Universidad de Valladolid

Escuela de Ingeniería Informática

**TRABAJO FIN DE GRADO**

Grado en Ingeniería Informática
Mención en Ingenieria del Software

# Algoritmo de control mediante RL para neuroestimuladores inteligentes


Alumno:
**Dña.Marina Yagüe González**

Tutores:
**Dña.Mª Aránzazu Simón Hurtado**
**D.Ángel Canal Alonso**

# Agradecimientos

Me gustaría expresar mi agradecimiento a todas las personas que me han ayudado en la realización de este Trabajo de Fin de Grado.

En primer lugar, a mis tutores, Mª Aránzazu Simón Hurtado y Ángel Canal Alonso, por vuestra ayuda y dedicación. A Jesús María Vegas Hernández, por todos sus consejos sobre el desarrollo de la planificación.

Al Air Institute por brindarme la oportunidad de trabajar en este proyecto.

A mis padres y a mi familia por su apoyo y paciencia durante toda la carrera.

# Resumen

El creciente avance de las tecnologías sanitarias ha supuesto la utilización de nuevos métodos de tratamiento para las enfermedades del sistema nervioso, como pueden ser el Parkinson o la epilepsia. Estos trastornos afectan gravemente a la calidad de vida de los pacientes, ya que producen de forma incremental temblores, rigidez muscular y dificultad para moverse.

Con la finalidad de reducir estos síntomas, se ha desarrollado una técnica neuroquirúrgica llamada Estimulación cerebral profunda. Este método consiste en enviar, mediante unos electrodos implantados en el cerebro, pulsos eléctricos a las zonas causantes de los temblores y convulsiones, reduciendo así su frecuencia e intensidad a largo plazo. Sin embargo, en la actualidad, estos tratamientos no se adaptan de forma dinámica al estado del paciente, ya que los parámetros del dispositivo de control deben ser ajustados manualmente por los sanitarios.

En este Trabajo de Fin de Grado se ha desarrollado un algoritmo de control, mediante inteligencia artificial y aprendizaje por refuerzo, que regula de forma automática los parámetros de frecuencia e intensidad de los pulsos eléctricos, adaptándose a la evolución del paciente y a las señales cerebrales enviadas en cada instante de tiempo.

# Abstract

The increasing progress in healthcare technologies has led to the use of new treatment methods for diseases of the nervous system, such as Parkinson's disease and epilepsy. These disorders severely affect the quality of life of patients, as they cause incremental tremors, muscle stiffness and difficulty in moving.

In order to reduce these symptoms, a neurosurgical technique called Deep Brain Stimulation has been developed. This method consists of sending, by means of electrodes implanted in the brain, electrical pulses to the areas causing tremors and convulsions, thus reducing their frequency and intensity in the long term. However, these treatments are not dynamically adapted to the patient's condition, since the parameters of the control device must be manually adjusted by the health care professionals.

In this Bachelor's Thesis, a control algorithm has been developed using artificial intelligence and reinforcement learning, which automatically regulates the frequency and intensity parameters of the electrical pulses, adapting to the evolution of the patient and the brain signals sent at each instant of time.

# Índice general

# Lista de Figuras

# Lista de Tablas

# Capítulo 1

# Introducción

## 1.1. Contexto

Las enfermedades neuronales, como son la epilepsia o el Párkinson causan la aparición de convulsiones, temblores y rigidez muscular en las personas que las padecen, afectando gravemente a su calidad de vida.

Con el objetivo de tratar estos transtornos de forma efectiva se utilizan una técnicas neuroquirurgicas como la Estimulación Cerebral Profunda o Deep Brain Stimulation (DBS). Este procedimiento consiste en la implantación de electrodos en áreas especificas del cerebro, el cual envía pulsos electromagnéticos a las zonas del cerebro que originan las convulsiones, haciendo que estas se reduzcan en número y en intensidad.

Sin embargo, actualmente los patrones de estimulación utilizados en los dispositivos de DBS son programados por el personal clínico basándose en una evaluación cualitativa de los pacientes, la cual puede tener una larga duración y supone que los pacientes se vean expuestos de forma continua a frecuencias altas que pueden no ser necesarias para ellos[16]. Por lo tanto, estos dispositivos son incapaces de adaptarse a las necesidades de cada paciente a tiempo real.

Debido a esto, en este Trabajo de fin de Grado (TFG) se propone un algoritmo de control basado en inteligencia artificial y entrenado con aprendizaje por refuerzo para controlar los patrones de estimulación del dispositivo en base a las señales cerebrales recibidas y el estado subyacente del sistema nervioso, con el objetivo de ajustar las señales enviadas al caso de cada paciente.

## 1.2. Objetivos

A continuación, se describe el objetivo general del proyecto y cada uno de los objetivos específicos en los que se divide.

### Objetivo General

El objetivo general de este TFG es desarrollar un algoritmo de control para dispositivos de Estimulación cerebral profunda mediante la inteligencia artificial y el aprendizaje por refuerzo que modifique las señales electromagnéticas enviadas en función de las condiciones del paciente.

### Objetivos Específicos

Para alcanzar el objetivo general, este ha sido dividido en una serie de objetivos específicos que se desarrollarán a lo largo del proyecto. Estos hitos se definen de la siguiente forma:

1. **Investigación sobre Dispositivos de Estimulación Cerebral Profunda**: Entender el funcionamiento de los dispositivos especializados en Estimulación Cerebral profunda y el tratamiento de las enfermedades neuronales, analizando sus motivos y algoritmos de control utilizados.
2. **Investigación sobre Arquitecturas de aprendizaje por refuerzo**: Comprensión de las estructuras utilizadas en los modelos de aprendizaje de cada tipo de dispositivo, analizando sus módulos principales, acciones, estados y parametrización del entorno en el que se encuentran.
3. **Aprendizaje del uso de frameworks de aprendizaje por refuerzo**: Documentación sobre el uso librerías de Python utilizadas en el desarrollo, como son Keras y TensorFlow.
4. **Tratamiento de los datos encefalográficos**: Procesamiento de los datos aportados para un correcto entrenamiento del modelo de aprendizaje por refuerzo.
5. **Codificación correcta del modelo**: Programación del modelo debe ser acorde a los requisitos del proyecto.
6. **Análisis de rendimiento del modelo**: Análisis de los resultados obtenidos en el entrenamiento del modelo.

## 1.3. Estructura de la memoria

Este documento se estructura de la siguiente forma:

**Capítulo 1 Introducción**: Descripción del contexto en el que se encuentra el proyecto y explicación de los objetivos principales

**Capítulo 2 Metodología, planificación y presupuesto**: Exposición de los riesgos a los que se somete el proyecto, así como las metodologías de planificación del proyecto utilizadas y la estimación de los presupuestos reales y esperados.

**Capítulo 3 Aprendizaje por refuerzo y redes neuronales:** Explicación y definición de los conceptos fundamentales en los que se basa el aprendizaje por refuerzo de inteligencias artificiales, así como las ideas fundamentales de la estructura de las redes neuronales utilizadas en el proyecto.

**Capítulo 4 Estado del arte:** Exposición del desarrollo actual del campo de la estimulación cerebral profunda en conjunto con el aprendizaje por refuerzo

**Capítulo 5 Conjunto de datos encefalográficos:** Descripción de las técnicas de procesamiento de datos de señales cerebrales.

**Capítulo 6 Análisis:** Descripción y análisis de los elementos que forman el sistema desarrollado así como de las relaciones que hay entre ellos.

**Capítulo 7 Diseño:** Descripción de la estructura del sistema de acuerdo con las decisiones de diseño elegidas.

**Capítulo 8 Tecnologías utilizadas**: Descripción de las herramientas utilizadas durante la etapa de codificación del proyecto.

**Capítulo 9 Diseño experimental:** Exposición de la configuración de los experimentos realizados.

**Capítulo 10 Resultados y discusión:** Estudio en profundidad de los resultados obtenidos.

**Capítulo 11 Conclusiones y Trabajo futuro:** Recapitulación de todas las ideas presentadas en el documento.

# Capítulo 2

# Metodología, planificación y presupuesto

## 2.1. Metodología

Debido a las características del proyecto, se han elegido dos tipos de metodologías diferentes para su desarrollo, las cuales son el modelo en cascada, para el avance general del proyecto, y el modelo evolutivo para la sección de codificación y aprendizaje del modelo. A continuación, se presentan las propiedades fundamentales de cada metodología y cómo han sido adaptadas a las necesidades del trabajo.

### 2.1.1. Metodología en cascada

La metodología en cascada es un método de gestión de proyectos de software clásico y lineal, caracterizado por dividir el proyecto en una serie secuencial de actividades que deben ser completadas en orden descendente, simulando la estructura de una cascada. De esta forma, cada tarea es finalizada completamente antes de que se pueda empezar con la siguiente, evitando así el gran coste que supone realizar cambios sobre actividades anteriores en fases más avanzadas del proyecto.

Por lo tanto, uno de los mayores atributos de este modelo es la necesidad de contar con unos requisitos de proyecto claros y bien interpretados, con el objetivo de prevenir modificaciones durante el desarrollo y aportar confianza y control sobre su avance. Además, este modelo garantiza la creación de etapas[17] en las que el trabajo realizado hasta el momento puede ser evaluado por los clientes, los cuales aportan sus impresiones y especifican si es necesario realizar cambios antes de que termine la fase actual del proyecto.

Las fases en las que se divide un proyecto con metodología en cascada son las siguientes:

1. **Requisitos**: Definición y documentación completa de los requisitos de software para el desarrollo del proyecto, siendo estos los requisitos funcionales y no funcionales. Esta etapa es una de las más importantes en la metodología por cascada, ya que se especifican todas las funciones que debe ser capaz de llevar a cabo el sistema al final del proyecto, determinando. Por lo tanto, los objetivos finales del proyecto, no deben ser modificados durante el desarrollo. Así mismo, también se realiza la identificación de los usuarios finales.

2. **Diseño**: Especificación de la estructura del sistema que se va a utilizar, dividiendo el producto en módulos, indicando su funcionamiento y cómo se relacionan entre ellos en la arquitectura global. De igual forma, se determinan las herramientas de codificación que serán empleadas, así como los algoritmos necesarios para cumplir con los requisitos elegidos en la fase anterior.

3. **Implementación**: Fase de programación del código que cumple con las especificaciones definidas durante el diseño.

4. **Pruebas**: Prueba del código creado en la fase anterior con el objetivo de detectar errores y asegurarse que el resultado cumple con los requisitos del cliente.

5. **Implantación**: Instalación del software en el sistema del cliente.

6. **Mantenimiento**: Conservación del software mediante el desarrollo de actualizaciones, corrigiendo errores de funcionamiento y creando mejoras que garanticen el funcionamiento del sistema y la satisfacción del cliente.

#### Integración del modelo cascada en el proyecto

A la hora de aplicar esta metodología al proyecto ha sido necesario realizar una serie de cambios a las fases generales del modelo, ya que, debido a que el proyecto es fundamentalmente de investigación, no tiene como objetivo desarrollar un producto comercial, por lo que se añade una fase de redacción de la memoria, y se eliminan la implantación y el mantenimiento. En cuanto al resto de fases siguen la estructura en cascada, incluyendo reuniones con los tutores durante cada una de las fases. Se detallan de la siguiente forma :

1. **Requisitos**: Esta fase se compone de una sección de búsqueda de información sobre aprendizaje por refuerzo en dispositivos médicos y sus arquitecturas. Después se definen los requisitos funcionales del proyecto, realizando un análisis funcional y estableciendo sus objetivos. Finalmente, se describe en detalle la arquitectura elegida.

2. **Diseño**: Diseño de la arquitectura utilizada en el aprendizaje automático, definición de las características del agente y planificación de la ingestión de datos del modelo.

3. **Implementación y Pruebas**: Codificación y entrenamiento del modelo, así como los correspondientes ajustes de parámetros de aprendizaje. La fase de pruebas se corresponde a cada uno de las iteraciones que realiza el agente durante el aprendizaje.

4. **Documentación**: Redacción de la memoria del proyecto.

Es necesario destacar que la metodología en cascada solo es utilizada para la planificación general del proyecto, ya que para la fase de codificación y pruebas se ha optado por una metodología evolutiva, ya que es mucho más flexible a los cambios.

### 2.1.2. Metodología evolutiva

La metodología evolutiva consiste en crear de forma iterativa un prototipo[18] que tenga la misma funcionalidad que el proyecto que se quiere desarrollar. Con cada iteración, el prototipo se desarrolla y modifica hasta que este puede realizar todos los servicios especificados en los requisitos, momento en el que el prototipo se convierte en el sistema final. Esto es posible, ya que en cada fase se anota la retroalimentación ofrecida por los usuarios del prototipo actual y se modifica de acorde a sus necesidades, garantizado que se cumplen todos los objetivos acordados y la satisfacción del cliente.

Por lo tanto, el modelo evolutivo resulta útil en proyectos de software en los que hay cierta incertidumbre sobre los requisitos, ya que si es necesario realizar cambios sobre el trabajo hecho hasta el momento, se pueden hacer en la siguiente iteración del proyecto sin causar costes elevados.

Las fases de las que consta esta metodología en cada iteración son:

1. **Requisitos**: Definición de los requisitos funcionales que debe cumplir el prototipo en la iteración actual.
2. **Diseño** : Diseño del prototipo según los requisitos de la fase anterior
3. **Construcción**: Codificación del prototipo.
4. **Evaluación**: Evaluación de las funcionalidades del prototipo actual por los usuarios finales y documentación de la retroalimentación obtenida.
5. **Modificación**: Modificación del prototipo teniendo como base las respuestas de los usuarios.

#### Integración del modelo evolutivo en el proyecto

Se ha decidido adaptar las fases de codificación y pruebas del modelo utilizando la metodología evolutiva, ya que, es necesario modificar constantemente los parámetros iniciales en cada iteración del aprendizaje del agente, dado que se tiene como objetivo optimizar los resultados obtenidos. De esta forma, se crea en primera instancia un modelo de funcionalidad sencilla que evoluciona constantemente en cada fase, volviéndose más complejo con el tiempo a medida que se obtienen resultados cada vez más positivos, hasta que cumple con todas las funcionalidades del producto final.

Por lo tanto, cada iteración se compone de las siguientes fases :

1. **Requisitos**: Definición de los objetivos que el modelo debe cumplir al terminar la prueba.

2. **Diseño de los parámetros de la prueba**: Se establecen los parámetros de ejecución con los que se va a realizar el entrenamiento del modelo, modificando su estructura y ratio de aprendizaje.

3. **Entrenamiento y validación del modelo**: Desarrollo del aprendizaje ejecutando el código en un servidor externo.

4. **Retroalimentación**: Análisis de los resultados obtenidos. Estos se comparan con los de las fases anteriores, esperando una mejora en la efectividad del agente.

5. **Modificación**: Codificación de cambios en el prototipo siguiendo la retroalimentación de la fase anterior

## 2.2. Planificación

Con el objetivo de asegurar que el alcance del proyecto esté claramente definido y para garantizar que sus objetivos sean conseguidos dentro del tiempo estimado, se ha descompuesto el proyecto en grupos de actividades correspondientes a las fases de documentación, desarrollo del producto y redacción de la memoria.

En primer lugar, la fase de documentación se basa en entender en profundidad la estructura y el funcionamiento de las principales arquitecturas utilizadas en proyectos de aprendizaje por refuerzo, para después, aplicar dichos conocimientos en la fase de desarrollo, a la hora de codificar el modelo y de cambiar los parámetros de ejecución. Por ultimo, todo avance del proyecto queda registrado en la escritura de la memoria.

Así mismo, al inicio del proyecto se asignaron a las actividades individuales una duración aproximada dependiendo de su complejidad y de los conocimientos previos que se poseen sobre el tema. Se debe tener en cuenta que el tiempo empleado en las tareas puede ser modificado a lo largo del desarrollo debido a errores o imprevistos,como puede ser, debido a la obtención de resultados inesperados durante la fase de entrenamiento del modelo. Además, se ha considerado que se emplea una media de dos horas y media diarias en el avance del proyecto.

Para una mejor visualización de la planificación del proyecto, se ha realizado el diagrama de Gantt dividido por fases, que se muestra al final del capítulo. (Figuras 2.4,2.5,2.6)

Cabe destacar que durante la fase de desarrollo se han planificado una serie de experimentos que se realizan de forma iterativa. En ellos se modifican los parámetros de entrenamiento del modelo para observar los efectos que tienen los valores de entrada en su rendimiento. Se realizan un total de seis experimentos por parámetro del modelo, los cuales tienen las siguientes fases:

**Requisitos**: Especificación de los valores que deben alcanzar las métricas de rendimiento resultantes después de la ejecución del modelo. Estas métricas son la recompensa acumulada, el éxito acumulado y la velocidad de aprendizaje.

**Ajuste de los parámetros de la prueba**: Modificación de la magnitud del parámetro estudiado, dejando el resto en su valores iniciales. Los parámetros del modelo son: la tasa de aprendizaje, la tasa de descuento, el número de capas, el número de neuronas en cada capa y el tipo de capa utilizado.

**Entrenamiento y validación del modelo**: Entrenamiento y validación del modelo en los servidores del centro de supercomputación SCAYLE[19].

**Retroalimentación**: Análisis de los resultados obtenidos. Se estudian las métricas resultantes comparándolas con el rendimiento que se esperaba al comienzo del experimento, y con el resto de iteraciones anteriores para el parámetro actual.

**Modificación**: Evaluación y ajuste del valor del parámetro teniendo en cuenta el análisis de los resultados obtenidos en la fase anterior.

## 2.3. Riesgos

Se ha realizado un análisis de los riesgos que pueden suceder durante el inicio, desarrollo y finalización del proyecto. Según la guía de los fundamentos para la dirección de proyectos (PMBOK) un riesgo se define como un evento o situación incierta, que si llega a ocurrir, puede tener efectos positivos o negativos en los objetivos del proyecto. De esta forma, se identifican como riesgos, tanto a las amenazas que perjudican a la finalización del proyecto, como a las oportunidades que ofrecen un mejor alcance de los requisitos establecidos.

Para cada riesgo se determina la probabilidad de que estos hechos ocurran y el grado de impacto en el desarrollo del proyecto. A estos factores se les asigna un nivel de importancia: alto, medio o bajo, con los que después se calcula el nivel de riesgo total, siguiendo la Tabla 2.1. Los riesgos identificados en el proyecto se muestran desde la Tabla 2.2 a la Tabla 2.17.

Tabla 2.1: Matriz de nivel de riesgo según su probabilidad e impacto

| **Prob/Imp** | Baja | Media | Alta |
|---|---|---|---|
| **Bajo** | Bajo | Bajo | Medio |
| **Medio** | Bajo | Medio | Alto |
| **Alto** | Medio | Alto | Alto |

Así mismo, se aporta una serie de acciones, tanto para reducir los efectos negativos de cada riesgo una vez han sucedido, acciones correctivas, como para reducir su probabilidad de aparición, acciones de mitigación. Los riesgos de categoría personal son aquellos que son causados o afectan en gran medida a los participantes del proyecto, mientras que los riesgos de hardware y software tienen como desencadenante herramientas físicas y digitales respectivamente.

Tabla 2.2: Riesgo 1: Enfermedad de los autores del proyecto

| | |
|---|---|
| **Riesgo 1** | Enfermedad |
| **Descripción** | Algunos de los participantes en el proyecto pueden enfermar durante su realización, lo cual causaría retrasos en la planificación acordada. |
| **Categoría** | Personal |
| **Probabilidad** | Media |
| **Impacto** | Medio |
| **Riesgo Total** | Medio |
| **Mitigación** | Realizar las reuniones de forma virtual. |
| **Acciones Correctivas** | Evaluar y ajustar la planificación del proyecto de acuerdo con los posibles retrasos en la finalización de los objetivos del proyecto |

Tabla 2.3: Riesgo 2: Falta de comprensión de las arquitecturas utilizadas

| | |
|---|---|
| **Riesgo 2** | Falta de comprensión de las arquitecturas utilizadas |
| **Descripción** | Debido a la necesidad de obtener nuevos conocimientos sobre arquitecturas utilizadas en aprendizaje por refuerzo, es posible que se den retrasos en la planificación del proyecto, ya que se necesita más tiempo para comprender los conceptos en profundidad. |
| **Categoría** | Planificación |
| **Probabilidad** | Alta |
| **Impacto** | Alto |
| **Riesgo Total** | Alto |
| **Mitigación** | Informarse sobre las arquitecturas utilizando la documentación que ha sido aportada por los tutores |
| **Acciones Correctivas** | Realizar reuniones con los tutores en las que preguntar todas las dudas relacionadas con el modelo |

Tabla 2.4: Riesgo 3: Falta de experiencia con el lenguaje utilizado

| | |
|---|---|
| **Riesgo 3** | Falta de experiencia con el framework utilizado |
| **Descripción** | En el proyecto se utilizan las librerías Numpy y TensorFlow, con las cuales no se tiene experiencia previa. Por lo tanto, será necesario aprender desde el principio su funcionamiento, invirtiendo un mayor tiempo de documentación de lo que había sido estimado. |
| **Categoría** | Planificación |
| **Probabilidad** | Media |
| **Impacto** | Medio |
| **Riesgo Total** | Medio |
| **Mitigación** | Consultar la documentación sobre el uso de las librerías antes de comenzar con la codificación |
| **Acciones Correctivas** | Informarse sobre las funciones definidas en las librerías y realizar pequeños ejemplos para familiarizarse con su funcionamiento. Planificar la fase de codificación aumentando su tiempo estimado. |

Tabla 2.5: Riesgo 4: Inactividad del servicio de computación utilizado

| | |
|---|---|
| **Riesgo 4** | Inactividad del servicio de computación utilizado |
| **Descripción** | El servicio utilizado para compilar y ejecutar el código fuente no se encuentra disponible durante el periodo de codificación del proyecto, por lo que es imposible comprobar si el programa es correcto y obtener los resultados generados. |
| **Categoría** | Hardware |
| **Probabilidad** | Baja |
| **Impacto** | Alto |
| **Riesgo Total** | Medio |
| **Mitigación** | Confirmar que el servidor tiene la suficiente capacidad como para ejecutar la carga prevista. |
| **Acciones Correctivas** | Esperar a que los servidores vuelvan a estar operativos, y ejecutar pruebas de menor intensidad en la máquina local. |

Tabla 2.6: Riesgo 5: Pérdida de la base de datos utilizada

| | |
|---|---|
| **Riesgo 5** | Pérdida de la base de datos utilizada |
| **Descripción** | Eliminación accidental de los datos utilizados para el aprendizaje del modelo. Como consecuencia no se podrá continuar el entrenamiento del modelo, ralentizando el avance del proyecto. |
| **Categoria** | Software |
| **Probabilidad** | Baja |
| **Impacto** | Alto |
| **Riesgo Total** | Medio |
| **Mitigación** | Almacenar la base de datos en un servicio remoto, con el objetivo de no depender únicamente de la copia guardada en local y garantizar que la información sea fácilmente recuperable en un futuro. |
| **Acciones Correctivas** | Solicitar al tutor los datos perdidos. |

Tabla 2.7: Riesgo 6: Inactividad de los servidores de Overleaf

| | |
|---|---|
| **Riesgo 6** | Inactividad de los servidores de Overleaf |
| **Descripción** | Caída de los servidores de Overleaf en los que se guarda y escribe la documentación del proyecto. |
| **Categoría** | Software |
| **Probabilidad** | Media |
| **Impacto** | Alto |
| **Riesgo Total** | Alta |
| **Mitigación** | Realizar copias de seguridad del documento en la máquina local cada vez que se haya hecho un cambio en la memoria. |
| **Acciones Correctivas** | Esperar a que los servidores vuelvan a estar operativos y escribir la información que se iba a añadir en otro editor de texto. |

Tabla 2.8: Riesgo 7: Pérdida del código fuente

| | |
|---|---|
| **Riesgo 7** | Pérdida del código fuente |
| **Descripción** | Pérdida del programa desarrollado durante el proyecto. Debido a que el código es modificado de forma local, es posible que se cometa un error y este quede borrado del ordenador. |
| **Categoría** | Software |
| **Probabilidad** | Baja |
| **Impacto** | Alto |
| **Riesgo Total** | Medio |
| **Mitigación** | Utilización de GitHub, una plataforma de desarrollo de código en la nube, en la que guardar diferentes versiones del código del modelo mientras se esta realizando el desarrollo del mismo. De esta forma se pueden recuperar versiones anteriores del programa en caso de un borrado o de una modificación incorrecta. |
| **Acciones Correctivas** | El entorno de desarrollo Visual Studio Code ofrece un servicio de recuperación de archivos borrados, ya que guarda versiones antiguas del proyecto. En caso contrario, se puede recurrir a la recuperación y restauración de archivos de Microsoft. |

Tabla 2.9: Riesgo 8: Modificación de los objetivos del proyecto

| | |
|---|---|
| **Riesgo 8** | Modificación de los objetivos del proyecto |
| **Descripción** | Los objetivos del proyecto son cambiados durante su desarrollo, como puede ser un cambio de las librerías o arquitecturas utilizadas. Esto puede causar que el código desarrollado hasta el momento deba ser modificado en gran medida, siendo necesario emplear una mayor cantidad de tiempo del estimado en primer lugar. |
| **Categoría** | Planificación |
| **Probabilidad** | Media |
| **Impacto** | Alto |
| **Riesgo Total** | Alto |
| **Mitigación** | Definición precisa y clara de los objetivos fundamentales durante la fase de planificación del proyecto. |
| **Acciones Correctivas** | Aceptar los nuevos objetivos y adaptar tanto la planificación como el modelo a los nuevos objetivos |

Tabla 2.10: Riesgo 9: Mala comprensión de los objetivos del proyecto

| | |
|---|---|
| **Riesgo 9** | Mala comprensión de los objetivos del proyecto |
| **Descripción** | Error de interpretación de los objetivos del proyecto que puede derivar en una desviación del alcance del proyecto, llegándose a trabajar en tareas que no resultan relevantes para la finalización exitosa del trabajo. |
| **Categoria** | Planificación |
| **Probabilidad** | Alta |
| **Impacto** | Alto |
| **Riesgo Total** | Alto |
| **Mitigación** | Definir los objetivos en detalle y preguntar a los tutores en caso de duda, durante la fase de planificación del proyecto. |
| **Acciones Correctivas** | Estudiar los requisitos con detenimiento y acordar una reunión con los tutores para solventar cualquier duda. |

Tabla 2.11: Riesgo 10: Poca disponibilidad de los tutores

| | |
|---|---|
| **Riesgo 10** | Poca disponibilidad de los tutores |
| **Descripción** | No es posible planificar reuniones con los tutores debido a una incompatibilidad de horarios. Puede conllevar un aumento del tiempo estimado dado que el alumno no puede consultar sus dudas con los tutores. |
| **Categoría** | Planificación |
| **Probabilidad** | Baja |
| **Impacto** | Medio |
| **Riesgo Total** | Bajo |
| **Mitigación** | Planificar con antelación las reuniones que sean necesarias y realizarlas utilizando aplicaciones de videoconferencia para evitar desplazamientos. |
| **Acciones Correctivas** | Revisar los horarios para encontrar periodos de tiempo libres en los que realizar las reuniones. Así mismo, utilizar métodos de comunicación alternativos como son el correo electrónico, Teams o Google Meet, y aprovechar el tiempo de espera para escribir la memoria del proyecto. |

Tabla 2.12: Riesgo 11: Avería del equipo utilizado durante el desarrollo

| | |
|---|---|
| **Riesgo 11** | Avería del equipo utilizado durante el desarrollo |
| **Descripción** | El ordenador con el que se realiza el proyecto ha sufrido una avería y no se puede utilizar durante el desarrollo, causando retrasos en la codificación del modelo y en la escritura de la memoria. |
| **Categoría** | Hardware |
| **Probabilidad** | Baja |
| **Impacto** | Medio |
| **Riesgo Total** | Bajo |
| **Mitigación** | Garantizar que se dispone de un equipo de repuesto, el cual no necesita muchos requisitos, ya que la ejecución del código se realiza en un servidor remoto |
| **Acciones Correctivas** | Encontrar un ordenador auxiliar de la forma más rápida posible. |

Tabla 2.13: Riesgo 12: Aprendizaje excesivamente lento del modelo

| | |
|---|---|
| **Riesgo 12** | Aprendizaje excesivamente lento del modelo |
| **Descripción** | La evolución del modelo programado es demasiado lenta, por lo que se va a tardar mucho más tiempo en obtener los resultados esperados, afectando a la planificación acordada. |
| **Categoría** | Software |
| **Probabilidad** | Alta |
| **Impacto** | Medio |
| **Riesgo Total** | Alto |
| **Mitigación** | Realizar un estudio previo a la codificación de los parámetros de aprendizaje utilizados en proyectos similares de aprendizaje automático |
| **Acciones Correctivas** | Modificar los parámetros de aprendizaje del modelo de acuerdo a la documentación aportada por los tutores. |

Tabla 2.14: Riesgo 13: Pérdida de los resultados obtenidos

| | |
|---|---|
| **Riesgo 13** | Pérdida de los resultados obtenidos |
| **Descripción** | Borrado accidental de los resultados obtenidos por el modelo. Puede causar la necesidad de repetir la ejecución actual aumentando el tiempo necesario para realizar la fase de análisis de los datos. |
| **Categoría** | Planificación |
| **Probabilidad** | Media |
| **Impacto** | Medio |
| **Riesgo Total** | Medio |
| **Mitigación** | Guardar los resultados en un repositorio remoto inmediatamente después de cada ejecución, garantizando su fácil recuperación. |
| **Acciones Correctivas** | Utilizar software especializado en recuperación de archivos borrados para restaurar el archivo original. |

Tabla 2.15: Riesgo 14: Cambio de la arquitectura de aprendizaje por refuerzo utilizada

| | |
|---|---|
| **Riesgo 14** | Cambio de la arquitectura de aprendizaje por refuerzo utilizada |
| **Descripción** | Traslado de la arquitectura de aprendizaje reforzado del modelo utilizado, Q-Learning, a otros similares como puede ser MDP (Markov Decision Process). |
| **Categoría** | Software |
| **Probabilidad** | Baja |
| **Impacto** | Alto |
| **Riesgo Total** | Medio |
| **Mitigación** | Estudiar de forma detallada cada una de las arquitecturas recomendadas por el tutor, asegurándose de escoger desde el principio del desarrollo aquella que se ajusta mejor a los objetivos del proyecto. |
| **Acciones Correctivas** | Realizar las modificaciones necesarias al modelo para adaptarlo a la nueva arquitectura |

Tabla 2.16: Riesgo 15: Mala aplicación de la metodología evolutiva

| | |
|---|---|
| **Riesgo 15** | Mala aplicación de la metodología evolutiva |
| **Descripción** | Mal seguimiento de la metodología de planificación evolutiva, causando un alargamiento de las iteraciones de la fase de desarrollo del modelo. |
| **Categoría** | Planificación |
| **Probabilidad** | Medio |
| **Impacto** | Alto |
| **Riesgo Total** | Alto |
| **Mitigación** | Estudio de la aplicación de la metodología en otros proyectos similares antes de comenzar con el desarrollo |
| **Acciones Correctivas** | Planificar la estructuración del cronograma para adaptarlo a una duración correcta de cada una de las iteraciones. |

Tabla 2.17: Riesgo 16: Datos de entrenamiento de mala calidad

| | |
|---|---|
| **Riesgo 16** | Datos de entrenamiento de mala calidad |
| **Descripción** | Los datos aportados para el entrenamiento tienen errores o están corruptos. |
| **Categoría** | Software |
| **Probabilidad** | Medio |
| **Impacto** | Alto |
| **Riesgo Total** | Alto |
| **Mitigación** | Tener varios conjuntos de datos de reserva |
| **Acciones Correctivas** | Contactar con la empresa y pedir que envíen unos nuevos ficheros. |

## 2.4. Seguimiento

En esta sección se expondrá el seguimiento de las tareas de cada fase del proyecto. Para ello, se presentan unas tablas de comparación entre la estimación planificada en días de la duración de cada tarea al comienzo del proyecto, y el tiempo real que ha sido empleado. Estas tablas se dividen en las fases de documentación, desarrollo y redacción de la memoria que se muestran a continuación.

| TAREA | INICIO ESTIMADO | FINAL ESTIMADO | INICIO REAL | FIN REAL | DURACIÓN ESTIMADA (DIAS) | DURACIÓN REAL (DIAS) |
|---|---|---|---|---|---|---|
| **Reunion de Arranque del TFG** | 21-2-23 | 21-2-23 | 21-2-23 | 21-2-23 | 1 | 1 |
| Busqueda RL para dispositivos médicos | 1-3-23 | 13-3-23 | 1-3-23 | 20-3-23 | 13 | 20 |
| Reunión de comunicación con el tutor | 13-3-23 | 13-3-23 | 20-3-23 | 20-3-23 | 1 | 1 |
| Funcionamiento de Arquitecturas de interes | 14-3-23 | 26-3-23 | 21-3-23 | 9-4-23 | 13 | 20 |
| Gestion de riesgos | 27-3-23 | 29-3-23 | 10-4-23 | 13-4-23 | 3 | 4 |

Figura 2.1: Seguimiento de las tareas de la fase de documentación

| TAREA | INICIO ESTIMADO | FINAL ESTIMADO | INICIO REAL | FIN REAL | DURACIÓN ESTIMADA (DIAS) | DURACIÓN REAL (DIAS) |
|---|---|---|---|---|---|---|
| **Desarrollo del producto** | | | | | | |
| Requisitos minimos | 30-3-23 | 2-4-23 | 14-4-23 | 17-4-23 | 4 | 4 |
| Definición de arquitecturas | 3-4-23 | 6-4-23 | 18-4-23 | 23-4-23 | 4 | 6 |
| Analisis funcional | 7-4-23 | 11-4-23 | 24-4-23 | 28-4-23 | 5 | 5 |
| Diseño funcional | 12-4-23 | 16-4-23 | 29-4-23 | 4-5-23 | 5 | 6 |
| Codificación ingestión de datos y base del mod | 17-4-23 | 30-4-23 | 5-5-23 | 22-5-23 | 14 | 18 |
| Codificación del modelo | 1-5-23 | 14-5-23 | 23-5-23 | 11-6-23 | 14 | 20 |
| Entrenamiento del modelo | 15-5-23 | 21-5-23 | 12-6-23 | 21-6-23 | 7 | 10 |
| Mantenimiento - Ajustes de parámetros | 20-5-23 | 25-5-23 | 22-6-23 | 27-6-23 | 6 | 6 |

Figura 2.2: Seguimiento de las tareas de la fase de desarrollo

En la fase de documentación, expuesta en la Figura 2.1, las mayores diferencias entre las duraciones estimadas y reales se encuentran en las tareas de búsqueda, lectura y comprensión de la información, tanto sobre los dispositivos médicos de aprendizaje por refuerzo, como de las arquitecturas que se utilizan. Esto se debe a que los artículos de investigación que se encontraron sobre estos temas eran muy complejos para los conocimientos que se tenían en aquel momento, por lo que era necesario emplear más tiempo para comprenderlos. Además se tuvo que buscar información adicional sobre técnicas de aprendizaje automático y redes neuronales para entender mejor el contexto de los documentos.

| TAREA | INICIO ESTIMADO | FINAL ESTIMADO | INICIO REAL | FIN REAL | DURACIÓN ESTIMADA (DIAS) | DURACIÓN REAL (DIAS) |
|---|---|---|---|---|---|---|
| **Redacción del TFG** | | | | | | |
| Documentacion - Nombre del TFG | 15-5-23 | 15-5-23 | 12-6-23 | 12-6-23 | 1 | 1 |
| Documentacion - Resumen | 15-5-23 | 15-5-23 | 12-6-23 | 12-6-23 | 1 | 1 |
| Documentacion - Estado del arte | 16-5-23 | 17-5-23 | 13-6-23 | 15-6-23 | 2 | 3 |
| Documentacion - Sistemas similares en el mercado | 18-5-23 | 19-5-23 | 16-6-23 | 18-6-23 | 2 | 3 |
| Documentacion - Gestion de riesgos | 20-5-23 | 21-5-23 | 19-6-23 | 21-6-23 | 2 | 3 |
| Documentacion - Requisitos minimos | 22-5-23 | 23-5-23 | 22-6-23 | 23-6-23 | 2 | 2 |
| Documentacion - Definición de arquitecturas | 24-5-23 | 25-5-23 | 24-6-23 | 25-6-23 | 2 | 2 |
| Documentacion - Analisis funcional | 26-5-23 | 27-5-23 | 26-6-23 | 26-6-23 | 2 | 1 |
| Documentacion - Diseño funcional | 28-5-23 | 29-5-23 | 26-6-23 | 26-6-23 | 2 | 1 |
| Documentacion - Codificación ingestión de datos y base del | 30-5-23 | 31-5-23 | 27-6-23 | 28-6-23 | 2 | 2 |
| Documentacion - Codificación del modelo | 1-6-23 | 3-6-23 | 29-6-23 | 30-6-23 | 3 | 2 |
| Documentacion - Entrenamiento del modelo | 4-6-23 | 6-6-23 | 1-7-23 | 4-7-23 | 3 | 4 |
| Documentacion - Mantenimiento - Ajustes de parámetros | 7-6-23 | 7-6-23 | 5-7-23 | 5-7-23 | 1 | 1 |
| Documentacion - Resultados | 7-6-23 | 9-6-23 | 5-7-23 | 7-7-23 | 3 | 3 |
| Documentacion - Conclusiones | 10-6-23 | 12-6-23 | 8-7-23 | 10-7-23 | 3 | 3 |
| **Defensa del TFG** | | | | | | |
| Preparación de la defensa | 13-6-23 | 13-6-23 | 11-7-23 | 11-7-23 | 1 | 2 |

Figura 2.3: Seguimiento de las tareas de la fase de redacción

Durante la fase de desarrollo de la Figura 2.2, se distinguen dos diferencias significativas en el tiempo de codificación de la ingestión de datos y del modelo. Esta desviación en la planificación esta causada por la inexperiencia a la hora de utilizar el lenguaje Python, en especial el uso de las librerías encargadas de instanciar la red neuronal que forma el modelo. Además, fue necesario documentarse acerca del uso de funciones de calculo de entropía espectral, de optimizadores del modelo y de procesamiento de señales digitales.

En la fase de redacción del la Figura 2.3, se observa una menor discrepancia de días empleados, siendo la más notable la descripción del entrenamiento del modelo, ya que fue preciso consultar más documentación de la estimada para explicar los conceptos con claridad.

Por último, en la Tabla 2.18 se puede observar un resumen de las horas empleadas en cada fase en comparación con las estimadas inicialmente.

| **Fase** | **Horas reales** | **Horas estimadas** |
|---|---|---|
| Documentación | 77,5h | 115h |
| Desarrollo | 147,5h | 187,5h |
| Redacción | 80h | 85h |
| **Total** | **305h** | **387,5h** |

Tabla 2.18: Comparación de horas reales y horas estimadas por fases

## 2.5. Presupuesto

Se ha elaborado un presupuesto para determinar el coste monetario que supondría llevar a cabo el proyecto. Sin embargo, se debe tener en cuenta que se trata de un Trabajo de fin de Grado, es decir, de un contexto académico no orientado a ser desarrollado por una empresa en el mercado real.

Es necesario estimar el salario medio de un programador especializado en inteligencia artificial en España[20], el cual se aproxima a **32.000€** al año para un trabajador, resultando en una media de **2285€** al mes. Por lo tanto, dado que se considera que hay una media de 22 días laborables al mes, resulta en un total de **3894€** de sueldo neto por las 300 horas de trabajo en el proyecto. Sin embargo, a este salario se le debe restar el un 32,95 por ciento que la empresa paga a la seguridad social[21] por cada trabajador, obteniendo una cantidad de 2611,54€.

A continuación, se analizan los costes del equipo informático. Durante el desarrollo se utiliza un ordenador ASUS, que cuenta con un procesador Intel(R) Core(TM) i7-9750H, 16 GB de memoria RAM y una tarjeta gráfica NVIDIA GeForce GTX 1650. Este equipo tiene un precio comercial de 1500€ con una vida útil de 5 años aproximadamente.

Así mismo, hay que destacar que el código desarrollado es compilado y ejecutado en los servidores de un centro de procesamiento de datos, concretamente del SCAYLE[19]. Sus servidores cuentan con dos procesadores Intel Xeon Platinum 8358 de 32 cores, 1TB de memoria RAM, una conexión Infiniband HDR 100 Gbps y una tarjeta gráfica NVidia Tesla A100. Además, la tarifa aplicada por sus servicios es de 0,08€ por hora de GPU y 0,0075€ por core de CPU. Dado que se utilizan 1GPU y 64 cores durante 127 horas de computo, la cifra total asciende a 71.12€.

Por otro lado, se calcula los costes de las herramientas de software empleadas por el trabajador. Entre ellas destacan el entorno de desarrollo utilizado, Visual Studio Code, el cual es gratuito, la aplicación de coworking Microsoft Teams, que cuenta con una subscripción de 5,60€ al mes para empresas, siendo un total de 22,40€ para los 4 meses estimados de trabajo y la licencia anual de MathLab que suma 367€.

Finalmente, se determina un margen de contingencia del 10 % para subsanar posibles aumentos en el gasto esperado debido a retrasos en el desarrollo del producto o por otras

causas, como pueden ser la necesidad de obtener nuevos equipos o licencias de software. Los gastos totales se pueden observar en la Tabla 2.19.

| **Gastos** | **Total** |
|---|---|
| Salario programador | 3894,00€ |
| Seguridad Social del trabajador | 2611,54€ |
| Equipo informático | 1500,00€ |
| Licencia Microsoft Teams | 22,40€ |
| Licencia MathLab | 367,00€ |
| Servicio de computo en CPD | 71,12€ |
| Subtotal | 8466,06€ |
| Margen de contingencia 10 % | 846,61€ |
| **Total** | **9312,67€** |

Tabla 2.19: Gastos totales del proyecto

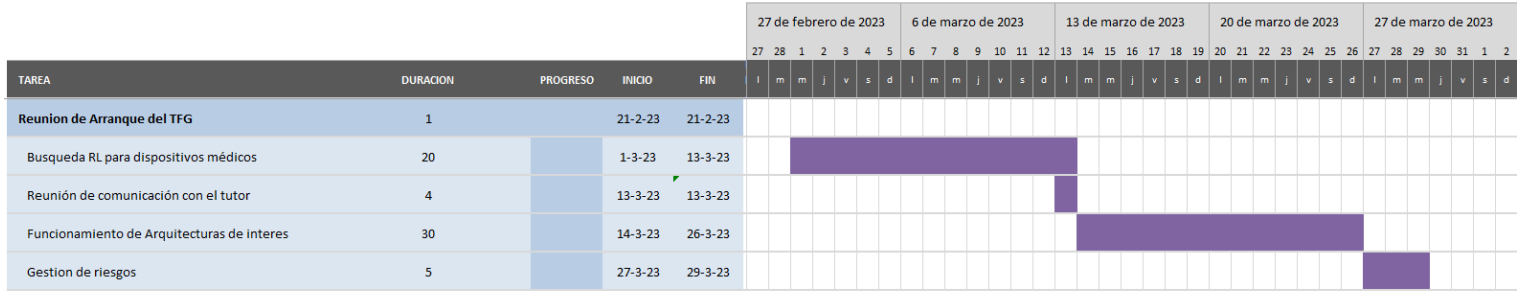



Figura 2.4: Diagrama de Gantt fase de documentación.

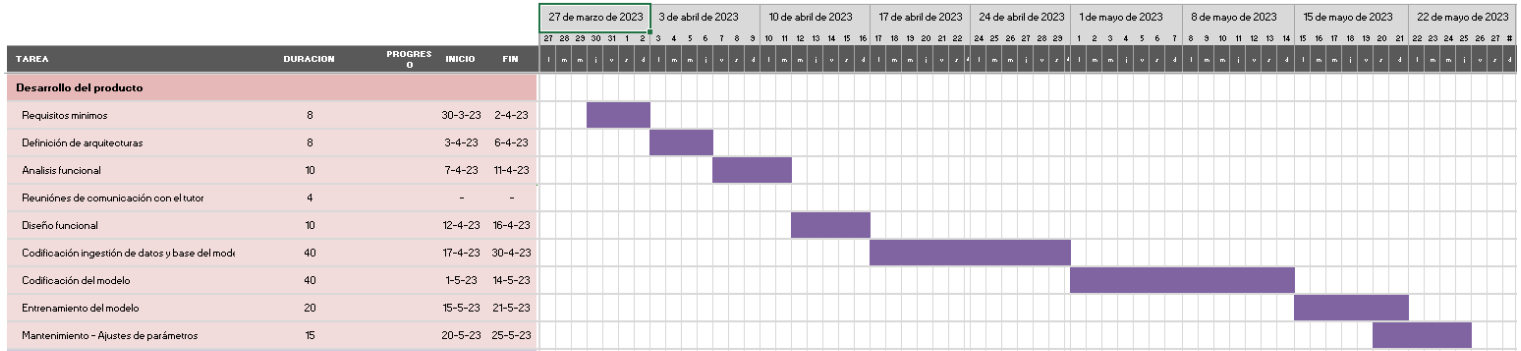
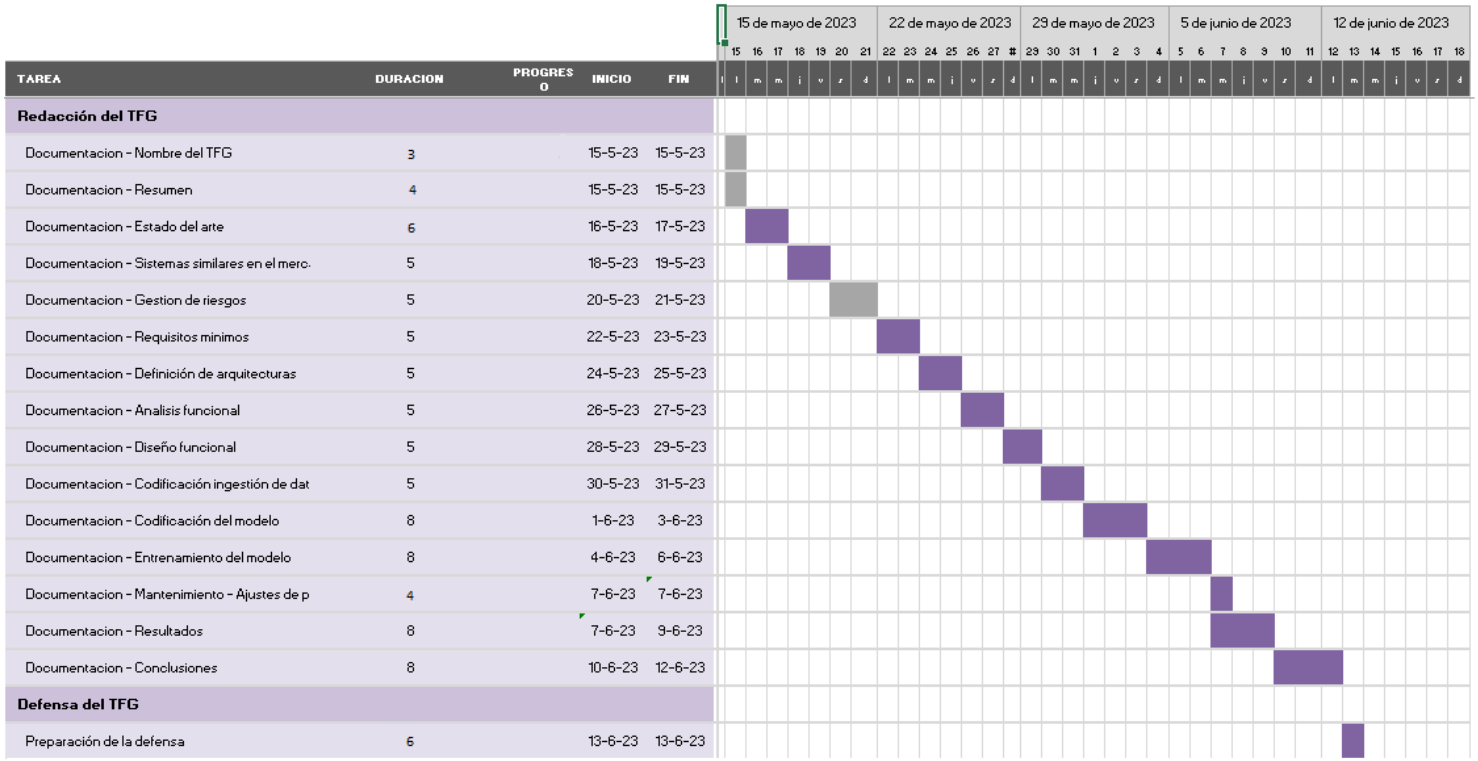



Figura 2.5: Diagrama de Gantt fase de desarrollo.




| TAREA | DURACION | PROGRESO | INICIO | FIN |
|---|---|---|---|---|
| **Redacción del TFG** | | | | |
| Documentacion - Nombre del TFG | 3 | | 15-5-23 | 15-5-23 |
| Documentacion - Resumen | 4 | | 15-5-23 | 15-5-23 |
| Documentacion - Estado del arte | 6 | | 16-5-23 | 17-5-23 |
| Documentacion - Sistemas similares en el merc | 5 | | 18-5-23 | 19-5-23 |
| Documentacion - Gestion de riesgos | 5 | | 20-5-23 | 21-5-23 |
| Documentacion - Requisitos minimos | 5 | | 22-5-23 | 23-5-23 |
| Documentacion - Definición de arquitecturas | 5 | | 24-5-23 | 25-5-23 |
| Documentacion - Analisis funcional | 5 | | 26-5-23 | 27-5-23 |
| Documentacion - Diseño funcional | 5 | | 28-5-23 | 29-5-23 |
| Documentacion - Codificación ingestión de dat | 5 | | 30-5-23 | 31-5-23 |
| Documentacion - Codificación del modelo | 8 | | 1-6-23 | 3-6-23 |
| Documentacion - Entrenamiento del modelo | 8 | | 4-6-23 | 6-6-23 |
| Documentacion - Mantenimiento - Ajustes de p | 4 | | 7-6-23 | 7-6-23 |
| Documentacion - Resultados | 8 | | 7-6-23 | 9-6-23 |
| Documentacion - Conclusiones | 8 | | 10-6-23 | 12-6-23 |
| **Defensa del TFG** | | | | |
| Preparación de la defensa | 6 | | 13-6-23 | 13-6-23 |

Figura 2.6: Diagrama de Gantt fase de redacción de la memoria.

# Capítulo 3

# Aprendizaje por refuerzo y redes neuronales

## 3.1. Aprendizaje automático

El aprendizaje automático [22] o machine learning es una rama de la inteligencia artificial centrada en la utilización de un conjunto de algoritmos, estructuras de datos y patrones que permiten que un ordenador consiga aprender, razonar y tomar decisiones sin influencia humana. Por lo tanto, para un problema específico, como puede ser la clasificación de imágenes o el procesamiento del lenguaje natural, un sistema computacional es capaz de aprender a realizar tareas de forma eficaz basándose sobre su propia experiencia.

Dentro del campo del aprendizaje automático se encuentran tres categorías, el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo, las cuales son diferenciadas por los datos de entrada que recibe el sistema y por el tipo de algoritmo utilizado durante el entrenamiento del agente. Se define como agente a un algoritmo que mediante la interacción del entorno es capaz de aprender sobre su propia experiencia, con el objetivo de maximizar la recompensa obtenida. Los métodos de aprendizaje automático[23] actuales se explican a continuación.

### 3.1.1. Aprendizaje Supervisado

En esta categoría el agente aprende a través del ejemplo. Para ello, se utiliza una gran cantidad de datos etiquetados, en los que se incluyen tanto las entradas deseadas como las salidas que deben ser obtenidas. El trabajo del agente consiste en crear una función que permita hallar una correspondencia correcta entre las entradas y las salidas esperadas. De modo que, con el tiempo suficiente, el agente es capaz de hacer observaciones e identificar patrones dentro de las entradas con las que realizar predicciones sobre el valor de las salidas. Estos resultados, en caso de ser incorrectos, son corregidos por los programadores hasta llegar al nivel deseado de exactitud y eficiencia.

Entre los escenarios en los que se utiliza el aprendizaje supervisado encontramos:

1. **Clasificación**: El agente debe reconocer características especificas de los elementos de entrada, para poder agruparlos en clases según su definición. Los algoritmos de clasificación más utilizados son las máquinas de vectores de soporte (SVM) y los clasificadores lineales.

2. **Regresión**: El agente debe determinar las relaciones entre una variable dependiente y varias independientes. Este algoritmo es utilizado en casos de análisis de ventas y para realizar proyecciones matemáticas.

El flujo de un algoritmo de aprendizaje supervisado se puede observar en la Figura 3.1.

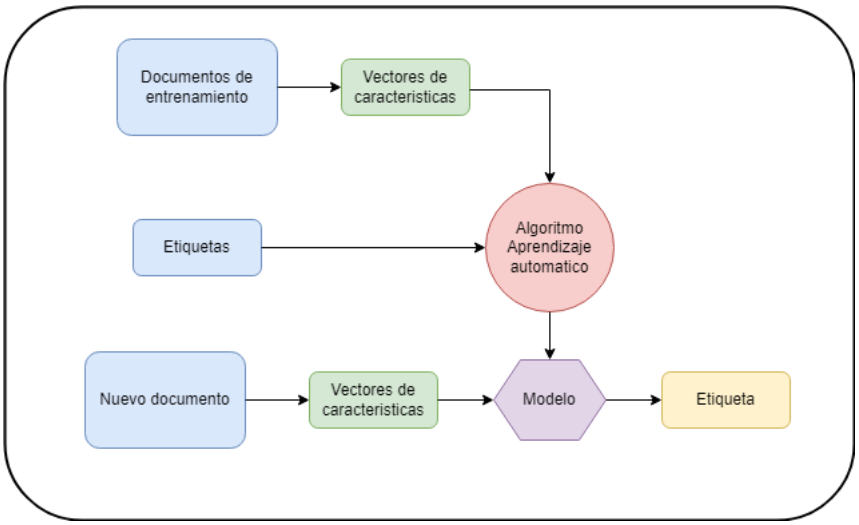



Figura 3.1: Algoritmo de Aprendizaje supervisado

### 3.1.2. Aprendizaje no Supervisado

El aprendizaje no supervisado tiene como objetivo estudiar estructuras de datos no etiquetados, sobre los que identificar patrones ocultos, sin la intervención de ningún programador que realice correcciones, como se ve en la Figura 3.2 . El agente interpreta grandes cantidades de datos y los agrupa en conjuntos, llamados generalmente clústeres, en los que se encuentran elementos con un grado alto de similitud. El algoritmo mas común es la agrupación en clústeres de k-medias.

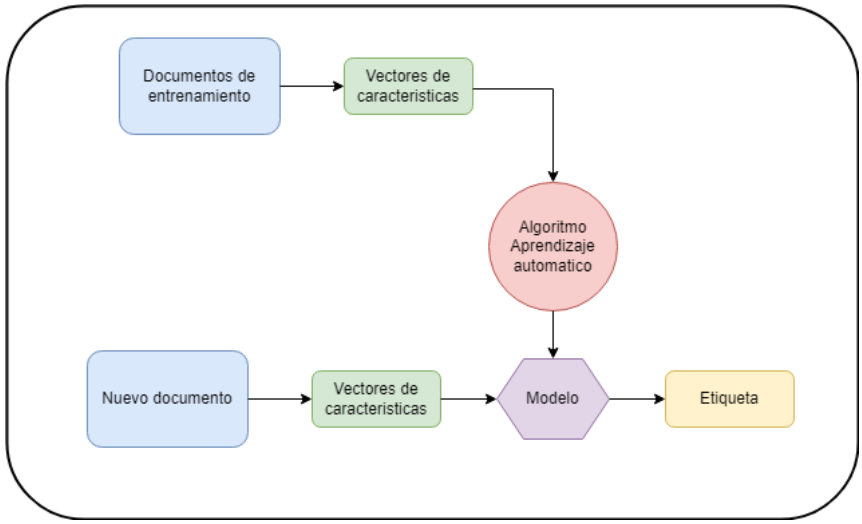



Figura 3.2: Algoritmo de Aprendizaje no supervisado

### 3.1.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo es el tipo de modelo utilizado en este proyecto. El algoritmo consiste en el entrenamiento mediante prueba y error. Se permite que el agente aprenda a tomar decisiones correctas a través de la experiencia obtenida al interactuar con el entorno, con el objetivo de maximizar el valor de recompensa obtenido. Cada acción realizada por el agente sobre el entorno tiene como resultado una recompensa, que puede ser positiva para los comportamientos que se quieren potenciar, y negativa para aquellos que se consideran incorrectos.

Dentro de este algoritmo podemos distinguir los siguientes conceptos básicos:

1. **Agente**: Entidad que es capaz de observar el entorno que le rodea, tomar decisiones dependiendo del estado en el que se encuentre, y llevar acabo dichas decisiones mediante acciones que estén relacionadas con el objetivo del problema.

2. **Entorno**: Contexto en el que el agente realiza las acciones, ya sea real o virtual. El entorno define las restricciones a las que se somete el agente a la hora de tomar decisiones, además de indicar datos importantes al sistema, como son el conjunto de acciones que se pueden llevar a cabo, el estado en el que se encuentra el mundo en un momento determinado y el valor de las recompensas asociadas a las acciones anteriores.

   Se debe tener en cuenta que existen varios tipos de entornos en los problemas de aprendizaje por refuerzo, siendo un entorno determinista aquel que para una determinada acción del agente va a suponer siempre la misma respuesta, y entornos estocásticos en los que hay un grado de incertidumbre para cada resultado.

3. **Estado**: Situación concreta en la que se encuentra el entorno. Su estructura depende de cada problema, ya que determina la relación que tiene el agente con las distintas características del entorno.

4. **Acción**: Elección realizada por el agente para un estado determinado que le permite interactuar con el entorno. Las acciones deben ser definidas como un conjunto finito y especifico para cada problema.

5. **Recompensa**: Retroalimentación obtenida por el agente como consecuencia de realizar una acción. Se utiliza para medir el grado de éxito o fracaso de ejecutar una elección.

6. **Política**: Estrategia que utiliza el agente para seleccionar la acción que considera que devolverá la mayor recompensa posible. Representa una relación entre las acciones y las recompensas.

Teniendo estas definiciones en cuenta, el proceso de entrenamiento de un modelo de aprendizaje por refuerzo se desarrolla en ciclos de interacción del agente con el entorno. Esto se puede ver en la Figura 3.3.

En una iteración, el agente observa el estado actual del entorno, selecciona una acción del conjunto de elecciones definidas y la lleva a cabo. En consecuencia, el entorno responde enviando una recompensa, ya sea positiva para las acciones correctas o negativa para las incorrectas, además de indicar al agente el nuevo estado en el que se encuentra.

Por último, el agente aprende de la interacción utilizando la recompensa obtenida para mejorar las políticas de selección de acciones.

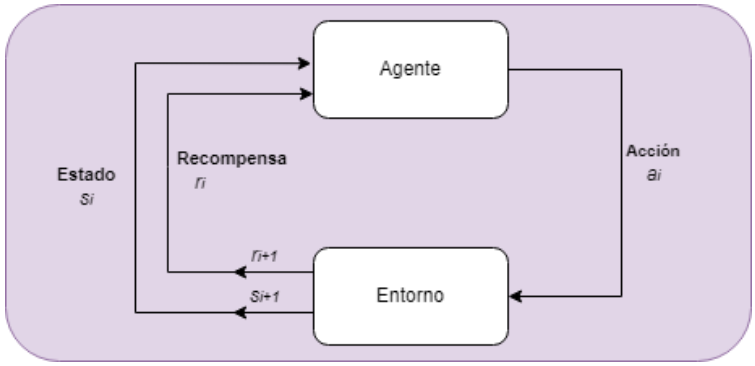



Figura 3.3: Bucle de aprendizaje por refuerzo

## 3.2. Redes neuronales

A continuación, se realiza una explicación sobre las ideas fundamentales de las redes neuronales.

Las redes neuronales [24] son un conjunto de algoritmos capaces de reconocer patrones y clasificar información a partir de unas entradas de vectores numéricos, como pueden ser imágenes y vídeos. Como su nombre sugiere, su diseño esta basado en el propio cerebro humano, por lo que cada red neuronal consta de varias capas de neuronas interconectadas entre sí, las cuales son capaces de procesar datos. Los elementos de los que se compone una red neuronal se explican en los siguientes apartados.

### 3.2.1. Nodos

Los nodos o neuronas son los elementos de procesamiento de información más básicos de los que se compone una red neuronal. Cada nodo recibe una serie de entradas numéricas, las cuales son multiplicadas pon un conjunto de coeficientes, llamados pesos. Estos pesos determinan el grado de importancia que tiene cada una de las entradas en la creación del valor de la salida. Por lo tanto, si un peso asignado a una entrada tiene un valor muy bajo, esta no tendrá mucho efecto en la salida, mientras que si el peso es muy alto condicionará en gran medida el resultado. Todos estos pesos representan en cierta medida la fuerza de las conexiones que hay entre las neuronas de cada capa, que son ajustadas en cada iteración para asegurar el mejor resultado posible. Todo este proceso esta ilustrado en la Figura 3.4

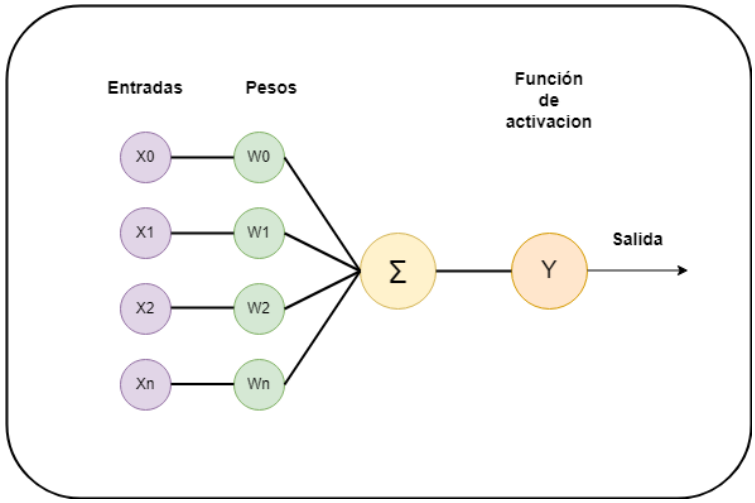



Figura 3.4: Procesamiento de un nodo en una red neuronal

### 3.2.2. Capas

Se define como capa[25] de una red neuronal a un conjunto de neuronas interconectadas entre sí, que envían y reciben información de otras capas que forman la red, como se observa en la Figura 3.5. Dependiendo de la posición que ocupe cada capa dentro del modelo se pueden distinguir tres tipos :

**Capa de Entrada:** Primera capa de la red neuronal. Se encarga de recibir los datos de entrada del entorno y enviarlos a las siguientes capas. En esta capa no se realiza ningún tipo de procesamiento de la información y su número de neuronas coincide con el número de variables del problema que se está estudiando.

**Capa Oculta:** Capas interiores del modelo que se ocupan de procesar las entradas aplicándoles sus pesos correspondientes y ejecutando la función de activación. Cada capa oculta se ocupa de un tipo diferente de procesamiento, y pasa los resultados al siguiente nivel.

**Capa de Salida:** Ultima capa de la red donde se obtienen los resultados finales, tiene tantas neuronas como características de estudio.

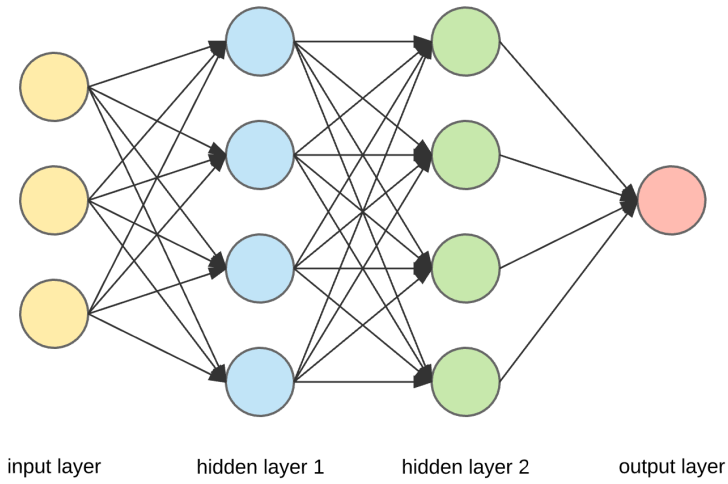



Figura 3.5: Capas de una red neuronal [1]

### 3.2.3. Función de activación

La función de activación es una expresión matemática que se encuentra dentro de cada nodo de la red neuronal, y se encarga de generar la salida correspondiente a las entradas aportadas. Sin embargo, esta función solamente se lleva a cabo cuando la suma ponderada de los datos de entrada es lo suficientemente grande para que el nodo se active, ya que en caso contrario no se produce la salida. Existen tres grupos de funciones de activación:

1. **Funciones de cresta** : Funciones de activación con una forma similar a una cresta, ya que su valor es cero en la mayoría de sus puntos, menos en una región en la que alcanza su valor máximo, se utiliza para dar importancia a ciertas características de los datos.

2. **Funciones radiales** : Se basa en calcular el resultado de la función dependiendo de la distancia que hay entre el punto de entrada y un centro definido.

3. **Funciones de pliegues** : Funciones con forma de pliegue o doblez, ya que cambia entre valores altos y bajos. Se utiliza en algunas redes neuronales para capturar patrones no lineales de datos.

Debido a las características de este proyecto, se han utilizado para construir el modelo la función ReLu y la función Softmax[26].

La función ReLU (Ver. Figura 3.6) se utiliza en muchos casos de aprendizaje automático. En esencia, la función devuelve valores positivos para entradas positivas y cero para entradas negativas. Este comportamiento garantiza un rápido aprendizaje de la red neuronal, ya que cada función solamente se activa para valores positivos de las entradas, causando que solamente se procese información importante y descartando el resto. Además, es extremadamente útil para problemas en los que se da efectos de interacciones entre variables.

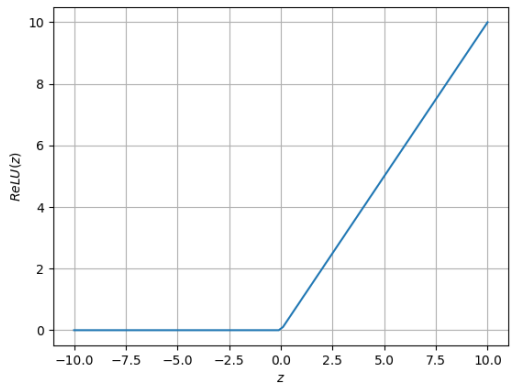



Figura 3.6: Función ReLU [2]

La función Softmax (Ver. Figura 3.7) se utiliza normalmente en las capas de salida del una red neuronal, ya que permite realizar una clasificación multiclase de las entradas que vienen de las capas anteriores. Softmax toma como entrada un vector de valores y devuelve la probabilidad de que cada instancia pertenezca a una clase. Estas probabilidades suman uno, y ayudan a interpretar de forma más simple los resultados calculados por la red.

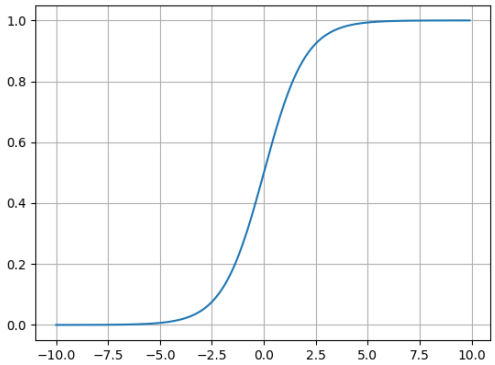



Figura 3.7: Función Softmax [3]

### 3.2.4. Entrenamiento y Función de pérdida

El entrenamiento del agente tiene como objetivo dotarle de la capacidad de hallar una correspondencia entre los datos que le llegan como entrada y las salidas que genera el propio modelo. Sin embargo, las predicciones que realiza el modelo, en especial en las primeras iteraciones, pueden no ser totalmente correctas.

Como se ha explicado anteriormente, primero se realiza una propagación hacia adelante, en la que con las entradas y los pesos se calculan las salidas. Después, en la propagación hacia atrás, en la que se comparan los resultados con las salidas ideales esperadas. La diferencia entre estos dos valores se denomina pérdida[27](Ver. Figura 3.8), la cual es utilizada para adaptar los hiperparámetros del modelo que se utilizan en las predicciones de las salidas. De esta forma, con un tiempo de entrenamiento prolongado se puede conseguir reducir la pérdida entre los valores estimados y los esperados.

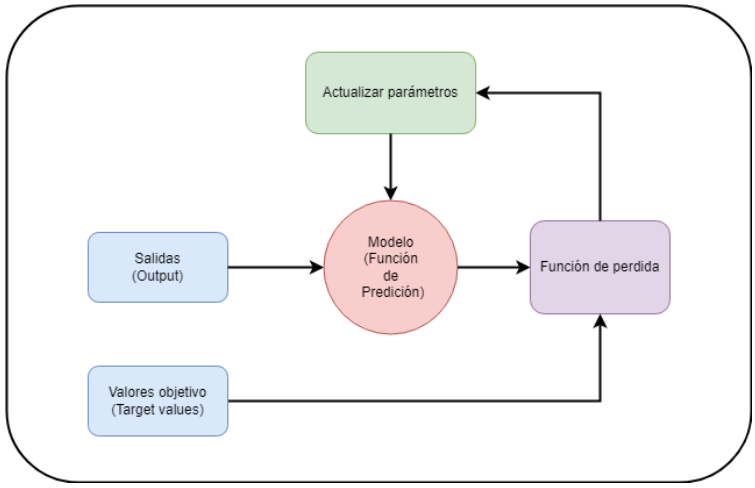



Figura 3.8: Función de perdida

### 3.2.5. Optimizadores

Los optimizadores [28] son un conjunto de algoritmos utilizados para ajustar los pesos del modelo, con el objetivo de mejorar su rendimiento. De esta forma, los optimizadores se encargan de hallar en cada iteración como se deben ajustar los parámetros del modelo para reducir la pérdida obtenida. Esto significa que determinan para que magnitudes de los parámetros la diferencia entre las salidas estimadas y los valores reales es menor.

Existen varios tipos de optimizadores:

1. **Descenso de gradiente** : Algoritmo de optimización basado en una función convexa, el cual reduce la función de perdida mediante un ajuste iterativo de los parámetros del modelo. Al modificar los parámetros se consigue que el valor de la función para esa iteración concreta sea el mínimo local. De esta forma, el algoritmo encuentra el menor punto de la función moviéndose en dirección contraria al incremento más significativo de los valores, es decir, del gradiente.

   Este método destaca por ser sencillo de implementar, pero tiene el inconveniente de necesitar todos los datos de entrada para calcular el gradiente, haciendo que sea lento y costoso en recursos.

2. **Descenso de gradiente estocástico**: Variación del algoritmo de descenso de gradiente en el que se actualizan los parámetros después de calcular la perdida de cada iteración. Por lo tanto, los parámetros serán actualizados tantas veces como filas tenga el conjunto de datos de entrada.

   La ventaja de utilizar este método es que consume mucha menos memoria que el descenso de gradiente, y puede llegar a converger con mayor rapidez. Sin embargo, también se puede producir ruido y variación en las actualizaciones.

3. **Adam [29]**: Es uno de los algoritmos de optimización más populares en proyectos de aprendizaje por refuerzo, y es el que se ha decidido utilizar en este trabajo. Este algoritmo, a diferencia del descenso de gradiente estocástico, no utiliza únicamente una tasa de aprendizaje para todos los parámetros, si no que cada parámetro tiene la suya propia. Por lo tanto, cada tasa de aprendizaje es adaptada en función del la estimación del momento, es decir, de la velocidad de convergencia del modelo, y de la magnitud de los gradientes anteriores. Esto conlleva que Adam pueda trabajar de eficazmente con conjuntos de datos con mucho ruido, además de converger más rápido que otros algoritmos similares.

## 3.3. Q-Learning

Q-learning[30] es un algoritmo de aprendizaje por refuerzo que tiene como objetivo encontrar la mejor acción posible para un estado del entorno determinado. Para ello se basa en el entrenamiento mediante la utilización de valores, llamados Q-values, y la optimización de la propia función que los calcula dependiendo del estado del entorno.

Los Q-values determinan la calidad de una acción para un determinado estado del entorno, es decir, cuanto mayor sea el valor correspondiente a un par estado-acción mayor sera la recompensa que aportará al sistema a largo plazo y más favorable será llevar a cabo esa elección comparada con el resto de acciones posibles.

Para entornos no muy extensos, cada uno de estos valores se guardan en una tabla indexada, llamada Q-table, según todas las posibles combinaciones de acciones y estados que puedan producirse en el problema estudiado. Durante el entrenamiento del modelo, se calcula el valor correspondiente a cada celda utilizando la Ecuación de Bellman.

La Ecuación de Bellman es una función matemática que determina una correlación entre un conjunto estado-acción, la combinación de la recompensa actual obtenida y el máximo de todas las posibles recompensas que se obtendrán en un futuro. La forma general de la ecuación se presenta a continuación:

$$Q(s,a) = Q(s,a) + alpha * [R(s,a) + gamma * max'_a Q(s',a') - Q(s,a)] \tag{3.1}$$

Consta de los siguientes elementos:

1. **Q(s, a)**: Representa el Q-value estimado de un par estado-acción. Este valor describe el grado de calidad que tendría llevar a cabo la acción especificada para ese estado del entorno. Las celdas de la Q-table correspondientes a esos estados y acciones son actualizados con el valor obtenido al aplicar la ecuación de Bellman.

2. **Ratio de Aprendizaje (alfa)** : Hiperparámetro que expresa la rapidez con la que el modelo actualiza las estimaciones de los Q-values actuales. Suele estar definido como un valor en un rango de 0 a 1. Siendo cero, la falta de actualización, y por tanto de aprendizaje, y 1 la total modificación de los valores.

3. **R(s, a)**: Recompensa obtenida después de realizar una acción sobre el estado actual del entorno. El agente recibe una recompensa que puede ser positiva si la acción ha sido correcta respecto a los objetivos del sistema o negativa si se considera errónea. Dado que el agente tiene como objetivo maximizar la recompensa obtenida, intentará realizar aquellas acciones que devuelvan una recompensa positiva.

4. **Factor de descuento**: Coeficiente que establece el grado de importancia que da el modelo a las recompensas actuales frente a las recompensas futuras. Para valores entre el 0 y el 1, cuanto más se acerque al uno más prioridad dará el modelo a las recompensas futuras sobre las inmediatas.

5. **max_a' Q(s', a')**: Representa el valor máximo de todas las acciones posibles para el siguiente estado del entorno. De esta forma, se estima cual será el mejor Q-value que se puede obtener del siguiente estado.

Por otro lado, un factor importante del algoritmo de Q-Learning es su capacidad para establecer un equilibro entre la elección de decisiones conocidas y desconocidas. Si para tomar una acción siempre se selecciona aquella con el Q-value más alto, es posible que el modelo acabe dando prioridad a las mismas acciones sin encontrar otras que puedan tener una mejor recompensa con el tiempo. Esto se llama el fenómeno de exploración-explotación[30] en el que se utiliza una función Epsilon-Greedy, donde se define que si valor aleatorio es menor que un coeficiente épsilon se realizará una acción aleatoria, y si no se tomará la de mayor recompensa conocida.

Teniendo en cuenta todos estos factores, el algoritmo de aprendizaje con Q-learning, mostrado en la Figura 3.9 consta de varias etapas.

1. **Observar el estado del entorno**: El agente obtiene cómo está el entorno en el momento actual.

2. **Determinar la acción actual**: Se elige la acción que va a llevar a cabo dependiendo del valor de épsilon, dando como resultado un movimiento aleatorio o conocido.

3. **Recoger el estado actual del entorno y la recompensa**: El entorno reacciona al cambio realizado devolviendo su nuevo estado y el valor de la recompensa.

4. **Actualizar la Q-table**: Actualizamos el valor correspondiente al par estado-acción en la tabla.

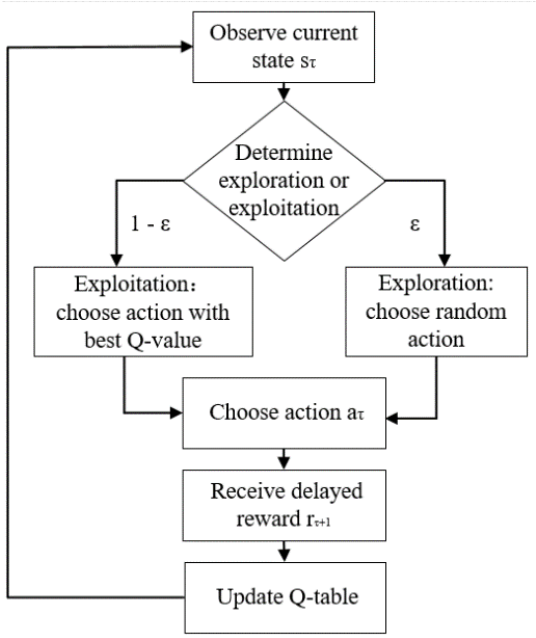



Figura 3.9: Diagrama de aprendizaje en Q-learning [4]

Finalmente, para problemas con muchos estados y acciones, se puede utilizar una variante del algoritmo llamada Deep Q-Learning[31]. Esta estrategia aumenta la eficiencia del algoritmo mediante el uso de redes neuronales a la hora de predecir los valores de las recompensas futuras, calcular la función de pérdida y actualizar los pesos de la propia red neuronal.

# Capítulo 4

# Estado del Arte

En este capítulo se explica el estado actual de los dispositivos y algoritmos de control utilizados en los tratamientos de Estimulación Cerebral Profunda en pacientes con trastornos neurológicos como el Parkinson o la epilepsia.

## 4.1. Estimulación cerebral profunda

La estimulación cerebral profunda, o en inglés Deep Brain Stimulation (DBS), es un tratamiento médico enfocado en el envío de señales eléctricas a ciertas áreas del cerebro, con la finalidad de regular los impulsos perjudiciales que ocurren en pacientes con enfermedades neuronales.

Para ello es necesario implantar electrodos en áreas especificas del cerebro de los pacientes, los cuales son controlados por un neuroestimulador colocado cerca del abdomen, esto se observa en la Figura 4.1. Esta máquina se encarga de generar los pulsos eléctricos, los cuales reducen el nivel de activación de las neuronas donde se están aplicando los impulsos, previniendo los síntomas[32]. La regulación de la frecuencia y la intensidad de los pulsos eléctricos es regulada por el personal sanitario de forma especializada y adaptada para cada paciente.

El tratamiento mediante DBS se considera una práctica efectiva para reducir los síntomas causados por enfermedades neuronales que no pueden ser reguladas de formas convencionales. De entre las enfermedades en las que se ha utilizado DBS destacan las siguientes:

**Enfermedad de Parkinson[33]:** Se utiliza DBS para aliviar los sintomas motores que caracterizan a la enfermedad del Parkinson en estados avanzados, reduciendo los temblores, la rigidez en las extremidades y la falta de equilibrio.

**Epilepsia[34]:** Utilizado en pacientes con resistencia ante los medicamentos antiepilepticos, ayuda a disminuir la intensidad y la frecuencia de las crisis convulsivas provocadas por la epilepsia.

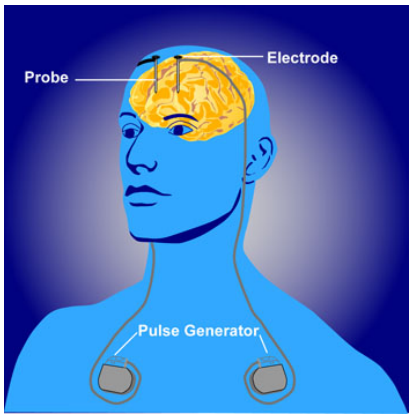



Figura 4.1: Estimulación cerebral profunda (DBS) [5]

## 4.2. Aprendizaje por refuerzo aplicado al DBS

Como se ha mencionado en el apartado anterior, para realizar el tratamiento mediante DBS es necesario que los ajustes de los parámetros de la frecuencia e intensidad de los pulsos sean adaptados manualmente por los médicos. De esta forma los sanitarios observan en cada sesión mediante prueba y error qué parámetros son los más eficaces para un paciente en particular.

Este procedimiento, aunque significativamente efectivo en aliviar los síntomas, puede causar efectos secundarios en los pacientes como puede ser una sobreestimulación al estar expuestos a altas intensidades durante un periodo prolongado de tiempo.

Debido a esto, se ha optado por la utilización de un modelo de aprendizaje por refuerzo (RL) que ajuste de forma automática los parámetros. Este modelo es capaz de aprender con qué parámetros reacciona mejor cada paciente y aplicarlos cuando sea necesario[35].

Para adaptar el dispositivo de control de DBS al modelo de aprendizaje por refuerzo se han definido los siguientes bloques[6]:

### 4.2.1. Entorno

El entorno en el contexto de la DBS se define como el sistema nervioso del paciente, en especial el cerebro, del que se puede observar las señales que está produciendo en un momento determinado y las respuestas que realiza a la estimulación eléctrica. El entorno se modela con un número de neuronas, capas e interconexiones que permitan simular los patrones de un sistema afectado por una patología, tomando como referencia estudios reales [36].

### 4.2.2. Acción

Las acciones se corresponden con los patrones de estimulación que aplican los electrodos en el cerebro del paciente. El modelo puede realizar una serie de acciones, que consisten en aumentar o disminuir la frecuencia o la intensidad de las señales. Estas variaciones deben ser lo más pequeñas posibles, ya que de esta forma resultan menos intrusivas para los pacientes. Además, para simular los pulsos que ocurren de forma natural, se introduce ruido en cada acción.

### 4.2.3. Recompensa

Retroalimentación que ayuda al modelo a aprender qué señales son óptimas para la condición del paciente, se definen en función de los resultados clínicos deseados dependiendo de la enfermedad del paciente.

### 4.2.4. Agente

Algoritmo que controla el dispositivo de estimulación y que tiene como objetivo maximizar las recompensas.

### 4.2.5. Entrenamiento

El proceso de entrenamiento expuesto en la Figura 4.2 consiste en que el agente observa el estado del cerebro y detecta anomalías, para después, enviar un pulso electromagnético basada en la acción escogida por el agente. Este pulso afecta a una región del cerebro disminuyendo su intensidad. Por último, el entorno devuelve un nuevo estado y una recompensa positiva o negativa dependiendo del nivel de eficacia de la acción anterior.

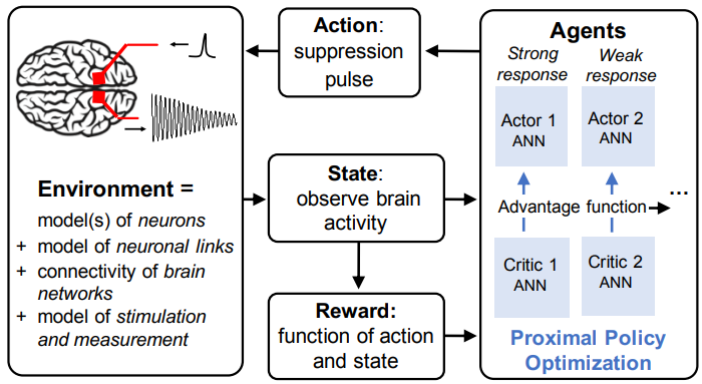



Figura 4.2: Ciclo de entrenamiento [6]

## 4.3. Métodos de regulación de pulsos en DBS

En los sistemas de DBS actuales se suelen utilizar dos tipos distintos de métodos de control de los pulsos eléctricos, la estimulación abierta y los bucles de estimulación cerrada[37](Ver. Figura **??**). Estos se diferencian en especial en la frecuencia con la que se actualizan los parámetros del dispositivo de control.

### 4.3.1. Estimulación abierta

En el método de estimulación abierta se establecen unos valores fijos de frecuencia e intensidad para los pulsos eléctricos en los primeros meses de tratamiento. Después de ese periodo, se analiza cómo ha respondido el paciente a esa estimulación y se ajustan las configuraciones del dispositivo en consecuencia.De esta forma, se obtiene un progreso personalizado al paciente con mínimos efectos adversos.

Sin embargo, al pasar tanto tiempo entre periodos, la efectividad es limitada, ya que no se tiene en cuenta los posibles cambios en la naturaleza de los síntomas del paciente.

### 4.3.2. Bucles de retroalimentación cerrados

En la retroalimentación cerrada el dispositivo de control obtiene información continua a través de unos monitores que observan en todo momento las condiciones en las que se encuentra el paciente. De esta forma, es posible ajustar los parámetros en tiempo real, haciendo que el neurotransmisor genere los pulsos correctos de forma dinámica. Este método aporta una mayor personalización, pero suelen ser más complejos de desarrollar. Cabe destacar que este tipo de estimulación es el que se ha elegido para este proyecto.

## 4.4. Algoritmos de control

A continuación se explican los algoritmos de control de dispositivos de DBS más utilizados hoy en día.

### 4.4.1. Actor-Critic

El algoritmo Actor-Crítico [38] representado en la Figura 4.3, es un método de aprendizaje que como su nombre indica, se compone de dos elementos principales, el actor y el crítico. Estos componentes trabajan en conjunto para elegir las acciones que lleven a la mayor recompensa posible

### Actor

El actor tiene como objetivo aprender a desarrollar una política de selección de acciones que le permita tomar la mejor decisión posible para el estado actual del entorno. Al igual que otros algoritmos de aprendizaje por refuerzo, el actor puede estar conformado de una red neuronal con sus respectivos hiperparámetros de aprendizaje.

### Crítico

El elemento crítico se encarga de evaluar la efectividad de las acciones llevadas a cabo por el actor para esa iteración. Esta evaluación se obtiene mediante la estimación del valor de la acción a largo plazo, que se corresponde con las recompensas acumuladas que se recibirían a partir del estado actual. Cabe destacar que este componente puede estar representado por una función de valor, por ejemplo el calculo de los Q-values para un par estado-acción, o por una función de pérdida que calcula la diferencia entre los valores reales y los estimados.

### Entrenamiento

Durante el entrenamiento, el actor elige una función dependiendo de su política actual de selección y la lleva a cabo. Después, el entorno reacciona al cambio, devolviendo el nuevo estado en el que se encuentra y la recompensa generada. Es entonces cuando el crítico evalúa la calidad de la acción, y se actualizan los parámetros tanto del actor como del critico con métodos de optimización y gradientes, a partir de la diferencia de los valores estimados y calculados. Estas actualizaciones se hacen en cada iteración para garantizar una mejoría constante.

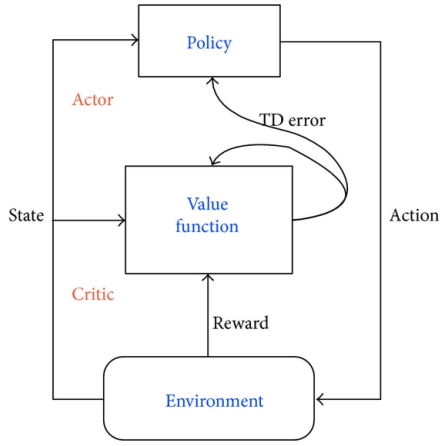



Figura 4.3: Ciclo de entrenamiento Actor-critic [7]

### 4.4.2. Proximal Policy Optimization

Proximal Policy Optimization (PPO)[39] es un método de aprendizaje por refuerzo, el cual, además de tener como objetivo obtener la mayor recompensa acumulada, también optimiza las políticas de elección de acciones a partir de otras anteriores.

Este algoritmo fue desarrollado por OpenIA, y tiene como base garantizar que las actualizaciones de las políticas del agente se hagan de forma segura y estable, evitando cambios drásticos que alteren el funcionamiento del modelo.

Para ello se calcula el ratio de diferencia entre la política actual y las anteriores, haciendo que sea mucho menos favorable alejarse en exceso del comportamiento anterior, ya que realizar cambios pequeños en el sistema, con el tiempo suficiente, llega a construir modelos mas eficientes.

## 4.5. Arquitecturas de redes en DBS

En los proyectos de DBS se utilizan distintos tipos de redes neuronales para configurar el modelo utilizado, así como para reconocer el estado que describen las señales cerebrales y predecir las posibles respuestas del paciente.

### 4.5.1. Redes neuronales convolucionales (CNN)

Las redes convolucionales[40] es una de las arquitecturas de redes neuronales mas utilizadas en proyectos de DBS. Se caracterizan por ser capaces de procesar de forma eficiente datos estructurados, que pueden ser, en el caso de la estimulación cerebral profunda, encefalogramas o impulsos eléctricos.

Esta arquitectura tiene la habilidad de extraer de los datos de entrada patrones y características, que después pueden ser utilizados en problemas de clasificación, reconocimiento y predicción de datos.

Además, a diferencia de otros modelos, las redes CNN tienen capas ocultas llamadas capas convolucionales, las cuales, se encargan de procesar los datos, aprendiendo los conceptos más fundamentales de las entradas a medida que se va enviando la información de una capa a otra. De esta forma, las primeras capas obtienen los detalles más básicos, como puede ser en el caso de una imagen los colores de los píxeles de los que se compone, mientras que las últimas procesan características más complejas como su forma.

También cuentan con una capa de *polling*, que en ciertos tramos de la red neuronal, realiza un resumen de las salidas generadas hasta el momento, reduciendo las dimensiones de las matrices de características creadas por las capas convoluciones. De esta forma, se filtra la información para operar únicamente con los valores mas representativos e importantes, además de reducir el coste de computación.

### 4.5.2. Redes neuronales densas

Las redes neuronales densas son aquellas en las que cada neurona está conectada a todas las neuronas de la capa anterior. De esta forma cada nodo recibe las entradas de todas la neuronas previas y envía su salida a todas las siguientes. Son el tipo de redes más utilizado en problemas de inteligencia artificial, ya que pueden adaptarse a todo tipo de situaciones. Además, dado que sus neuronas están totalmente conectadas, son capaces de identificar patrones complejos en los datos de entrada, resultando en un aprendizaje del modelo mucho más complejo.

Sin embargo, este tipo de redes tienen una gran cantidad de parámetros de entrenamiento, por lo que es necesario disponer tanto de una mayor capacidad computacional, como de una gran cantidad de datos de entrenamiento.

# Capítulo 5

# Conjunto de datos electroencefalográficos

Como en la mayoría de proyectos centrados en el aprendizaje por refuerzo, es necesario contar con una gran cantidad de datos representativos del problema, con los que realizar tanto el entrenamiento del modelo como las pruebas posteriores. En este capítulo se especifican el significado y las características más importantes de los datos aportados, así como el tratamiento que se realiza sobre ellos durante la ejecución del programa.

## 5.1. Descripción del conjunto de datos

En este proyecto se han utilizado dos conjuntos distintos de datos, siendo estos los provenientes de un estudio médico[41], los cuales han sido cedidos por la empresa tutora de este Trabajo de Fin de Grado, Air Institute, y otros creados por un generador de forma dinámica. Sin embargo, todos tienen la misma estructura, dividiéndose en dos tipos, los valores de señal y las etiquetas.

Los valores de señal corresponden con una serie temporal de magnitudes de una señal obtenida de forma dinámica de un corte hipocampal del cerebro, mediante un array de microelectrodos.

Por otro lado, las etiquetas son valores booleanos que se emparejan con cada una de las señales. Si para un determinado valor su etiqueta se encuentra a cero, significa que la actividad cerebral esta transcurriendo con normalidad, mientras que si su valor es un uno, entonces se ha detectado una anomalía. Estos cambios en la actividad neuronal del cerebro pueden ser causados por una crisis derivada de enfermedades como la epilepsia, y son aquellos que el sistema tiene como objetivo reducir.

La forma en la que se encuentran los datos es extremadamente sencilla, ya que cada tipo de datos se encuentra en su propio fichero con formato .mat en una única columna. Además,

en cada archivo también se encuentra la tasa de muestreo, es decir es número de muestras recibidas por segundo.

Para una mejor visualización de los datos se ha realizado un submuestreo equidistante, obteniendo la Figura 5.1.

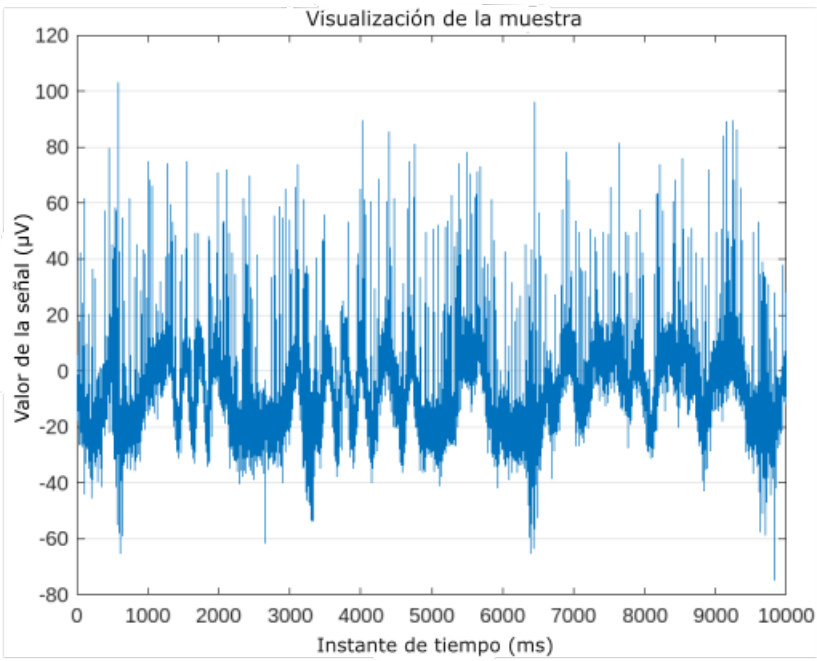



Figura 5.1: Muestra de datos de las señales

Sin embargo, para el total del conjunto de datos los valores se encuentran en un rango con un valor máximo de 139 µV y un mínimo de -96.5 µV. En cuanto a las etiquetas, se caracterizan por alterar entre periodos continuos de actividad normal y de convulsiones.

Por último, en cuanto a la cantidad total de datos, se ha dispuesto de un total de 116 archivos de datos encefalográficos que suman 2320 minutos de grabación, además de los datos creados por el generador que asciende a 16042 minutos de señales simuladas.

## 5.2. Procesamiento

Debido a que los datos utilizados se corresponden con una señal digital, es necesario llevar acabo un preprocesamiento de las señales con el objetivo de hacer más fácil la extracción de sus características, así como la reducción de fenómenos como la fuga espectral.

La fuga espectral[42] es un fenómeno que ocurre cuando se analizan señales que se encuentran delimitadas por un periodo de tiempo, en este caso el periodo se corresponde con la duración de estudio. Debido a que el análisis se realiza con secciones acotadas de una onda, esta presenta discontinuidades en los tramos de inicio y final. Esta discontinuidad genera valores de frecuencia altos que se extienden fuera de los niveles de la señal original.

Como consecuencia, las técnicas de análisis de señales como la Transformada de Fourier, que asumen que la onda simplemente se repite de la misma forma fuera de la sección de tiempo estudiada, se ven afectadas por la discontinuidades, haciendo que no se pueda reconstruir la señal de entrada. Esto se observa en la Figura 5.2.

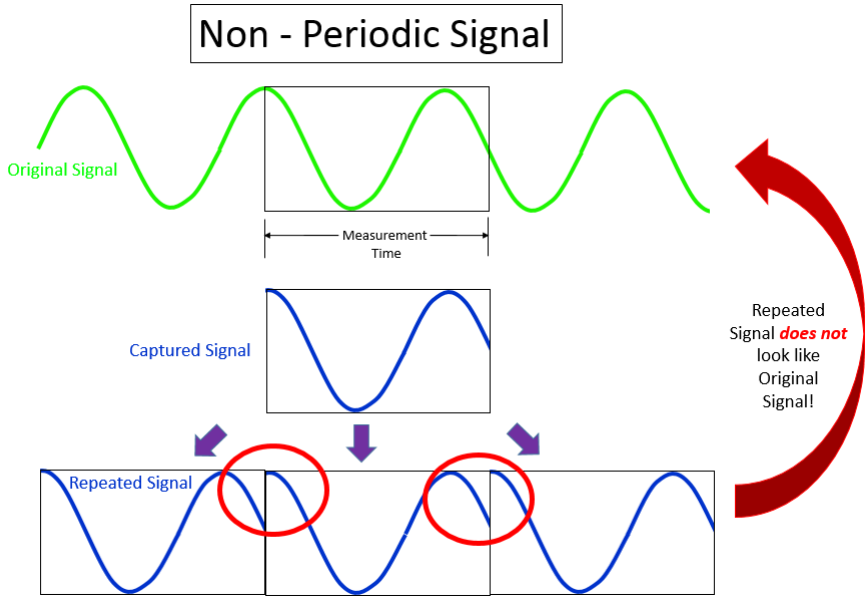



Figura 5.2: Muestra de discontinuidades en la señales [8]

Por lo tanto, la fuga espectral es un fenómeno muy problemático, ya que puede crear frecuencias erróneas que hagan muy difícil la correcta identificación y análisis de la señal. Para minimizar sus efectos se utilizan técnicas de aplicación de ventanas como la ventana de Hanning.

### 5.2.1. Ventana de Hanning

Una ventana de Hanning[43] es una técnica de preprocesamiento de señales digitales utilizada para reducir el efecto de las discontinuidades en una señal digital. Esta ventana está representada por una función matemática parecida a la función del coseno, ya que los valores mas altos se encuentran en el centro, mientras que los extremos tienen valores próximos a cero. Su expresión es la siguiente:

$$w(n) = 0{,}5 - 0{,}5cos((2pi * n)/(M - 1)) \tag{5.1}$$

Para realizar la aplicación de una ventana de Hanning, es necesario multiplicar punto a punto la serie temporal por una instancia de una ventana de Hanning de longitud proporcional y constante a los datos que se quieren tratar. De esta manera, la ventana adapta la forma de la señal, suavizando las discontinuidades y reduciendo su amplitud, minimizando así el efecto de la fuga espectral a pequeñas zonas, sin que afecte a todo el recorrido de la onda.

Como resultado, los extremos de cada sección de la señal se unen de forma gradual y sin producir saltos. Tanto la aplicación de una ventana de Hanning como la reducción de las discontinuidades, se representan en las figuras 5.3 y 5.4 respectivamente.

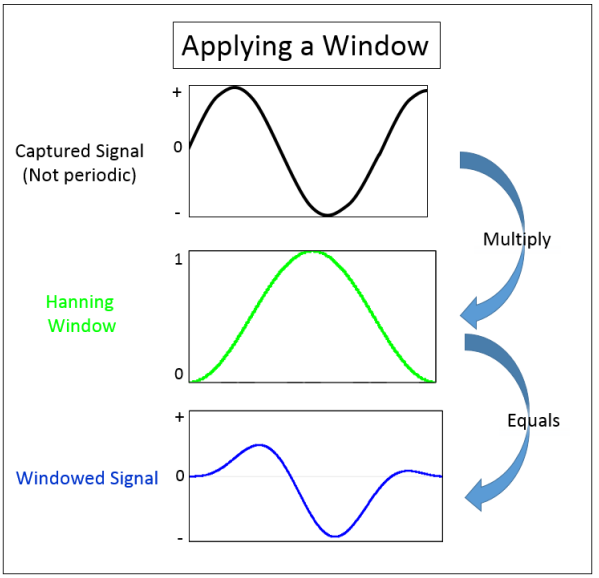



Figura 5.3: Aplicación de una ventana de Hanning [8]

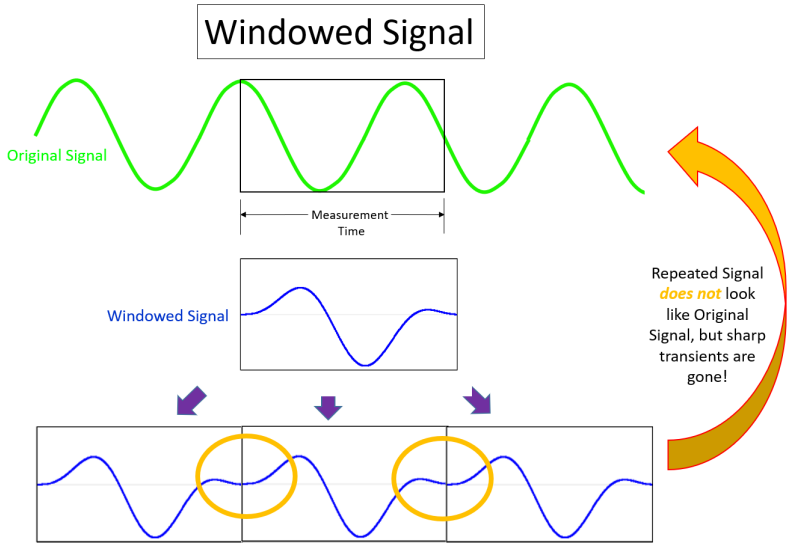



Figura 5.4: Reducción de discontinuidades en la señal [8]

# Capítulo 6

# Análisis

Durante el periodo de análisis del proyecto, se ha llevado acabo un estudio de las capacidades que el sistema debe poseer una vez terminado el proyecto, especificadas en requisitos funcionales y no funcionales. Sobre estos requisitos se presentan el diagrama de dominio del sistema, así como un diagrama de actividad que destaca la interacción entre los elementos de agente y entorno.

## 6.1. Requisitos Funcionales

A continuación, se especifican los requisitos funcionales identificados a partir de los objetivos del proyecto descritos anteriormente:

**RF1.** El sistema deberá ser capaz de recopilar los datos relacionados con la actividad cerebral del paciente.

**RF2.** El sistema deberá ser capaz de entrenar un modelo de aprendizaje por refuerzo mediante los datos aportados.

**RF3.** El sistema deberá ser capaz de procesar los datos para el entrenamiento del modelo.

**RF4.** El sistema deberá ser capaz de analizar los datos de entrada para identificar señales en los que se produce un ataque epiléptico.

**RF5.** El sistema deberá ser capaz de desarrollar una política de selección de acciones.

**RF7.** El sistema deberá ser capaz de generar señales con diferentes frecuencias e intensidades.

**RF8.** El sistema deberá ser capaz de simular interferencias entre las señales cerebrales de entrada y los nuevos pulsos generados.

**RF9.** El sistema deberá ser capaz de obtener las recompensas correspondientes a cada acción del agente.

**RF10.** El sistema deberá ser capaz de predecir los posibles valores para una serie de estados futuros.

**RF11.** El sistema deberá ser capaz de analizar los valores de retroalimentación al aplicar un pulso y determinar el estado en el que se encuentra el paciente.

## 6.2. Requisitos no Funcionales

Los requisitos no funcionales que deben caracterizar al sistema son los siguientes

**RNF1.** El sistema debe devolver respuestas rápidas y eficientes en el análisis de datos y la generación de pulsos.

**RNF2.** Las políticas generadas por el modelo deben ser precisas y confiables.

**RNF3.** Las recompensas obtenidas por el agente deben estar fuertemente relacionadas con los objetivos del sistema.

## 6.3. Diagrama de Dominio

Teniendo como fundamento los requisitos descritos, se ha realizado un diagrama de dominio(Ver. Figura 6.1) del sistema, donde se especifican las clases principales en las que se divide el sistema y la relación entre ellas. Se distinguen los conceptos de agente y entorno.

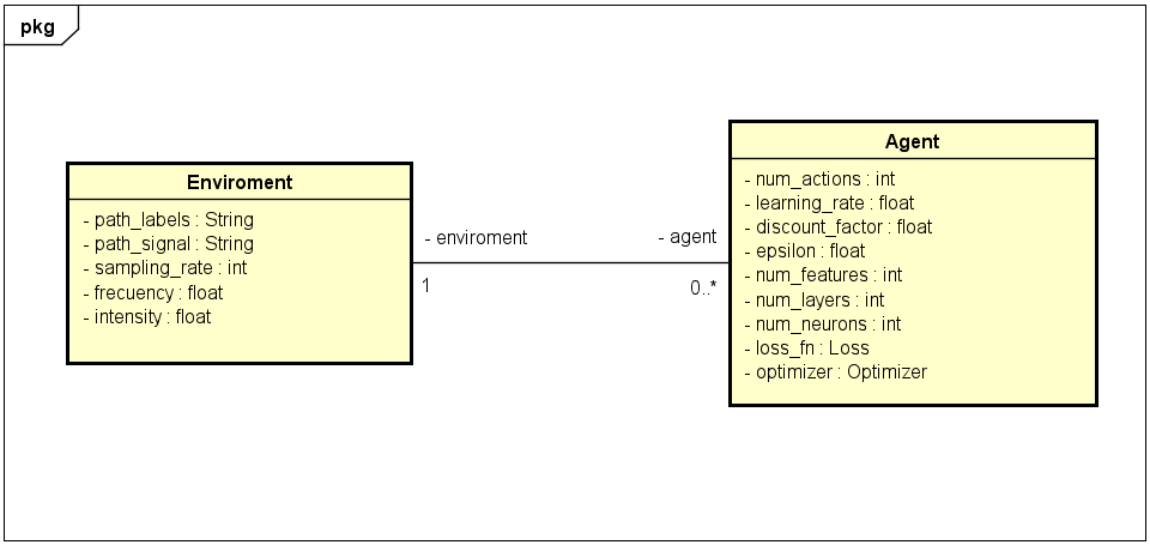



Figura 6.1: Diagrama de Dominio

Como se puede observar, el diagrama esta compuesto de dos clases, el entorno, el cual se encarga de modelar en este caso tanto la obtención de los datos como la generación de las ondas eléctricas. Para ello, tiene como atributos las direcciones donde se encuentran los datos obtenidos, ratio de muestreo, y las intensidades y frecuencias de las señales generadas.

Por otro lado, el agente dispone de los atributos correspondientes a la estructura de la red neuronal, así como los hiperpárametros del entrenamiento del modelo y su optimizador. También cuenta con la función de cálculo de pérdida.

Cabe destacar que se podría tener a varios tipos de modelos entrenando bajo el mismo entorno, pero esto no está contemplado en los objetivos del proyecto.

## 6.4. Diagrama de Clases de Análisis

En el diagrama de clases de Análisis (Ver. Figura 6.2) se muestran las operaciones que deben ser implementadas para el funcionamiento del sistema. En la clase de entrono se encuentran la función de obtención del directorio en el que se encuentran los ficheros y la función de carga de los datos de los archivos en memoria. Por otro lado, en la clase del agente están las funciones de creación y actualización del modelo, además de las operaciones de selección de acciones y tratamiento de valores de entrada.

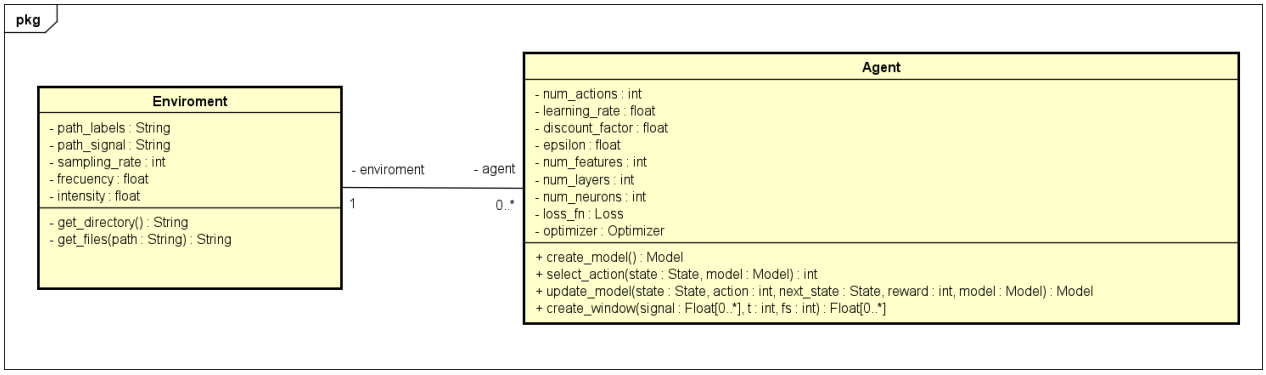



Figura 6.2: Diagrama de Clases de Análisis

## 6.5. Diagrama de decisiones del sistema

A continuación, se muestra un diagrama de flujo de la Figura 6.3 el conjunto de decisiones que debe tomar el sistema durante la ejecución. En el diagrama destacan los nodos de decisión encargados de evaluar si la entradas representan un estado epiléptico del paciente, así como de analizar si dicho estado es reversible o si las estimulaciones que se han enviado han sido efectivas. Por otro lado, se encuentran las acciones correspondientes a la evaluación de los datos, la monitorización, la generación de las señales de estimulación y el ajuste de parámetros del modelo.

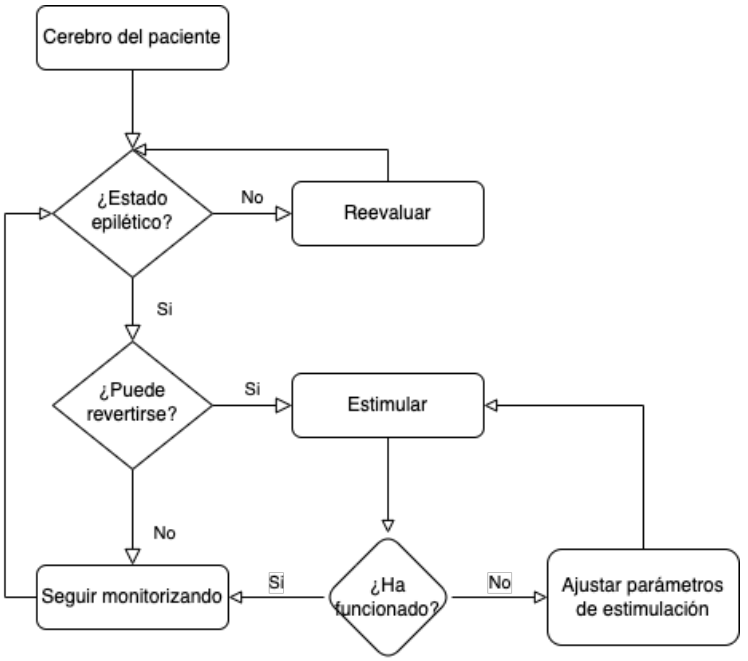



Figura 6.3: Diagrama de flujo del sistema

## 6.6. Diagrama de actividad

Como último apartado del análisis, se muestra un diagrama de actividades (Ver. Figura 6.4) que representa el procedimiento del bucle general del sistema, desde la observación del estado actual hasta la actualización de los parámetros del modelo.

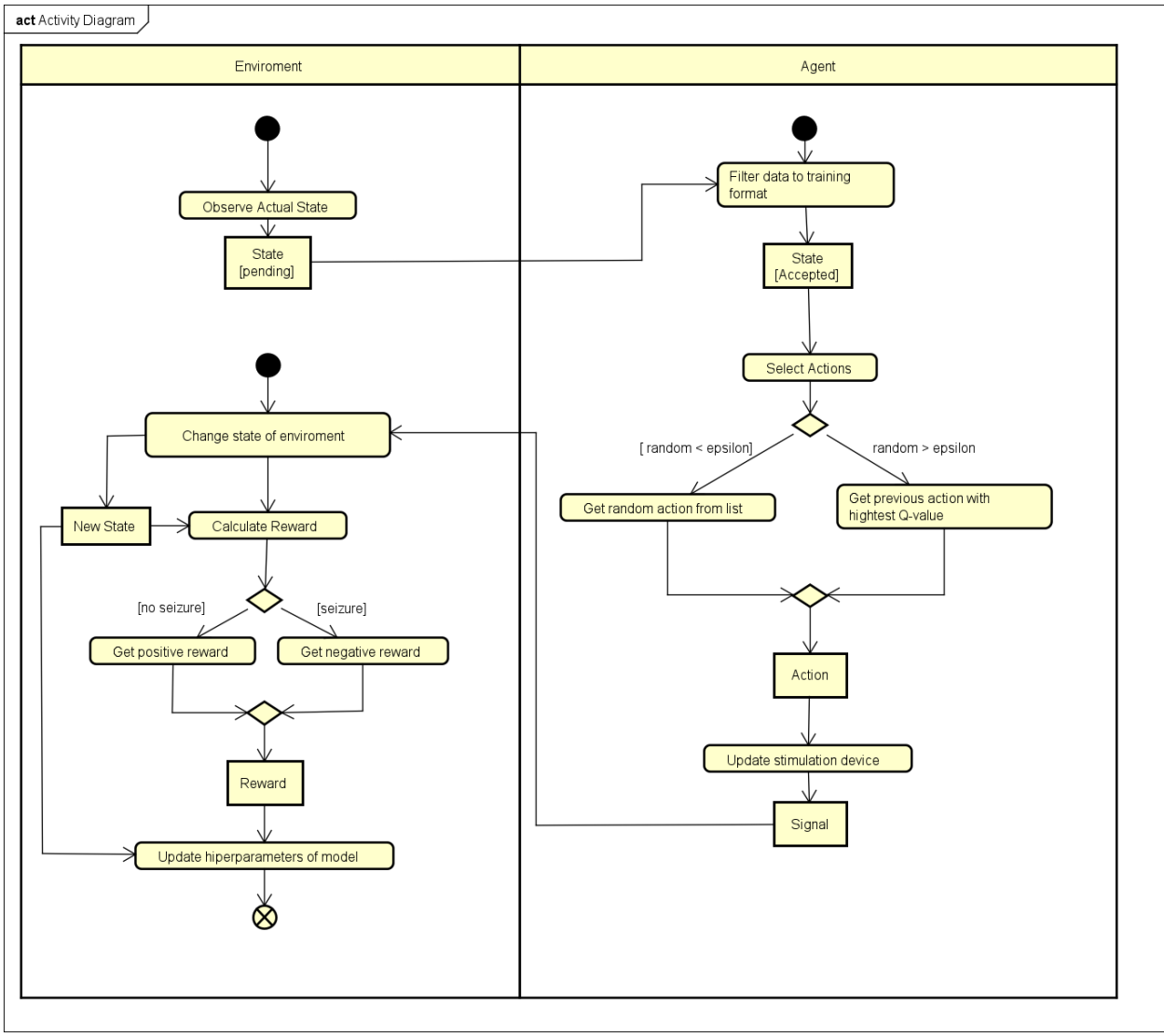



Figura 6.4: Diagrama de Actividad

# Capítulo 7

# Diseño

En esta sección, se muestran las decisiones de diseño que se han tomado en el desarrollo del proyecto, con el objetivo de cumplir con los requisitos establecidos anteriormente. A partir de una serie de diagramas, se especifica la arquitectura del sistema, así como las dependencias que este mismo tiene con las librerías externas, ya sean de análisis de datos o de aprendizaje por refuerzo.

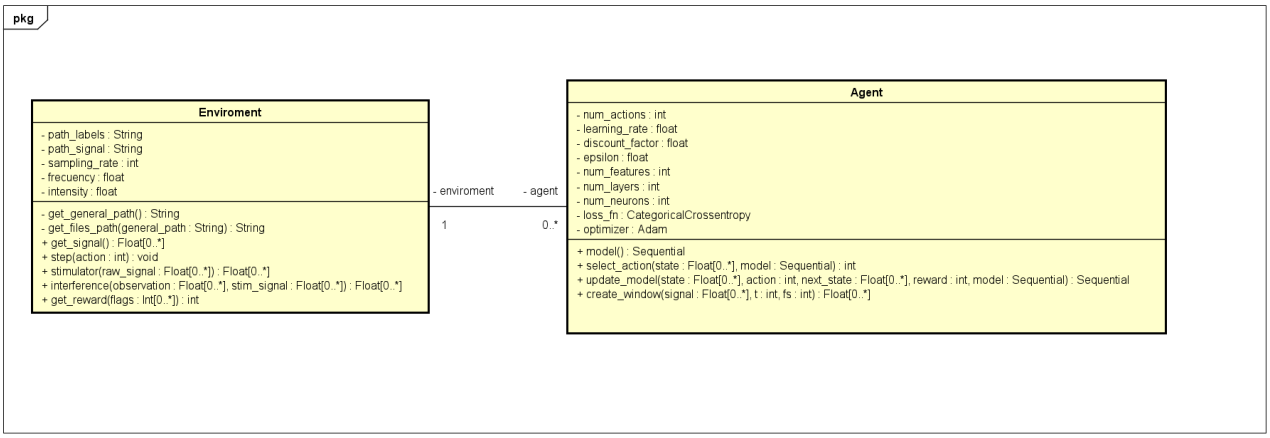



Figura 7.1: Diagrama de clases de diseño

Este diagrama de clases de diseño (Ver. Figura 7.1) describe la relación de asociación entre la clase de entorno y del agente. Además, se detallan las operaciones de cada elemento, anotando los parámetros de cada función y su valor de retorno.

Por otro lado, se puede observar que la clase agente tiene una dependencia sobre las clases externas provenientes de la librería de TensorFlow, y en consecuencia sobre la de Keras. Estas clases externas se utilizan para instanciar los optimizadores de los parámetros del modelo

(tf.keras.optimizers.Adam) así como la función de pérdida (tf.keras.losses.CategoricalCrossentropy), mientras que la última permite la creación del modelo secuencial (tf.keras.models.Sequential). Dichas dependencias serán señaladas con detalle a continuación.

## 7.1. Diagrama de paquetes

En este diagrama de paquetes de la Figura 7.2 se puede observar la estructura del sistema, así como las dependencias del paquete Agente con la librería de Keras. Dado que Keras es una librería de alto nivel que opera sobre TensorFlow, esta se encuentra contenida dentro de paquete. De igual forma, ambos paquetes son externos al sistema. Así mismo, también está representado el paquete SignalSimulator, con su correspondiente clase Eeg_gen, la cual actúa como generador de señales cerebrales simuladas para el entrenamiento del modelo. Por último, el agente depende del entorno para realizar su funcionalidad.

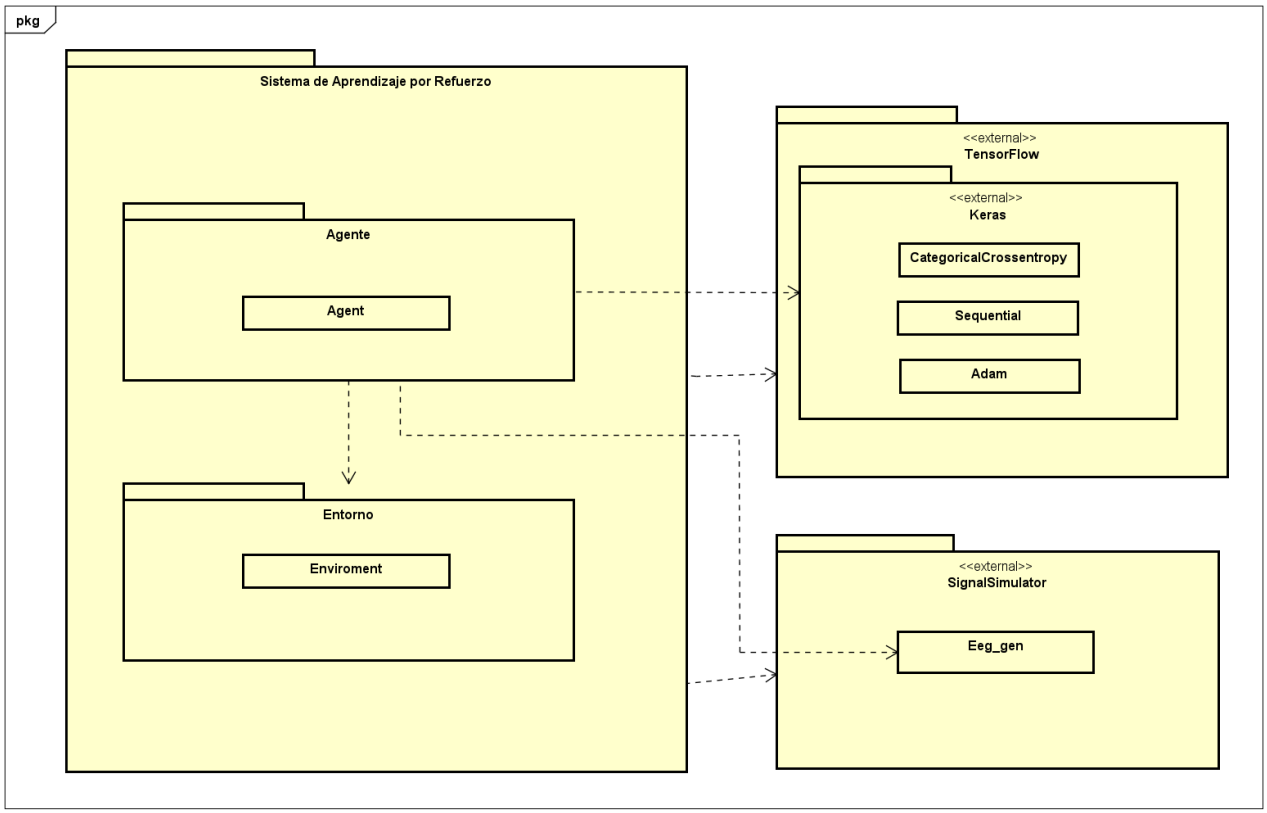



Figura 7.2: Diagrama de paquetes.

# Capítulo 8

# Tecnologías utilizadas

## 8.1. Numpy

Numpy [44] (Ver. Figura 8.1) es una librería de código abierto que permite la computación científica para el lenguaje de programación de Python. Numpy fue desarrollado en 2005 por Travis Oliphant, el cual era un científico de datos y desarrollador de software que tenía como objetivo implementar una forma sencilla de trabajar con vectores extremadamente grandes y realizar operaciones matemáticas complejas sobre ellos.


Figura 8.1: Logotipo de Numpy. [9]

En la actualidad, la librería es desarrollada por una comunidad de programadores internacionales, y se encuentra como uno de los paquetes más populares para el análisis de datos de forma global.

Esta librería ofrece una variedad de funcionalidades [45] como son:

1. **Matrices y Arrays multidimensionales**:

   Numpy permite la creación de objetos de arrays, llamados ndarrays, con un tamaño fijo y un tipo de datos seleccionados para todos los elementos que lo componen. Esto permite que los elementos estén guardados en espacios continuos de memoria ya que tienen el mismo tamaño.

   Existen varias formas de crear arrays con Numpy, sin embargo las que han sido utilizadas en el proyecto son las siguientes:

   - **np.zeros**: Creación de arrays inicializados a cero mediante funciones intrínsecas de Numpy.

- **np.array** Creación de arrays de n-dimensiones mediante la conversión de estructuras de Python

2. **Operaciones matemáticas y estadísticas sobre los arrays**: Permite realizar operaciones matemáticas complejas sobre los elementos de los arreglos, entre ellas destacan funciones de álgebra lineal, la trasformación de Fourier y estimaciones estadísticas, como pueden ser las siguientes:

   - **np.argmax**: Devuelve el índice correspondiente al valor máximo que se encuentra a lo largo del eje del array.
   - **np.random.uniform**: Crea un array obteniendo valores de una distribución uniforme, especificando un máximo y un mínimo.

3. **Indexación y segmentación de arrays**: Hace posible un acceso rápido y eficiente a elementos específicos de las matrices, además de permitir aumentar su tamaño inicial si es necesario insertar nuevos elementos durante la ejecución.

4. **Entrada y salida de datos**: Implementa la lectura y escritura de ficheros de una forma más eficiente que con las funciones estándar de Python.

5. **Integración con otros paquetes**: Compatibilidad con otras librerías de Python como son Pandas y TensorFlow.

## 8.2. TensorFlow

TensorFlow [46](Ver. Figura 8.2) es una librería de código abierto, desarrollada en 2015 por el Google Brain Team, para el lenguaje de programación Python, que implementa la programación de redes neuronales y la creación de modelos de aprendizaje automático. Esta librería cuenta con una gran variedad de funciones que permiten simplificar la programación de modelos de deep learning y hace más accesible su entrenamiento aportando algoritmos de optimización y funciones de cálculo de costes.

Figura 8.2: Logotipo de Tensorflow. [10]

TensorFlow aporta muchas funcionalidades para el desarrollo de inteligencias artificiales, como son una gran versatilidad de plataformas en las que puede ejecutarse, pudiendo ser utilizado tanto en dispositivos móviles como en la nube. Así mismo, permite la creación de tensores y de grafos computacionales, que constituyen los modelos.

Esta plataforma se utiliza frecuentemente en conjunto con la API de Keras para una mayor versatilidad.

## 8.3. Keras

Keras [47](Ver. Figura8.3) es una API de alto nivel de aprendizaje profundo escrita en Python, que se encuentra implementada dentro de la plataforma de Tensorflow. Esta librería proporciona un nivel de abstracción a la hora de crear y entrenar redes neuronales profundas, a través de funciones preestablecidas, facilitando la implementación de soluciones en las que se utiliza la inteligencia artificial.

Figura 8.3: Logotipo de Keras. [11]

Los modelos construidos mediante la librería se basan en combinar distintos tipos de capas que forman las redes neuronales, disponiendo de dos modelos preestablecidos dependiendo de su complejidad:

- **tf.keras.Sequential**: El modelo Secuencial [48] consta de una pila lineal de capas que tienen un único valor de entrada y de salida. Dadas las características del proyecto, se ha elegido este tipo de modelo.
- **Modelo funcional**: Modelos más complejos, con capas no lineales y con varias entradas y salidas.

En cuanto a los tipos de capas proporcionadas, destacan las capas densamente conectadas, en las que todos los nodos de la capa actual están relacionados con cada uno de la capa anterior. En la librería de Keras se encuentra la siguiente función de creación:

- **tf.keras.layers.Dense**: Crea una capa densa [49] que se añade al modelo anteriormente creado, tomando como parámetros el número de neuronas de que contiene y la función de activación que ejecutan.

Por último, Keras implementa funciones características de algoritmos de aprendizaje por refuerzo, como son el cálculo de las pérdidas [50] y los optimizadores [51]. Se distinguen las siguientes funciones:

- **tf.keras.losses.CategoricalCrossentropy**: se utiliza para calcular la diferencia de la intersección entre las predicciones y las etiquetas.
- **tf.keras.optimizers.Adam**: implementa el algoritmo de optimización Adam en atributos de la red neuronal para reducir la pérdida generada y mejorar el modelo.

## 8.4. Scipy

SciPy [52](Ver. Figura 8.4) es una librería de uso gratuito para Python basada en las estructuras de datos de Numpy, permitiendo a los programadores realizar funciones matemáticas sobre matrices y vectores de gran tamaño. En especial se utiliza para implementar algoritmos de optimización, álgebra lineal e interpolación dentro de problemas estadísticos.

Figura 8.4: Logotipo de Scicy. [12]

En este proyecto se ha utilizado la siguiente funcion de Scipy:

- **scipy.io.loadmat**: Permite cargar el contenido de un archivo de formato .mat de MathLab en forma de un array de Numpy dentro de una variable de Python.

## 8.5. Gitlab

Gitlab [53](Ver. Figura 8.5) es un servicio de repositorios de código abierto utilizado para el desarrollo de software. Esta aplicación permite a los usuarios gestionar de forma visual, a través de una interfaz de usuario, repositorios de Git. De esta forma, todas la operaciones que se realizarían a través de la consola de comandos, como son crear un repositorio o unir y crear ramas, pueden realizarse desde el navegador en la página de usuario de Gitlab.


Figura 8.5: Logotipo de GitLab. [13]

En concreto para este proyecto se ha utilizado el repositorio de GitLab ofrecido por la Escuela de Ingeniería Informática de Valladolid.

## 8.6. Overleaf

Overleaf [54](Ver. Figura 8.6) es un software de edición de documentos en la nube basado en LaTex. Este permite crear y editar archivos de texto online, además de poder importar plantillas predefinidas a proyectos nuevos o exportar documentos a formato PDF.

Para este proyecto se ha utilizado Overleaf para redactar y editar la documentación.


Figura 8.6: Logotipo de Overleaf. [14]

## 8.7. Matlab

Matlab [55](Ver. Figura 8.7) es una aplicación de software y un lenguaje de programación creado por MathWorks, con el objetivo de facilitar cálculos numéricos complejos a los programadores.

Mediante su uso se pueden realizar con facilidad operaciones de análisis de datos como son el análisis e implementación de algoritmos, creación de gráficos o manipulación de matrices de gran tamaño a través de una interfaz gráfica comprensible para el usuario.

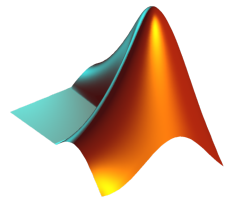


Figura 8.7: Logotipo de Matlab. [15]

En especial destaca por su eficiencia a la hora de realizar operaciones matriciales, por lo que es comúnmente utilizado para el procesamiento de señales, como en este proyecto, así como para imágenes o datos estadísticos.

# Capítulo 9

# Diseño Experimental

Una vez presentados en capítulos anteriores la estructura de los datos utilizados como entrada en el algoritmo de control, se va a proceder a explicar las decisiones tomadas a la hora de configurar cada experimento realizado.

Se debe tener en cuenta que el objetivo de la ejecución de las pruebas es concluir para qué valores de los parámetros el modelo resulta más productivo, distinguiendo después un modelo optimizado sobre el que realizar un análisis de eficiencia.

Cabe destacar que, debido a la gran cantidad de datos que se está manejando durante el entrenamiento del modelo, es necesario ejecutar las pruebas en un sistema con un gran poder de computación. Por lo tanto, cada experimento se ha realizado gracias al centro de Supercomputación Castilla y León (SCAYLE)[19] ya que el código de cada prueba se ha procesado dentro de sus servidores, reduciendo significativamente el tiempo empleado.

## 9.1. Configuración del Sistema

Para la realización de cada experimento se han modificado de forma iterativa, en ensayos aislados, tanto los parámetros de aprendizaje del modelo, como la propia topología de las capas y número de neuronas que conforman su red neuronal. Estos ajustes provocan efectos en el rendimiento y la capacidad de aprendizaje del modelo, ya sean positivos o negativos dependiendo de la naturaleza de cada parámetro.

La configuración de cada experimento es individual y aislada del resto, ya que se realizan con distintos conjuntos de valores. Antes de comenzar una prueba, se selecciona un único parámetro al cual aumentar o disminuir su valor actual, mientras que el resto se mantienen como se encontraban anteriormente. De esta forma, se puede observar de forma individual el efecto que causa la modificación de esa característica sobre el aprendizaje del modelo.

Los valores que se han asignado por defecto a los parámetros cuando no están siendo estudiados son:

1. Tasa de aprendizaje: 0,001
2. Tasa de descuento: 0,99
3. Número de capas: 2
4. Número de neuronas en capas ocultas: 32
5. Tipo de capa: Capas densas.

Por otro lado, el cambio de magnitudes de los parámetros está programado de forma iterativa, modificando su valor dentro de un rango a lo largo de seis iteraciones.

Los parámetros del sistema que han sido ajustados son los siguientes:

**Tasa de aprendizaje[56]** Parámetro que define la rapidez y el tamaño de los ajustes que se realizan a los pesos del modelo.

Es necesario elegir su valor con precaución, ya que si se selecciona una magnitud demasiado alta, los parámetros serán actualizados aplicando pasos demasiado grandes y rápidos. Esto estabiliza el modelo en un corto espacio de tiempo, pero cabe la posibilidad que pase por alto la solución más óptima.

Por otro lado, si su valor es bajo, el entrenamiento llevará más tiempo, pero se obtendrá una solución más precisa.

Los valores de la tasa de aprendizaje seleccionados correspondientemente para cada iteración fueron: {0,001, 0,002, 0,003, 0,004, 0,005, 0,006 }. En este conjunto se incrementan un 0,001 cada magnitud en cada iteración.

**Tasa de descuento[57]** Parámetro que determina la importancia que el modelo asigna las recompensas inmediatas frente a las que se obtendrán a largo plazo. Se encuentra en un rango entre cero y uno, causando que cuanto mayor sea su valor, más se preocupará el modelo por las recompensas a largo plazo, mientras que si es cercano a cero, se centrará en una perspectiva a corto plazo.

Los experimentos se han realizado con tasas de descuento con las siguientes magnitudes: { 0,99, 0,90, 0,85, 0,80, 0,75, 0,70 }

**Número de neuronas** Indica el número de neuronas que forman cada capa oculta de la red neuronal del modelo, ya que las neuronas de las capas de entrada y salida se mantienen constantes. La capa de entrada tiene un total de nodos equivalente a la longitud de la secuencia de datos creada por el generador, siendo por lo tanto de 5000 neuronas. Así mismo, la capa de salida se compone de tantas neuronas como tipos de acciones puede realizar el modelo, siendo en este caso un total de dos. Al incrementar la cantidad de neuronas aumenta la capacidad de abstracción de la red, pero puede darse una situación en la que la red se vuelve demasiado compleja para hallar patrones en los datos de entrada de forma eficiente, creando una situación de sobreajuste.

Se realizaron las pruebas multiplicando por dos el número de neuronas de la iteración anterior, siendo: { 32, 64, 128, 256, 512, 1024 }

**Número de capas** Cantidad de capas que forman la red neuronal, a mayor número de capas mayor capacidad de procesamiento, pero requiere una mayor capacidad de recursos del sistema y puede darse un caso de sobreajuste del modelo.

Se han utilizado los siguientes números de capas, incluyendo tanto la capa de entrada como de salida: { 2, 3, 4, 5, 6, 7}

**Tipo de capa** Se distinguen dos tipos de capas diferentes que se han utilizado en las pruebas, estas son las capas densas y las capas convolucionales.

Las capas densas[58] son aquellas en las que cada neurona de una capa esta conectada con todas la neuronas de la capa anterior, y con todas las de la siguiente, es decir, cada neurona de la capa actual recibe y procesa las salidas de todas la neuronas de la capa previa y envía su resultado a cada neurona de la capa que se encuentra a continuación. Las únicas excepciones a esta regla son las capas de entrada y salida. La capa de entrada, aunque esta completamente conectada con la primera capa oculta, recibe sus entradas del exterior, mientras que la capa de salida no envía sus valores a otra capa, sino fuera de la red.

Por otro lado, se encuentran las capas convolucionales[59], las cuales pueden procesar de forma rápida datos estructurados en matrices o vectores, extrayendo sus características fundamentales de forma progresiva. A medida que la información va siendo enviada de capa a capa, se van obteniendo las ideas fundamentales de ese conjunto de datos, las cuales se hacen más complejas en cada paso.

El objetivo de cambiar el tipo de capas utilizadas es estudiar las diferencias en el comportamiento del modelo y determinar con qué capas es más eficiente .

Además de los parámetros anteriores, se encuentran elementos de configuración cuyos valores son constantes para cada experimento realizado.

**Tamaño de ventana[60]** Como se ha descrito en el capítulo de datos encefalográficos, las señales cerebrales que se utilizan como entrada se deben dividir en ventanas para su procesamiento. El tamaño de esta ventana especifica el número de datos que se seleccionarán de la señal original al ser multiplicada por la ventana de Hanning. Si este valor es pequeño, conllevará un análisis localizado de los cambios de la señal en un corto espacio de tiempo, mientras que si es una magnitud grande, será capaz de detectar tendencias generales de un periodo de tiempo más grande. Se ha utilizado un tamaño de ventana: $t = 50$.

**Frecuencia inicial** Describe la frecuencia inicial de la señal de entrada. Para cada experimento su valor comienza en 120Hz, aumentando o disminuyendo en cada iteración en función de las acciones elegidas por el modelo y del ruido aplicado a la frecuencia actual.

En los dispositivos de estimulación cerebral profunda se definen dos rangos de estimulación, uno de alta frecuencia, más de 100Hz, y uno de baja frecuencia, menos de 100Hz[61]. Debido a que la mayoría de dispositivos utilizan la alta frecuencia, se ha decidido comenzar con una frecuencia de 120Hz para favorecer que esta se encuentre en ese rango la mayor parte del tiempo, a pesar de las variaciones del estimulador.

Además, se han establecido unos límites superior e inferior para garantizar que el valor de la frecuencia se mantiene en un rango real a las frecuencias naturales del cerebro. Por lo tanto, su valor se encuentra siempre en un rango entre 75Hz y 200Hz.

**Intensidad inicial** Expresa la intensidad inicial de las señales de entrada. En cada prueba su valor se inicializa a 20V, aunque esta magnitud varía en función de la selección de acciones del modelo, ya que aumenta o disminuye al aplicarse un ruido externo a la intensidad actual. Sin embargo, aunque en dispositivos de control similares se utilizan intensidades entre 4V y 10V, el que se usa en este proyecto no es lo suficientemente sensible como para responder a intensidades tan bajas, por lo que se ha configurado a una intensidad de 20V.

**Frecuencia de muestreo[62]** Constituye el número de puntos de una señal que se obtienen por segundo. Esto se debe a que, a diferencia de las señales analógicas, las cuales son continuas, las señales digitales deben dividirse en valores discretos, llamados muestras. Por lo tanto, cuanto mayor sea el valor de la frecuencia de muestreo, mayor será el número de puntos recogidos y más se ajustará la representación de la amplitud y forma de la señal a la original.

En este proyecto se ha calculado la frecuencia de muestreo mediante la siguiente formula: $fs = 1/T$ donde T corresponde con el intervalo de muestreo, es decir, el tiempo que transcurre entre la adquisición de cada muestra, definido a $T = 0{,}0005s$. Por lo tanto el valor de la frecuencia de muestreo es $fs = 2000Hz$.

**Valor de recompensa** Define la magnitud de la recompensa obtenida por el modelo después de ejecutar una acción sobre el entorno, siendo positiva para las acciones que se quieren reforzar y negativa para las que no deben aplicarse. En la mayoría de algoritmos de control de dispositivos médicos se aplicarían recompensas positivas únicamente si no se ha dado ningún caso de ataque. Sin embargo, para agilizar el proceso de entrenamiento, se ha optado por dar valores entre el -1 y el 1 proporcionales al número de indicadores de actividad normal que se encuentran en la muestra actual de la señal.

**Número de características** Cantidad de características de las entradas estudiadas por el modelo, en este caso únicamente se analiza la entropía espectral, la cual se explicará más adelante.

**Número de acciones** Cantidad de acciones que puede realizar el modelo sobre el entorno, en este caso, son solamente dos, definidas como la modificación de la frecuencia o de la intensidad de la señal actual.

## 9.2. Entropía espectral

Es importante destacar el uso de la entropía espectral a la hora de analizar los valores de las señales de entrada.

La entropía espectral[63] en el contexto del análisis de señales es una medida que caracteriza la cantidad de estructura u orden que se encuentra en la distribución de los valores de una señal en el dominio de la frecuencia.

De esta forma, cuando la entropía espectral tiene un valor bajo, indica que hay poco orden en la señal. Esto se debe a que no hay una distribución uniforme de la energía en todo el espectro de la frecuencia, haciendo que la señal esté más distribuida y que no presente picos o frecuencias dominantes.

Por otro lado, si la entropía espectral aumenta, representa que la señal tiene un mayor grado de orden y estructura, haciendo que su energía este distribuida de forma uniforme. En este caso, la potencia de la señal tiende a presentar frecuencias dominantes.

En el caso de los datos encefalográficos (EEG) se ha observado una distribución no equitativa, ya que la energía de la señal se concentra más en las frecuencias bajas que en las altas. Al estar cerca de un ataque las señales cambian, haciendo que la potencia este más ordenada. Esto se debe a que pasan a favorecerse las frecuencias altas mientras las bajas disminuyen, causando la presencia de frecuencias dominantes. Todo esto tiene como consecuencia un aumento de la entropía espectral.

Teniendo en cuenta este comportamiento, se ha decidido utilizar la entropía espectral de los datos de entrada como característica a estudiar por el modelo. Por lo tanto, en cada iteración en la que el modelo obtiene los datos de la señal mediante una ventana de Hanning, se aplica a dicho vector de valores una función de cálculo de la entropía espectral.

## 9.3. Etapas y conjuntos de datos experimentales

Dentro de la fase de experimentación se pueden distinguir tres etapas: entrenamiento del modelo, validación y pruebas.

En cada experimento, una vez modificados los parámetros del modelo y enviado el programa al servidor, se entrena cada modelo, analizando después las características de rendimiento que ha presentado en su ejecución. Al hacer esto de forma iterativa para cada ajuste de parámetros descritos anteriormente, se obtiene un conjunto de resultados que representan el comportamiento del modelo para cada ajuste.

A continuación, se lleva a cabo una comparación de todos los modelos, distinguiendo el valor de cada parámetro con el que el modelo trabaja de forma más eficiente. Una vez se ha asignado un valor óptimo a cada parámetro se entrena un modelo final con esos ajustes, sobre el que después se hará un análisis de rendimiento en la fase de prueba.

Para llevar acabo todo esto se han utilizado como entrada varios conjuntos de datos, entre los que se distinguen dos tipos, los datos reales provenientes de ficheros que se encuentran dentro del servidor durante la ejecución, y los que son creados dinámicamente en cada iteración por el generador.

Los datos de los archivos se dividen en dos partes, siendo el 70 % de ellos orientados al entrenamiento del modelo, mientras que el 30 % restante se utiliza para la primera iteración de la validación. Esto se debe a que para el resto de iteraciones se emplean los datos creados por el generador.

El funcionamiento del generador consiste en simular el funcionamiento del cerebro del paciente, para ello se compone de neuronas configuradas de forma que su comportamiento

se aproxime a las neuronas reales de un paciente con enfermedades neuronales. Además, el generador es capaz de calcular que efecto tendría sobre el cerebro la señal eléctrica enviada por el dispositivo, de forma que, si una misma señal se repite de forma constante, el generador entiende que el cerebro se acostumbraría a esa intensidad y modifica los datos de salida para que no se reduzcan tantas señales de ataque como sería normal.

De esta forma, se imita el procedimiento de obtención de la información en tiempo real, es decir, el generador actúa como si fuese el cerebro del paciente enviando señales al dispositivo de control. Así mismo, se encuentran los datos de prueba con los que se mide el rendimiento del modelo óptimo, los cuales también son creados por el generador.

Cabe destacar que la duración total de un experimento es de 15000 iteraciones, ya que para realizar el procesamiento de datos se realizan 1500 iteraciones, en cada una de las cuales el generador crea datos nuevos, que son procesados en pequeños conjuntos a través de un bucle anidado de 10 iteraciones.

## 9.4. Métricas de rendimiento

Con el objetivo de evaluar y comparar el rendimiento del conjunto de modelos creados, se han elegido una serie de métricas:

**Velocidad de aprendizaje** Se define como la rapidez a la que el modelo puede mejorar su rendimiento mediante su interacción con el entorno y obtención de recompensas. En concreto para este proyecto se considerará como el número de iteraciones que ha necesitado un modelo para converger.

Un modelo de aprendizaje por refuerzo converge cuando alcanza un estado óptimo y estable, es decir, cuando a pesar de ajustar los parámetros del modelo o aumentar la cantidad de iteraciones de entrenamiento, no se observan cambios significativos en su rendimiento. Por lo tanto, se entiende que el modelo ha conseguido optimizar sus políticas de decisión.

**Éxito acumulado[64]** Métrica que representa la suma del número de veces que el algoritmo ha elegido la acción correcta de la Q-table después de haber sido entrenado. Por lo tanto, un ratio de éxito alto indica que el modelo está obteniendo buenos resultados de forma estable, mientras que uno bajo indica que el modelo está teniendo dificultades para aprender o necesita más tiempo.

**Recompensa acumulada** Constituye la suma de las recompensas obtenidas por el modelo para una ejecución. Si el valor total obtenido es alto, quiere decir que el modelo ha tomado una gran cantidad de acciones acertadas. Esto se debe a que solo recibe recompensas positivas cuando, para el estado siguiente del entorno, se han encontrado un gran porcentaje de indicadores de actividad normal en la señal. Sin embargo, si el valor es pequeño, se considera que el modelo no está eligiendo buenas acciones ya que la proporción de indicadores normales es menor, por lo que no está contrarrestando de forma efectiva los ataques.

Todas estas métricas son calculadas durante la ejecución en el servidor, mediante la utilización de la librería Weights & Biases[65] que ofrece una interfaz con la que manejar los datos resultantes de cada uno de los experimentos, además de mostrar gráficos de la evolución de los modelos.

# Capítulo 10

# Resultados y Discusión

En este capítulo, se presentan los resultados obtenidos de cada uno de los experimentos, destacando la importancia del ajuste de los parámetros de entrenamiento de los modelos. Se estudiará por separado el efecto que tiene la variación de cada parámetro en el rendimiento del modelo, mediante la evaluación de las métricas de rendimiento anteriormente explicadas, como son la velocidad de aprendizaje, el ratio de éxito y la recompensa acumulada.

Por último, se analiza el rendimiento de un modelo final, el cual ha sido entrenado y evaluado con los ajustes de parámetros considerados más eficaces.

## 10.1. Resultados

Con el objetivo de facilitar el análisis y la comparación de los datos obtenidos, se han utilizado una serie de gráficos de líneas para cada parámetro, los cuales están dispuestos en dos grupos, correspondientes a las métricas de la recompensa y el éxito acumulado. En cada gráfico se señala en el título el parámetro al que se está haciendo referencia en ese apartado, incluyendo su valor actual, mientras que en el eje Y se detalla la magnitud del éxito o recompensa acumulado, y en el eje X se determina el número de epoch transcurridos durante el experimento.

Así mismo, para representar la velocidad de aprendizaje del modelo para cada parámetro, se han creado unos gráficos de barras. En ellos, se detalla en el título el parámetro actual, mientras que, en el eje Y se encuentra la velocidad de aprendizaje como el número de epoch transcurridos y en el eje X se indica la magnitud del parámetro utilizado en la prueba. Los gráficos se pueden ver desde la Figura 10.1 hasta la Figura 10.11.

Esta distribución tiene como finalidad destacar el efecto que tiene la variación de cada uno de los parámetros de forma individual, ya que únicamente se modifica el valor de un parámetro en cada experimento.

Además, los resultados representados en las gráficas corresponden a la fase de validación del modelo, en la que se toman como entrada los datos encefalográficos reales aportados por

el Air Institute, además de aquellos creados por el generador de señales importado en el proyecto.

### 10.1.1. Tasa de aprendizaje

Las gráficas correspondientes al ajuste de la tasa de aprendizaje se muestran de forma incremental para el rango de valores desde $alfa = 0,001$ hasta $alfa = 0,006$, siendo la duración total del experimento de 15000 epochs.

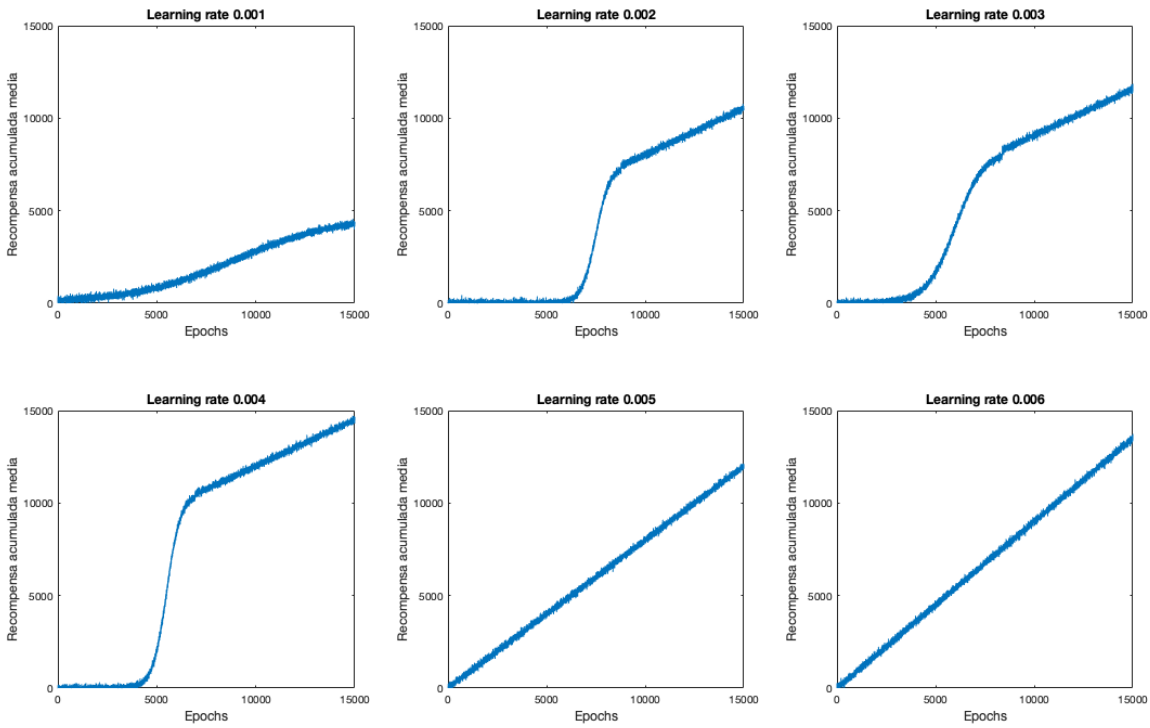



Figura 10.1: Recompensa acumulada para la tasa de aprendizaje.

La Figura 10.1 se representa el comportamiento de la recompensa acumulada para las variaciones de la tasa de aprendizaje. En primer lugar, para la magnitud mas pequeña se observa cómo la gráfica no alcanza valores de recompensa demasiado altos, ya que su crecimiento, aunque constante, es significativamente lento.

A continuación, desde la segunda hasta la cuarta gráfica se encuentra un cambio de tendencia. En estas tres funciones se calculan valores extremadamente bajos para las primeras 5000 iteraciones del algoritmo, para después incrementar rápidamente y finalizar con un crecimiento más prolongado hasta terminar la prueba. De esta forma, se puede distinguir un

crecimiento similar a un función sigmoide, que después comienza a adquirir un comportamiento lineal. En cuanto a la quinta y sexta gráfica, su forma es totalmente lineal para todo el experimento.

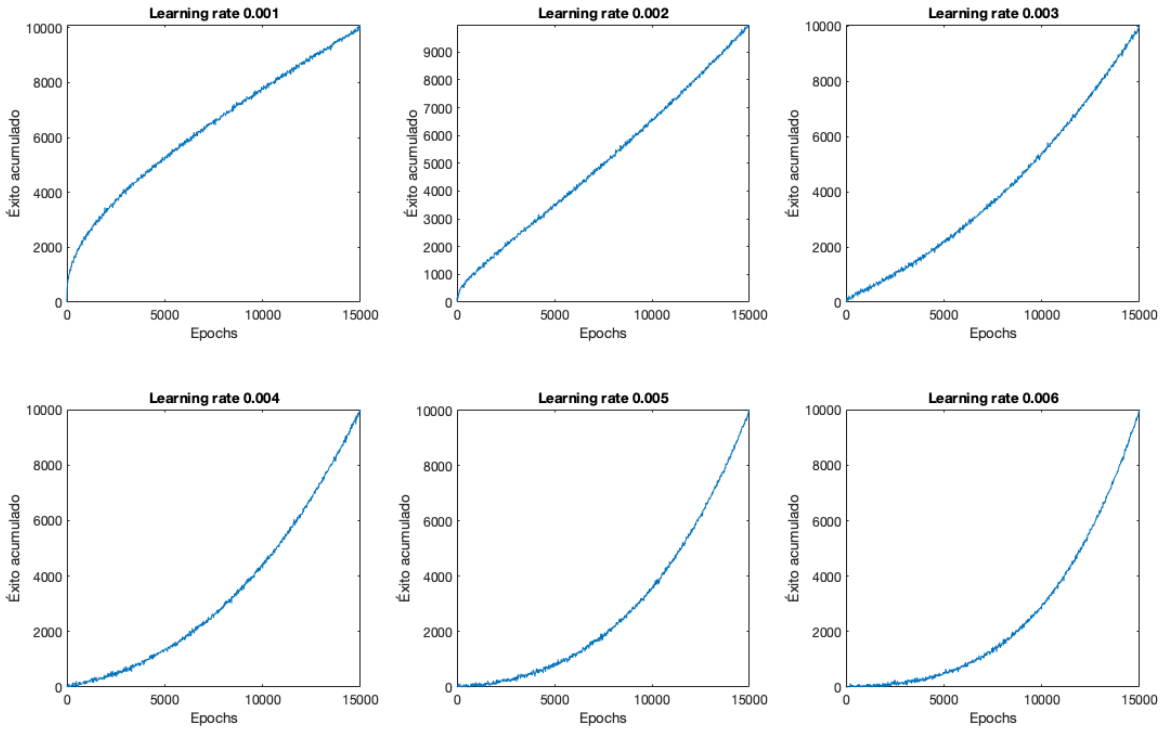



Figura 10.2: Éxito acumulado para la tasa de aprendizaje.

Para la tasa de éxito acumulado de la Figura 10.2, la tendencia general de todas las gráficas es creciente. Sin embargo, para el primer valor de la tasa de aprendizaje, se distingue un aumento rápido al comienzo del experimento.

Se puede observar que aumentar el valor de la tasa de aprendizaje provoca que el crecimiento del éxito acumulado pase de asemejarse a una función logarítmica a una exponencial. De esta forma, el éxito inicial de cada experimento crece de forma más lenta con cada aumento de la tasa de aprendizaje, pero termina incrementando rápidamente al final de cada experimento

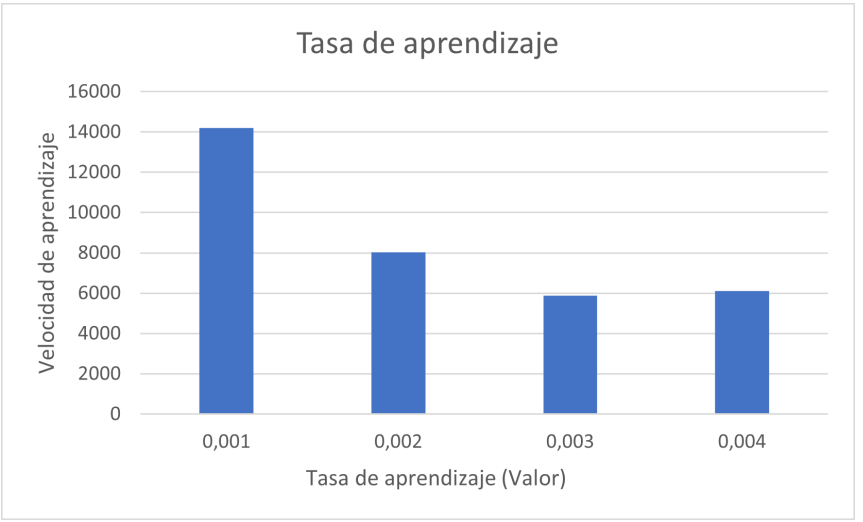



Figura 10.3: Número de epoch hasta convergencia para la tasa de aprendizaje.

En la Figura 10.3 se pueden observar los resultados obtenidos para la métrica de la velocidad de aprendizaje. Inicialmente se muestra que un incremento de la tasa de aprendizaje provoca una disminución considerable del número de epochs necesarios para que el modelo alcance la convergencia, alcanzando un mínimo para $alfa = 0,003$.

A partir de dicho valor, la velocidad de aprendizaje vuelve a crecer, causando que el número de epoch se dispare, llegando a provocar que para los dos últimos modelos, con unas tasas de $alfa = 0,005$ y $alfa = 0,006$ el modelo no llegue a converger. Esto quiere decir que su comportamiento nunca se estabiliza y seguirá creciendo independientemente de la duración del experimento. Debido a esta falta de convergencia se han omitido la representación de los resultados de las últimas pruebas.

### 10.1.2. Tasa de descuento

El rendimiento del modelo se representa en las siguientes gráficas, reduciendo la tasa de descuento desde un valor de $gamma = 0,99$ hasta $gamma = 0,07$.

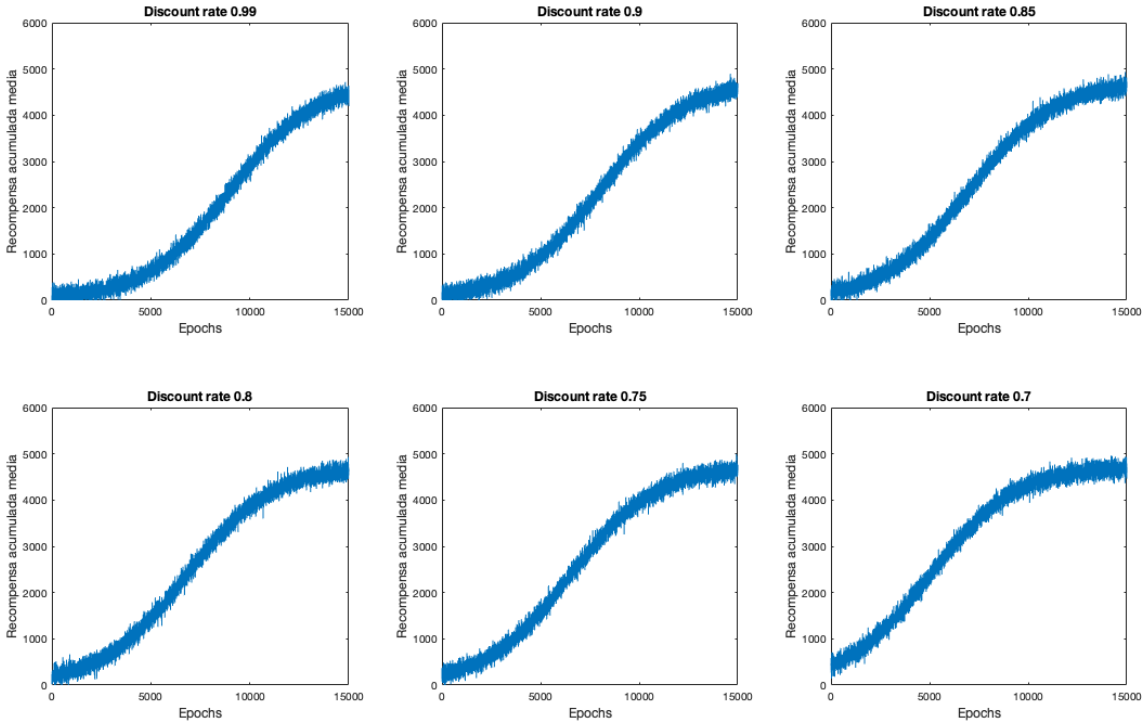



Figura 10.4: Recompensa acumulada para la tasa de descuento.

En el conjunto de gráficos de la Figura 10.4 se ve que el comportamiento de la recompensa acumulada en función de la magnitud de la tasa de descuento, es similar en todas la iteraciones.

Se caracteriza por iniciar la prueba con un crecimiento lento de las recompensas, que va aumentando con cada epoch transcurrido, hasta empezar a estabilizarse en las últimas iteraciones, siguiendo un comportamiento sigmoide. Sin embargo, en la última gráfica se encuentra una diferencia con respecto al resto, ya que los valores de recompensa iniciales son ligeramente mayores. Además, para esa tasa de descuento se llega a la cantidad más grande de recompensa total.

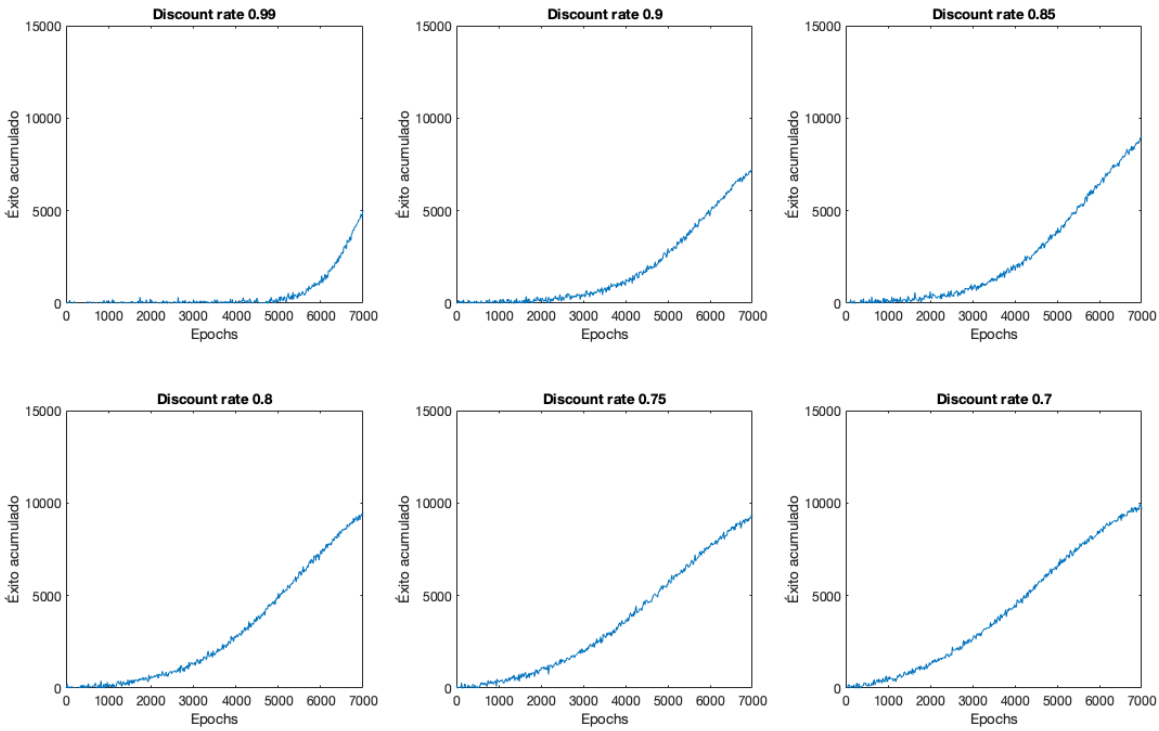



Figura 10.5: Éxito acumulado para la tasa de descuento.

Para la métrica del éxito acumulado representado en la Figura 10.5 la disminución de la tasa de descuento provoca un aumento del número de acciones correctas. En las gráficas se puede ver que a medida que se reduce el valor de la tasa de descuento, los valores iniciales de éxito acumulado crecen más temprano en la transcurso de la prueba.

Además, el éxito resultante aumenta progresivamente entre experimentos, ya que la diferencia de los valores acumulados del primer experimento y el último es de más del doble de aciertos.

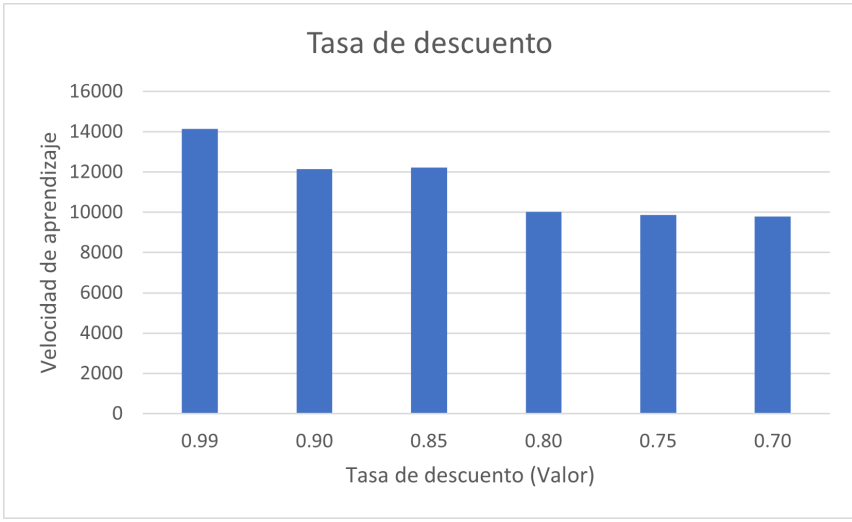



Figura 10.6: Velocidad de aprendizaje para la tasa de descuento.

El comportamiento de la velocidad de aprendizaje del modelo en función del valor de la tasa de descuento esta representada en la Figura 10.6. Se distingue una tendencia claramente descendente a medida que la tasa de descuento disminuye, por lo que el modelo tarda menos iteraciones en converger de forma progresiva.

Las variaciones más notables entre experimentos se encuentran entre los valores $gamma = 0,99$ y $gamma = 0,90$ donde el número de epoch disminuye en 2000 iteraciones. De igual forma, también ocurre otro descenso significativo para $gamma = 0,85$ y $gamma = 0,80$. Finalmente, el valor más pequeño alcanzado por la velocidad de aprendizaje se da para $gamma = 0,70$.

### 10.1.3. Número de neuronas

La variación del comportamiento del modelo respecto al número de neuronas de las que se compone cada una de sus capas se muestra para el rango desde 32 hasta 1024 neuronas, duplicando su cantidad en cada experimento.

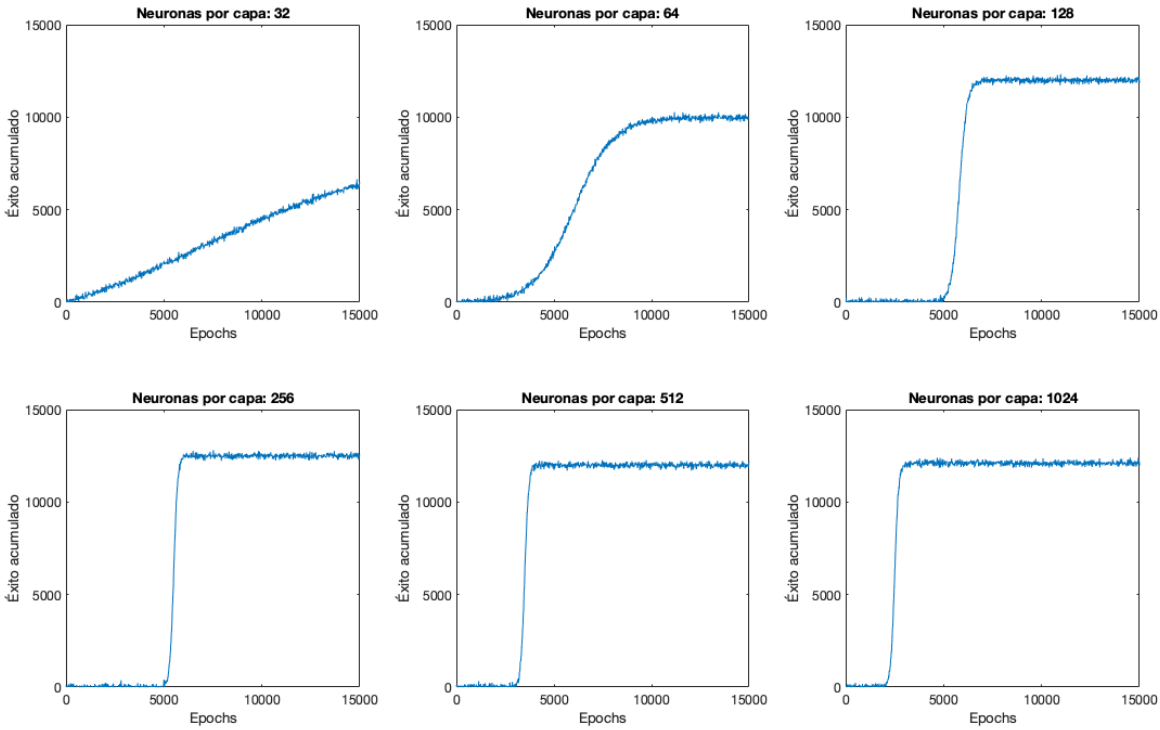



Figura 10.7: Éxito acumulado para el número de neuronas por capa.

El comportamiento del éxito acumulado para la variación del número de neuronas está representado en la Figura 10.7.

En la serie de experimentos se observa cómo para magnitudes mayores de 32 neuronas por capa se dan casos de sobreajuste del modelo a los conjuntos de datos de entrada. El fenómeno se caracteriza por valores de éxito extremadamente pequeños para las primeras iteraciones, seguido por un incremento brusco e inmediato, el cual resulta en una estabilidad constante de valores altos de éxito para el resto de la duración de la prueba.

Sin embargo, estos buenos resultados finales no se deben al correcto aprendizaje, sino a que el modelo ha conseguido memorizar el conjunto de datos empleado, por lo que únicamente es efectivo para estos valores de entrada, siendo incapaz de generalizar o encontrar patrones.

### 10.1.4. Número de capas

El rendimiento del modelo con respecto al número de capas que lo forman se expone en la Figura 10.8 y en la Figura 10.9. Para estos experimentos la cantidad de capas utilizadas se encuentra en el rango entre 2 y 7 capas.

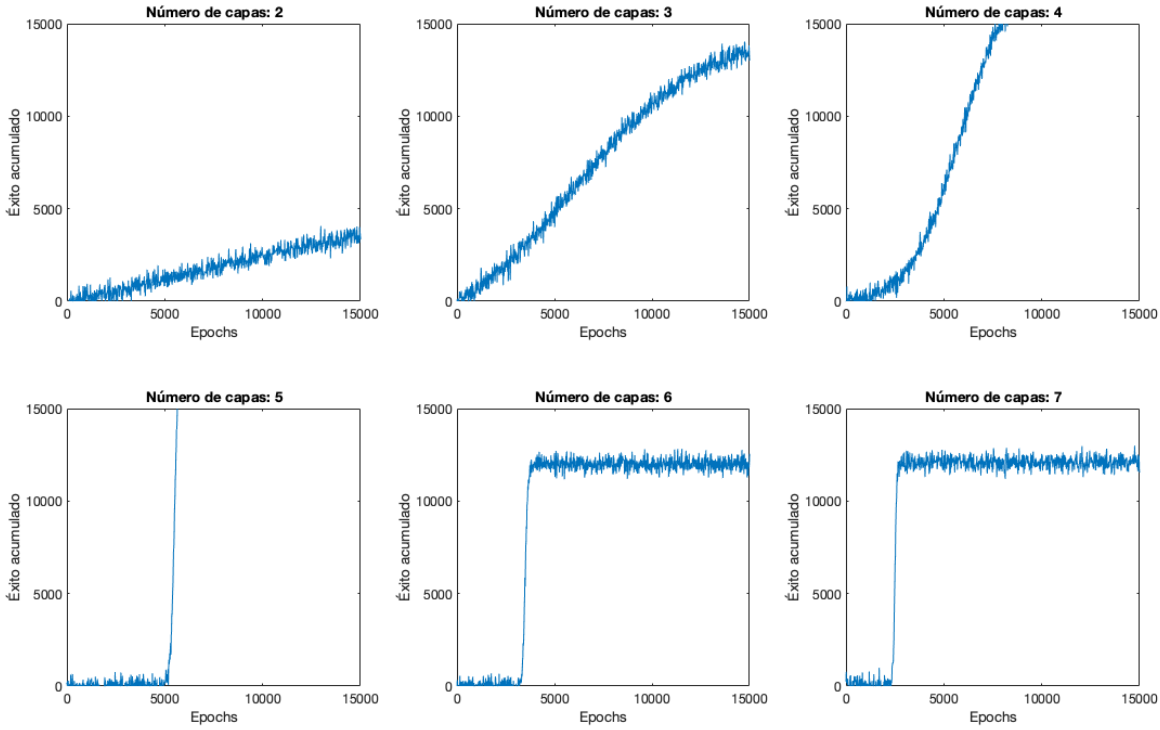



Figura 10.8: Éxito acumulado para el número de neuronas por capa.

El éxito acumulado durante estas pruebas se caracteriza por tener un incremento estable para los tres primeros experimentos, mientras que en los últimos casos se dan fenómenos de no convergencia y sobreajuste.

En primer lugar, para un modelo de dos capas el crecimiento es constante aunque no alcanza valores demasiado altos en comparación con el resto de gráficas. Después, para modelos con tres y cuatro capas se distingue la misma tendencia, ya que comienzan con valores de éxito pequeños, los cuales aumentan progresivamente hasta estabilizarse para el resto de la prueba. Cabe destacar que la cuarta gráfica se estabiliza más pronto y alcanza un mayor valor de éxito acumulado.

Así mismo, para el modelo de cinco capas se observa que el modelo no converge, es decir, no consigue llegar a un estado estable, ya que el crecimiento del éxito acumulado no es

constante a lo largo de la ejecución. Para los últimos experimentos su comportamiento es similar a los obtenidos para el número de neuronas, comenzando con valores bajos de éxito, los cuales aumentan repentinamente para después estabilizarse. Esto indica, al igual que en el apartado anterior, un caso de sobreajuste del modelo.

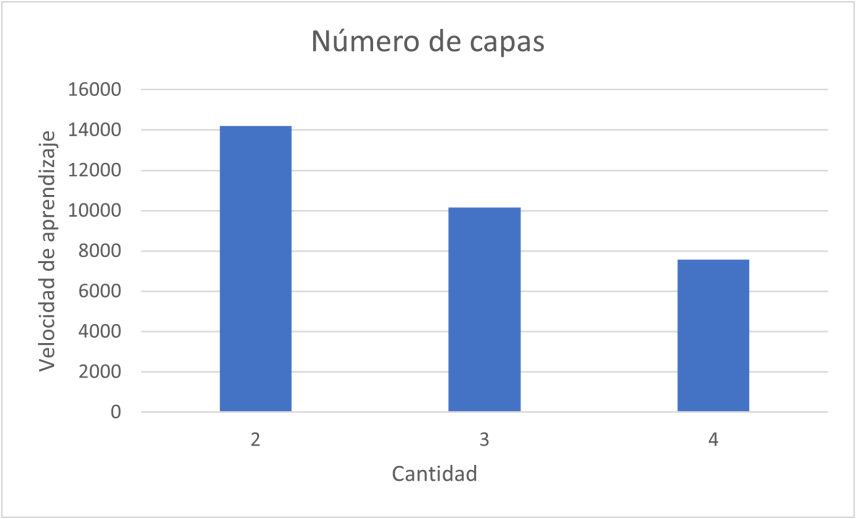



Figura 10.9: Velocidad de aprendizaje para el número de neuronas.

En cuanto a la velocidad de aprendizaje se han omitido las representaciones de los experimentos en los que se ha producido sobrejuste o no convergencia del modelo. La tendencia general analizada es que a medida que se aumenta el número de capas, disminuye la cantidad de epoch transcurridos hasta que el modelo converge.

### 10.1.5. Tipo de capas

Se han utilizado dos tipos distintos de capas durante la experimentación, siendo estas las capas densas y las convolucionales. La diferencia en el rendimiento del modelo se representa en la Figura 10.10 y en la Figura 10.11.

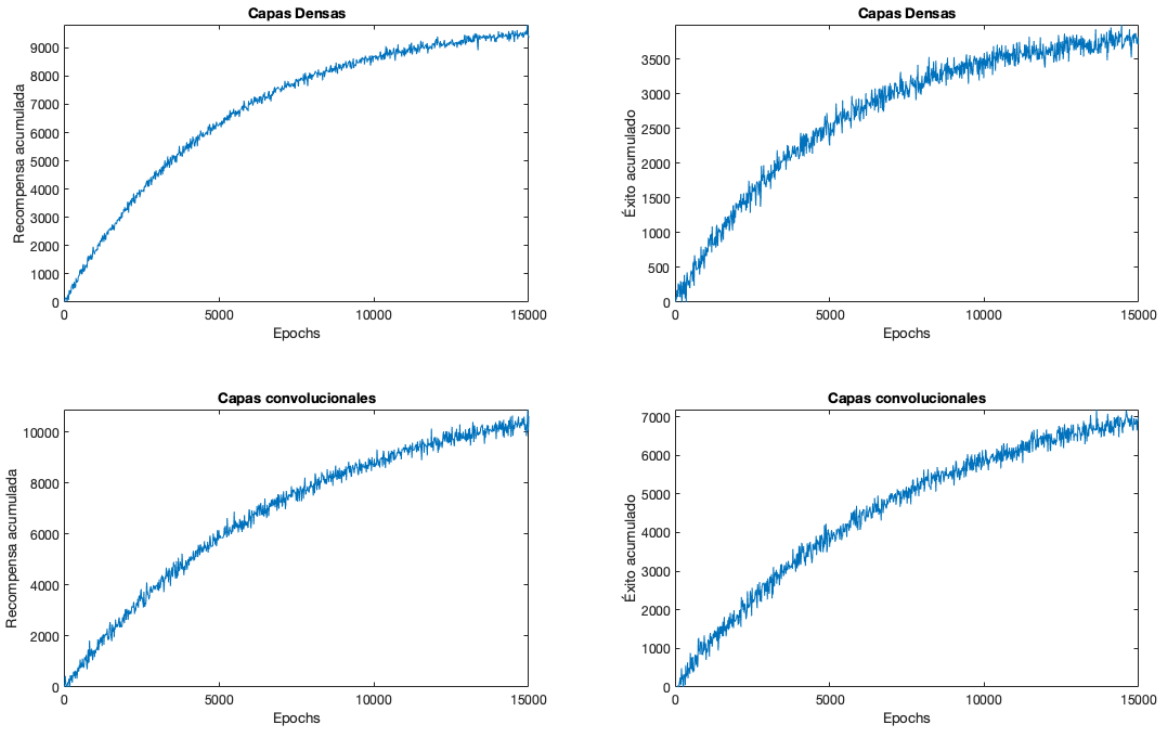



Figura 10.10: Éxito acumulado para las capas densas y convolucionales.

Cómo se puede ver en la Figura 10.10 el modelo es más eficiente utilizando capas convolucionales que densas. Por un lado, para la recompensa acumulada se observa que la gráfica tiene en ambos casos una forma similar, asemejándose a una función logarítmica. La distinción más notable entre los dos experimentos es que el modelo con capas convolucionales alcanza una mayor recompensa acumulada al final de la prueba.

De la misma forma, para el éxito acumulado la mayor diferencia entre los dos tipos de capas es que el valor total acumulado por el modelo con capas convolucionales es mayor que el de las capas densas, llegando a duplicar su cantidad.

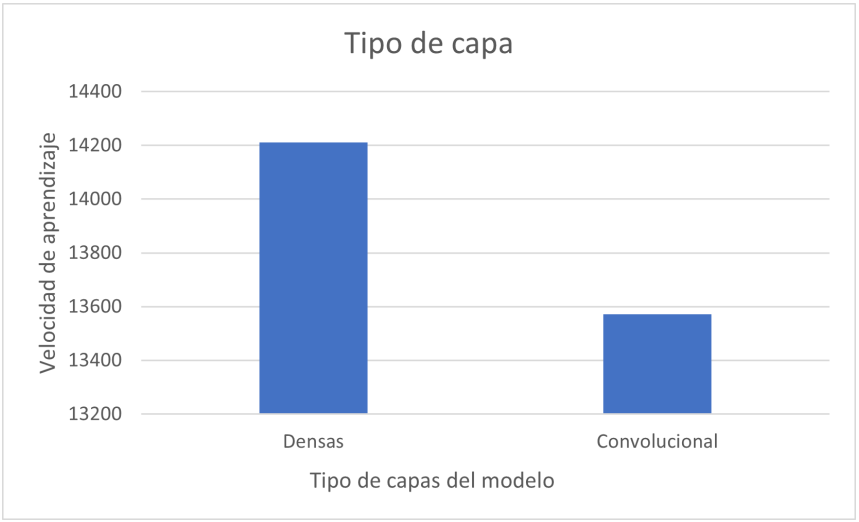



Figura 10.11: Velocidad de aprendizaje para las capas densas y convolucionales.

Finalmente, la velocidad de aprendizaje para el tipo de capa tiene una gran variación en su comportamiento. Como se ve en la Figura 10.11, el número de iteraciones que transcurren hasta que el modelo converge es mucho menor cuando se utilizan las capas convolucionales que cuando se usan las densas.

## 10.2. Discusión

A través de los resultados obtenidos es posible estudiar cómo la variación de los parámetros de aprendizaje afecta al rendimiento del modelo, pudiendo seleccionar para qué valores es más eficiente.

**Tasa de aprendizaje [66].** Cabe destacar que el ajuste de este parámetro tiene grandes efectos sobre el rendimiento del modelo, ya que se encarga de definir la rapidez con la que se actualizan los parámetros del modelo.

En consecuencia, cuando la tasa de aprendizaje es demasiado pequeña se pueden dar casos de sobreajuste del modelo al conjunto de datos de entrada, . Este sobreajuste se debe a que una tasa de aprendizaje pequeña hace que los parámetros se actualicen demasiado rápido, causando que el modelo se adapte de forma tan precisa a los datos que pierda su capacidad de generalización y de búsqueda de patrones.

En este caso, se hacen ajustes tan pequeños de los parámetros durante el entrenamiento que vuelve al modelo demasiado complejo y sensible a las variaciones en el comportamiento de los datos. Por lo tanto, el modelo no aprende de forma correcta, ya que tendrá un alto rendimiento para ese conjunto de datos en particular, pero será mucho menos eficiente para conjuntos de datos con los que no haya entrenado.

En cambio, si la tasa de aprendizaje es mayor se puede evitar el sobreajuste haciendo que las actualizaciones ocurran con menor frecuencia. Sin embargo, si su valor es

demasiado grande se puede dar una divergencia en el entrenamiento del modelo, es decir, que no converja a una solución óptima. En el modelo estudiado, esto ocurre para valores mayores a $alfa = 0,004$, lo cual indica que los ajustes de los parámetros son demasiado grandes, haciendo que varíen drásticamente y no puedan estabilizarse. Sin embargo, para $alfa = 0,003$ el modelo aprende de forma constante y alcanza valores de recompensa y éxito altos, además de converger en el menor número de iteraciones.

**Tasa de descuento[67].** El ajuste de la tasa de descuento afecta a la importancia que da el modelo a las recompensas inmediatas frente a las que se obtienen en un futuro.

Si se utiliza una tasa de descuento baja durante el entrenamiento, se dará prioridad a las recompensas inmediatas, obteniendo por lo tanto un mejor rendimiento en los primeros momentos de la ejecución. Sin embargo, esto puede provocar que el modelo pase por alto mejores recompensas que se dan a largo plazo, ya que una tasa de descuento pequeña reduce la exploración de nuevas acciones del modelo, pudiendo causar que se quede bloqueado en un valor óptimo local. Se observa que cuanto menor sea la tasa de descuento, mayor es el resultado obtenido, siendo los mejores resultados los obtenidos por el modelo entrenado para $gamma = 0,70$.

En contraste, si la tasa de descuento es alta, como ocurre para $gamma = 0,99$, las recompensas futuras se consideran más importantes, haciendo que el modelo decida elegir acciones con recompensas inmediatas más pequeñas con el objetivo de mejorar las que se obtienen a largo plazo. Por lo tanto, cuanto mayor sea la tasa, más largo es el periodo en el que los valores son bajos, ya que el modelo esta planificando a futuro, para después incrementar la recompensa de forma constante.

**Número de neuronas[68].** Determina el número de neuronas comprendidas en cada capa del modelo. Se puede observar cómo para una cantidad de neuronas por capa mayor a 32, el modelo sufre un efecto de sobreajuste. Esto se debe a que, aunque el modelo tiene más capacidad de representación y puede ser más preciso a la hora de identificar patrones complejos en los datos de entrada, también puede conllevar que si la cantidad de nodos es demasiado alta se adapte únicamente para ese conjunto de datos.

Por otro lado, si el número de neuronas es demasiado bajo se dan casos de subajuste, en el cual, al tener tan pocos nodos, el modelo no tiene la suficiente capacidad como para entender la complejidad de los datos de entrada. Esto lleva a un cálculo simplificado e impreciso de los resultados, además de un mal rendimiento para conjuntos de datos desconocidos.

**Número de capas[69].** De forma similar a la variación del número de neuronas, en este caso en particular, si el modelo esta comprendido por más de cuatro capas, hay un riesgo de sobreajuste o de no convergencia. De esta forma, cuantas más capas tiene el modelo, mejor puede adaptarse a los datos de entrada, además de detectar patrones y comportamientos con mayor rapidez. Sin embargo, se da un sobreajuste si el modelo se centra demasiado en pequeños detalles de los datos, como ocurre para seis y siete capas, maximizando su comportamiento únicamente para ese conjunto de entradas.

Así mismo, si el número de capas es pequeño, se puede dar un subajuste, ya que el modelo no es capaz de identificar el comportamiento de los datos si este es demasiado complejo, dando lugar a un rendimiento pobre.

**Tipo de capas.** La variación entre el tipo de capas utilizadas en el modelo, ya sean capas densas o convolucionales, afecta al rendimiento del modelo. Por un lado, las capas densas son eficientes para el manejo de datos estructurados, así como para el procesamiento de conjuntos de datos de entrenamiento pequeños y medianos. Mientras que las convolucionales tienen un alto rendimiento con grandes cantidades de datos, además de ser eficaces a la hora de detectar patrones y tendencias. Además, en las capas convolucionales las neuronas están conectadas a una única entrada de datos, a diferencia de las densas que están unidas a todas las neuronas de la capa anterior, lo cual hace que las convolucionales tengan una mayor eficiencia de procesamiento. Así mismo, las redes convolucionales también cuentan con capas de pooling que permiten que se entrene únicamente con la información más relevante. Por lo tanto, para este experimento, las capas convolucionales tienen un mejor rendimiento, ya que obtienen resultados más altos en las ejecuciones.

Estos resultados se pueden observar en la literatura [70] de otros modelos de aprendizaje por refuerzo. En ellos, se utilizan algoritmos mucho más complejos que Q-learning, como pueden ser el actor-critico, pero también se muestran en su métricas de recompensa acumulada un comportamiento parecido a una función sigmoide, que se presenta en los resultados de los experimentos realizados en este proyecto.

Teniendo en cuenta la información anterior, se han seleccionado una serie de valores para los parámetros de entrenamiento del modelo final. Estos se pueden observar en la Tabla 10.1.

| **Parámetro** | **Valor** |
|---|---|
| Tasa de aprendizaje | 0,003 |
| Tasa de descuento | 0,7 |
| Número de neuronas | 32 |
| Número de capas | 4 |
| Tipo de capa | Convolucionales |

Tabla 10.1: Configuración de los parámetros del modelo óptimo.

## 10.3. Modelo óptimo

El rendimiento del modelo óptimo está representado en las figuras 10.12 y 10.13, destacando sus métricas de recompensa y éxito acumulados, además de la reducción de los segundos de crisis presentes.

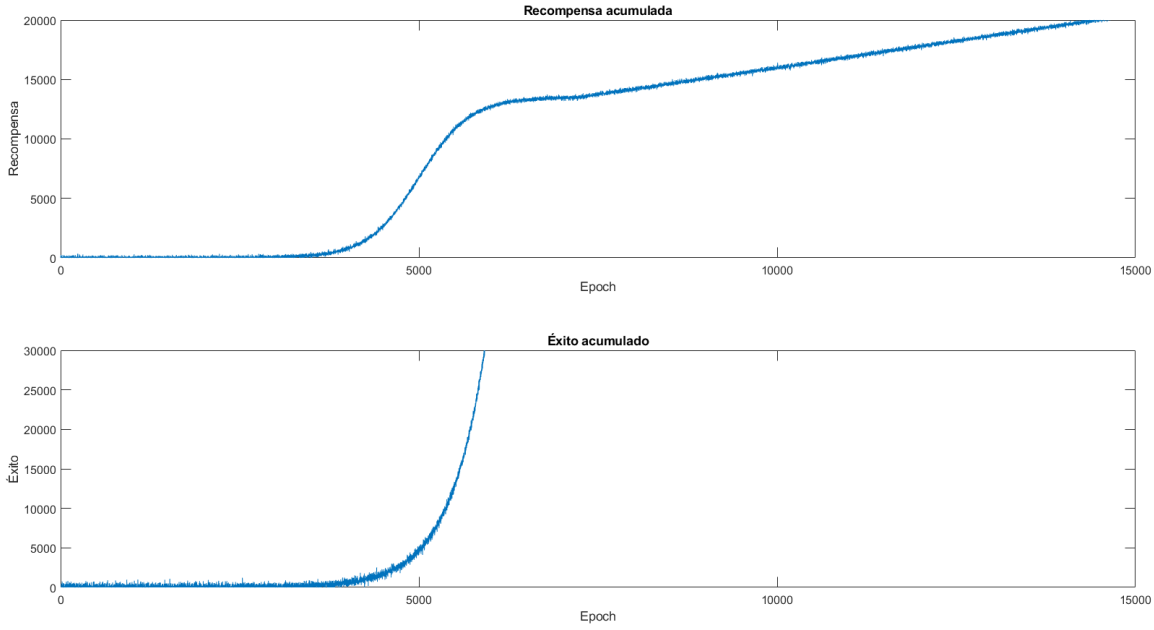



Figura 10.12: Éxito y recompensa acumuladas del modelo óptimo

En primer lugar, la recompensa acumulada del modelo sigue un crecimiento similar a una función sigmoide, la cual, cambia de tendencia durante la mitad de la ejecución, pasando a tener un comportamiento lineal. Esto se debe a que se obtienen recompensas muy pequeñas durante los momentos iniciales de la ejecución, las cuales, van creciendo constantemente hasta estabilizarse, para después incrementar rápidamente durante el resto de la ejecución. Además, cabe destacar que la recompensa total obtenida es mucho mayor para este modelo óptimo que para el resto de experimentos realizados, ya que alcanza un valor total de 20000.

En cuanto al éxito acumulado, se distingue un comportamiento exponencial, caracterizado por crecimiento acelerado durante la mayoría del tiempo de la prueba. Se observa cómo el número de éxitos durante el inicio de la ejecución es extremadamente bajo, aumentando significativamente a partir de las 5000 iteraciones. Esta tendencia exponencial significa que el modelo esta mejorando rápidamente su rendimiento, distinguiendo correctamente los patrones de los datos, y por lo tanto, siendo capaz de reducir, una vez que converge el modelo, el 100 % de los ataques epilépticos que general el simulador de señales cerebrales. Así mismo,

el éxito acumulado resultante llega a una magnitud superior a 30000, que es una cantidad mucho mayor que las obtenidas en los experimentos anteriores.

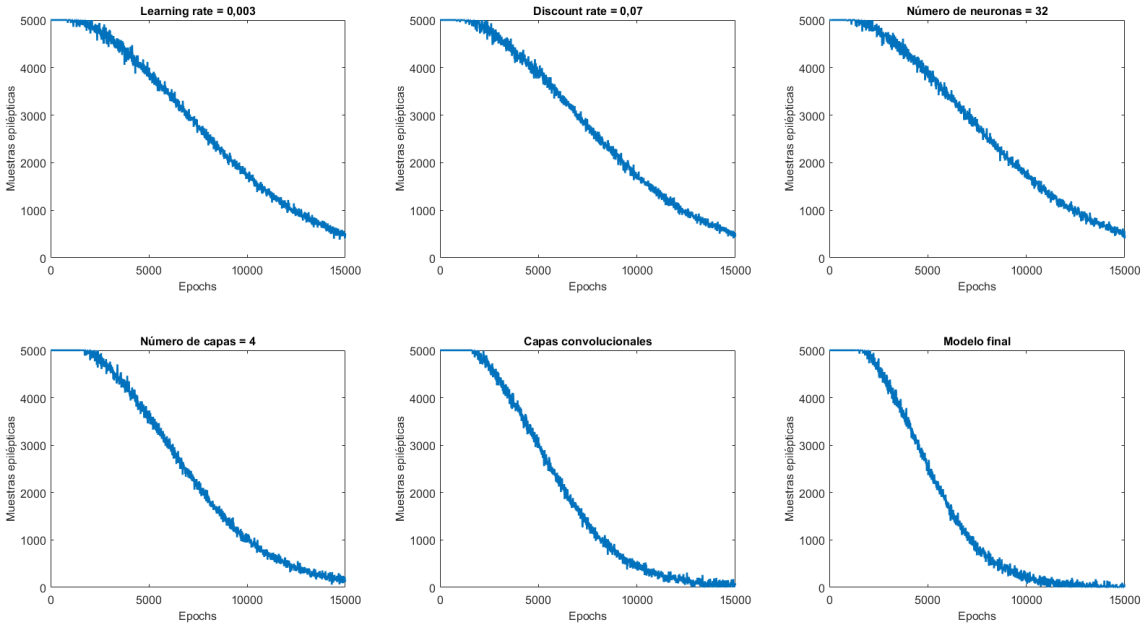



Figura 10.13: Reducciones de crisis epilépticas en el modelo óptimo

Por otro lado, se puede observar la efectividad del modelo a través de la reducción de los segundos epilépticos acontecidos. En la Figura 10.13 está representado cómo modificando únicamente los parámetros elegidos de forma individual, se consigue una disminución significativa de las crisis a medida que transcurren las iteraciones. Sin embargo, el mejor rendimiento se encuentra para el modelo final, donde al aplicar todos los ajustes en conjunto, se consigue reducir mucho antes la cantidad de crisis. Finalmente, gracias a todas las ideas explicadas anteriormente, se puede afirmar que el modelo final tiene un mejor rendimiento que los modelos ejecutados durante la experimentación, y por lo tanto, los parámetros elegidos para el entrenamiento pueden considerarse óptimos.

# Capítulo 11

# Conclusiones y Trabajo futuro

## 11.1. Conclusiones

En el trabajo expuesto se ha estudiado e implementado un algoritmo de control de dispositivos para la Estimulación Cerebral Profunda, basado en la inteligencia artificial y en el aprendizaje por refuerzo. De esta forma, se han cumplido con los objetivos planteados inicialmente en este proyecto que se resume a continuación.

En primer lugar, se ha realizado una investigación en profundidad de los conceptos fundamentales de los que se compone el tratamiento de enfermedades neuronales mediante DBS. Esto ha permitido entender el funcionamiento de los dispositivos de control, estudiando su estructura formada por electrodos y un generador de señales. Así mismo, se han estudiado los algoritmos más utilizados para gestionar el envió de pulsos eléctricos a ciertas áreas del cerebro del paciente.

También se ha investigado sobre las ideas principales del funcionamiento de las arquitecturas de aprendizaje por refuerzo y de las redes neuronales. Para los métodos de aprendizaje por refuerzo se han diferenciado cada uno de sus elementos, como son los conjuntos de estados, acciones, parámetros y bucle de entrenamiento del modelo.

Así mismo, se ha realizado un proceso de documentación sobre el uso de las librerías de código abierto especializadas en la implementación de proyectos de inteligencia artificial, como son TensorFlow y Keras.

En cuanto al conjunto de datos cerebrales utilizados como entrada para el entrenamiento del modelo, han sido procesados utilizando técnicas de tratamiento de señales como las ventanas de Hanning, con el objetivo de eliminar interferencias de la señal original.

A continuación, se ha implementado el sistema mediante la codificación de un agente, un entorno y un generador de señales. El agente es entrenado mediante un algoritmo de Q-Learning, en el que este elige y realiza acciones sobre el entorno para obtener una recompensa positiva. De esta forma el agente ha aprendido a tomar aquellas decisiones que permitan reducir los ataques epilépticos simulados por el generador.

Una vez terminado el desarrollo, se realizaron una serie de experimentos en los qué se ajustaban los parámetros de aprendizaje del modelo a distintos valores, para observar y analizar con que magnitudes el rendimiento del modelo es mejor. Por lo tanto, a través de las métricas de éxito acumulado, recompensa acumulada y velocidad de aprendizaje, se determinó que los parámetros de aprendizaje óptimos son una tasa de aprendizaje del 0,003, una tasa de descuento del 0,7, además de una topología de red neuronal de 4 capas con 32 neuronas cada una, siendo estas capas convolucionales.

Con estos parámetros se entrenó un modelo final y se observó mediante las métricas elegidas que su rendimiento era mucho mejor que el resto de modelos entrenados. Por lo tanto, se ha llegado a la conclusión de que las magnitudes más eficientes de los parámetros del modelo son aquellas cuyo valor es moderado dentro de los rangos elegidos, ya que si se escogen valores demasiado grandes se dan problemas de sobreajuste o de divergencia que evitan el correcto funcionamiento del algoritmo.

## 11.2. Trabajo futuro

Este proyecto ha definido unas bases sobre las que desarrollar un algoritmo de control de dispositivos de Estimulación Cerebral Profunda. El modelo desarrollado es capaz de aprender sobre los datos aportados, y reducir el número de ataques producidos. Sin embargo, existen varias áreas de mejora y de trabajo futuro que pueden llevarse a cabo en otros proyectos.

Se puede explorar la utilización de otras arquitecturas de redes neuronales, ya que en este proyecto se han utilizado únicamente las redes densas y las convolucionales. Es posible extender el desarrollo al uso de redes neuronales recurrentes (CNN), dado que cuentan con una memoria interna que las hace capaces de capturar la dependencia temporal de los datos, siendo extremadamente eficientes a la hora de procesar series temporales como las señales cerebrales.

También se podría mejorar el procesamiento de los datos de entrada, ya que en lugar de utilizar la entropía espectral de los datos como medida de variación, se podría procesar directamente la señal en bruto. Esto permitiría utilizar todo el conjunto de información de la que se dispone y mejorar el rendimiento del algoritmo, ya que no se realizarían transformaciones adicionales sobre los datos.

Finalmente, el modelo actual únicamente se puede ejecutar en sistemas con gran capacidad computacional, debido a los recursos que necesita para su funcionamiento. Por lo tanto, una línea de mejora sería conseguir que se pudiese ejecutar en dispositivos clínicos, mediante la incorporación de librerías como TensorFlow Lite. Estas librerías son capaces de cuantizar los modelos mediante una reducción del número de bits utilizados para representar los parámetros del modelo, reduciendo los recursos necesarios para ejecutarlos y haciéndolos adaptables a dispositivos móviles.

# Apéndice A

# Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código:
  https://gitlab.inf.uva.es/maryagu/tfg-marina/-/blob/main/TFG_Marina_Yague.py

# Bibliografía

[1] Pablo Huet. Qué son las redes neuronales y sus aplicaciones. https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/, 2023. Accedido: 2023-6-18.

[2] Miguel Sotaquirá. La función de activación. https://www.codificandobits.com/blog/funcion-de-activacion/, 2018. Accedido: 2023-6-19.

[3] Jahaziel Ponce. Conoce que son las funciones de activacion. https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tenso
2020. Accedido: 2023-6-19.

[4] Fangyuan Chang. Q-learning algorithm flowchart. https://www.researchgate.net/figure/Q-learning-algorithm-flowchart_fig1_334244533, 2019. Accedido: 2023-6-21.

[5] Raquel Marin. La depresión puede mejorar con estimulación eléctrica. https://raquelmarin.net/la-depresion-puede-mejorar-con-estimulacion-electrica/, 2022. Accedido: 2023-7-10.

[6] Dmitrii Krylov, Remi Tachet, Romain Laroche, Michael Rosenblum, and Dmitry V. Dylov. Reinforcement learning framework for deep brain stimulation study. https://arxiv.org/abs/2002.10948, 2020. Accedido: 2023-6-23.

[7] Víctor Uc-Cetina. Advances in artificial intelligence. https://www.researchgate.net/figure/The-actor-critic-architecture_fig1_258394387, 2013. Accedido: 2023-7-10.

[8] Windows and spectral leakage. https://community.sw.siemens.com/s/article/windows-and-spectral-leakage, 2019. Accedido: 2023-7-10.

[9] Numpy. https://seeklogo.com/vector-logo/398690/numpy, 2023. Accedido: 2023-7-10.

[10] Tensorflow. https://www.vectorlogo.zone/logos/tensorflow/index.html, 2023. Accedido: 2023-7-10.

[11] Keras. https://commons.wikimedia.org/wiki/File:Keras_logo.svg, 2023. Accedido: 2023-7-10.

[12] Scipy. https://commons.wikimedia.org/wiki/File:SCIPY_2.svg, 2023. Accedido: 2023-7-10.

[13] Gitlab. https://commons.wikimedia.org/wiki/File:GitLab_logo.svg, 2023. Accedido: 2023-7-10.

[14] Overleaf official logos. https://www.overleaf.com/for/partners/logos, 2023. Accedido: 2023-7-10.

[15] Matlab. https://commons.wikimedia.org/wiki/File:Matlab_Logo.png, 2023. Accedido: 2023-7-10.

[16] Eric Lowet, Krishnakanth Kondabolu, Samuel Zhou, Rebecca A. Mount, Yangyang Wang, Cara R. Ravasio, and Xue Han. Deep brain stimulation creates informational lesion through membrane depolarization in mouse hippocampus. https://www.nature.com/articles/s41467-022-35314-1, 2022. Accedido: 2023-4-10.

[17] Anastasia Stsepanets. Modelo de cascada (waterfall): qué es y cuándo conviene usarlo. https://blog.ganttpro.com/es/metodologia-de-cascada/, 2023. Accedido: 2023-4-10.

[18] Modelo de prototipos: ¿qué es y cuáles son sus etapas? https://www.hostingplus.com.es/blog/modelo-de-prototipos-que-es-y-cuales-son-sus-etapas/, 2021. Accedido: 2023-4-12.

[19] Supercomputacion castilla y león. https://www.scayle.es/, 2023. Accedido: 2023-7-03.

[20] Experta en inteligencia artificial. https://www.infoempleo.com/guias-informes/empleo-it-mujeres/perfiles/experta-inteligencia-artificial.html#:~:text=En%20el%20caso%20de%20ingenieros,los%20profesionales%20con%20m%C3%A1s%20experiencia, 2023. Accedido: 2023-4-12.

[21] Cuánto se paga de seguros sociales por cada trabajador. https://ayudatpymes.com/gestron/cuanto-se-paga-de-seguros-sociales-por-cada-trabajador/, 2023. Accedido: 2023-4-13.

[22] ¿qué es el aprendizaje automático? https://www.netapp.com/es/artificial-intelligence/what-is-machine-learning/, 2023. Accedido: 2023-5-18.

[23] Descubre los principales beneficios del 'machine learning'. https://www.iberdrola.com/innovacion/machine-learning-aprendizaje-automatico, 2023. Accedido: 2023-5-25.

[24] Javier Luna Gonzalez. Redes neuronales. https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2, 2021. Accedido: 2023-6-4.

[25] Amit Singh Rathore. Layers in neural network. https://medium.com/nerd-for-tech/layers-in-neural-network-90d48a5a42fb, 2023. Accedido: 2023-6-20.

[26] Funciones de activación. https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb, 2019. Accedido: 2023-6-18.

[27] Laxman Singh. Forward and backward propagation — understanding it to master the model training process. https://medium.com/geekculture/forward-and-backward-propagation-understanding-it-to-master-the-model-training-p 2021. Accedido: 2023-6-19.

[28] Krish Naik. Understanding all optimizers in deep learning. https://krishnaik.in/2022/03/28/understanding-all-optimizers-in-deep-learning/, 2022. Accedido: 2023-6-23.

[29] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, 2017. Accedido: 2023-7-05.

[30] Sergio Pérez. Q-learning. https://blog.damavis.com/aprendizaje-por-refuerzo-q-learning/, 2023. Accedido: 2023-6-21.

[31] Q-learning. https://datascience.eu/es/aprendizaje-automatico/q-learning/, 2021. Accedido: 2023-6-21.

[32] Todd M. Herrington, Jennifer J. Cheng, and Emad N. Eskandar. Mechanisms of deep brain stimulation. https://journals.physiology.org/doi/full/10.1152/jn.00281.2015, 2016. Accedido: 2023-6-22.

[33] Estimulación cerebral profunda en la enfermedad de parkinson. https://cdincbarcelona.com/es/estimulacion-cerebral-profunda-en-la-enfermedad-de-parkins 2018. Accedido: 2023-6-22.

[34] Un implante dbs trata la epilepsia médicamente refractaria. https://www.hospimedica.es/tecnicas-quirurgicas/articles/294777210/un-implante-dbs-trata-la-epilepsia-medicamente-refractaria.html, 2019. Accedido: 2023-6-23.

[35] Meili Lu, Xile Wei, Yanqiu Che, Jiang Wang, and Kenneth A Loparo. Application of reinforcement learning to deep brain stimulation in a computational model of parkinson's disease. https://pubmed.ncbi.nlm.nih.gov/31715567/, 2019. Accedido: 2023-7-8.

[36] F. Wendling, F. Bartolomei, J. J. Bellanger, and P. Chauvel. Epileptic fast activity can be explained by a model of impaired gabaergic dendritic inhibition. https://onlinelibrary.wiley.com/doi/abs/10.1046/j.1460-9568.2002.01985.x?sid=nlm%3Apubmed, 2002. Accedido: 2023-7-8.

[37] P. Ghasemi, T. Sahraee, and A. Mohammadi. Closed and open-loop deep brain stimulation: Methods, challenges, current and future aspects. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6015649/, 2018. Accedido: 2023-6-23.

[38] Thomas Simonini. Advantage actor critic (a2c). https://huggingface.co/blog/deep-rl-a2c#the-actor-critic-process, 2021. Accedido: 2023-6-24.

[39] Thomas Simonini. Proximal policy optimization (ppo). https://huggingface.co/blog/deep-rl-ppo#the-intuition-behind-ppo, 2021. Accedido: 2023-6-24.

[40] Mayank Mishra. Convolutional neural networks, explained. https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939, 2023. Accedido: 2023-6-25.

[41] Gabriella Panuccio, Davide Caron, and Angel Canal-Alonso. Mimicking ca3 temporal dynamics controls limbic ictogenesis. https://zenodo.org/record/6278046, 2022. Accedido: 2023-7-8.

[42] Mathuranathan Viswanathan. Fft and spectral leakage. https://www.gaussianwaves.com/2011/01/fft-and-spectral-leakage-2/, 2011. Accedido: 2023-6-25.

[43] Window types: Hanning, flattop, uniform, tukey, and exponential. https://community.sw.siemens.com/s/article/window-types-hanning-flattop-uniform-tukey-and-exponential, 2019. Accedido: 2023-6-25.

[44] Numpy. https://numpy.org/, 2023. Accedido: 2023-5-1.

[45] Numpy fundamentals. https://numpy.org/doc/stable/user/basics.html, 2023. Accedido: 2023-5-1.

[46] Tensorflow. https://www.tensorflow.org/?hl=es-419, 2023. Accedido: 2023-5-1.

[47] Keras. https://keras.io/, 2023. Accedido: 2023-5-2.

[48] François Chollet. The sequential model. https://keras.io/guides/sequential_model/, 2020. Accedido: 2023-5-5.

[49] Tensorflow dense layer. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense, 2023. Accedido: 2023-5-8.

[50] Tensorflow losses. https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy, 2023. Accedido: 2023-5-11.

[51] Tensorflow adam. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam, 2023. Accedido: 2023-5-11.

[52] Scipy. https://scipy.org/, 2023. Accedido: 2023-5-13.

[53] Gitlab. https://about.gitlab.com/, 2023. Accedido: 2023-5-1.

[54] J Hammersley, J. & Lees-Miller. Overleaf. https://es.overleaf.com/, 2012. Accedido: 2023-2-15.

[55] Matlab. https://es.mathworks.com/?s_tid=gn_logo, 2023. Accedido: 2023-5-2.

[56] Learning rate. https://encord.com/glossary/learning-rate-definition/, 2023. Accedido: 2023-7-03.

[57] Sajil C. K. Discount factor in reinforcement learning. https://intuitivetutorial.com/2020/11/15/discount-factor/, 2020. Accedido: 2023-7-03.

[58] Palash Sharma. Keras dense layer explained for beginners. https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/, 2020. Accedido: 2023-7-03.

[59] Assaad Moawad. Dense layers explained in a simple way. https://medium.com/datathings/dense-layers-explained-in-a-simple-way-62fe1db0ed75, 2019. Accedido: 2023-7-03.

[60] Ashutosh Makone. Window size. https://support.ircam.fr/docs/AudioSculpt/3.0/co/Window%20Size.html, 2018. Accedido: 2023-7-03.

[61] Michael R. Gionfriddo, Alexandra J. Greenberg, Abhijeet L. Wahegaonkar, and Kendall H. Lee. Rangos de frecuencia. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4057890/+, 2013. Accedido: 2023-7-04.

[62] Digital signal processing: Sampling rates, bandwidth, spectral lines, and more. https://community.sw.siemens.com/s/article/digital-signal-processing-sampling-rates-bandwidth-spectral-lines-and-more#:~:text=Sampling%20rate%20(sometimes%20called%20sampling,as%202000%20Hertz%20sample%20frequency., 2020. Accedido: 2023-7-04.

[63] Diego Coulombie and Susana Blanco. Predicción de ataque epiléptico usando entropía espectral. http://sedici.unlp.edu.ar/bitstream/handle/10915/121940/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y#:~:text=an%C3%A1lisis%20de%20se%C3%B1ales%3B%20entrop%C3%ADa%20espectral&text=Todas%20son%20discontinuidades%20en%20la,para%20que%20ocurra%20el%20ataque., 2011. Accedido: 2023-7-05.

[64] Aditya Mishra. Metrics to evaluate your machine learning algorithm. https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234, 2018. Accedido: 2023-7-05.

[65] Weights & biases. https://wandb.ai/site, 2023. Accedido: 2023-7-06.

[66] Leslie N. Smith. A disciplined approach to neural network hyper-parameters. https://arxiv.org/abs/1803.09820, 2018. Accedido: 2023-7-06.

[67] Vincent François-Lavet and Raphael Fonteneau. How to discount deep reinforcement learning: Towards new dynamic strategies. https://arxiv.org/abs/1512.02011, 2016. Accedido: 2023-7-06.

[68] Miller Trujillo, Mario Linares-Vásquez, Camilo Escobar-Velásquez, Ivana Dusparic, and Nicolás Cardozo. Does neuron coverage matter for deep reinforcement learning? a preliminary study. https://dl.acm.org/doi/10.1145/3387940.3391462, 2020. Accedido: 2023-7-07.

[69] Alexios Koutsoukas, Keith J Monaghan, Xiaoli Li, and Jun Huan. Deep-learning: investigating deep neural networks hyper-parameters and comparison of performance to shallow methods for modeling bioactivity data. https://pubmed.ncbi.nlm.nih.gov/29086090/, 2017. Accedido: 2023-7-07.

[70] Qitong Gao, Michael Naumann, Ilija Jovanov, Vuk Lesi, Karthik Kamaravelu, Warren M. Grill, and Miroslav Pajic. Model-based design of closed loop deep brain stimulation controller using reinforcement learning. `https://ieeexplore.ieee.org/document/9096004`, 2020. Accedido: 2023-7-10.