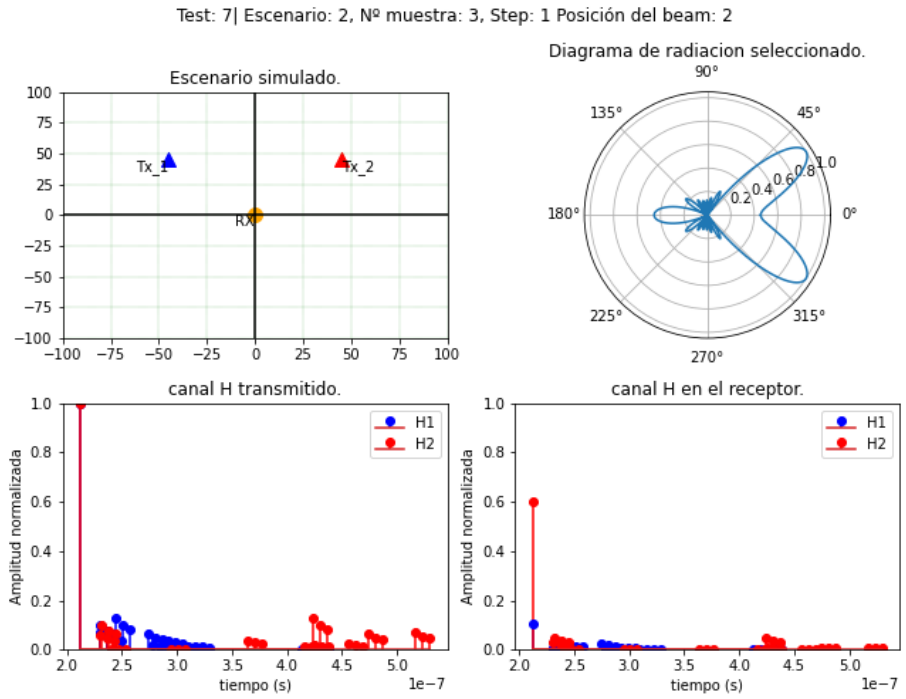


Figura 38 Escenario 2 simulado: Muestra: 3, Step: 1, Posición del beam: 2.

Una vez iniciada la prueba, el receptor debe decidir si ir hacia la derecha o hacia la izquierda dentro del codebook, con el fin de llegar a seleccionar el diagrama más adecuado en esta situación.

En este caso, tal y como se puede ver en la siguiente figura 39, el receptor avanza hacia la derecha, cambiando el diagrama radiación, y, por lo tanto, el canal recibido.

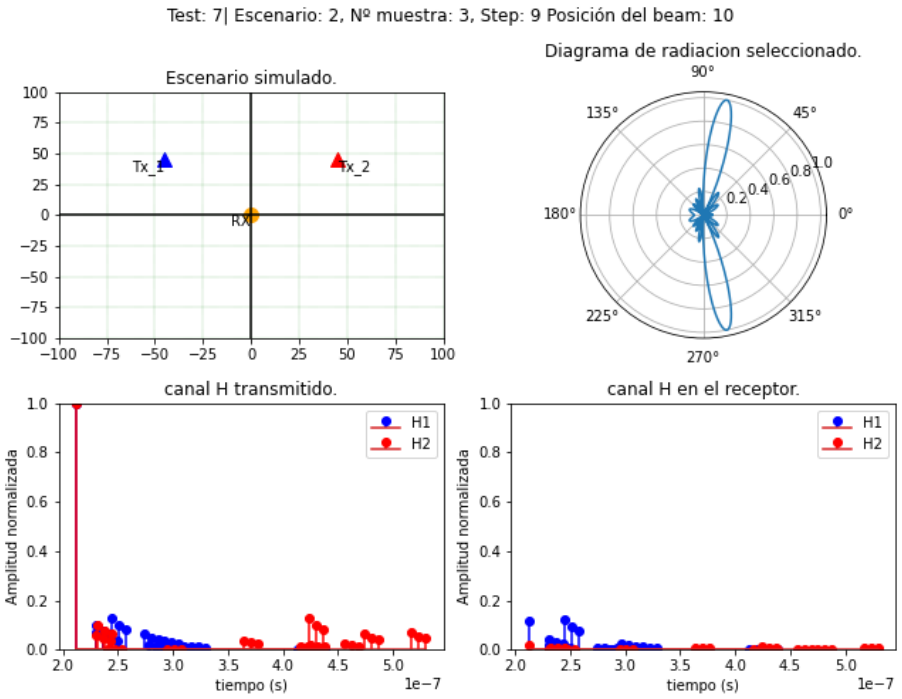


Figura 39 Escenario 2 simulado: Muestra: 3, Step: 9, Posición del beam: 10.

Finalmente, el receptor avanza dentro del codebook hasta dar con el resultado deseado. Este resultado se ve reflejado en la siguiente figura 40, donde la solución se encuentra en el beam con la posición 20 dentro del codebook.

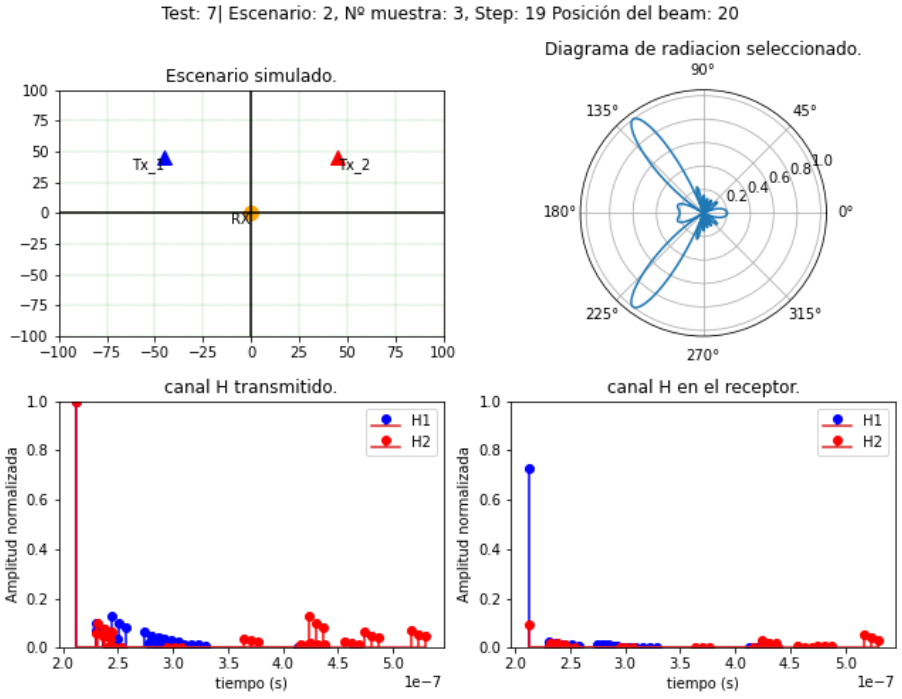


Figura 40 Escenario 2 simulado: Muestra: 3, Step: 19, Posición del beam: 20.

En esta situación, se puede ver como el beam seleccionado atenúa el canal H2, referente a la señal de interferencia, y obtiene la mayor ganancia posible para el canal H1, referente a la señal deseada.

Por otro lado, a la hora de testear cada escenario, se realizan un total de 25 steps, con el fin de comprobar si una vez alcanzado el objetivo, el receptor es capaz de mantener el resultado. En la siguiente figura 41 se puede comprobar como tras haber recorrido 25 steps, el diagrama sigue siendo el correspondiente a la posición 20.

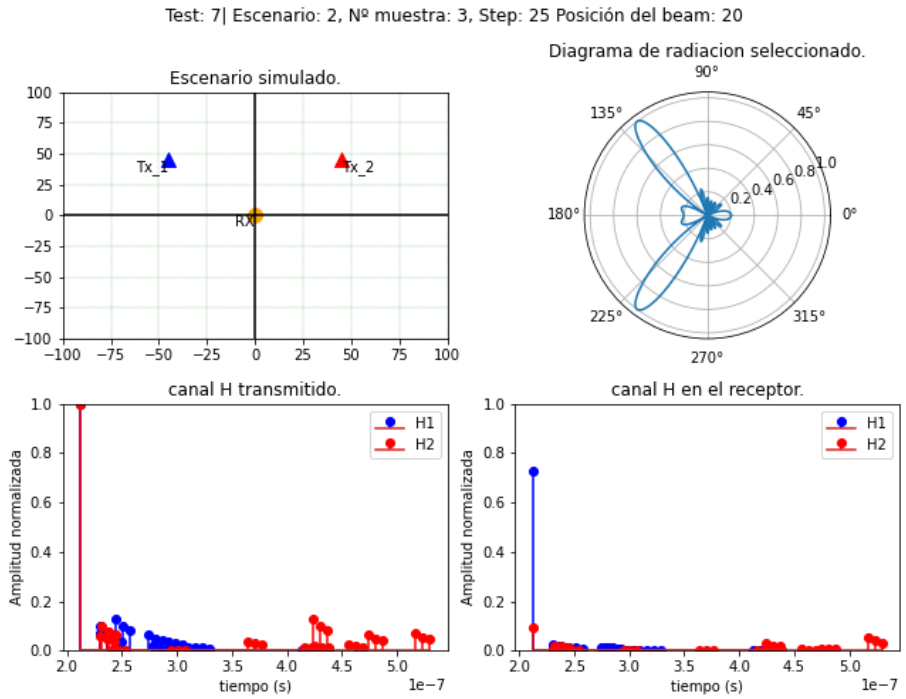


Figura 41 Escenario 2 simulado: Muestra: 3, Step: 25, Posición del beam: 20.

Para una mejor visualización de estos resultados, se ha creado un video disponible en el link [19].

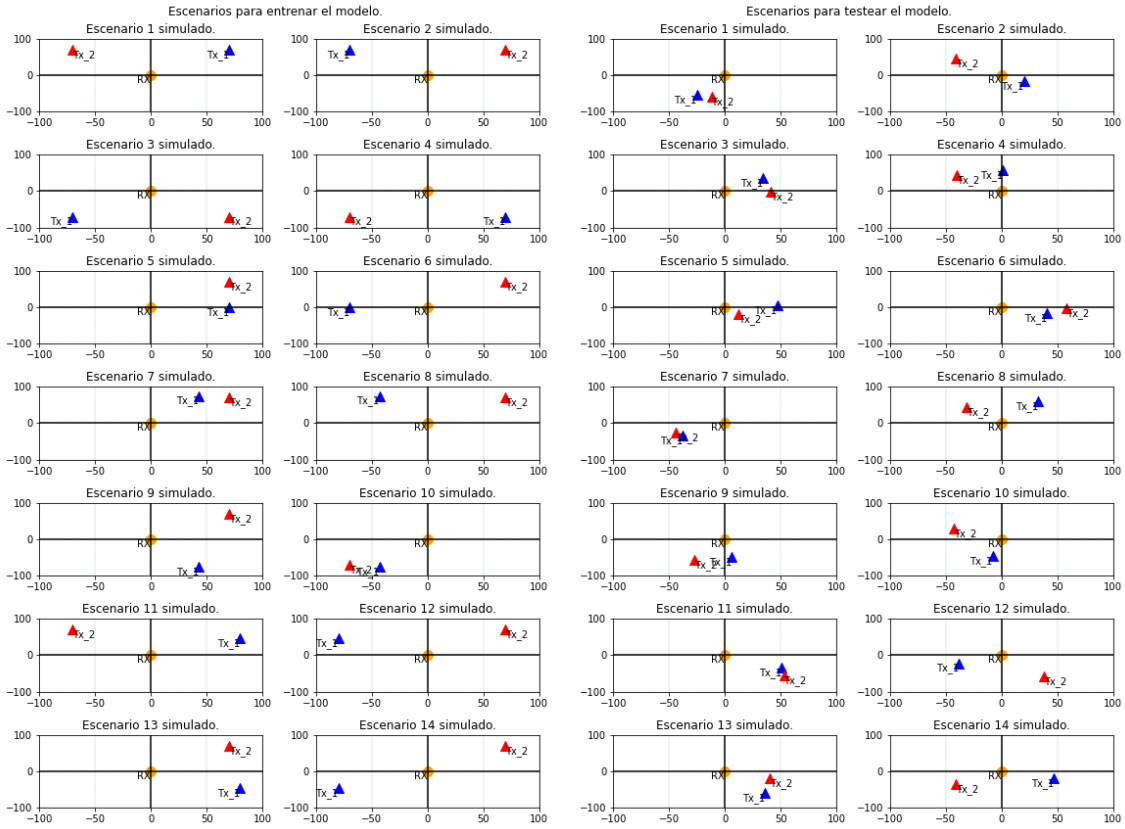
Evaluación con datos diferentes a los del entrenamiento:

Para comprobar que el modelo entrenado puede extrapolar su conocimiento a escenarios para los cuales no ha sido entrenado, se generaron nuevos escenarios donde las posiciones de los transmisores se generaron de una forma aleatoria. Antes de ver los resultados, se hará una comparativa de las posiciones con las que se entrenó al algoritmo, y las posiciones con las que se evaluó. En la siguiente tabla, aparecen las posiciones que tienen los transmisores sobre un plano de rango [-100, 100]. Así mismo, los ángulos indicados, hacen referencia al ángulo que forma cada transmisor respecto al eje horizontal, y tomando como punto de origen el receptor.

Posiciones random para el Test				Posiciones para el Entrenamiento			
Transmisor 1	ángulo 1	Transmisor 2	ángulo 2	Transmisor 1	ángulo 1	Transmisor 2	ángulo 2
[-25, 55]	295	[-12, -58]	282	[70, 70]	45	[-70, 70]	135
[20, -17]	320	[-41, 46]	132	[-70, 70]	135	[70, 70]	45
[34, 36]	46	[41, -1]	359	[-70, -70]	225	[70, -70]	315
[1, 57]	88	[-40, 45]	133	[70, -70]	315	[-70, -70]	225
[47, 4]	4	[12, -19]	303	[70, 0]	0	[70, 70]	45
[41, -17]	338	[58, -3]	358	[-70, 0]	180	[70, 70]	45
[-38, -33]	220	[-44, -26]	210	[43, 74.6]	60	[70, 70]	45
[33, 59]	60	[-31, 45]	125	[-43, 74.6]	120	[70, 70]	45
[6, -48]	278	[-27, -56]	244	[43, -74.6]	300	[70, 70]	45
[-8, -47]	260	[-43, 29]	147	[-43, -74.6]	240	[-70, 70]	135
[51, -34]	330	[53, -55]	314	[80, 47]	30	[-70, 70]	135
[-38, -22]	210	[38, -58]	304	[-80, 47]	150	[70, 70]	45
[36, -59]	302	[40, -19.]	335	[80, -47]	330	[70, 70]	45
[47, -19]	338	[-41, -36]	221	[-80, -47]	210	[-70, 70]	135

Cuadro 1 Posiciones y ángulos de los escenarios de entrenamiento y de evaluación.

Estas mismas posiciones y ángulos se pueden ver reflejados de una forma gráfica en la siguiente figura:



(a) Escenarios de entrenamiento. (b) Escenarios aleatorios para la evaluacion.

Para estos nuevos escenarios, donde las posiciones fueron generadas de una forma aleatoria, se consiguió un porcentaje de acierto superior al 71 %.

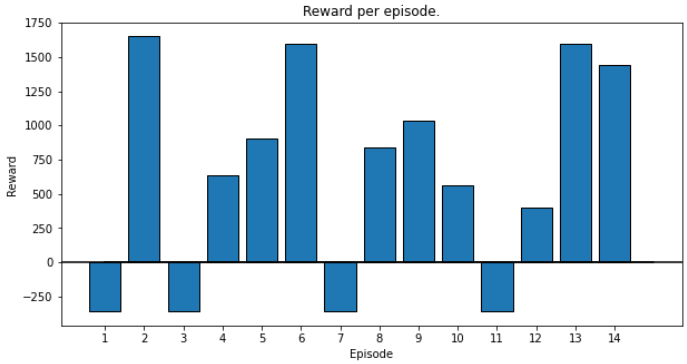


Figura 43 Reward por episodio de evaluación.

Como se puede ver en la figura 43, los escenarios 1, 3, 7 y 11 son los únicos que no se pudo resolver, obteniendo un reward negativo en cada uno de ellos. Los escenarios para los cuales no se ha cumplido el objetivo son los siguientes:

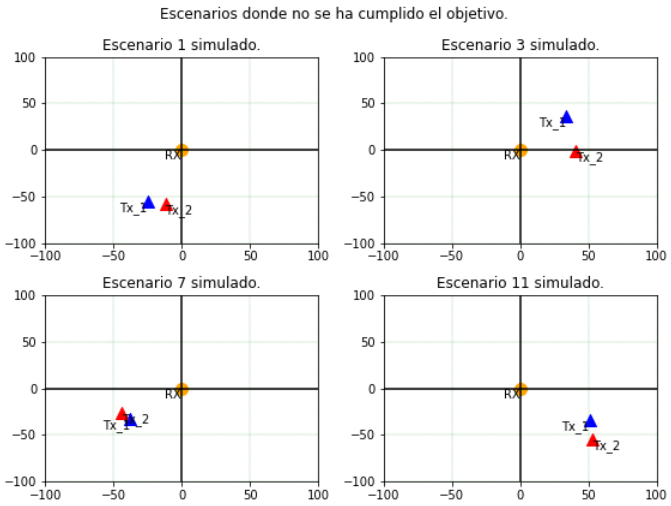


Figura 44 Escenarios de evaluación no cumplidos.

Analizando estos escenarios se puede ver que, en cada uno de ellos, los transmisores se encuentran a una distancia muy cercana entre sí, y con los diagramas de radiación propuestos, es difícil conseguir una diferenciación correcta entre las señales transmitidas. Estos se solventaría aumentando el número de elementos que componen la antena tipo ULA, dando así una mayor directividad a los diagramas de radiación. Al igual que se hizo anteriormente, para este caso también se ha realizado un video que permita visualizar la actuación del modelo entrenado dentro de estos nuevos escenarios. Este video está disponible a través del link [20].

9 Plan de gestión.

En este apartado se describe el plan de gestión realizado para llevar a cabo la elaboración del proyecto. En él se describen las fases en las que se distribuye el proyecto, y dentro de cada una, se explican las tareas a realizar, agrupadas en paquetes de trabajo.

9.1 Descripción de los paquetes de trabajo.

Las siguientes tablas describen los paquetes de trabajo que se realiza en cada una de las fases. En cada una, se hace una breve descripción del paquete de trabajo, y se enumeran las tareas a realizar junto con los plazos previstos.

9.1.1 Fase completa del proyecto.

Este paquete de trabajo refleja las tareas y el plazo en el que deben ser realizadas, tanto por los coordinadores del proyecto, como por el alumno.

PT1	Fecha de inicio.	Fecha de finalización.	Duración
Gestión del proyecto: Monitorización y administración necesaria para asegurar el cumplimiento de los objetivos marcados.	02-05-2022	10-10-2022	107
PT1.1: Gestión, monitorización y supervisión de trabajo: Coordinación, supervisión y monitorización del proyecto, desde el inicio hasta la finalización del mismo.	02-05-2022	10-10-2022	107

Cuadro 2 Paquete de trabajo Nº: 1.

9.1.2 Fase 1 del proyecto.

En esta fase inicial, se adquieren los conocimientos iniciales y se define la planificación a seguir.

PT2	Fecha de inicio.	Fecha de finalización.	Duración
Preparación del proyecto: Adquisición del conocimiento necesario previo al desarrollo de la solución planteada.	02-05-2022	31-05-2022	22
PT2.1: Conocimiento previo: Lectura de artículos relacionados con los escenarios IDL/ITCN. Estudio de los algoritmos de Reinforcement Learning.	02-05-2022	13-05-2022	10
PT2.2: Definición del proyecto: Descripción del ámbito del proyecto y el flujo de trabajo a seguir.	16-05-2022	20-05-2022	5
PT2.3: Estado del arte: Búsqueda de información en forma de estudios, proyectos y publicaciones relacionados con el proyecto.	23-05-2022	31-05-2022	7

Cuadro 3 Paquete de trabajo Nº: 2.

9.1.3 Fase 2 del proyecto.

En esta fase se centra todo el desarrollo del código, tanto para la obtención de los datos con los que se alimentara el algoritmo, como para su entrenamiento.

PT3	Fecha de inicio.	Fecha de finalización.	Duración
Definición del entorno de simulacion: Matlab y Quadriga. Creación de un escenario que simule él problema de auto-interferencia que se da en los escenarios IDL/ITCN.	01-06-2022	24-06-2022	18
PT3.1: Conocimiento previo: Lectura sobre la documentación de Quadriga, y elaboración de varios ejemplos para su aprendizaje.	01-06-2022	10-06-2022	8
PT3.2: Definición del escenario: Definición del escenario equivalente que simulará el problema de auto-interferencia.	13-05-2022	17-05-2022	5
PT3.3: Pruebas de validación: Realizar varias pruebas utilizando los diferentes espacios que proporciona Quadriga, como LOSonly, freeSpace y 3GPT. Creación de las primeras gráficas de los canales obtenidos con Quadriga.	20-06-2022	24-06-2022	5

Cuadro 4 Paquete de trabajo Nº: 3.

PT4	Fecha de inicio.	Fecha de finalización.	Duración
Diseño del algoritmo Deep Q-Network. Definición, pruebas y primeros ajustes de los parámetros e hiperparámetros del algoritmo DQN.	27-06-2022	10-08-2022	33
PT4.1: Preparación de los datos: Análisis, preprocesamiento y partición de los datos obtenidos con Quadriga. Definir los datos con los que se va a alimentar la red.	27-06-2022	01-07-2022	5
PT4.2: Definición de los actores y función recompensa: Definición del agente, entorno, acciones y rewards de cada estado.	04-07-2022	08-07-2022	5
PT4.3: Elaboración del código dentro de Spyder. Crear el código referente a la replay memory, red neuronal, estrategia épsilon y demás componentes del algoritmo.	11-07-2022	22-07-2022	10
PT4.4:Pruebas de validación: Comprobar que el modelo funciona correctamente, y realizar los ajustes correspondientes en cada hiperparámetro.	25-07-2022	10-08-2022	13

Cuadro 5 Paquete de trabajo Nº: 4.

9.1.4 Fase 3 del proyecto.

La fase final se centra en documentar lo realizado y los resultados obtenidos, así como, la defensa del proyecto frente a un tribunal.

PT5	Fecha de inicio.	Fecha de finalización.	Duración
Documentación y presentación del proyecto: Escritura de la memoria del proyecto y Presentación oral.	10-08-2022	10-10-2022	44
PT5.1: Documentación del proyecto: Elaboración del documento que define el contexto del proyecto, objetivos, beneficios, metodología, descripción de la solución y conclusiones.	10-08-2022	18-09-2022	34
PT5.2: Presentación del proyecto: Elaboración y ejecución de la presentación del proyecto frente al tribunal.	26-09-2022	10-10-2022	10

Cuadro 6 Paquete de trabajo Nº: 5.

9.2 Diagrama de Gantt.

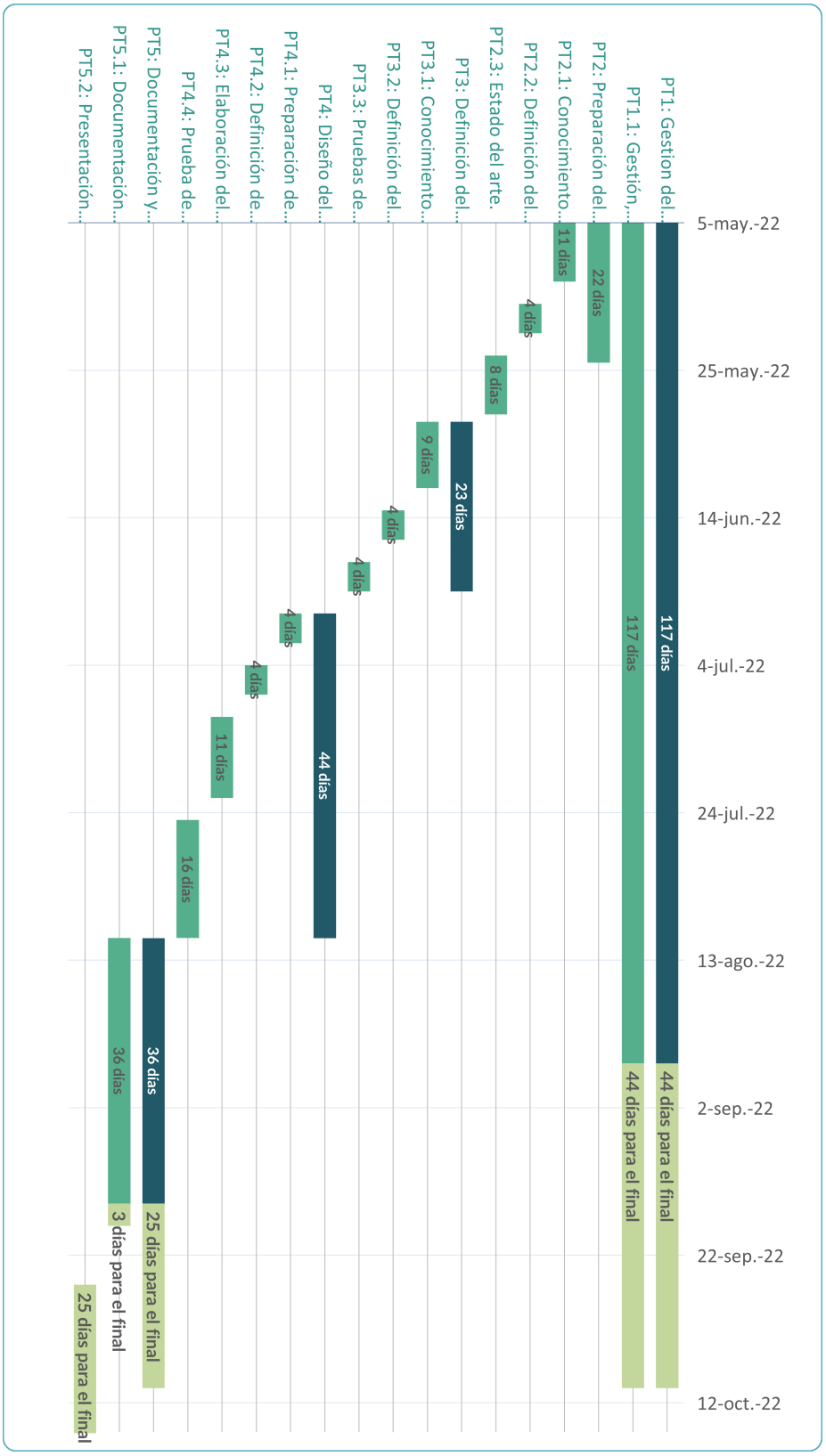


Figura 45 Diagrama de Gantt del proyecto.

10 Conclusiones.

Los resultados obtenidos hasta este momento demuestran que, el objetivo marcado al inicio de este proyecto, ha sido conseguido. Como ya se ha visto, este modelo Beamforming, basado en un algoritmo Deep Q-Nerwork, es una buena opción para minimizar el problema de auto-interferencia existente en una comunicación IBFD, pudiendo así, ser utilizado dentro de las comunicaciones IBFD realizadas en los entornos ICTN/IDL. Para ello, hay que tener en cuenta que, los datos de entrenamiento deberían ser los adecuados para esa situación, ya que en este caso, los datos utilizados, pertenecen a un escenario equivalente al real.

En cuanto al escenario propuesto en el apartado 7.3.1, como ya se ha visto, no fue posible llevar a cabo la auto-generación del diagrama de radiación, sin embargo, no significa que no sea posible. El problema de no haber llegado a una solución coherente, puede deberse a múltiples motivos, como el incorrecto ajuste de los hiperparámetros, o una mala definición de las acciones a realizar por parte del agente.

En la definición de las acciones, se definieron 2 posibles acciones para cada elemento del array, más una acción común para todos los elementos. Esto hace que el número de acciones a elegir por parte del agente, dependa directamente del número de elementos que componen la antena tipo ULA, convirtiendo al número de antenas, en un hiperparámetro a tener en cuenta. En este proyecto, se decidió usar 4 antenas, lo que hizo un total de 9 posibles acciones, sin embargo, cabe la posibilidad de que usando más antenas, y reajustando los hiperparámetros, se pueda llegar a una mejor solución. Aun así, se debe tener en cuenta que, cuantas más antenas se definan, más acciones deberá aprender a seleccionar el agente, haciendo que la tarea a resolver sea más compleja.

Por otra parte, las acciones definidas consistían en incrementar o decrementar el valor de las fases ϕ, correspondientes a cada elemento de la antena, sumando o restando un valor fijo a dicha fase. Este valor fijo se dio como otro hiperparámetro que debía ser ajustado, ya que en función de su valor, se podían generar diferentes diagramas de radiación. En este caso, quizás, la forma en la que se generaban o seleccionaban las fases ϕ, de cada elemento de array, podía ser el causante de un mal aprendizaje, ya que los diagramas generados con estas fases, probablemente no podían resolver los escenarios propuestos.

En definitiva, este primer planteamiento queda definido como un futuro trabajo a resolver, teniendo en cuenta las conclusiones obtenidas en este proyecto. Así mismo, queda pendiente la comparación de este modelo basado en Deep Q-Network, con

otro modelo iterativo tradicional para el sistema Beamforming que permita afirmar que, este tipo de algoritmos son computacionalmente más eficientes que los métodos tradicionales.

Bibliografía

[1] W. L. L. Zhang Y. Wu, S.-I. Park, J.-y. L. andH.-M. Kim y col., «ATSC 3.0 In-band Backhaul for SFN Using LDM with Full Backward Compatibility,» 2022. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8971918.

[2] L. Z. et. all, «Layered-Division-Multiplexing: Theory and Practice,» 2016. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7378924.

[3] L. Z. et al., «Using Layered Division Multiplexing for Wireless In-Band Distribution Links in Next Generation Broadcast Systems,» 2021. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9093864.

[4] W. L. et al., «Integrated Inter-Tower Wireless Communications Network for Terrestrial Broadcasting and Multicasting Systems,» 2021. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9444117.

[5] E. I. Iñigo Bilbao y J. Montalban, «AI-based Inter-Tower Communication Networks: First approach,» 2022. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9828767.

[6] J. H. L. Hyung Jun Kwon y W. Choi, «Machine Learning-Based Beamforming in Two-User MISO Interference Channels,» 2019. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8669027&tag=1.

[7] Q. S. Haoran Sun Xiangyi Chen, M. Hong, X. Fu y N. D. Sidiropoulos, «Learning to Optimize: Training Deep Neural Networks for Interference Management,» 2018. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8444648.

[8] H. Jia, Z.-Q. He, H. Rui y W. Lin†, «Robust Distributed MISO Beamforming Using Multi-Agent Deep Reinforcement Learning,» 2022. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9817604&tag=1.

[9] M. B. Mahdi Fozi Ahmad R. Sharafat, «Fast MIMO Beamforming via Deep Reinforcement Learning for High Mobility mmWave Connectivity,» 2022. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9612729.

[10] J. M. Iñigo Bilbao Eneko Iradier y P. Angueira, «AI-based Inter-Tower Communication Networks: Challenges and Benefits,» 2022. dirección: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9547159.

[11] Y. A. (John), *Fundamentals of a Uniform Linear Array (ULA)*, 2018. dirección: https://www.raymaps.com/index.php/fundamentals-of-a-uniform-linear-array-ula/.

[12] N. Almskou, «RL-for-Adaptive-Beamforming,» GitHub, inf. téc., 2022. dirección: https://github.com/Almskou/RL-for-Adaptive-Beamforming.

[13] Q. Team, *Quadriga Documentation*, 2018. dirección: https://quadriga-channel-model.de.

[14] V. M. et al., «Human-level control through deep reinforcement learning,» 2015. dirección: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15Natu
pdf.

[15] M. A. et al., *Introduction to RL and Deep Q Networks*, 2018. dirección: https://www.tensorflow.org/agents/tutorials/0_intro_rl.

[16] EMCoS, *Simulation of 28 GHz Series-Fed Patch Antenna Array for 5G Applications*, 2018. dirección: https://www.emcos.com/?application-examples=simulation-of-28-ghz-series-fed-patch-antenna-array-for-5g-applications.

[17] J. Achiam, «Introduction to RL,» Open AI, inf. téc., 2018. dirección: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[18] Q. Larson, «An introduction to Q-Learning: reinforcement learning,» freeCodeCamp, inf. téc., 2018. dirección: https://www.freecodecamp.org/news/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc/.

[19] D. Chuga, *Video representativo de un modelo beamforming basado en RL para comunicaciones IBFD*, 2022. dirección: https://youtu.be/yFGav_xrp3c.

[20] ——, *Video representativo de la evaluación de un modelo beamforming basado en RL para comunicaciones IBFD*, 2022. dirección: https://youtu.be/yFGav_xrp3c.