Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Generación de señales cerebrales mediante técnicas de Reservoir Computing

Autor:
Dña. Miriam Arconada Pedriza

Tutores:
Dña. María Aránzazu Simón Hurtado
D. Ángel Canal Alonso

"Mientras el cerebro sea un misterio,
el universo continuará siendo un misterio."

SANTIAGO RAMÓN Y CAJAL

Agradecimientos

Me gustaría dedicar este apartado a todos aquellos que me han acompañado a lo largo de estos cuatro años de carrera, tanto en lo bueno como en lo malo. Me llevo una experiencia enriquecedora, tanto profesional como personal, que me alegro de haber adquirido.

En primer lugar me gustaría agradecer a mis tutores, Ángel y Arancha, que han hecho todo lo posible para que este trabajo salga adelante con éxito.

En segundo lugar me gustaría agradecer a mi familia, que ha estado ahí cuando más lo necesitaba y han confiado plenamente en mí.

También me gustaría agradecer a mis compañeros de AIR Institute, en especial a Pablo Enrique Guillem, que me ha ayudado todo lo posible durante el desarrollo de este proyecto.

Por último, me gustaría agradecer a mis amigos más cercanos, que saben por todo lo que he pasado este último año y han sabido sacarme siempre una sonrisa y darme fuerzas para seguir.

Sin todos ellos esto no habría sido posible.

Resumen

Los últimos avances en inteligencia artificial generativa han permitido una imitación casi completa del lenguaje humano. Sin embargo, el lenguaje interno del cerebro es un terreno poco explorado dentro de este campo, habiéndose limitado a modelos matemáticos simples que reproducen el comportamiento individual de neuronas pero son incapaces de replicar estructuras cerebrales completas.

Si entendemos el cerebro como un sistema dinámico no lineal, existen diversas aproximaciones que pueden modelizarlo mediante técnicas de Inteligencia Artificial para después generar o predecir señales sin necesidad de conocer las ecuaciones del sistema. Entre estas aproximaciones están las Recurrent Neural Network (RNN), que en sus diversas formas son capaces de retener información temporal del sistema y reconstruirlo internamente con sus pesos.

En este Trabajo de Fin de Grado (TFG) se busca utilizar cálculo de reservoir, un tipo específico de redes recurrentes, para el modelado y generación de señales cerebrales de forma automatizada. Para ello se usaran grabaciones de array multielectrodo en roedores con epilepsia como entrada de diferentes tipos de redes recurrentes para establecer una comparativa entre las diferentes aproximaciones.

Abstract

The latest advances in generative artificial intelligence have enabled an almost complete imitation of human language. However, the brain's internal language remains a largely unexplored territory within this field, having been limited to simple mathematical models that reproduce the individual behavior of neurons but are unable to replicate complete brain structures.

If we understand the brain as a nonlinear dynamic system, there are various approaches that can model it using artificial intelligence techniques to then generate or predict signals without needing to know the equations of the system. Among these approaches are Recurrent Neural Networks (RNN), which in their various forms are capable of retaining temporal information from the system and internally reconstructing it with their weights.

In this Bachelor´s Degree Final Project, the aim is to use Reservoir Computing, a specific type of recurrent networks, for the automated modeling and generation of brain signals. For this purpose, recordings of multi-electrode arrays in rodents with epilepsy will be used as input for different types of recurrent networks to establish a comparison between the different approaches.

Índice general

Índice de Acrónimos

BPTT Back Propagation Through Time. 46, 47, 55

CPU Central Processing Unit. 23, 26, 29, 111

CRISP-DM CRoss-Industry Standard Process for Data Mining. 5, 6

DL Deep Learning. 32, 43, 44

ESN Echo State Networks. 17, 27, 34, 50–53, 55, 69, 99, 101, 112

FNN Feedforward Neural Network. 46

FPTT Forward Propagation Through Time. 46

GPU Graphics Processing Unit. 23, 26, 29

IA Inteligencia Artificial. 1, 23, 31, 32, 43, 48, 61

IRPF Impuesto sobre la renta de las Personas Físicas. 22, 23

LSM Liquid State Machine. 17, 34, 50, 53–56, 63, 80, 81, 99, 101, 113

LSTM Long Short-Term Memory. 1–3, 17, 23, 25, 26, 34, 43, 44, 48, 59–61, 65, 91, 94, 98, 99, 101, 111, XX

MEA arrays multielectrodos. 1, 2, 7, 17, 22, 26, 28, 31–36, 39, 40, 43, 49–53, 56, 57, 60–63, 69, 72, 79, 80, 86, 87, 90, 91, 99, 102, 109

ML Machine Learning. 22, 23, 32, 43, 61, 62, 101, 102

MSE error cuadrático medio. 80, 91

NGRC Next Generation Reservoir Computing. 17, 34, 43, 44, 57, 87, 99, 101, 114

NRMSE raíz del error cuadrático medio normalizado. 69, 79, 90

NVAR Nonlinear Vector AutoRegression. 34, 57

RC Reservoir Computing. 1, 2, 16, 27, 34, 43, 44, 48, 50, 53, 57–59, 99, 101

RNN Recurrent Neural Network. 1, 2, 26, 33, 34, 43–49, 57, 59, 60, 65, 99, 101, V

SD Sistema Dinámico. 31, 32

TFG Trabajo de Fin de Grado. 1–3, 6, 8, 9, 22, 26–29, 31, 32, 34–36, 38, 43, 49–52, 54, 56–59, 61, 65, 69, 90, 101, 102, 109, 115, V

TPU Tensor Processing Unit. 25, 26, 62, 66

Índice de Ecuaciones

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

Debido al continuo desarrollo de las ciencias de la computación, en concreto en el campo de la Inteligencia Artificial (IA), el ser humano ha sido capaz de crear máquinas capaces de realizar tareas que normalmente requerirían de inteligencia humana.

A día de hoy, esta tecnología se divide en muchos tipos y modalidades, entre los que se encuentra la inteligencia artificial generativa. Este tipo de IA es capaz de aprender sobre la estructura y los patrones de los datos de entrenamiento para generar unos nuevos con características similares a los proporcionados.

Su aplicación abarca muchas áreas de investigación, entre las que se encuentra la medicina y, en concreto, el análisis de señales cerebrales mediante arrays multielectrodos (MEA), aunque existen pocas referencias.

1.2. Motivación

El cerebro es un sistema complejo, difícil de interpretar y analizar en su totalidad. A pesar de que existen modelos capaces de interpretar partes del cerebro de forma individual [1], aún no se ha conseguido replicar la funcionalidad de estructuras anatómicas cerebrales completas.

Existen muchas aproximaciones que intentan replicar su comportamiento intentando mantener la mayor cantidad de información posible. Entre ellas se encuentra interpretar el cerebro como un sistema dinámico no lineal [2] y, mediante técnicas de IA, conseguir generar o predecir el funcionamiento de las señales emitidas por el cerebro [3].

Diversos modelos y técnicas aseguran ser capaces de reproducir eficazmente señales MEA, como pueden ser las RNN, capaces de procesar datos de forma secuencial. Como el caso de cada paciente es distinto, y por ello su actividad cerebral también, en este TFG se ha propuesto buscar y comparar modelos capaces de adaptarse de manera satisfactoria a registros reales de este tipo de señales provenientes de roedores con epilepsia.

La técnica principal seleccionada en este TFG es la conocida como Reservoir Computing (RC), ya que permite el manejo de dinámicas no lineales y patrones temporales complejos inherentes a varios tipos de datos, asegurando ser más eficiente que otros modelos. La comparativa se centrará tanto a nivel interno, comparando modelos contenidos en el marco del RC, como con otras variantes de RNN, como puede ser la ya conocida LSTM.

1.3. Objetivos

Para establecer los objetivos de forma clara y concisa, se ha seguido lo que en gestión de proyectos se conoce como SMART [4].

Cada letra de SMART representa los aspectos importantes que los objetivos bien definidos deben cumplir, debiendo de esta forma ser: específicos (*Specific*), medibles (*Measurable*), alcanzables (*Achievable*), relevantes (*Relevant*) y con tiempo límite asociado (*Time constrained*).

Teniendo esta definición en cuenta, se ha definido el objetivo general y se ha optado por su descomposición en objetivos más pequeños para conseguir adaptarse a la definición de SMART y poder cumplir con el desarrollo del proyecto.

1.3.1. Objetivo general

Como se ha descrito en el apartado de la motivación del proyecto (1.2), en este TFG se ha puesto el foco en buscar diferentes modelos dentro de la familia de las RNN y compararlos, en concreto varios modelos de RC y la LSTM. Por ello, el objetivo general se podría definir de la siguiente manera:

OG. Aplicar diversos modelos relacionados con las RNN para la generación automática de señales cerebrales artificiales para obtener una serie de resultados y conclusiones.

1.3.2. Objetivos específicos

Una vez descrito el objetivo general del proyecto, a continuación se muestra una división en objetivos más específicos para lograr su consecución:

OE1. Investigar y revisar la literatura existente relacionada con los modelos a desarrollar para adquirir un conocimiento profundo de sus fundamentos teóricos y aplicaciones prácticas, además de conceptos relevantes, ventajas y limitaciones de cada uno de ellos.

OE2. Preprocesar las señales MEA de la forma adecuada para ajustarse a las necesidades de ejecución de los modelos.

OE3. Desarrollar cada modelo, incluyendo sus respectivas etapas de revisión bibliográfica, análisis, implementación, evaluación del rendimiento y optimización de los hiperparámetros.

OE4. Realizar una comparación exhaustiva de los resultados obtenidos de cada modelo y extraer conclusiones relevantes de dicha comparación.

A mayores, se pretende lograr la consecución de los siguientes objetivos personales, ya que este documento se trata de un TFG con finalidades académicas:

OA1. Redactar el informe final en su completitud de forma clara y concisa que cumpla la estructura establecida con el fin de presentar la investigación realizada asegurando la coherencia y la calidad requerida por la Universidad de Valladolid.

OA2. Presentar de forma oral ante el tribunal el proyecto, obteniendo una calificación satisfactoria adecuada al trabajo realizado.

1.4. Estructura de la memoria

Esta memoria se estructura en los siguientes capítulos:

1. **Introducción**: Descripción del contexto en el que se alberga el proyecto y la motivación a su realización, además de la exposición de los objetivos principales.
2. **Gestión del proyecto**: Explicación de la forma de administrar el proyecto, documentando la planificación, la metodología usada, el plan de riesgos, así como el seguimiento del proyecto y su respectiva estimación de costes.
3. **Estado del arte**: Redacción de la situación actual de la investigación que se trata en este TFG.
4. **Conjunto de datos y preprocesamiento**: En este capítulo se describen los datos y se muestran los pasos realizados para llevar a cabo su preprocesamiento.
5. **Modelos**: Enumeración y descripción de los diferentes modelos, tanto de su base teórica como de lo implementado para que se ajusten a los datos preprocesados.
6. **Implementación en hardware**: Justificación de lo realizado para conseguir implementar el modelo de LSTM en un dispositivo de la gama de Google Coral.
7. **Comparación y discusión de los resultados**: Extracción y aclaración de los resultados obtenidos de los diferentes modelos.
8. **Conclusiones y líneas futuras**: Deducción y sintetización de los resultados obtenidos de forma que puedan servir en trabajos futuros, además de unas pautas en caso de que se quiera seguir con dicha investigación.

Capítulo 2

Gestión del proyecto

A lo largo de este capítulo se va a exponer cómo se ha llevado a cabo la gestión del proyecto para poder cumplir con el alcance del mismo y asegurar que se consigan los objetivos definidos en el Capítulo 1. Para ello, se irá señalando la metodología usada, el plan inicial, el plan de riesgos, el seguimiento real del proyecto y una estimación de costes.

2.1. Metodología CRISP-DM

CRoss-Industry Standard Process for Data Mining (CRISP-DM) [5] es la metodología más usada para proyectos de minería de datos y de investigación.

Cuenta con 6 fases que, a su vez, pueden contar con otras subfases. Se exponen a continuación, de forma resumida, las principales fases de esta metodología:

1. **Comprensión del negocio**: en esta fase se definen los objetivos del proyecto y la gestión de este.
2. **Comprensión de los datos**: se identifican, recolectan y analizan los datos que permiten cumplir con los objetivos definidos en la fase anterior.
3. **Preparación de los datos**: ya conseguidos los datos, se procesan de forma que se adecuen a los requisitos del modelo al que se quieran adaptar.
4. **Construcción del modelo**: una vez conseguidas las fases anteriores, se seleccionan los algoritmos y se construye el modelo deseado. En teoría, esta fase se debería iterar hasta que se considere que se ha conseguido el mejor modelo antes de pasar a posteriores fases. En la práctica, primero se consigue un modelo inicial que sea lo suficientemente bueno como para seguir avanzando con el ciclo de CRISP-DM, y posteriormente se mejora en iteraciones futuras.
5. **Evaluación del modelo**: en esta fase se evalúan los resultados obtenidos por el modelo.
6. **Despliegue**: para finalizar, se realiza la documentación y la revisión final del proyecto, proporcionando el entregable final.

Como se puede ver en la Figura 2.1, las fases pueden ser flexibles, ya que se puede navegar entre ellas tanto como se requiera durante el proyecto. Si además se itera de manera rápida, pone en manifiesto algunos principios y prácticas que se encuentran dentro del Manifesto de Ágile [6], como 'aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo' y 'entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible'.

En cambio, si se opta por seguir las fases de forma más rígida y definiéndolas desde el inicio, sin apenas modificaciones e iteraciones, se estaría aproximando más a una metodología en cascada como se podría ver en un proyecto de software [4]. La elección de una aproximación u otra viene dada por la naturaleza del proyecto y los objetivos que se tengan que cumplir [7].

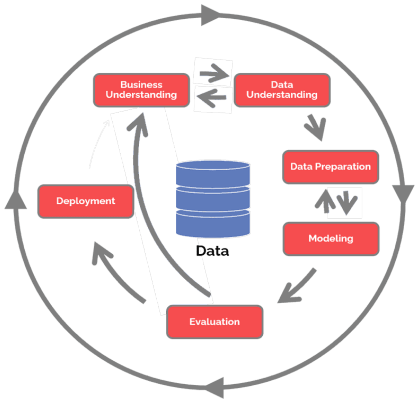

Figura 2.1: Esquema del ciclo CRISP-DM estándar [8].

2.2. Planificación

El plan inicial cuenta con un plan de fases, que se definió al inicio de este TFG como adaptación de la metodología CRISP-DM en su versión ágil (como se ha mencionado en el apartado 2.1).

Para conseguir que se siga el manifesto ágil [6], al final de cada fase se comunicará a los profesores los resultados obtenidos. De esta forma, conseguimos satisfacer la mayor prioridad de este manifesto, que es conseguir entregables de forma temprana y que contengan valor para la investigación. Además, mediante esta realimentación continua, se consigue una mejora sustancial de la productividad y comunicación constante, evitando de esta forma retrasos inesperados.

Este plan de fases se ha elegido debido a su gran versatilidad y eficacia, ya que es una estrategia que organiza las diversas etapas de manera flexible para completar el proyecto en su totalidad. Cada fase representa un conjunto de actividades relacionadas entre sí que conducen a la consecución de los objetivos ya definidos.

Se ha adaptado la metodología descrita de manera que cada fase se enfoque en un objetivo específico para lograr descomponer el proyecto en partes más pequeñas y manejables, siendo de especial interés a la hora de llevar a cabo el seguimiento real del proyecto y poder realizar una mejor estimación de los tiempos realizados.

Las fases adaptadas en comparación a CRISP-DM con las que cuenta este proyecto son:

1. **Definición del proyecto y objetivos generales de cada fase (relación con la fase 1 'Comprensión del negocio')**: En esta fase se expone la investigación que se va a llevar a cabo, además de la realización de la planificación y los objetivos que se tienen que cumplir para finalizar el proyecto con éxito.

   Una vez asentados los objetivos, se han diseñado las siguientes fases de acuerdo al plan inicial para conseguir definir el alcance del proyecto.

2. **Revisión bibliográfica general**: En esta fase se busca obtener el conocimiento necesario de la técnica del *Reservoir Computing*, nutriéndose tanto del marco teórico actual mediante artículos prestigiosos como de las aplicaciones prácticas que han dado mejores resultados.

   Esta fase sustituye a la fase 2 'Comprensión de los datos', ya que el conjunto de datos viene determinado por la empresa. Esta fase se realizará como una actividad extra de la fase asociada en este plan a la fase del ciclo CRISP-DM 3 'Preparación de los datos'.

Cabe destacar que esta búsqueda es preliminar e introductoria exclusivamente de dicha técnica, puesto que en cada modelo cuenta con su etapa específica de revisión bibliográfica para adentrarse en profundidad a los conceptos clave para satisfacer sus respectivas necesidades de implementación (ver la actividad 4).

3. **Preparación de los datos (relación directa con la fase 3)**: Para poder implementar las señales MEA adecuadamente, en esta fase se trataron de forma que los datos obtenidos después del preprocesamiento fueran representativos y de utilidad para la investigación.

4. **Construcción de los modelos (relación directa con la fase 4)**: Una vez se haya hecho una revisión exhaustiva de la bibliografía general y los datos estén correctamente tratados, empieza el desarrollo e implementación de los modelos.

   Debido a la densa concentración de tareas para cada modelo que alberga esta fase, se ha optado por realizar un flujo de actividades que se sigue de forma común para cumplir con 4 iteraciones (ya que se han desarrollado 4 modelos). El flujo de actividades de la iteración que se va a repetir por cada modelo consta de:

   - Revisión bibliográfica del modelo: De forma similar a la fase 2, pero haciendo una revisión exhaustiva de las ventajas y desventajas de cada modelo, así como aplicaciones teóricas y prácticas.
   - Implementación del modelo: Se implementa el modelo de acuerdo a lo establecido en la bibliografía revisada de forma que acepte las señales MEA.
   - Obtención de resultados iniciales: Una vez implementado el modelo, se evalúan los resultados obtenidos.
   - Ajuste de hiperparámetros: Se realiza una búsqueda de los hiperparámetros que mejor se adapten a las características de nuestros datos y se vuelve a reevaluar el modelo para obtener unos resultados mejorados.
   - Creación del entregable Al final de cada iteración, se realiza un entregable con la implementación y los resultados del modelo en desarrollo en esa iteración para enseñárselo a los tutores. De esta forma se busca que lo obtenido sea coherente y acorde a las características del conjunto de datos.

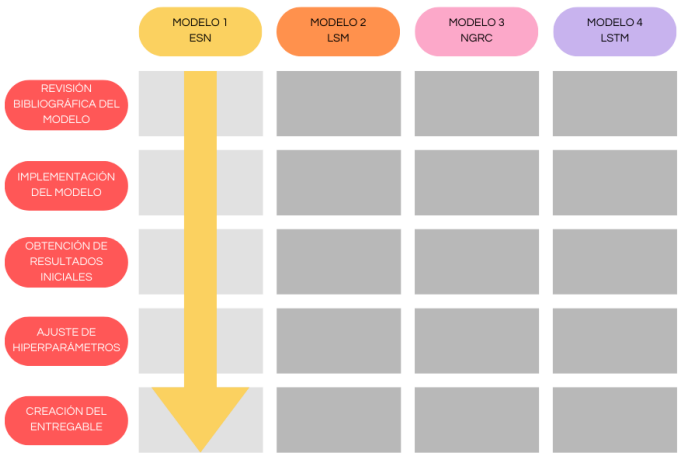

Figura 2.2: Imagen animada que muestra el flujo de actividades en cada iteración dentro de la fase 4. En caso de que no se admitan imágenes animadas, se mostrará la primera iteración. Elaboración propia, basada en [7].

Esta fase es crucial para finalizar el proyecto con éxito, por lo que se ha mantenido un contacto continuo y estrecho con los tutores durante su ejecución.

5. **Extracción de resultados y comparaciones (relación con la fase 5 'Evaluación del modelo')**: Cuando los modelos se encuentren ya adaptados a los datos, comienza la extracción de resultados que finalmente formará parte de las conclusiones finales del proyecto. Se va ejecutando cada modelo y, mediante el ajuste de los hiperparámetros realizado en las iteraciones de la fase anterior, se evalúan los resultados mediante visualizaciones y métricas para comparar los modelos entre sí y obtener las conclusiones resultantes.

6. En cuanto a la fase 6 'Despliegue', se ha divido en dos fases principales de la planificación de este TFG, ya que la fase de 'Redacción de la memoria' contiene un gran flujo de actividades y, de esta forma, se consigue una iteración más rápida.

   a) **Redacción de la memoria**: A medida que se vayan obteniendo resultados consistentes, se va a proceder a redactar este TFG. Al final de cada capítulo se contactará con los tutores para tener una retroalimentación continua de lo redactado en el informe y poder realizar los cambios pertinentes.
   Asimismo, se ha intentado mantener una estructura de la memoria (ver el apartado 1.4) similar a este plan de fases para un mejor seguimiento del mismo.

   b) **Presentación ante el tribunal**: Se preparará una presentación con los contenidos más importantes de este proyecto para exponerlo ante el tribunal.

2.3. Riesgos

Según el Project Management Body of Knowledge (PMBOK) [9], se define riesgo como 'un evento o condición incierta que, si ocurriese, tendría una repercusión positiva o negativa sobre los objetivos del proyecto'. Por ello, en este apartado se creará un plan de riesgos que analizará los posibles riesgos que pueden surgir a lo largo de este proyecto.

Para cada riesgo, se utilizará la Tabla 2.1. Se va a seguir un indicador de alto, medio o bajo para definir la probabilidad y el grado de riesgo. Después, se calculará el grado de riesgo total en función de dicha tabla.

Tabla 2.1: Matriz de nivel de riesgo según la probabilidad y el impacto de su suceso.

	Impacto		
Probabilidad	**Baja**	**Media**	**Alta**
Bajo	Bajo	Bajo	Medio
Medio	Bajo	Medio	Alto
Alto	Medio	Alto	Alto

2.3.1. Registro de riesgos

Para los riesgos identificados en este proyecto, se ha creado el siguiente registro de riesgos que contiene las siguientes secciones:

- **ID Riesgo**: identifica el riesgo de forma única.
- **Título**: enuncia el nombre del riesgo identificado.
- **Categoría**: clasifica el riesgo en personal, tecnología o gestión según la naturaleza del riesgo.
- **Estado**: progreso dentro del plan de mitigación de riesgos.
- **Descripción del riesgo**: breve introducción del riesgo.
- **Amenaza**: describe el factor de riesgo con potencial de provocar un perjucio o daño en el proyecto.
- **Mitigación recomendada por el riesgo**: medida para evitar que se materialice el riesgo.
- **Acciones correctivas**: medida para corregir en caso de que el riesgo se haya materializado.
- **Valores de probabilidad e impacto**: valores otorgados a los factores de probabilidad e impacto antes y después de crear una medida de mitigación del riesgo.

A continuación, se muestra el contenido del registro de riesgos:

Tabla 2.2: Riesgo 1 - Equipo de trabajo pequeño. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK001	**Título**	Equipo pequeño		
Categoría	Personal		**Estado**	Abierto	
Descripción del riesgo Debido a que el equipo de desarrollo de este TFG solamente cuenta con una persona, se depende de que esa persona cumpla con los plazos de entrega.					
Amenaza Puede tener una ausencia, una enfermedad o algún problema que haga que no se pueda avanzar de la forma deseada con el proyecto.					
Mitigación recomendada para el riesgo Contar con una amortiguación de retrasos, dejando al final de cada fase un breve periodo de tiempo para poder repasar lo realizado o actuar en consecuencia.					
Acciones correctivas Realizar una replanificación que se adecue a la situación concebida.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Medio
Postmitigación			Baja	Medio	Bajo

Tabla 2.3: Riesgo 2 - Modificación de los objetivos. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK002	**Título**	Modificación de los objetivos		
Categoría	Gestión		**Estado**	Abierto	
Descripción del riesgo Los objetivos se definen mal o tienen que ser modificados a la mitad del proyecto.					
Amenaza Retrasos en la entrega o incapacidad de finalizar el proyecto a tiempo.					
Mitigación recomendada para el riesgo Hablar con los tutores y resolver todas las dudas conceptuales que se tengan de los objetivos.					
Acciones correctivas Replanificar y ajustarse a los nuevos objetivos intentando no retrasar demasiado la entrega.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Media
Postmitigación			Baja	Medio	Bajo

Tabla 2.4: Riesgo 3 - Falta de experiencia. Elaboración propia.

REGISTRO DE RIESGO				
ID Riesgo	RSK003	**Título**	Falta de experiencia	
Categoría	Tecnología		**Estado**	Abierto
Descripción del riesgo No se cuenta con experiencia de ningún tipo en el tratamiento y análisis de señales y otros factores relevantes del campo de la bioinformática.				
Amenaza Se puede requirir más tiempo del planificado para comprender los conceptos básicos, o incluso no llegar a ser capaz de aplicarlos correctamente, provocando retrasos.				
Mitigación recomendada para el riesgo Dedicar las primeras semanas a la formación adecuada, preguntando todas las dudas que surjan a los tutores.				
Acciones correctivas En caso de no saber cómo proceder, volver a la bibliografía proporcionada por los tutores y examinarla en profundidad antes de continuar con el proyecto.				
Valores de probabilidad e impacto				
	Probabilidad	**Impacto**	**Riesgo total**	
Premitigación	Alta	Alto	Alto	
Postmitigación	Baja	Alto	Medio	

Tabla 2.5: Riesgo 4 - Falta de tiempo para perfeccionar aspectos relevantes. Elaboración propia.

REGISTRO DE RIESGO				
ID Riesgo	RSK004	**Título**	Falta de tiempo para perfeccionar	
Categoría	Gestión		**Estado**	Abierto
Descripción del riesgo Aspectos relevantes del proyecto no han cumplido con las expectativas.				
Amenaza El proyecto queda pobre, causando una mala impresión al tribunal.				
Mitigación recomendada para el riesgo Esfuerzo, dedicación y compromiso para entregar el proyecto en las mejores condiciones posibles.				
Acciones correctivas Replanificación del proyecto				
Valores de probabilidad e impacto				
	Probabilidad	**Impacto**	**Riesgo total**	
Premitigación	Media	Alto	Alto	
Postmitigación	Baja	Alto	Medio	

Tabla 2.6: Riesgo 5 - Pérdida de material relevante. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK005	**Título**	Pérdida del material relevante		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo Debido a un imprevisto, se pierde la memoria del proyecto o el código realizado.					
Amenaza Retrasos o incluso pérdida total del proyecto y haya que volver a empezar.					
Mitigación recomendada para el riesgo En el caso del código, se utilizará GitHub, ya que esta plataforma permite recuperar versiones anteriores en caso de pérdida de código. En cuanto a la memoria, se trabajará con Overleaf ya que permite editar el texto en línea.					
Acciones correctivas Copias de seguridad de forma local y remota en ambos casos.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Medio
Postmitigación			Bajo	Bajo	Baja

Tabla 2.7: Riesgo 6 - Inactividad de los servidores. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK006	**Título**	Inactividad de los servidores		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo Caída de los servidores temporal o permanente de GitHub y/o Overleaf durante el desarrollo del código o de la memoria de forma que afecte a la planificación inicial.					
Amenaza Retrasos o incluso pérdida total del proyecto y haya que volver a empezar.					
Mitigación recomendada para el riesgo Recurrir a copias de seguridad de la memoria y el código en la máquina local cada vez que se realice un cambio.					
Acciones correctivas Redundancia de los documentos y del código.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Medio
Postmitigación			Baja	Bajo	Bajo

Tabla 2.8: Riesgo 7 - Destrucción total o parcial del equipo de trabajo. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK007	**Título**	Destrucción total o parcial del equipo de trabajo		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo El equipo en el que se trabaja se avería de forma parcial o total.					
Amenaza Retrasos en la entrega o incapacidad de finalizar el proyecto.					
Mitigación recomendada para el riesgo Copias de seguridad de todo lo que se vaya a entregar, además de redundancia de equipos de trabajo, como pueden ser el equipo de la oficina y otro equipo personal.					
Acciones correctivas Migrar todo el contenido de un equipo a otro y continuar con el trabajo.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Medio
Postmitigación			Baja	Bajo	Bajo

Tabla 2.9: Riesgo 8 - Poca disponibilidad de los tutores. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK008	**Título**	Poca disponibilidad de los tutores		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo Los tutores estan ocupados cuando surje un imprevisto y se demora la respuesta.					
Amenaza Retrasos en la entrega.					
Mitigación recomendada para el riesgo Acordar reuniones con tiempo para que ambas partes estén reunidas a tiempo.					
Acciones correctivas En el caso de que la duda surja en la parte del desarrollo de los modelos, continuar con el documento de la memoria en lo que se espera una respuesta. Por el contrario, en caso de que surja en la parte de la memoria, perfeccionar el código ya realizado.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Media	Medio	Medio
Postmitigación			Media	Bajo	Bajo

Tabla 2.10: Riesgo 9 - Mala aplicación de la metodología elegida. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK009	**Título**	Mala aplicación de la metodología elegida		
Categoría	Gestión		**Estado**	Abierto	
Descripción del riesgo No aplicar correctamente la metodología.					
Amenaza Retrasos en la entrega.					
Mitigación recomendada para el riesgo Planificar con antelación el cronograma y repartir de forma adecuada las tareas siguiendo esa metodología. Preguntar a profesores con conocimientos en planificación para adaptarlo correctamente.					
Acciones correctivas Replanificar el cronograma.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Media	Medio	Medio
Postmitigación			Media	Bajo	Bajo

Tabla 2.11: Riesgo 10 - Insuficiencia de datos. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK010	**Título**	Insuficiencia de datos		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo No hay datos suficientes para llevar a cabo la tarea.					
Amenaza Modelo entrenado de forma pobre, haciendo que no generalice correctamente.					
Mitigación recomendada para el riesgo Hablar con los tutores y seguir las pautas dictaminadas por ellos.					
Acciones correctivas Informar a los tutores y generar más datos o consultar unos nuevos.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Media	Alto	Alto
Postmitigación			Media	Bajo	Bajo

Tabla 2.12: Riesgo 11 - Déficits de rendimiento de los equipos de trabajo. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK011	**Título**	Déficits de rendimiento		
Categoría	Tecnología		**Estado**	Abierto	
Descripción del riesgo Debido a una mala programación del código o recursos insuficientes (como la capacidad de cómputo) por parte del equipo, el modelo tarde demasiado en ejecutarse.					
Amenaza El programa no se ejecuta correctamente, desencadenando fallos.					
Mitigación recomendada para el riesgo Optimizar el código y los recursos empleados por el equipo.					
Acciones correctivas Probar a ejecutarlo en otros equipos con mejores prestaciones si los fallos persisten.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Baja	Alto	Medio
Postmitigación			Baja	Medio	Baja

Tabla 2.13: Riesgo 12 - Interpretación incorrecta de los resultados. Elaboración propia.

REGISTRO DE RIESGO					
ID Riesgo	RSK012	**Título**	Interpretación incorrecta de los resultados		
Categoría	Personal		**Estado**	Abierto	
Descripción del riesgo Los resultados se interpretan de forma incorrecta, llegando a conclusiones erróneas.					
Amenaza Causar mala impresión al tribunal, afectando a la nota final del proyecto.					
Mitigación recomendada para el riesgo Comunicar los resultados a los tutores.					
Acciones correctivas Corregir lo erróneo.					
Valores de probabilidad e impacto					
			Probabilidad	**Impacto**	**Riesgo total**
Premitigación			Media	Alto	Alto
Postmitigación			Baja	Medio	Bajo

Tabla 2.14: Riesgo 13 - Pérdida de los resultados. Elaboración propia.

REGISTRO DE RIESGO				
ID Riesgo	RSK013	**Título**	Pérdida de los resultados	
Categoría	Gestión		**Estado**	Abierto
Descripción del riesgo Los resultados y las conclusiones obtenidas se pierden por errores humanos o del sistema.				
Amenaza Retrasos en la entrega o incapacidad de finalizar el proyecto a tiempo.				
Mitigación recomendada para el riesgo Usar la herramienta Excel en línea para tener guardado el documento con los resultados a medida que se vayan obteniendo en la nube y crear una copia local cada vez que se modifique.				
Acciones correctivas Replanificar y volver a obtener los resultados de nuevo.				
Valores de probabilidad e impacto				
		Probabilidad	**Impacto**	**Riesgo total**
Premitigación		Baja	Alto	Media
Postmitigación		Baja	Bajo	Bajo

Tabla 2.15: Riesgo 14 - Pérdida del contenido bibliográfico. Elaboración propia.

REGISTRO DE RIESGO				
ID Riesgo	RSK014	**Título**	Pérdida del contenido bibliográfico	
Categoría	Gestión		**Estado**	Abierto
Descripción del riesgo Al tener que indagar entre muchos artículos, no guardarlos debidamente y tener que escribir la bibliografía a mano al final del proyecto.				
Amenaza Ser totalmente ineficaz y malgastar tiempo valioso que se podría emplear en perfeccionar la memoria y el código.				
Mitigación recomendada para el riesgo Usar una herramienta en línea que gestione la bibliografía. Se recomienda el uso de Zotero, ya que se pueden sincronizar varios dispositivos y permite el guardado de los artículos con unos pocos pasos. Además, permite exportarlo directamente en el formato exigido por Overleaf.				
Acciones correctivas Replanificar e intentar recuperar la bibliografía lo antes posible.				
Valores de probabilidad e impacto				
		Probabilidad	**Impacto**	**Riesgo total**
Premitigación		Media	Alto	Alto
Postmitigación		Baja	Bajo	Bajo

2.4. Seguimiento

En este apartado se mostrará el seguimiento de las tareas realizadas durante el proyecto.

Para la realización de este proyecto, se han destinado 8 horas al día de lunes a viernes sin incluir festivos que se han repartido a lo largo de la duración total del proyecto.

Se han organizado distintas tablas, una para cada fase, en las que se muestra la duración de cada tarea. Esta duración se descompondrá en real y estimada durante la planificación en horas. También se expondrán las fechas de inicio y de final estimadas y reales.

Fase 1: Definición del proyecto y objetivos generales de cada fase						
Tarea	**Inicio Estimado**	**Final Estimado**	**Inicio Real**	**Final Real**	**Duración Estimada (horas)**	**Duración Real (horas)**
Determinar los objetivos	25-03-2024	25-03-2024	25-03-2024	25-03-2024	1	1
Elección de la metodología a seguir	25-03-2024	25-03-2024	25-03-2024	25-03-2024	3	4
Diseñar el plan de fases	25-03-2024	25-03-2024	25-03-2024	25-03-2024	2	1
Realizar el registro de riesgos	25-03-2024	25-03-2024	25-03-2024	25-03-2024	2	2

Figura 2.3: Seguimiento de la fase 1 y sus tareas. Elaboración propia.

La Figura 2.3 resalta el tiempo dedicado a la planificación del proyecto y la 2.4 el tiempo dedicado a la revisión bibliográfica de la arquitectura RC. Como se puede observar, la duración estimada y la real casi no difieren.

Fase 2: Revisión bibliográfica general						
Tarea	**Inicio Estimado**	**Final Estimado**	**Inicio Real**	**Final Real**	**Duración Estimada (horas)**	**Duración Real (horas)**
Recopilar bibliografía de RC	01-04-2024	01-04-2024	01-04-2024	01-04-2024	5	7
Evaluación y selección de la bibliografía relevante	01-04-2024	01-04-2024	01-04-2024	02-04-2024	3	2
Leer la bibliografía seleccionada	02-04-2024	02-04-2024	02-04-2024	02-04-2024	4	3.5
Realizar apuntes	02-04-2024	02-04-2024	02-04-2024	02-04-2024	4	3.5

Figura 2.4: Seguimiento de la fase 2 y sus tareas. Elaboración propia.

La figura (2.5) muestra el tiempo dedicado a la preparación de los datos. El tiempo real dedicado es bastante menor al estimado debido a que los datos de las señales MEA que se proporcionaron eran muy similares a los trabajados en las practicas curriculares.

Para trabajar más eficazmente, se modificó el código que se utilizó en dichas prácticas para tratar los datos. Los cambios fueron mínimos y se trabajó especialmente rápido porque ya se tenían los conocimientos básicos de cómo preparar los datos.

Fase 3: Preparación de los datos						
Tarea	**Inicio Estimado**	**Final Estimado**	**Inicio Real**	**Final Real**	**Duración Estimada (horas)**	**Duración Real (horas)**
Estudiar y limpiar de datos	03-04-2024	03-04-2024	03-04-2024	03-04-2024	4	**1**
Preprocesar las señales	03-04-2024	04-04-2024	03-04-2024	03-04-2024	**8**	**3**
Normalizar y estandarizar los datos	04-04-2024	04-04-2024	03-04-2024	03-04-2024	**2**	**0.5**
Realizar una reducción de los datos	04-04-2024	04-04-2024	03-04-2024	03-04-2024	2	**0.5**

Figura 2.5: Seguimiento de la fase 3 y sus tareas. Elaboración propia.

La Figura 2.6 tiene la mayor variabilidad en los tiempos estimados y reales. Esto se debe a que se sobrestimó la duración de algunos modelos y, como se muestra en el seguimiento de la fase 3 (ver Figura 2.5), se ganaron 9 horas que fueron de vital importancia para el desarrollo de los modelos.

Además, las iteraciones de los modelos Next Generation Reservoir Computing (NGRC) y Liquid State Machine (LSM) intercambiaron posiciones dentro del flujo de tareas de la fase 4 debido a tres factores:

1. El modelo Echo State Networks (ESN) se consiguió antes de lo previsto, dejando 24 horas libres.
2. Se encontraron pocas referencias prácticas de LSM, dificultando aún más la realización de este modelo.
3. Un compañero de la oficina había estado informándose del modelo NGRC con anterioridad, por lo que podría ser de gran ayuda.

Finalmente, teniendo en cuenta los factores mencionados, se decidió adelantar el desarrollo de este modelo. Esta decisión fue acertada, ya que agilizó el proceso de desarrollo y, comparando la duración estimada y real total de ambos modelos, se consiguieron 32 horas para futuras fases.

En cuanto al modelo de LSTM, se tenía que realizar en Google Colab, empleando tecnologías diferentes a las usadas en los otros modelos y, además, se informó que la curva de aprendizaje era lenta con respecto a la implementación en el dispositivo Dev Board de Google Coral, por lo que se estimó bastante tiempo de producción, siendo una de las tareas que más tiempo requeriría de todo el proyecto. No obstante, la realidad fue otra, pues la implementación del modelo se consiguió en la mitad del tiempo estimado, consiguiendo adelantar 80 horas que se destinarían a las fases posteriores.

Fases 4 : Construcción de los modelos						
Tarea	Inicio Estimado	Final Estimado	Inicio Real	Final Real	Duración Estimada (horas)	Duración Real (horas)
Realizar la iteración del modelo 1 (ESN)	08-04-2024	12-04-2024	04-04-2024	09-04-2024	40	32
Realizar la iteración del modelo 2 (LSM)	15-04-2024	22-04-2024	12-04-2024	24-04-2024	48	56
Realizar la iteración del modelo 3 (NGRC)	24-04-2024	03-05-2024	10-04-2024	11-04-2024	56	16
Realizar la iteración del modelo 4 (LSTM)	06-05-2024	17-05-2024	24-04-2024	30-04-2024	80	40

Figura 2.6: Seguimiento de la fase 4 y sus tareas. Elaboración propia.

En la fase 5, reflejada en la Figura 2.7, las horas estimadas y reales no difieren apenas, pero donde sí se puede ver una gran diferencia es en las fechas de inicio y final estimadas, ya que prácticamente difieren en 18 días, siendo útiles unas 88 horas aproximadamente.

Fases 5 : Extracción de conclusiones y resultados						
Tarea	Inicio Estimado	Final Estimado	Inicio Real	Final Real	Duración Estimada (horas)	Duración Real (horas)
Obtener resultados	20-05-2024	21-05-2024	02-05-2024	02-05-2024	10	8
Comparar resultados	21-05-2024	21-05-2024	03-05-2024	03-05-2024	3	3
Extraer conclusiones	21-05-2024	21-05-2024	03-05-2024	03-05-2024	3	1

Figura 2.7: Seguimiento de la fase 5 y sus tareas. Elaboración propia.

Antes de finalizar con el seguimiento del proyecto presentando la fase 6, cabe resaltar que existe un hito (el rombo resaltado de color rojo en la Figura 2.10) en esta fase.

En el periodo de tiempo que dura el hito, se han revisado fases anteriores para asegurar que todo estuviera en las mejores condiciones posibles.

Esto no computa dentro de las horas reflejadas en el seguimiento, pues la revisión se hacía de la fase que se estimase, sin planificación previa y sin seguir un orden establecido.

Teniendo esto en cuenta, se reservó este periodo de tiempo para evitar retrasos innecesarios en la redacción por tener que modificar algún fragmento de código.

Fase 6: Despliegue						
Tarea	**Inicio Estimado**	**Final Estimado**	**Inicio Real**	**Final Real**	**Duración Estimada (horas)**	**Duración Real (horas)**
6.a Redacción de la memoria						
Elaborar la portada, índices, resumen y abstract	22-05-2024	22-05-2024	03-05-2024	03-05-2024	**4**	**4**
Elaborar el resto de capítulos	23-05-2024	05-06-2024	15-05-2024	11-06-2024	**80**	**160**
6.b Presentación ante el tribunal						
Crear la presentación de la defensa	06-06-2024	07-06-2024	12-06-2024	13-06-2024	**16**	**16**
Preparar la defensa del TFG	10-06-2024	28-06-2024	17-06-2024	28-06-2024	-	-

Figura 2.8: Seguimiento de la fase 6 y sus tareas. Elaboración propia.

En esta fase, se estimaron menos horas de las reales para la elaboración de los capítulos centrales de la memoria.

Esto se debe a que no se precisaron adecuadamente los tiempos de revisión del documento, ni los tiempos en los que tardaba el documento en cargar en la nube de Overleaf, suponiendo un retraso del doble de lo estimado.

Aún así, gracias al tiempo ahorrado en otras fases, esto no ha afectado negativamente al proyecto, pudiendo seguir adelante sin afectar a la entrega final de la memoria.

Además, se puede contemplar en la Figura 2.8 que la tarea 'Preparar la defensa del TFG' no tiene duración asignada.

Esto se debe a que es una tarea que tampoco computa en las horas totales del proyecto, pues esta tarea se realiza para repasar la presentación sin un orden predefinido ni unas horas asignadas a su realización.

El cómputo total de horas, que se puede extraer de la Figura 2.9, asciende a **365 horas reales**, frente a las **380 horas estimadas** inicialmente.

Color	EDT	Título / Nombre de tarea	Fecha de inicio	Fecha de fin	Duración (horas)
	1	**FASE 1**	**3/25/2024**	**3/25/2024**	**8**
	1.1	Determinar los objetivos	3/25/2024	3/25/2024	1
	1.2	Elección de la metodología a seguir	3/25/2024	3/25/2024	4
	1.3	Diseñar el plan de fases	3/25/2024	3/25/2024	1
	1.4	Realizar el registro de riesgos	3/25/2024	3/25/2024	2
	2	**FASE 2**	**4/1/2024**	**4/2/2024**	**16**
	2.1	Recopilar bibliografía de RC	4/1/2024	4/1/2024	7
	2.2	Evaluar y seleccionar bibliografía relevante	4/1/2024	4/2/2024	2
	2.3	Leer la bibliografía seleccionada	4/2/2024	4/2/2024	3.5
	2.4	Realizar apuntes	4/2/2024	4/2/2024	3.5
	3	**FASE 3**	**4/3/2024**	**4/3/2024**	**5**
	3.1	Estudiar y limpiar datos	4/3/2024	4/3/2024	1
	3.2	Prepocesar las señales	4/3/2024	4/3/2024	3
	3.3	Normalizar y estandarizar los datos	4/3/2024	4/3/2024	0.5
	3.4	Realizar una reducción de los datos	4/3/2024	4/3/2024	0.5
	4	**FASE 4**	**4/4/2024**	**4/30/2024**	**144**
	4.1	**Realizar la iteración del modelo 1 (ESN)**	**4/4/2024**	**4/9/2024**	**32**
	4.1.1	Revisión bibliográfica del modelo	4/4/2024	4/4/2024	8
	4.1.2	Implementación del modelo	4/5/2024	4/5/2024	4
	4.1.3	Obtención de resultados iniciales	4/5/2024	4/5/2024	4
	4.1.4	Ajuste de hiperparámetros	4/8/2024	4/9/2024	12
	4.1.5	Creación del entregable	4/9/2024	4/9/2024	4
	4.2	**Realizar la iteración del modelo 2 (LSM)**	**4/12/2024**	**4/22/2024**	**56**
	4.2.1	Revisión bibliográfica del modelo	4/12/2024	4/16/2024	20
	4.2.2	Implementación del modelo	4/16/2024	4/18/2024	16
	4.2.3	Obtención de resultados iniciales	4/18/2024	4/18/2024	4
	4.2.4	Ajuste de hiperparámetros	4/19/2024	4/22/2024	12
	4.2.5	Creación del entregable	4/22/2024	4/22/2024	4
	4.3	**Realizar la iteración del modelo 3 (NGRC)**	**4/10/2024**	**4/11/2024**	**16**
	4.3.1	Revisión bibliográfica del modelo	4/10/2024	4/10/2024	2
	4.3.2	Implementación del modelo	4/10/2024	4/11/2024	8
	4.3.3	Obtención de resultados iniciales	4/11/2024	4/11/2024	1
	4.3.4	Ajuste de hiperparámetros	4/11/2024	4/11/2024	4
	4.3.5	Creación del entregable	4/11/2024	4/11/2024	1
	4.4	**Realizar la iteración del modelo 4 (LSTM)**	**4/24/2024**	**4/30/2024**	**40**
	4.4.1	Revisión bibliográfica del modelo	4/24/2024	4/24/2024	8
	4.4.2	Implementación del modelo	4/25/2024	4/29/2024	20
	4.4.3	Obtención de resultados iniciales	4/29/2024	4/30/2024	6
	4.4.4	Ajuste de hiperparámetros	4/30/2024	4/30/2024	1
	4.4.5	Creación del entregable	4/30/2024	4/30/2024	5
	5	**FASE 5**	**5/2/2024**	**5/3/2024**	**12**
	5.1	Obtener resultados	5/2/2024	5/2/2024	8
	5.2	Comparar resultados	5/3/2024	5/3/2024	3
	5.3	Extraer conclusiones	5/3/2024	5/3/2024	1
	6	**FASE 6**	**5/3/2024**	**6/13/2024**	**180**
	6.1	**Redacción de la memoria**	**5/3/2024**	**6/11/2024**	**164**
	6.1.1	Elaborar la portada, índices, resumen y abstract	5/3/2024	5/3/2024	4
	6.1.2	Revisar todo lo realizado en las fases anteriores	5/6/2024	5/14/2024	0
	6.1.3	Elaborar el resto de capítulos	5/15/2024	6/11/2024	160
	6.2	**Presentación ante el tribunal**	**6/12/2024**	**6/13/2024**	**16**
	6.2.1	Crear la presentación de la defensa	6/12/2024	6/13/2024	16

Figura 2.9: Seguimiento del diagrama de Gantt. Elaboración propia.

Figura 2.10: Diagrama de Gantt. Elaboración propia.

2.5. Estimación de costes

Para finalizar el capítulo de gestión del proyecto, se ha decidido hacer un presupuesto de lo que supondría trasladar la investigación a un equipo de trabajo fuera de la universidad. Cabe destacar que este proyecto no tiene ningún fin comercial, ya que está destinado única y exclusivamente a realizar un TFG meramente académico.

Para empezar con el presupuesto, se debería contar inicialmente con un equipo de trabajo. El rango del sueldo base anual de un ingeniero de Machine Learning (ML) en España oscila entre los 29.000€ - 50.000€ a día 16 de mayo de 2024 por [10], resultando en un sueldo medio de 42.641€. Si contamos con una remuneración adicional (que pueden incluir bonificaciones, comisiones, propinas, etc.) que comprende desde los 950€ hasta los 6400€, estimando un valor de dicha remuneración de 3.171€ (aprox 264,25€/mes extra), el sueldo anual total es de 45.842€, por lo que el sueldo final resultante es de aproximadamente **3.820,17€/mes**.

No obstante, para los conocimientos clave sobre señales MEA y las nociones sobre cómo realizar el posterior tratamiento de dichas señales, se debería contar con un profesional de la rama médica. Suponiendo que se desease contratar a un ingeniero biomédico, ya que sus destrezas tanto en el ámbito ingenieril como en el médico aportarían un gran valor añadido al resultado del proyecto, se sumarían los siguientes costes. Siguiendo la misma lógica de antes, el rango de sueldo base oscila entre los 22.000€ - 30.000€ a día 16 de mayo de 2024 por [11], dando lugar a un sueldo medio anual de 26.264€. Por otro lado, las remuneraciones adicionales oscilan entre los 1427€ y 1549€, con una estimación media de 1488€ y, por tanto, un sueldo medio total de 27.752€/año. Para la duración del proyecto, se estima que dicho ingeniero obtenga **2.344,17€/mes**.

El contratista no solo paga por los sueldos base, sino que también se tienen que añadir otros factores [12]. En concreto, el coste de un trabajador al mes esta formado por:

- Sueldo: salario bruto que recibe el empleado cada mes indicado en su nómina. En este caso es lo calculado en el párrafo anterior.
- Seguridad Social: cotizaciones que debe pagar la empresa por cada empleado a la Seguridad Social. Los factores con sus porcentajes respecto al salario bruto, suponiendo que los contratados no tienen condiciones especiales, que se han contemplado en este TFG son los siguientes:
  - Cotización por contingencias comunes: 23,6 %
  - Cotización por formación: 0,6 %
  - Cotización por desempleo: 6,7 %
  - Cotización por accidentes de trabajo y enfermedades profesionales: 3,5 %
  - Fondo de Garantía Salarial (FOGASA): 0,2 %
  - Mecanismo de Equidad Intergeneracional (MEI) en 2024: 0,58 %
- Costes adicionales: otros aportes sociales que se deben realizar. En este caso, solo se va a tener en cuenta el Impuesto sobre la renta de las Personas Físicas (IRPF) con una retención del 2 % (contrato temporal de menos de un año de duración).

Por tanto, se podría deducir a través de la Ecuación 2.1 el coste de un empleado al mes según lo mencionado.

$$C = SB + SS + IRPF \tag{2.1}$$

Donde SB se refiere al salario bruto del trabajador. Suponiendo que la fórmula de la Seguridad Social (2.2) al sumar todos los porcentajes de las cotizaciones que la componen es:

$$SS = SB * 0,3518 \tag{2.2}$$

Y la del IRPF:

$$IRPF = SB * 0,02 \tag{2.3}$$

Obtenemos mediante las Ecuaciones 2.1, 2.2 y 2.3 que lo que tiene que pagar el contratista por cada empleado al mes es **5.240,51€** por el ingeniero de ML y **3.215,73€** por el ingeniero biomédico. Como se puede observar en el apartado 2.4, el proyecto ha comprendido una duración aproximada de 3 meses, por tanto, el coste de salarios para la duración del proyecto es de **25.368,72€**.

Una vez tratados los salarios, se calcularán otros costes asociados al proyecto, como puede ser el equipo de trabajo y el coste de otros dispositivos asociados.

Ya que el equipo informático no contaba con una Graphics Processing Unit (GPU) integrada, todos los procesos se corren en Central Processing Unit (CPU). Este equipo contaba con el sistema operativo Windows 11 instalado, un procesador Intel(R) Core(TM) i5-11400 con una frecuencia básica de 2.60GHz [13], una memoria RAM instalada de 16GB (15,8GB usables) y con un Solid State Drive (SSD) de 500GB. Un equipo con estas características actualmente supone un coste aproximado de **497,60€** [14], aunque se pueden buscar otras alternativas con mejores prestaciones si se desea.

En el caso del modelo LSTM que se desarrolló en Google Colab no necesita de ningún plan, pero en el caso de que se quiera contar con características premium, el plan Colab Pro [15] supone 11,9€/mes, por lo que para la duración del proyecto se estimarían **35,7€**.

Algunas de las ventajas más relevantes para el proyecto que incluye este plan son 100 unidades informáticas al mes, GPU más rápidas y más memoria. Esto puede ser útil a la hora de ejecutar el modelo con muchos más datos.

Otras utilidades adicionales que proporciona este plan es la posibilidad de usar una terminal con acceso a la máquina virtual y asesoramiento mediante IA.

Para el dispositivo de Google Coral [16], las inferencias se han realizado sobre un Google Coral Dev Board, el precio asociado es **199€** [17].

En caso de que surjan imprevistos durante el proyecto o un aumento en los gastos de forma inesperada, se ha reservado un 10 % extra del total para sufragar dichos gastos. De esta forma, los gastos totales ascienden a la cantidad indicada en la Tabla 2.16.

Gasto	**Sin Colab Pro**	**Con Colab Pro**
Salarios	25.368,72€	25.368,72€
Equipo informático	497,60€	497,60€
Colab Pro (opcional)	-	35,7€
Google Coral Dev Board	199€	199€
Subtotal	26.065,32€	26.101,02€
Margen de contingencia (10 %)	2.606,53€	2.610,10€
Total	**28.671,85€**	**28.711,12€**

Tabla 2.16: Gastos totales del proyecto. Elaboración propia.

En conclusión, se deberían destinar aproximadamente 30.000€ en caso de querer replicar este proyecto.

Capítulo 3

Tecnologías empleadas

Este proyecto se ha desarrollado completamente en Python, en Jupyter Notebook. Para su almacenamiento, se ha empleado un repositorio de Github que contiene una carpeta para cada modelo con su respectivo código. La redacción de la memoria se ha hecho completamente con Overleaf.

Para la implementación, se han empleado:

- Visual Studio Code para la implementación de la mayoría de modelos por su gran versatilidad y comodidad.
- Para el desarrollo del modelo de la red LSTM se ha usado Google Colab. Esto se debe al uso de Tensor Processing Unit (TPU) en los dispositivos de Google Coral, y con Google Colab se abría la posibilidad de alojar tensores y desarrollar este modelo de forma remota, permitiendo así el poder trabajar con lo necesario en TPU y llevar a cabo este modelo en el equipo de trabajo del que se disponía.

Las bibliotecas empleadas son las siguientes:

3.1. Tensorflow

Tensorflow [18] es una biblioteca de software de código abierto para el desarrollo de modelos de aprendizaje automático desarrollada por Google. Fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015.

El nombre TensorFlow deriva de las operaciones que tales redes neuronales realizan sobre arrays multidimensionales de datos. Estos arrays multidimensionales son referidos como "tensores".

Los tensores son una generalización de los vectores y las matrices, permitiendo representar datos con más de dos dimensiones. Un vector unidimensional es un tensor de rango 1, una matriz bidimensional es un tensor de rango 2 y así sucesivamente.

Se pueden aplicar diversas operaciones matemáticas a los tensores, como suma, resta, multiplicación, convolución, activación y pooling, las cuales son esenciales para el entrenamiento y la ejecución de redes neuronales.

Estos tensores presentan grandes ventajas a la hora de programar los modelos, entre las que se encuentran:

- **Representación de datos multidimensionales**: Los tensores pueden almacenar y procesar datos multidimensionales, como es el caso de las series temporales y, en este contexto, de los datos disponibles de señales MEA.
- **Eficiencia computacional**: TensorFlow está optimizado para el procesamiento de tensores, lo que permite realizar cálculos numéricos y operaciones de aprendizaje automático de manera eficiente, incluso en conjuntos de datos grandes.
- **Flexibilidad y versatilidad**: Los tensores se pueden utilizar para construir una amplia gama de modelos de aprendizaje automático, siendo de gran utilidad para el desarrollo de RNN.
- Integración con otras bibliotecas: TensorFlow se integra fácilmente con otras bibliotecas de Python como NumPy (apartado 3.5), Pandas (apartado 3.2) y Scikit-learn (apartado 3.7), facilitando la manipulación y preprocesamiento de datos antes de alimentarlos a los modelos.

TensorFlow permite a los desarrolladores construir y entrenar modelos de aprendizaje profundo de manera eficiente. Esta arquitectura puede ejecutarse en una variedad de plataformas, como CPU, GPU y TPU.

En este TFG, se ha empleado esta biblioteca para alojar tensores y usar herramientas y bibliotecas propias para poder desarrollar e implementar el modelo LSTM en el dispositivo Google Coral. Entre estas bibliotecas se incluyen:

3.1.1. Keras

Keras [19] es una Application Programming Interface (API) diseñada para posibilitar la experimentación con redes de aprendizaje profundo. En 2017, el equipo de TensorFlow de Google decidió ofrecer soporte a Keras en la biblioteca de core de TensorFlow, con la llegada de Tensorflow 2.0.

Entre las ventajas de usar Keras para el desarrollo de una LSTM se encuentran:

1. **Simplicidad y facilidad de uso**: es una API intuitiva y de alto nivel que facilita la creación y entrenamiento de diversos tipos de RNN, incluyendo la LSTM.
2. **Flexibilidad y escalabilidad**: el modelo LSTM desarrollado en Keras permite una personalización de las capas internas y otros factores que influyen en los modelos, ayudando a una mejor optimización del modelo. En cuanto a la escalabilidad, Keras permite escalar los modelos de forma que se pueda manejar fácilmente grandes conjuntos de datos y tareas complejas.
3. **Eficiencia y rendimiento**: Keras optimiza automáticamente el entrenamiento de las redes LSTM, permitiendo obtener resultados precisos y eficientes en un tiempo razonable y aprovechando el hardware del que se dispone. Esto es de gran utilidad al utilizar una Google Coral, debido a su disponibilidad y requisitos para el uso de su hardware.

En este TFG, se ha empleado esta biblioteca para definir y guardar el modelo LSTM, ya que es capaz de definir modelos y ejecutarlos de forma temprana. Con esto se consigue facilitar el desarrollo de modelos, posibilitar la depuración de modelos utilizando herramientas estándar, y simplificar la definición de modelos dinámicos utilizando estructuras de control.

3.1.2. Tflite

TensorFlow Lite [20] es un marco de software de código abierto desarrollado por Google para ejecutar modelos de aprendizaje automático en dispositivos móviles y de baja potencia.

Permite convertir modelos entrenados en TensorFlow a un formato más ligero y eficiente para su implementación en dispositivos con recursos limitados, como puede ser un dispositivo de Google Coral.

Por ello, se ha utilizado en este TFG, ya que la combinación de TensorFlow Lite y Google Coral ofrece una solución potente y flexible para implementar modelos en dispositivos de baja potencia, ofreciendo así un alto rendimiento, baja latencia y eficiencia energética.

3.2. Pandas

Pandas [21] es una biblioteca de código abierto de alto rendimiento, construida sobre NumPy (apartado 3.5), diseñada específicamente para el trabajo con datos estructurados en Python. Proporciona estructuras de datos y operaciones potentes para la manipulación, análisis y visualización de datos.

Las estructuras de datos principales incluyen las series (array unidimensional), los DataFrames (estructura de datos bidimensional con forma de tabla) y el panel (estructura de datos tridimensional para almacenar datos con tres ejes de indexación).

Debido a la gran ventaja que supone representar los datos en formato de tabla para representar sus características y estructuras principales, se ha optado por usar en este TFG los DataFrame debido a la naturaleza de los datos.

3.3. ReservoirPy

ReservoirPy [22] es una biblioteca sencilla y fácil de usar basada en módulos científicos de Python. Proporciona una interfaz flexible para implementar arquitecturas eficientes de RC con un enfoque particular en ESN. Las funciones avanzadas de ReservoirPy permiten mejorar la eficiencia del tiempo de cálculo en comparación con la implementación básica de Python, con conjuntos de datos de cualquier tamaño.

Debido a que RC tiene como objetivo hacer que la complejidad surja de la aparente simplicidad, ReservoirPy proporciona a sus usuarios herramientas basadas en Scipy (apartado 3.4) y Numpy (apartado 3.5) que pueden realizar una amplia gama de tareas de aprendizaje automático, en particular cuando se trata de datos secuenciales.

Debido a todas las ventajas que ofrece esta biblioteca, se ha empleado en este TFG para el desarrollo de la ESN usando el tutorial y los consejos proporcionados en la guía de la página oficial de ReservoirPy con el objetivo de obtener los mejores resultados posibles.

3.4. Scipy

SciPy [23] es una biblioteca gratuita y de código abierto usada para computación científica y técnica en Python. Agrega un poder significativo a Python al proporcionar al usuario comandos y clases de alto nivel para manipular y visualizar datos.

Se creó a partir de distintos módulos de extensión para Python y fue lanzada en 1999 bajo el nombre de

Multipack, llamada así por los paquetes netlib que reunían a ODEPACK, QUADPACK, y MINPACK.

SciPy se basa en Numpy (apartado 3.5), proporciona módulos para optimización, álgebra lineal, integración, interpolación y otras tareas científicas y de ingeniería. Gracias a ello, se consigue una gran colección de funciones accesibles, fáciles de usar y completamente gratuitas.

Como cuenta con una amplia gama de funciones dedicadas al procesamiento de señales, se ha usado Scipy para el preprocesamiento y tratamiento de las señales MEA.

3.5. Numpy

NumPy [24] es una biblioteca de Python que proporciona arrays multidimensionales, varios objetos derivados y una variedad de rutinas para operaciones rápidas en matrices, incluidas matemáticas, lógicas, manipulación de formas, clasificación, selección, E/S., transformadas discretas de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.

Numpy se ha usado en este TFG, ya sea como parte de otras bibliotecas de forma intrínseca o como biblioteca individual con sus propias funciones.

Su uso se debe principalmente a que facilita el manejo de grandes volúmenes de datos de texto, y algunas de sus funciones han permitido desarrollar aspectos relevantes de este TFG como la extracción de los datos y el entrenamiento de los modelos de aprendizaje automático para realizar el análisis de las señales MEA.

3.6. Matplotlib

Matplotlib [25] es una biblioteca de código abierto para Python ampliamente utilizada para la creación de gráficos 2D de alta calidad.

Es una herramienta versátil y popular que se puede usar para crear una amplia gama de gráficos personalizados permitiendo su exportación (desde los más sencillos como gráficos de líneas o barras, hasta algunos más complejos como diagramas de caja y bigotes).

En este TFG, se ha empleado esta biblioteca principalmente para la visualización de diversas funciones con el submódulo "pyplot".

Esto ha sido de vital importancia en este proyecto, pues se han realizado visualizaciones tanto de los datos de las señales MEA en crudo, como del procesamiento y la elaboración de los resultados de cada modelo.

3.7. Scikit-learn

Scikit-learn [26] es una biblioteca gratuita y de código abierto construida sobre SciPy (apartado 3.4), NumPy (apartado 3.5) y Matplotlib (apartado 3.6) para aprendizaje automático en Python. Proporciona herramientas eficientes y fáciles de usar para una amplia gama de tareas de aprendizaje automático.

En este TFG, Scikit-learn ha sido de gran utilidad a la hora de la división de los datos, usando técnicas como train-test split o Stratified K-Fold (usado en el método personalizado de tratamiento de series temporales del Capítulo 6).

3.8. Optuna

Optuna [27] es un marco de código abierto para la optimización automática de hiperparámetros en Python.

Para encontrar el conjunto de hiperparámetros óptimo, Optuna utiliza un enfoque basado en la búsqueda Bayesiana. Este enfoque utiliza una distribución de probabilidad para representar la incertidumbre sobre los valores óptimos de los hiperparámetros. A medida que Optuna explora el espacio de hiperparámetros, actualiza la distribución de probabilidad en función del rendimiento del modelo en las combinaciones de hiperparámetros probadas. Esto le permite a Optuna enfocarse en las regiones del espacio de hiperparámetros con mayor probabilidad de contener los valores óptimos.

Al hacerlo de esta forma, se consigue un incremento del rendimiento y un gran ahorro de tiempo (gracias a su facilidad de exploración). Si se hace una comparativa con la búsqueda de hiperparámetros manual tradicional, se ha encontrado una mejora lo suficientemente grande como para considerar utilizar esta herramienta en este TFG.

3.9. Joblib

Joblib [28] es una biblioteca de Python para computación paralela y serialización de objetos. Se suele utilizar para ejecutar código en múltiples procesadores o núcleos.

Para entender su uso en este TFG, hay que tener en cuenta las siguientes definiciones:

- La **computación paralela** es un paradigma de computación que distribuye un problema en múltiples procesadores o núcleos para ejecutarlo simultáneamente. Esto puede mejorar significativamente el rendimiento, especialmente para tareas computacionalmente intensivas.

  Sin embargo, el uso de la computación paralela trae consigo algunas desventajas, como pueden ser las condiciones de carrera (que ocurre cuando varios hilos o procesos acceden o manipulan recursos compartidos sin sincronización, logrando de esta forma resultados erróneos o impredecibles), la posible pérdida de datos y el tiempo extra en transformar un código secuencial.

- La **serialización de objetos** es el proceso de convertir un objeto en una forma que se puede almacenar o transmitir. Esto puede ser útil para guardar datos en un archivo, enviar datos a través de una red o compartir datos entre diferentes procesos.

Ya que el equipo de trabajo no cuenta con una GPU, el tiempo de búsqueda de hiperparámetros sería excesivo, por lo que se ha utilizado esta biblioteca para distribuir la optimización de hiperparámetros realizada con Optuna (apartado 3.8) y así mejorar los tiempos de cómputo (para más información, ver el Capítulo 6).

Aún así, sería preferible usar una GPU, pues la computación paralela en CPU presenta algunas desventajas, como se han mencionado anteriormente.

Capítulo 4

Estado del arte

Para dar contexto y fundamentar la investigación propuesta en este TFG, este capítulo se va a enfocar en proporcionar una comprensión pormenorizada de los avances científicos más relevantes y recientes en la predicción y generación de señales MEA que han impulsado la realización de este proyecto.

4.1. Introducción

Debido a los grandes avances en el ámbito de la IA y su aplicación en las tareas diarias, se está intentando extrapolar a ámbitos más relevantes y cruciales dónde las predicciones juegan un papel muy importante, como sucede con la medicina. El cerebro humano siempre ha sido de gran atractivo para el estudio y elaboración de artículos científicos, y se ha buscado simular completamente su actividad para realizar diagnósticos completos y precisos de enfermedades tan presentes como la epilepsia. Se han buscado muchas aproximaciones, entre ellas interpretar el cerebro como un sistema dinámico.

4.2. Sistemas dinámicos

En el ámbito que envuelve la neurociencia computacional, una de las aproximaciones que más ha llamado la atención a lo largo de las últimas décadas es interpretar el cerebro como si de un sistema dinámico se tratase [29].

Se define Sistema Dinámico (SD) como un sistema cuyos estados internos evolucionan a lo largo del tiempo a través de una regla que determina la dinámica del sistema, siendo capaz de analizarse y modelarse mediante modelos matemáticos. A lo largo de los años, han surgido diferentes clasificaciones de los sistemas en función a sus fundamentos teóricos matemáticos [30].

La evolución que es capaz de experimentar un sistema dinámico a lo largo del tiempo puede mostrarse de forma discreta o continua [30]. Una evolución discreta del tiempo hace que los SD vengan definidos a través de ecuaciones en diferencias, que muestran relaciones recursivas entre variables. Por otra parte, los SD cuya evolución en el tiempo es continua vienen definidos principalmente por ecuaciones diferenciales ordinarias que capturan la relación entre las variables y sus derivadas. En caso de que esta relación no dependa explícitamente del tiempo, es decir, que no aparezcan en las ecuaciones del sistema, se considera que se esta hablando de SD autónomos.

Otra clasificación que tuvo una gran repercusión a partir de los siglos XVIII y XIX debido a diversas investigaciones en medios naturales es clasificar dichos sistemas en función de la variabilidad de sus condiciones [31]. De esta forma se diferencia entre SD deterministas y aleatorios o estocásticos. Los primeros engloban a los sistemas cuyo

valor de salida nunca varía dada una misma entrada, como puede suceder con un péndulo simple ya que partiendo desde el mismo punto, siempre vamos a obtener el mismo punto final después de un tiempo t determinado. En cuanto a los aleatorios o estocásticos, funcionan mediante la teoría de la probabilidad y del caos, pues dos experimentos que se efectúan bajo condiciones iniciales similares pueden tener resultados completamente distintos, como puede ser el ejemplo del péndulo doble.

Por último también se puede hacer una distinción en SD lineales si las variables que caracterizan su función son capaces de cumplir el principio de superposición (dadas dos entradas a la misma vez en el sistema, la salida es la suma de la salida en caso de que estas entradas se hubieran introducido al sistema independientemente). En caso de que no se cumpla este principio, hablaríamos de SD no lineales, que tienen bastante relación con la teoría del caos y los SD estocásticos [32].

Interpretar sistemas provenientes de la naturaleza como si fueran SD, de acuerdo a la teoría de los sistemas dinámicos [33], empezó a recibir cierta acogida en el área de la investigación neurocomputacional hace varias décadas. Varias investigaciones consiguieron deducir que la mayoría de los procesos físicos y biológicos que encontramos en la naturaleza se pueden formular de como SD continuos o discretos. Además, estos SD se pueden interpretar con funciones matemáticas, que a su vez pueden ser tratadas computacionalmente, haciendo posible la exploración de las dinámicas de un sistema a través de los ordenadores.

Aun así, cuando se intentaba trasladar esta teoría al sistema nervioso desde la vista de la neurociencia, que asienta sus bases teóricas en describir los cálculos del sistema nervioso como un SD no lineal, hace esta tarea algo más complicada [34]. Al tratarse de SD de este tipo, indica que sus trayectorias pueden tomar comportamientos caóticos y dependientes a sus condiciones iniciales, haciendo la representación mediante modelos matemáticos bastante ineficaz y compleja para extraer conocimientos útiles.

En los últimos años, gracias al gran avance de la IA, se ha permitido emplear distintas herramientas de ML y Deep Learning (DL) para automatizar estas tareas y conseguir representar las características propias de los SD no lineales y, consecuentemente, ser capaz de reproducir registros de señales MEA y otro tipo de series temporales.

4.3. Predicción, imputación y generación de series temporales

De acuerdo a [35], en las últimas investigaciones con series temporales (para un resumen de estas investigaciones, centradas en realizar diferentes tareas con series temporales mediante modelos de difusión, ver la Tabla 4.1), las tareas principales que se buscan modelar mediante IA para obtener conocimiento de este tipo de datos son la predicción, la imputación y la generación.

- Con la predicción de series temporales se consigue predecir el futuro próximo mediante un fragmento que capture información relevante del historial de la serie temporal.
- Mediante la imputación, se consigue rellenar valores del conjunto de datos que no estén completos debido a fallos de sensores o descuidos humanos. Si estos datos faltasen, el análisis y la predicción de estas series temporales sufriría un gran impacto negativo, puesto que la falta de observaciones hace que las inferencias futuras extraídas del modelo no sean capaces de generalizar correctamente.
- La generación, no obstante, consigue crear datos sintéticos que sean similares a los proporcionados en instantes de tiempo anteriores. Esta tarea está tomando mucha importancia en el sector de la investigación para generar datos de gran calidad a partir de otros para mejorar el funcionamiento del modelo y de esta forma conseguir inferencias mucho más precisas.

En este TFG las tareas se han orientado entorno a la predicción y generación de señales MEA, puesto que con la predicción, conseguimos averiguar si el modelo que se está probando es capaz de reproducir las dinámicas y fluctuaciones de la señal; y mediante la generación, logramos ver cómo de capaz es el modelo de continuar representando la señal sin impulsos externos, manteniendo las características propias de la señal. La imputación no

ha sido objetivo de esta investigación, puesto que se ha procurado que los datos que han sido provistos no contasen con valores incompletos durante la etapa de preprocesamiento.

Tareas	Tipo de datos	Referencia bibliográfica
Predicción de series temporales	Series temporales multivariantes	[36]
		[37]
		[38]
		[39]
		[40]
		[41]
	Grafos espacio-temporales	[42]
		[43]
Imputación de series temporales	Series temporales multivariantes	[44]
		[36]
		[45]
		[46]
		[47]
		[48]
	Grafos espacio-temporales	[49]
Generación de series temporales	Series temporales multivariantes	[50]
		[51]
		[52]
		[53]

Tabla 4.1: Aplicaciones de las series temporales en diferentes investigaciones [35].

4.4. Importancia de las RNN en señales MEA

Las RNN son muy variadas, con una gran cantidad de modelos computacionales. Estos modelos se diseñaron con la intención de imitar la funcionalidad del cerebro. Por ello, dentro del modelo, encontramos una serie de unidades de procesamiento conocidas como neuronas, que se entrelazan entre sí mediante unos enlaces que simulan las conexiones sinápticas. De esta forma, se intenta conseguir que las tareas se realicen como se haría en un cerebro: propagando la información de una neurona a otra mediante impulsos eléctricos. Pero lo más revelador fue cuando se emplearon ciclos, es decir, que este tipo de redes neuronales usase la recurrencia en sus conexiones, lo que supone que:

- Las RNN tienen la capacidad de mantener un patrón de activación sin necesidad de estímulos externos gracias a las conexiones recurrentes que contiene. Esta característica convierte a las RNN en sistemas dinámicos, lo que significa que su comportamiento puede cambiar con el tiempo de manera no lineal y compleja.
- Si es impulsado por una señal de entrada, un RNN conserva en su estado interno una transformación no lineal del historial de entrada; en otras palabras, tiene una memoria dinámica y es capaz de procesar información de series temporales.

Gracias a lo descrito en el párrafo anterior [54], los avances en el campo de la medicina y de la tecnología impulsan arquitecturas derivadas de RNN como campo de investigación para realizar inferencias sobre diferentes tipos de series temporales (como en algunas de las referencias de la Tabla 4.1).

Es oportuno mencionar que el uso de una RNN simple para las tareas de predicción y generación de series temporales no es recomendable debido a las limitaciones que tienen (ver el apartado 6.2.2). En el caso de las señales MEA, son secuencias temporales que pueden ser muy extensas y cuentan con relaciones a largo plazo que son muy importantes para su análisis [55], es por ello que como he mencionado anteriormente, han surgido muchos modelos derivados de las RNN que aseguran ser capaces de modelar series temporales adecuadamente para este tipo de tareas.

En los artículos científicos [56] y [57], se prueban diferentes arquitecturas de DL con diferentes conjuntos de series temporales para diferentes tareas, como por ejemplo análisis de sentimientos, reconocimiento automático de actividad humana, clasificación de imágenes y predicción del clima. El uso de la LSTM para estas tareas muestra unos resultados satisfactorios en comparación con otros modelos; por ello, se ha seleccionado este modelo para compararlo con la técnica de RC para predecir y generar las señales de las que se dispone en esta investigación.

4.5. Reservoir Computing en la predicción y generación de señales MEA

Debido a los problemas derivados del entrenamiento de una RNN convencional, se han buscado otras aproximaciones como se ha mencionado en el apartado anterior. Una de las más aceptadas por la comunidad científica para la predicción y generación de señales MEA es RC. Una de sus ventajas, que causó cierta revolución cuando se descubrió, es su capacidad de adaptar la recurrencia de la arquitectura de las RNN con tan solo entrenar una capa lineal [58].

Es por ello que a partir de este descubrimiento se solucionaban las limitaciones mencionadas en el apartado 6.2.2 de mejor manera que usando una LSTM, ya que aunque haya sido ampliamente utilizada, hay casos en los cuales no supera otras implementaciones para ciertas tareas con series temporales como sucede en este artículo [59].

En el artículo [60] se muestra una comparativa entre arquitecturas englobadas en RC, obteniendo que la ESN tiene un gran poder de generalización para series temporales.

En cuanto se trata del uso de modelos de RC para tareas como el reconocimiento de emociones mediante señales MEA, se obtienen buenos resultados en comparación a otras arquitecturas de RNN, como se trata en el articulo [55].

En la predicción y generación de series temporales caóticas provenientes de modelos conocidos como Mackey-Glass o Lorenz-63, se han conseguido muy buenos resultados en los modelos LSTM, ESN y NGRC en el experimento mencionado en el artículo [61]. Debido a la similitud entre las tareas desempeñadas en este artículo y las tareas que se quieren desempeñar en este TFG, se ha optado por incluir en este proyecto la variante Nonlinear Vector AutoRegression (NVAR), también conocida como NGRC por los creadores del artículo [62]. En esta memoria se va a denominar de la segunda forma para entender mejor el significado de este modelo, ya que con NGRC se busca mejorar ciertos aspectos de la técnica de RC, asegurando una nueva generación para el cálculo con reservorios.

Por último, también se ha estado investigando cómo mejorar el rendimiento sin comprometer la eficacia ni la tasa de acierto. Por ello, se están intentando implementar modelos cuantizados, como por ejemplo la LSM diseñada en este artículo [63], que operan con números de 2,3,4 u 8 bits para representar los parámetros de un modelo en vez de 32 bits, obteniendo un tamaño de modelo más pequeño y mayor eficacia de cálculo. Por consiguiente, este modelo se va a estudiar en este TFG para comprobar si realmente los tiempos mejoran en comparación al resto de modelos y si es capaz de recordar y representar las características de las señales MEA a lo largo de un largo periodo de tiempo con representaciones numéricas tan pequeñas.

Capítulo 5

Conjunto de datos y preprocesamiento

Para llevar a cabo este proyecto, se ha obtenido una serie de datos procedente de la empresa tutora de este TFG, AIR Institute. A continuación se van a explicar los aspectos más relevantes de este conjunto de datos, justificando su procedencia y los tratamientos realizados para su posterior uso en la ejecución de los modelos.

5.1. Procedencia de los datos

Los datos utilizados en este TFG provienen del dataset generado por el proyecto "Sistemas híbridos mejorados de medicina regenerativa"(HERMES) y las colaboraciones del AIR Institute con el Insituto Italiano de Tecnología y la Universidad de Módena; la versión pública del dataset se puede encontrar en Zenodo [64].

Las señales se han obtenido mediante grabación de potenciales de campo locales (LFP) usando un MEA de la empresa Multi Channel Systems. Para ello, cortes coronales de cerebro de roedor fueron sumergidos en líquido cerebroespinal artificial e incubados a 32^{o} en una cámara de grabación.

El array de la cámara de grabación cuenta con 6x10 electrodos de Titanio-Iridio para la captura de señales en tiempo real. Estos electrodos pueden usarse tanto para grabación como para estimulación eléctrica.

La ventaja de este método es que permite obtener grabaciones en un entorno controlado (la cámara permite regular temperatura, contenido del líquido cerebroespinal artificial, la mezcla de gases atmosféricos...) de diferentes animales con mayor resolución que un electroencefalograma o una resonancia magnética.

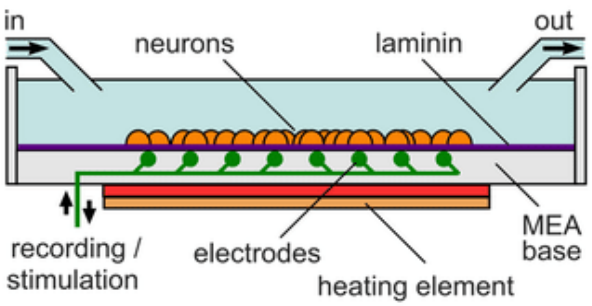

Figura 5.1: Esquema del montaje de MEA [65].

En la Figura 5.2 se puede ver un corte de un cerebro, también procedente de un roedor, posicionado en la placa del MEA.

Las terminaciones de color negro constituyen los electrodos, y aquellos que están rodeados de colores son los electrodos usados en ese experimento. Diferentes colores representan diferentes regiones del cerebro.

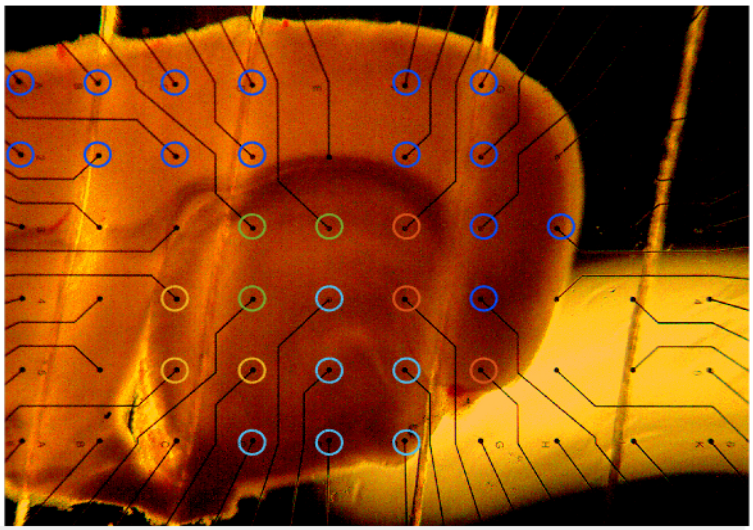

Figura 5.2: Corte de un cerebro de roedor encima de una placa MEA. Imagen cedida por AIR Institute.

5.2. Descripción de los datos empleados

Para obtener los datos de este TFG, se ha empleado un cerebro de un roedor con epilepsia en el lóbulo temporal, al cual se le ha aplicado un corte en el hipocampo.

El hipocampo cuenta con una serie de regiones (Figura 5.3) que se encuentran señalizadas por colores distintos en la Figura 5.4. En esta figura se puede ver cómo se representa el corte del hipocampo con los electrodos seleccionados para obtener la actividad cerebral del roedor y así extraer los registros a usar en este TFG. La correspondencia color-región representada en la Figura 5.4 es:

- Naranja - Región correspondiente a la corteza parahipocampal (CTX)
- Amarillo - Región correspondiente al primer subcampo del *cornu ammonis* (CA1)
- Azul claro - Región correspondiente al tercer subcampo del *cornu ammonis* (CA3)
- Azul oscuro - Región correspondiente al giro dentado (DG)
- Verde - Región correspondiente al subículo (SUB)

En investigaciones recientes como [66] y [67] se ha detectado la gran influencia que tiene la región CA3 en la epilepsia del lóbulo temporal dado que los roedores con este tipo de epilepsia tienen esta región dañada, por lo que se ha decidido obtener las señales MEA únicamente de esta región para ver si los modelos son capaces de reproducirla.

En consecuencia, para este TFG se han usado los electrodos marcados de color azul claro, correspondientes a los números 27, 65, 75 y 77 en el corte mostrado en la Figura 5.4.

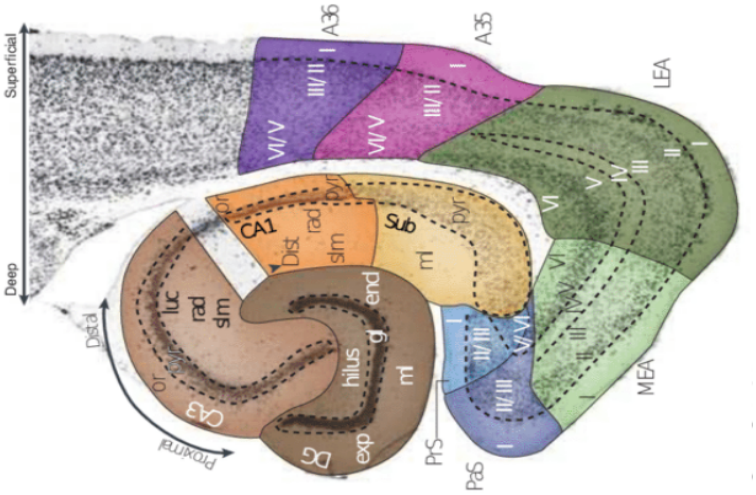

Figura 5.3: Esquema del corte del cerebro en la región del hipocampo de un roedor [68].

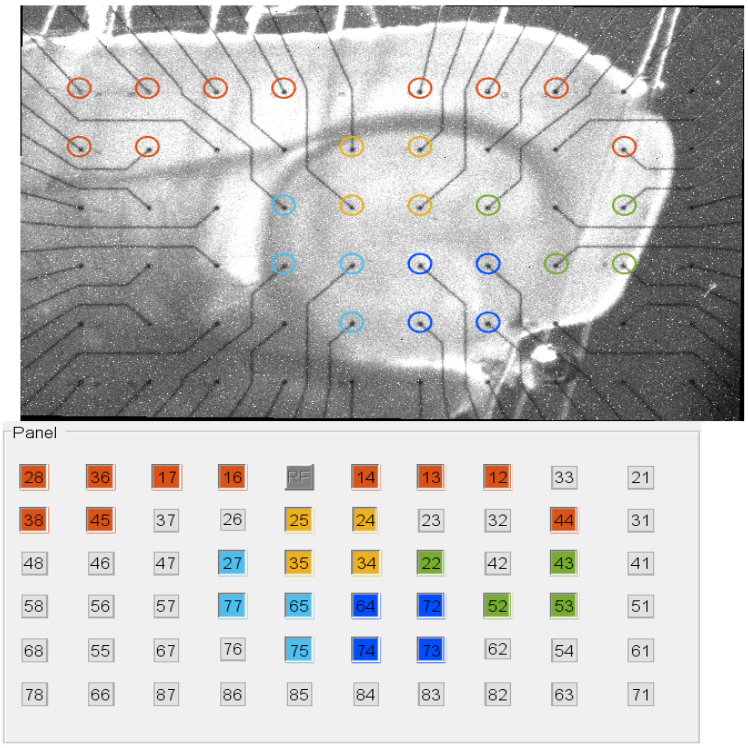

Figura 5.4: Corte de un cerebro de roedor encima de una placa MEA con un panel de los electrodos seleccionados. Imagen cedida por AIR Institute.

5.3. Preprocesamiento y limpieza de los datos

Los ficheros de los electrodos seleccionados estaban en formato .mat y nombrados de la forma

fecha-cortedecerebro-subexperimento-subsubexperimento-electrodo

Estos archivos se han pasado a CSV antes de su utilización en el preprocesamiento. Una vez obtenidos los ficheros, se obtuvieron unas gráficas como representación inicial de los datos en crudo (Figura 5.5).

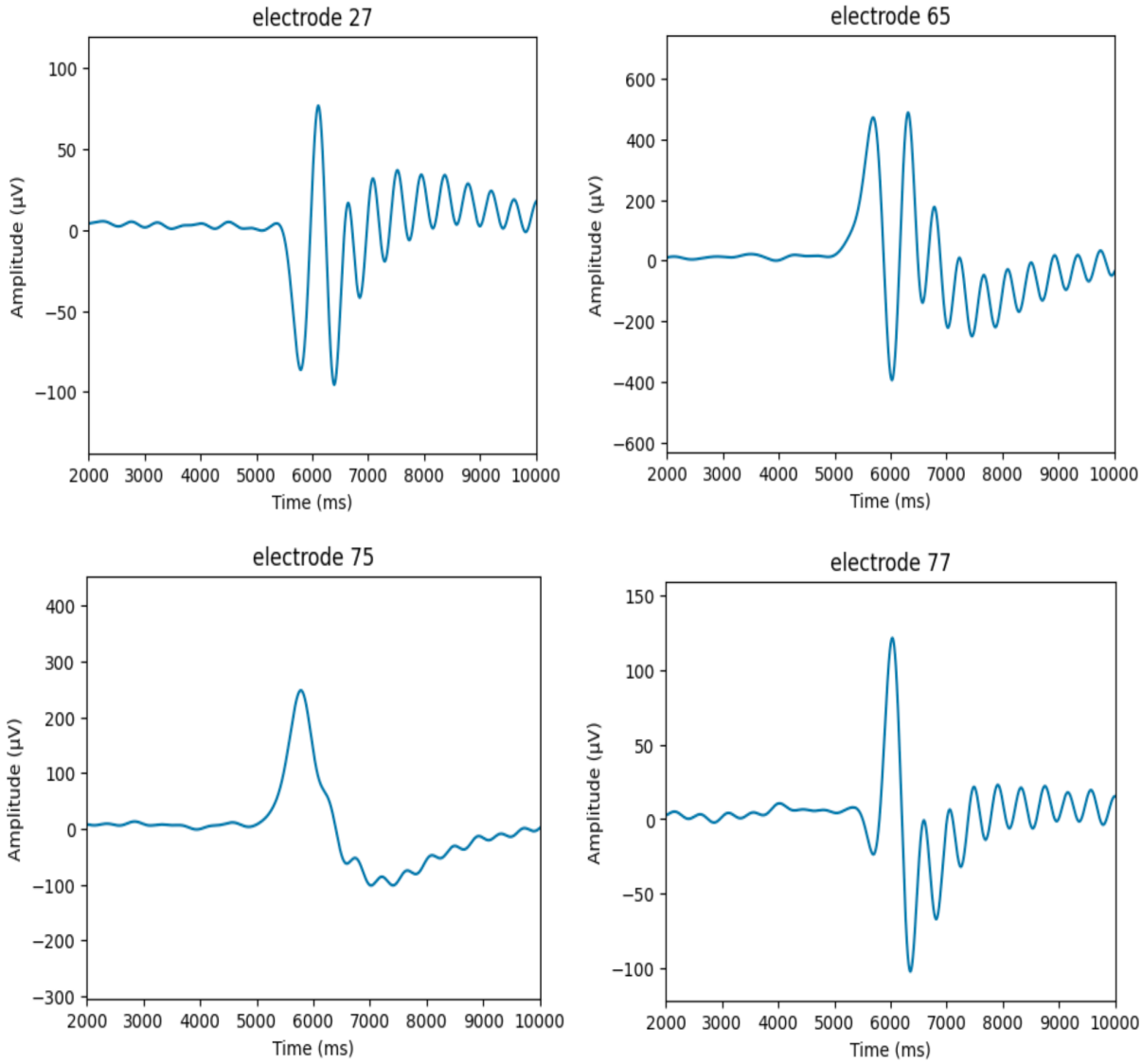

Figura 5.5: Muestra de las señales sin preprocesar en los diferentes electrodos. Elaboración propia.

Una vez mostrado, se ha realizado un filtrado pasabaja de 40Hz. Esto permite que las frecuencias superiores a ese valor sean ignoradas, ya que una frecuencia superior a 40Hz no es objeto de estudio en este TFG. Es importante resaltar que la tasa de muestreo de las señales es de 2000Hz.

Como se puede observar en la Figura 5.6, en el dominio del tiempo de las señales, no varían con respecto a las señales antes de usar el filtro.

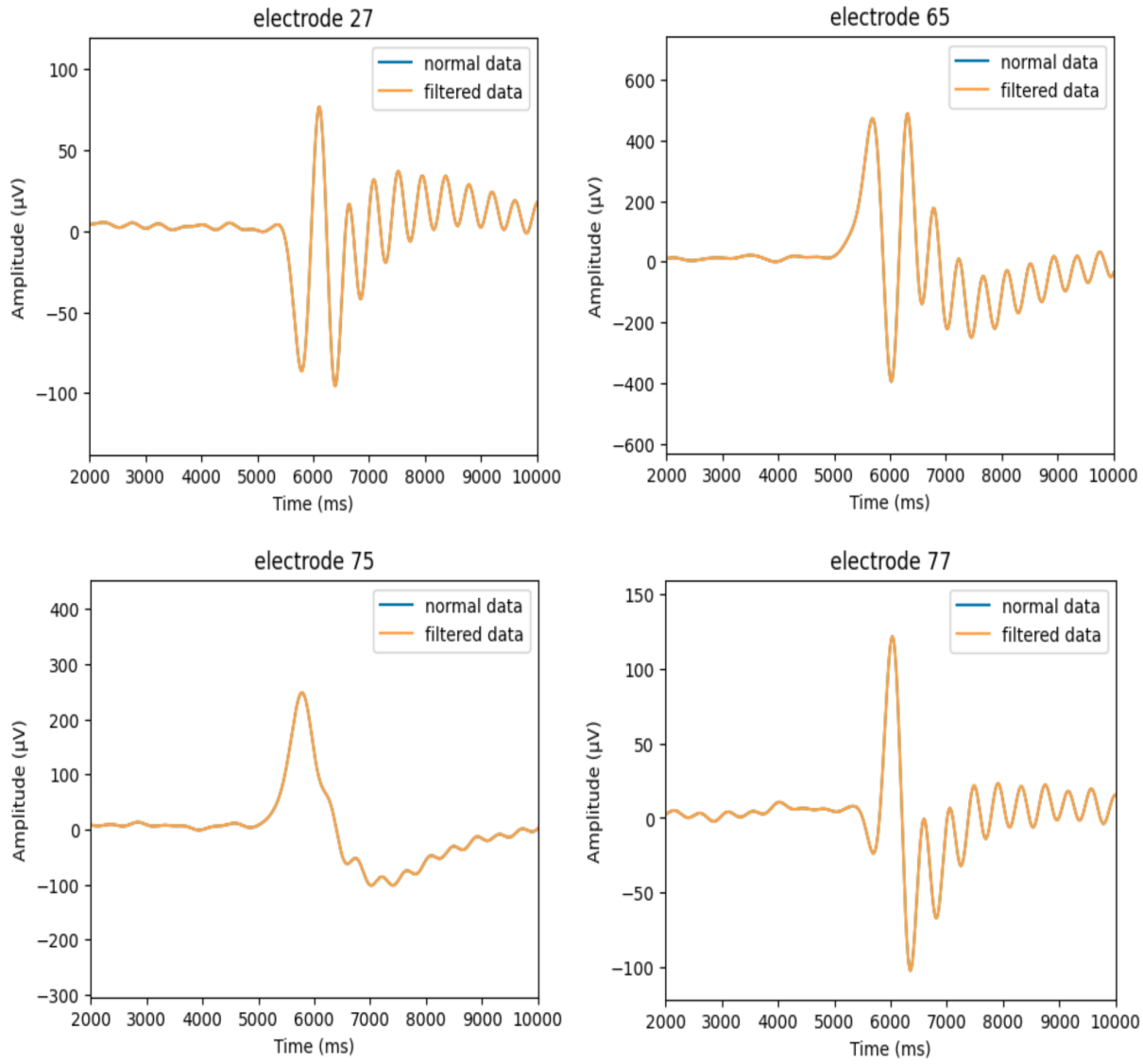

Figura 5.6: Muestra de la señales en el dominio del tiempo antes y después del filtrado en los diferentes electrodos. Elaboración propia.

Esto se debe a que el filtro actúa en el dominio de la frecuencia, atenuando las frecuencias mayores a 40Hz y, de esta forma, modificando la distribución espectral de la señal de origen, como se puede ver en la Figura 5.7.

Teniendo esto en cuenta, las sucesivas representaciones de las señales MEA se harán en el dominio del tiempo y, aunque en este dominio no se note variación alguna como se puede observar en la Figura 5.6 (ya que la línea naranja se superpone en su totalidad a la línea azul), realmente sí que están filtradas estas frecuencias superiores a 40Hz, solamente hay que representarlo en el dominio de la frecuencia para apreciar el filtrado, como se puede ver en la Figura 5.7.

Una vez filtradas las señales, se van a crear tres tipos de conjuntos que se van a usar en función de los prerrequisitos del modelo:

- El primer conjunto de datos no estará ni estandarizado ni normalizado.
- El segundo conjunto de datos estará normalizado en el rango [0,1].
- El tercer conjunto de datos estará estandarizado con media 0 y desviación estándar 1.

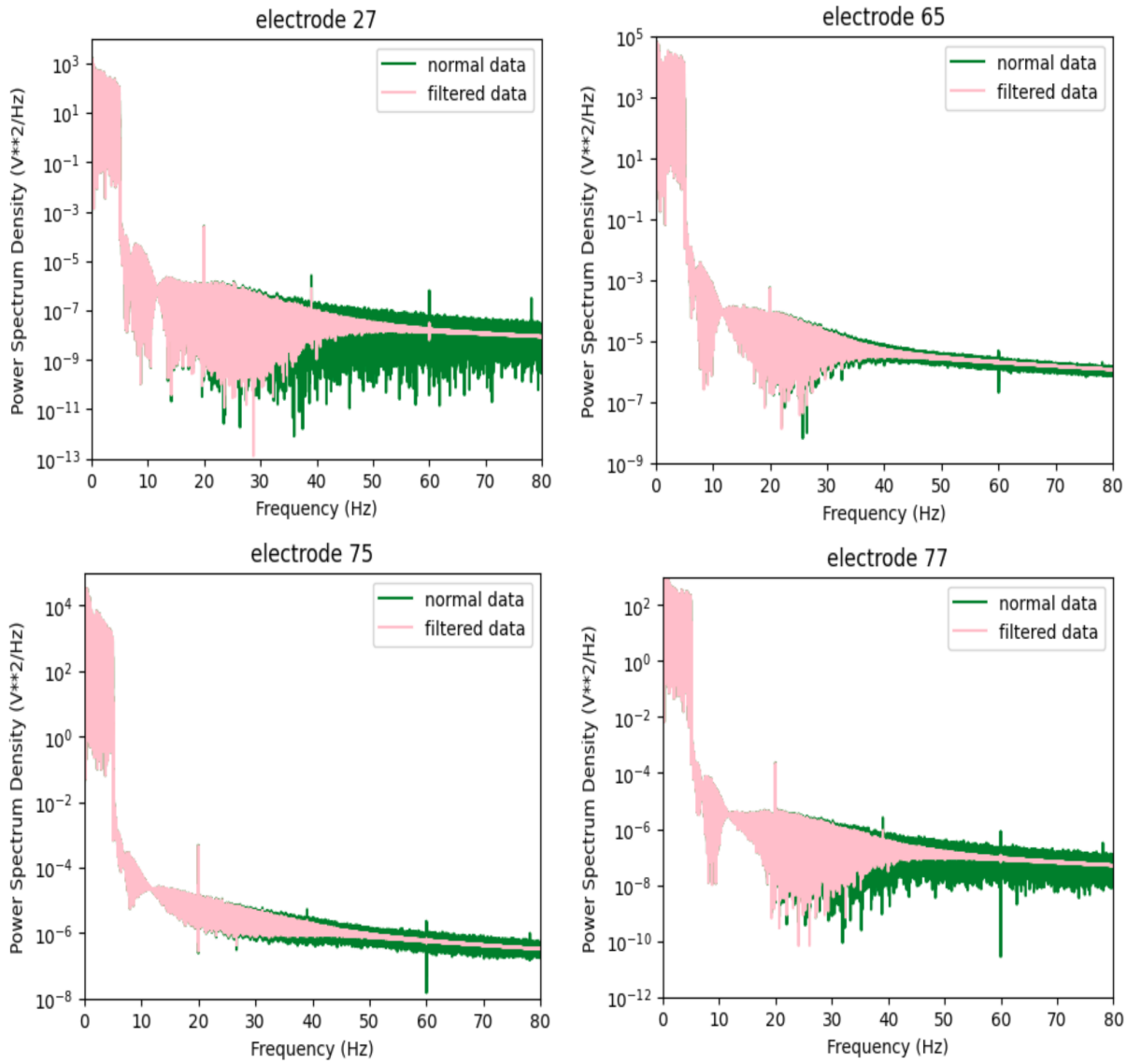

Figura 5.7: Muestra de la señales en el dominio de la frecuencia antes y después del filtrado en los diferentes electrodos. Elaboración propia.

Por último, se reducirá la tasa de muestreo de cada conjunto de datos a 1 de cada 10 puntos para simplificar el análisis y procesamiento de la señal. Al reducir la cantidad de datos en las señales MEA, se puede reducir la carga computacional y facilitar la identificación de patrones y características importantes en la señal.

En la Figura 5.8 se visualiza una comparativa entre la señal producida por el electrodo 27 antes de la reducción de la tasa de muestreo en color azul, acompañada de tres señales en color rojo también de este electrodo, que muestran la señal después de la reducción de la tasa de muestreo en los tres conjuntos de datos.

Ya finalizado todo el preprocesamiento, las señales resultantes se guardan en ficheros .csv y .npy para su posterior uso en la implementación de los modelos.

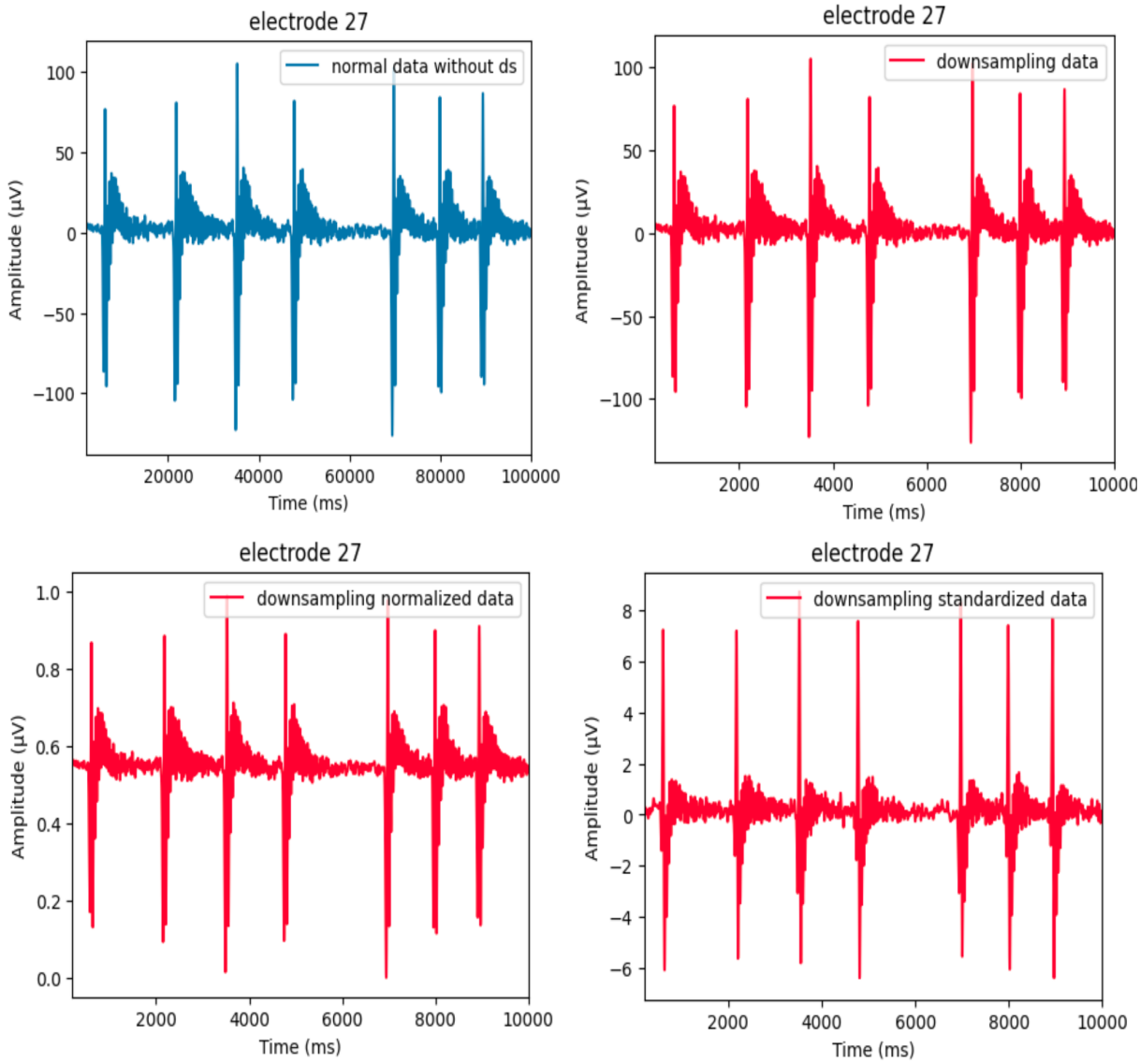

Figura 5.8: Muestra de la señal originada por el electrodo 27 con reducción de la tasa de muestreo en los tres conjuntos de datos. Elaboración propia.

Capítulo 6

Modelos

6.1. Machine Learning (ML) y Deep Learning (DL)

Como preámbulo, dentro de la IA encontramos una subdivisión en función de la naturaleza del aprendizaje [69]. Podemos distinguir dos términos muy reconocidos: ML y DL. Como se puede ver en la Figura 6.1, DL está contenido en ML.

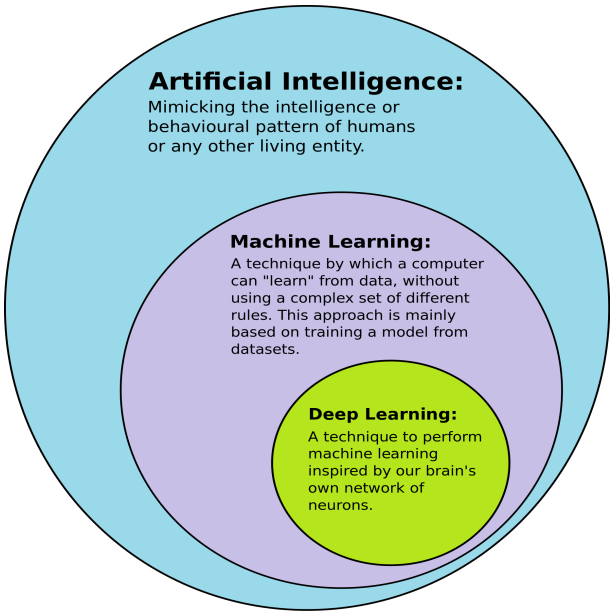

Figura 6.1: Composición IA, ML y DL [70].

Teniendo esto en cuenta, existen diferentes maneras de clasificar los algoritmos de ML. La primera de ellas es dividirlos en función de las tareas desempeñadas (como se puede ver de manera simplificada en la Figura 6.2). En el caso de este TFG, la tarea a desempeñar se clasifica dentro del **aprendizaje autosupervisado**, pues el sistema aprende a predecir y generar parte de las señales MEA a partir de trozos anteriores de dichas señales.

Otra forma de representarlo es enfocarse en las arquitecturas que conforman ML. Por simplificar, la Figura 6.3 muestra una clasificación de las arquitecturas de DL. Las arquitecturas señaladas de color verde representan las utilizados en este proyecto. Como se puede ver, se usa la LSTM de RNN, y después se hace especial énfasis en RC, la técnica de estudio en este TFG, que usa la arquitectura RNN. NGRC, por otro lado, es una técnica que intenta mejorar los modelos de RC, como se verá a continuación.

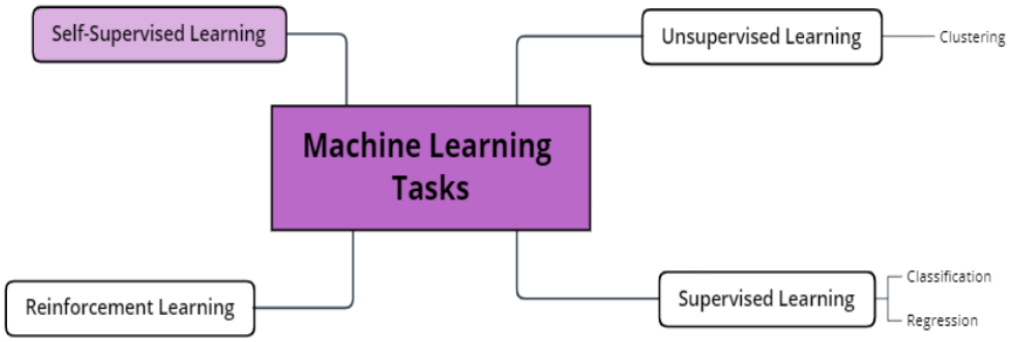

Figura 6.2: Clasificación resumida de las tareas de ML. Elaboración propia, basada en [71].

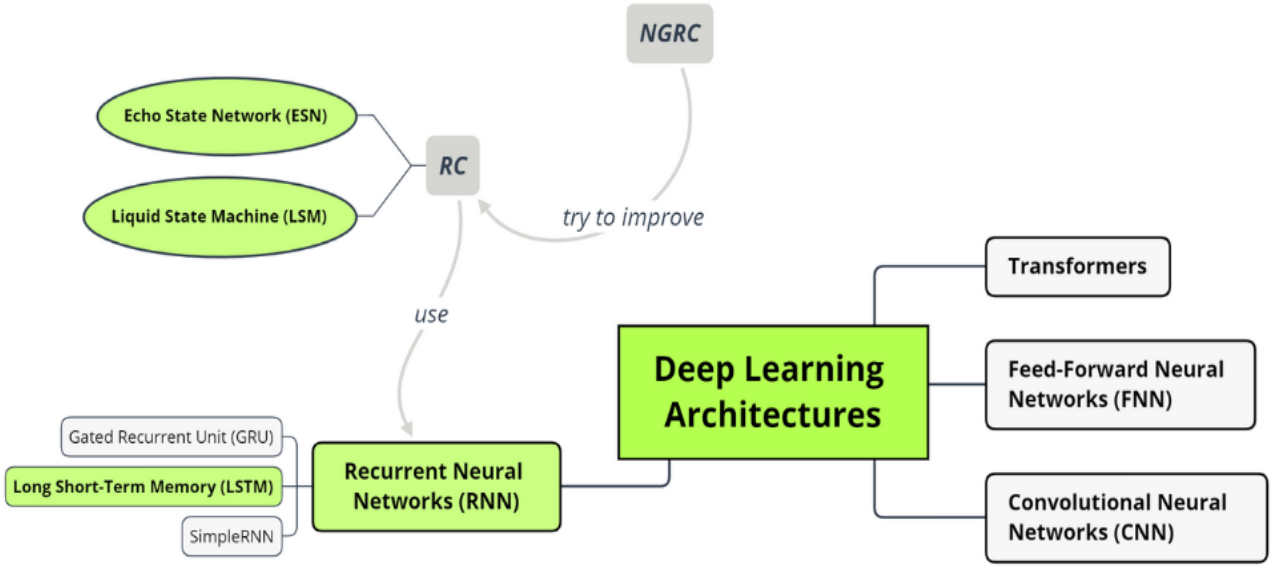

Figura 6.3: Clasificación resumida de las arquitecturas de DL. Elaboración propia, basada en [71].

A continuación, se procederá a explicar las arquitecturas resaltadas. Por ello, primero se hará una breve introducción a las RNN, posteriormente se procederá a explicar la técnica central de este trabajo: RC y sus respectivos modelos. Posteriormente, se explicará la técnica de NGRC y, por último, se hará una introducción a las LSTM antes de comentar su implementación en el dispositivo de Google Coral en el capítulo 7.

6.2. Redes Neuronales Recurrentes (RNN)

Como se ha mencionado con anterioridad, las RNN son un tipo de algoritmos pertenecientes a la familia de DL que ha ganado gran popularidad en la representación de sistemas dinámicos gracias a sus propiedades (ver Capítulo 4).

6.2.1. Base teórica

En cuanto se trata de procesar datos secuenciales, las RNN cuentan con conexiones recursivas que permite el flujo de datos a través de la red, siendo capaz de recordar información relevante y modelar patrones complejos de este tipo de datos.

Estructura

En cuanto a su estructura y funcionamiento [72], la RNN más sencilla está formada por 3 elementos principales: la capa de entrada, la capa oculta (aunque también puede haber varias capas ocultas) y la capa de salida.

Esta arquitectura se puede presentar como una sola neurona, que recibe una entrada, pasa por la neurona o neuronas que conformen la capa oculta, se produce una salida y esta salida obtenida de la capa oculta se reenvía de nuevo a la entrada de dicha capa (diagrama de la parte izquierda de la Figura 6.4).

Si desdoblamos esta representación inicial en el eje del tiempo, obtenemos que la capa oculta realmente recibe dos entradas en el instante de tiempo t: la salida obtenida en el instante *t-1* y la entrada correspondiente en el instante t (diagrama desdoblado de la parte derecha de la figura 6.4).

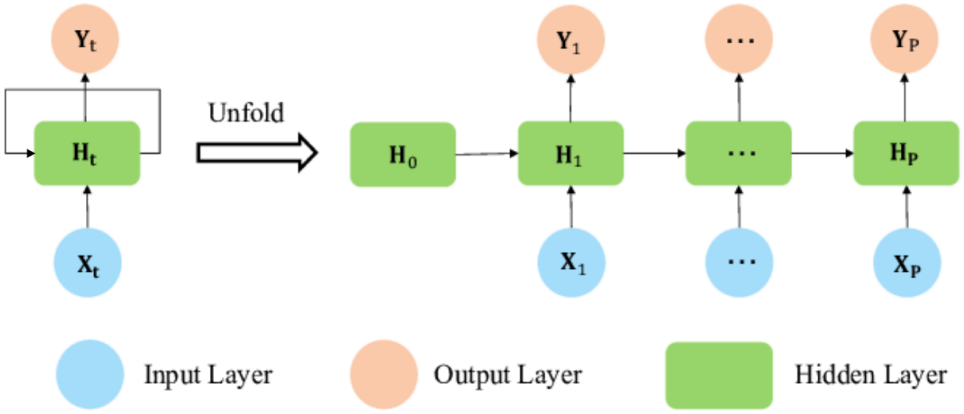

Figura 6.4: Desdoble de una RNN [73].

La capa o capas ocultas están formadas por una o varias neuronas. Esto es lo que constituye en sí la capacidad de recordar información y lo que dota de memoria a la red neuronal. Esta memoria, que no es más que una celda recurrente, se va actualizando en cada instante de tiempo t.

Para representar el funcionamiento de la capa oculta matemáticamente, se denota como capa oculta $H_t \in \mathbb{R}^{n*h}$ en el instante de tiempo t, y como entrada en el instante t a $X_t \in \mathbb{R}^{n*d}$, donde n es el número de ejemplares, d es el número de entradas de cada ejemplar y h es el número de neuronas en la capa oculta H_t. Con ello, obtenemos la siguiente expresión matemática:

$$H_t = \sigma_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h) \tag{6.1}$$

Donde σ es la función de activación (normalmente una sigmoide o una tangente hiperbólica), $W_{xh} \in \mathbb{R}^{d*h}$ es la matriz de pesos de la entrada x a la capa oculta h, $W_{hh} \in \mathbb{R}^{h*h}$ es la matriz de pesos de una capa oculta a otra, H_{t-1} es el estado de la capa oculta H en el instante *t-1* y $b_h \in \mathbb{R}^{1*h}$ es el sesgo del la capa oculta.

Así mismo, podemos obtener la ecuación representativa de la salida de una RNN:

$$y_t = \sigma_o(H_t W_{ho} + b_o) \tag{6.2}$$

Donde W_o es la matriz de pesos desde la capa oculta hasta la de salida y b_o es el sesgo del la capa de salida.

Entrenamiento

El flujo de datos en el entrenamiento de la RNN se basa en lo que se conoce como Forward Propagation Through Time (FPTT) y Back Propagation Through Time (BPTT).

Mediante la FPTT, la red procesa los datos de entrada en cada instante de tiempo y las neuronas de la capa oculta realizan una serie de operaciones matemáticas para calcular de esta forma la salida en el instante correspondiente (ver las Ecuaciones 6.1 y 6.2) [74].

A continuación, se calcula un error de la salida obtenida con los datos reales, y pasando este error a lo largo de esta red mediante el algoritmo de BPTT, se consigue ajustar los pesos de la red para conseguir el mínimo error posible referente a la salida de la red. Esto hace que se produzca una realimentación en ambos sentidos, y no solamente de manera unidireccional como pasaba con las Feedforward Neural Network (FNN).

Para conseguir minimizar este error y así mejorar el funcionamiento de la red [75], el algoritmo BPTT calcula lo que se conoce como el algoritmo del Descenso del Gradiente, en el cual se ajustan los parámetros para encontrar aquellos que proporcionen el coste más bajo posible.

Al ser una RNN, es necesario calcular los gradientes en cada uno de los instantes de tiempo, resultando en los gradientes desde el instante inicial 1 hasta el último instante de tiempo T para W_{xh}, W_{hh} y para W_{xh}, llegando a las siguientes ecuaciones:

$$\frac{\partial E}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial \sigma_o} \cdot W_{ho} \sum_{k=1}^{T} \frac{\partial H_t}{\partial H_k} \cdot \frac{\partial H_k}{\partial W_{hh}} \tag{6.3}$$

$$\frac{\partial E}{\partial W_{xh}} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial y_t} \cdot \frac{\partial y_t}{\partial \sigma_o} \cdot W_{ho} \sum_{k=1}^{T} \frac{\partial H_t}{\partial H_k} \cdot \frac{\partial H_k}{\partial W_{xh}} \tag{6.4}$$

Donde E es la suma de cada función de pérdida E_t. Cabe destacar que cuanto más grande sea T, mayor será el intervalo temporal que pueda recordar la red.

Una vez actualizada la red, estos pasos se repiten cíclicamente en lo que se conoce como épocas para conseguir en cada iteración una mayor tasa de acierto, y con ello mejores predicciones.

Tipos

Podemos distinguir entre diferentes tipos de RNN en función de las entradas y salidas de las que disponga [76]. Siguiendo la misma colorimetría que en la Figura 6.4, se han plasmado 5 tipos diferentes de RNN en la Figura 6.5.

- **One to one**: Es la representación más simple de procesar un vector de entrada para generar un vector de salida sin necesidad de una RNN, como puede darse en la clasificación de imágenes.
- **One to many**: En esta representación, la entrada es un único vector que se transforma en una secuencia de salida. Esto puede darse al extraer una descripción de una imagen, donde la entrada es una única imágen y la salida es una secuencia de palabras.
- **Many to one**: Es el caso inverso de la anterior, dónde la entrada es una secuencia y la salida es un único vector. Un ejemplo de este tipo es el reconocido analásis de sentimientos, donde dada una oración se clasifica en si el significado de esta es positivo o negativo.

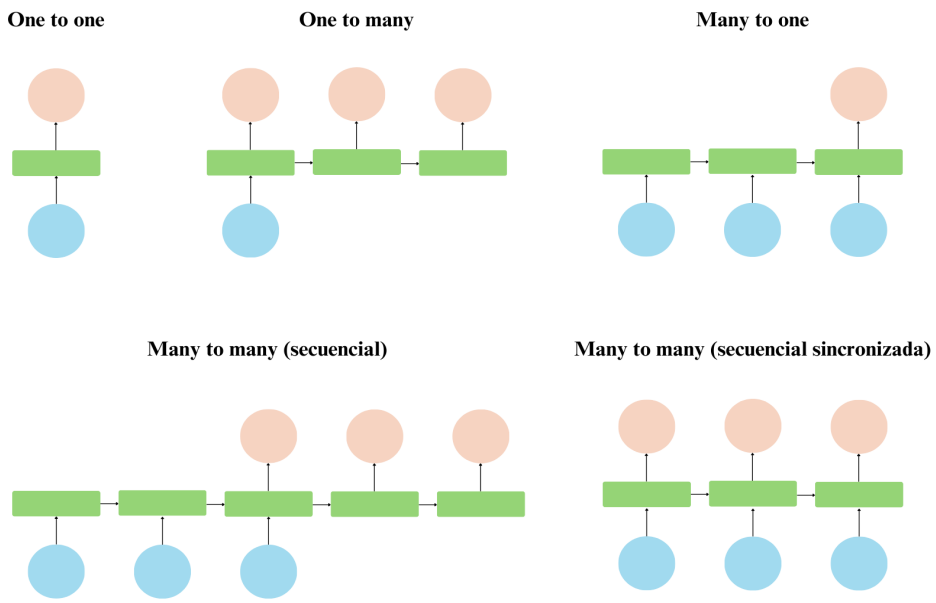

Figura 6.5: Tipos de RNN. Elaboración propia, basada en [76].

- **Many to many**: Dentro de este tipo, encontramos dos subtipos:
  - **Secuencial**: las entradas y las salidas son secuencias, como en la traducción de texto de un idioma a otro.
  - **Secuencial sincronizada**: las entradas y las salidas son secuencias, pero en este caso la salida en el instante t es dependiente de la entrada en ese mismo instante. Esto sucede en la clasificación de vídeo, cuando intentamos etiquetar cada fotograma de un vídeo.

6.2.2. Limitaciones

Las RNN son un tipo de redes neuronales muy populares, pero eso no significa que no tengan una serie de inconvenientes que han sido objetivo de mejora a lo largo de las últimas décadas.

Entre estos inconvenientes se encuentran [74]:

- Problemas con el algoritmo BPTT: durante la propagación hacia atrás para actualizar los pesos en series temporales, existen dos problemas con el término $\frac{\partial H_t}{\partial H_k}$ de las ecuaciones 6.3 y 6.4:
  - Explosión del Gradiente: si este término se hace mayor que 1 ($\frac{\partial H_t}{\partial H_k} > 1$), los gradientes de los pesos en las capas anteriores se vuelven excesivamente grandes durante la propagación hacia atrás. Esto hace que las actualizaciones de los pesos en cada iteración se vuelvan a su vez demasiado grandes, tendiendo de esta forma al infinito y provocando valores numéricos incapaces de ser tratados por un ordenador, dificultando la tarea de capturar dependencias temporales y con ello el aprendizaje de la red.

    Algunas de las soluciones propuestas son:
    - Realizar un truncamiento en el BPTT, dejando de realizar propagaciones hacia atrás a partir de cierto punto. Esto se conoce como BPTT truncado [77].
    - Penalizar en caso de que no se reduzca el gradiente, o en su defecto reducir el gradiente con otros procedimientos.
    - Limitar el número de capas ocultas.
    - Establecer un límite superior para el valor que tome el gradiente. Esto se conoce como Gradient Clipping [78].

- Desvanecimiento del Gradiente: Sucede de forma contraria a la Explosión del Gradiente, pues el término se vuelve menor que 1 ($\frac{\partial H_t}{\partial H_k} < 1$). Los gradientes se vuelven más pequeños a medida que se propagan hacia atrás, provocando como resultado que las actualizaciones de los pesos en las capas anteriores se vuelvan insignificantes en comparación con las capas posteriores, lo que dificulta el aprendizaje efectivo.
  Algunas de las soluciones propuestas son:
  - Inicializar los pesos de forma recomendada.
  - Usar otro tipo de arquitecturas que solucionen este problema, como puede ser la aproximación de las técnicas de RC y LSTM.

- Memoria limitada: las RNN tienen una capacidad limitada, por lo que a medida que se avanza en una secuencia temporal lo suficientemente larga se va perdiendo información de los instantes de tiempo más tempranos.

6.3. Reservoir Computing (RC)

Para intentar solucionar el problema que surgía a la hora de entrenar las RNN, se introdujo esta técnica. Actualmente ha adquirido mucha relevancia en otros campos aparte de la neurociencia y la IA debido a una de sus grandes ventajas: su simplicidad conceptual [79].

6.3.1. Base teórica

El principal objetivo de RC es solucionar la inestabilidad propia del entrenamiento de una RNN y prescindir del método del Descenso del Gradiente. Esto se hace restringiendo el entrenamiento solamente a la última capa que conforma la arquitectura de un modelo de RC.

Estructura

Como se ha mencionado, la simplicidad es una de las características más destacables de RC, y es que su arquitectura esta formada por únicamente tres componentes principales: la capa de entrada, el reservorio y la capa de salida (denominada comúnmente *readout*).

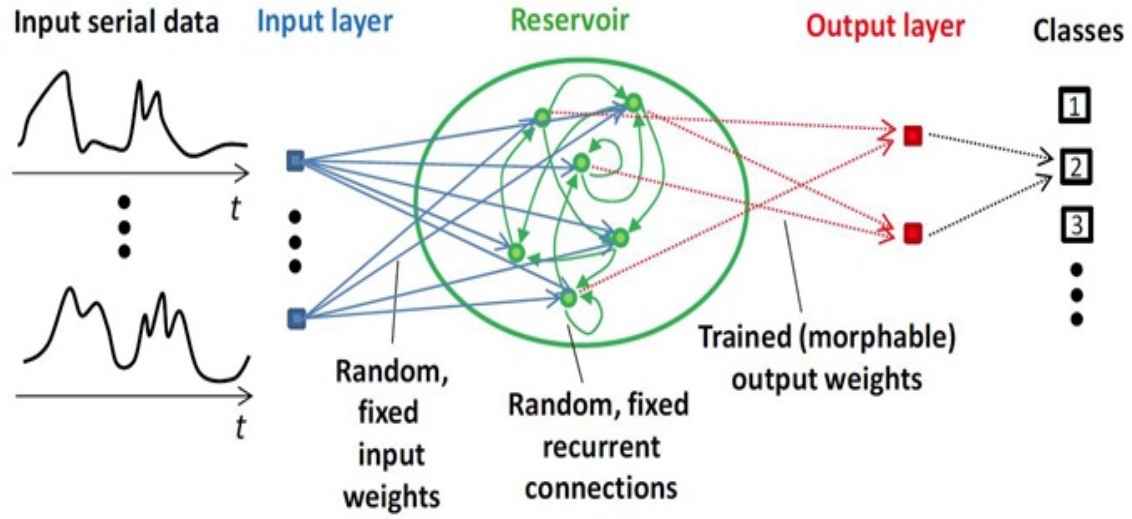

Figura 6.6: Arquitectura general de la técnica RC [80].

A raíz de la Figura 6.6, se puede entender el funcionamiento y la arquitectura básica de cualquier modelo de RC.

La capa de entrada recibe una serie de datos temporales, en este caso señales MEA. Esta capa de entrada introduce estas señales en una RNN inicializada de forma aleatoria y que cuenta con conexiones recurrentes fijas. Esto es lo que constituye la parte del reservorio (o *reservoir*). Por último, las neuronas que constituyen la capa de salida (o *readout*) obtienen la información necesaria para reproducir la señal de salida deseada [79].

En el caso de la imagen, lo que hace la capa de salida es clasificarlas en diferentes clases numeradas, pero esta arquitectura también tiene la capacidad de realizar predicciones y generaciones de series temporales, que es el objetivo de este TFG.

Entrenamiento

La principal diferencia de esta arquitectura con la de una RNN es el entrenamiento, puesto que como se ve en la Figura 6.6, los pesos que se encuentran entre la capa de entrada y dentro del reservorio no se cambian una vez inicializados, dejando solamente los pesos que se encuentran entre el reservorio y la salida para ser modificados y optimizados durante la etapa del entrenamiento [79].

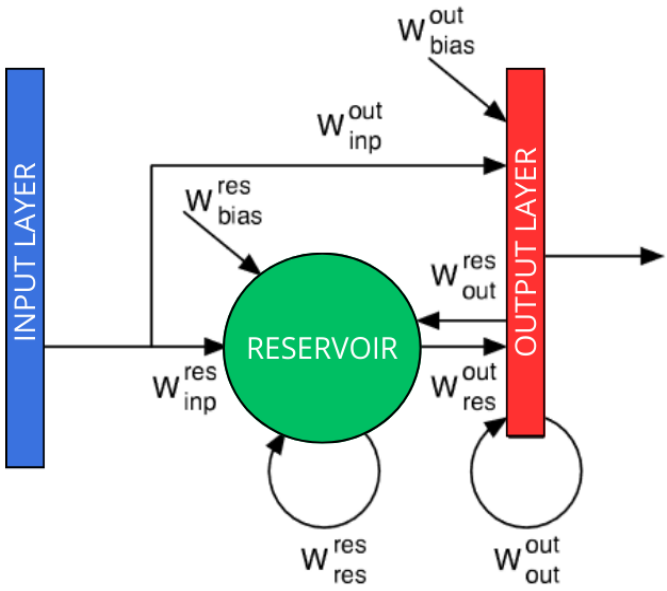

Figura 6.7: Arquitectura general de RC con las matrices de pesos representadas [79].

Siguiendo la Figura 6.7, podemos llegar a las siguientes Ecuaciones:

$$x(t+1) = f(W^{res}_{res}x(t) + W^{res}_{inp}u(t) + W^{res}_{out}y(t) + W^{res}_{bias}) \tag{6.5}$$

$$y(t+1) = W^{out}_{res}x(t+1) + W^{out}_{inp}u(t) + W^{out}_{out}y(t) + W^{out}_{bias} \tag{6.6}$$

Donde la Ecuación 6.5 representa la ecuación de los estados internos y la Ecuación 6.6 representa la ecuación de la salida, ambos en el instante *t+1*. Como se puede ver, corresponden al funcionamiento del entrenamiento mencionado en el apartado anterior, donde solo se entrenan las conexiones W_{out} y las demás son fijas (W_{res}).

Tipos

Existen diversos modelos que pueden encajar dentro de la arquitectura RC, pero los más reconocidos son los que se han estudiado en este TFG, que son la ESN propuesta por Herbert Jaeger [81] y LSM presentada por Wolfgang Maass [82].

Ambos modelos tienen en común que permiten un cierto equilibrio para que se procese la información de la dinámica del sistema de manera efectiva y se generen resultados útiles. Esto lo consiguen mediante el reservorio, que permite reconstruir el sistema dinámico de alta dimensión. Sin embargo, mientras que las LSM utilizan ecuaciones diferenciales que describen neuronas, las ESN emplean nodos simples pero abstractos en forma de ecuaciones diferenciales no lineales para lograr los mismos resultados [79].

Para una mejor comprensión de estos modelos, se desarrollarán de forma más extensa estos tipos de RC mencionados, comentando sus características principales y su implementación para aceptar las señales MEA de las que se disponen. Por otra parte, se comentarán algunas de las limitaciones teóricas y prácticas que se han encontrado a lo largo de este proyecto.

6.3.2. Echo State Network (ESN)

Base teórica

Este modelo contiene una estructura similar a la mencionada en el apartado 6.3.1, ya que es considerado el modelo más sencillo que se puede encontrar en RC . Por este motivo, un modelo de una ESN genérica discreta en el tiempo está formada por H neuronas que forman la capa de entrada del modelo para recibir las series temporales de las señales MEA, N neuronas en el interior del reservorio que procesan la información y L neuronas en la capa de salida del modelo que computan las señales resultantes.

Como en el modelo general de RC, las neuronas de la ESN se conectan entre sí mediante pesos que se almacenan en las matrices $W_{in} \in R^{N \times H}$ para los pesos de las neuronas de entrada, $W \in R^{N \times N}$ para las conexiones internas, $W_{out} \in R^{L \times (H+N+L)}$ para las de salida y $W_{back} \in R^{N \times L}$ para las conexiones de retroalimentación donde la información de la salida se envía de vuelta a las neuronas internas [81].

A lo largo de la ejecución del modelo, el modelo sufre una serie de activaciones que representan el nivel de actividad de cada neurona por cada punto de la serie temporal. Por ello, se denotará como *u(n)*, *x(n)* y *y(n)* a las activaciones en el punto n en la capa de entrada, en el interior del reservorio y en la capa de salida respectivamente.

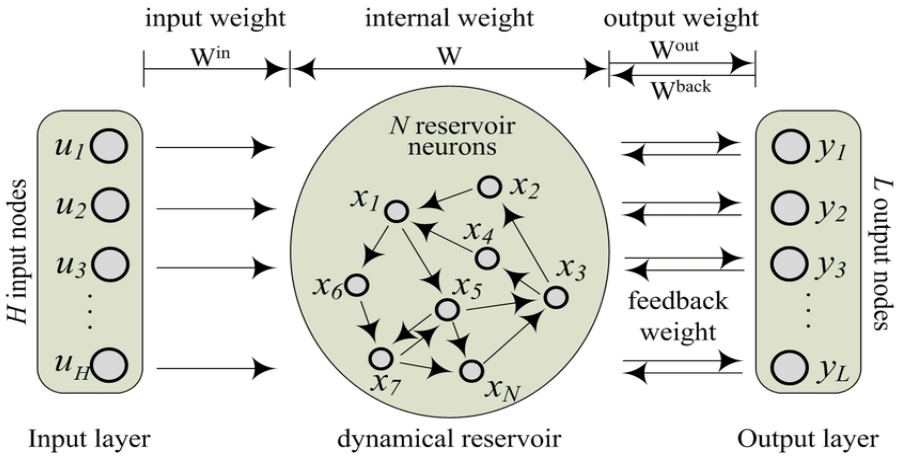

Figura 6.8: Estructura general de la ESN [83].

La activación de las neuronas internas del reservorio se actualiza de la siguiente forma:

$$x(n+1) = f(W_{in}u(n+1) + Wx(n) + W_{back}y(n)) \tag{6.7}$$

donde $f = (f_1, ..., f_N)$ son las funciones de salida de las neuronas internas del reservorio. En cuanto a la salida:

$$y(n+1) = f_{out}(W_{out}(u(n+1), x(n+1), y(n))) \tag{6.8}$$

donde $f_{out} = (f_{out1}, ..., f_{outL})$ son las funciones de salida de las neuronas de la capa de salida. Los pesos de salida W_{out} se multiplican a los vectores de activación previos de entrada, internos y de salida.

Un aspecto fundamental para el correcto funcionamiento de las ESN es la Echo State Property (ESP), la cual garantiza que la influencia de las condiciones iniciales del reservorio se desvanece con el tiempo, dejando solo la respuesta a la entrada actual. De esta forma, se consigue que el reservorio no se estanque en un estado especifico y se permite que la red se adapte a nuevas entradas.

Teniendo en cuenta todas estas posibles optimizaciones y eligiendo el mejor algoritmo de aprendizaje para nuestra tarea a realizar (en el caso de este trabajo, la regresión Ridge como se ha explicado), se consigue que la ESN sea capaz de predecir y generar señales MEA, como se mostrará a continuación.

Implementación

Para implementar este modelo, se ha utilizado la herramienta ReservoirPy en Python, como se ha mencionado en el Capítulo 3.

Para ello, se han importado las señales MEA estandarizadas y se ha creado un modelo ESN que cuenta con dos nodos reservorio y un nodo de capa de salida con regularización Ridge usando dicha herramienta en Python.

Esta implementación se considera de DeepESN [84], y se ha elegido debido a su mejora de rendimiento en comparación a un único nodo reservorio, tanto en este TFG como en otras investigaciones ([84] y [85]).

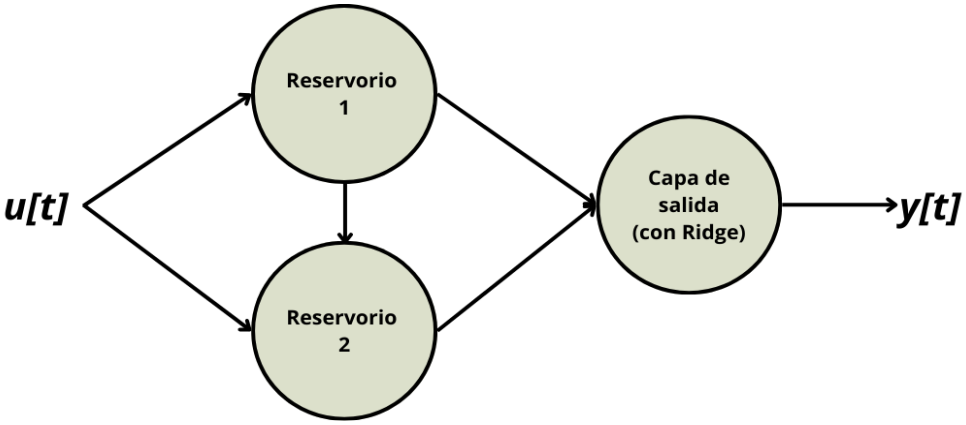

Figura 6.9: Implementación DeepESN en ReservoirPy. Elaboración propia, basada en [22].

Una vez creado el modelo y comprobado que funcione con los datos disponibles, se ha realizado una búsqueda de hiperparámetros para que el modelo se ajuste lo máximo posible a las señales MEA.

Para que la tarea de búsqueda de los hiperparámetros óptimos no se demorase demasiado, se realizó una división de los datos en bloques [86] (ver Figura 6.10 para un mejor entendimiento).

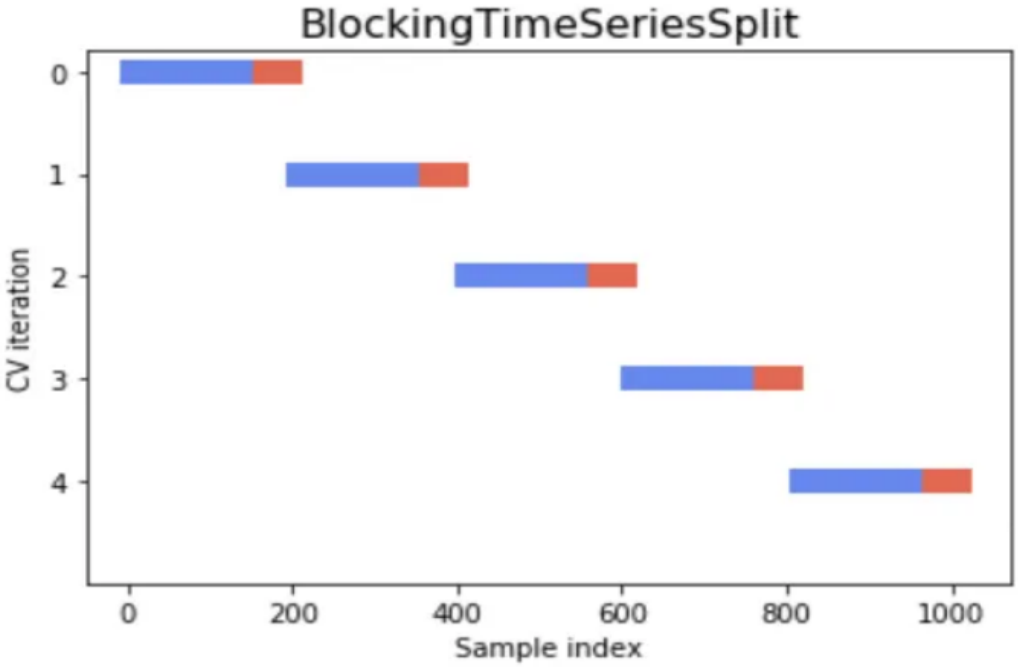

Figura 6.10: División en bloques para las series temporales [86].

Normalmente, para separar el conjunto de datos en entrenamiento y prueba para introducirlo en un modelo se emplea la validación cruzada.

Este método permite prevenir el sobreajuste y evaluar el rendimiento del modelo de una manera más robusta que el simple método de división entrenamiento-prueba.

Pero la validación cruzada en series temporales no es sencilla, ya que no se pueden elegir muestras aleatorias y asignarlas al conjunto de prueba o al conjunto de entrenamiento porque no tiene sentido utilizar valores del futuro para predecir valores del pasado. Esto nos indica que existe una dependencia temporal entre las observaciones, y debemos preservar esa relación durante la prueba.

Mediante esta división en entrenamiento (azul) y prueba (rojo) nos aseguramos de que, al ser series temporales, no estemos introduciendo al modelo el futuro de la serie temporal al hacer validación cruzada. De esta forma obtenemos todas las ventajas que nos ofrece la validación cruzada para las series temporales.

Una vez preparados los datos para realizar esta optimización, se ha empleado Optuna como herramienta para llevar a cabo esta tarea.

Los resultados obtenidos en cada búsqueda se mostraran en el Capítulo 8.

Limitaciones

La implementación práctica de las ESN presenta diversos desafíos que limitan su rendimiento y efectividad. En este TFG se han detectado los siguientes.

- Las ESN cuentan con conexiones y parámetros que se inician de forma aleatoria, que pueden conducir a un rendimiento inestable en la predicción de las señales MEA a largo plazo.
- Si los datos de entrada que se proporcionan a la ESN contienen demasiado ruido, las predicciones pueden ser negativamente afectadas debido a la gran sensibilidad que muestra este tipo de redes.

- Las ESN solo pueden predecir y generar dichas señales a corto plazo debido a su capacidad de representación.

No obstante, esto ya ha sido objetivo de estudio, como se puede ver en las posibles soluciones de [87].

Algunas de estas mitigaciones se han puesto en práctica en este trabajo para conseguir el mejor rendimiento de la red, aunque no eliminen el problema por completo. Para ello habría que recurrir a arquitecturas más complejas, como se menciona en ese artículo.

En este caso, para una investigación académica con un periodo de tiempo limitado, se han modificado ciertos aspectos de la estructura de la ESN original. Entre ellas se encuentra la regularización del entrenamiento con la regularización Ridge.

Otra medida que se ha tenido en cuenta ha sido utilizar el método implementado BlockedTimeSeries, como se ha visto en el apartado anterior, para mejorar la robustez de la red. Este método de división de los datos se empleará también en posteriores modelos debido a su efectividad con las series temporales [86].

Por último, se ha buscado una optimización del entorno de aprendizaje para una mejor adaptabilidad a los datos. Para ello se ha realizado un preprocesamiento adecuado de los datos (como se menciona en el Capítulo 5) y se han buscado los hiperparametros adecuados para la predicción y generación de estas señales MEA.

6.3.3. Liquid State Machine (LSM)

Las LSM son, junto a las ESN, los modelos referentes del marco RC, ya que estos son los dos modelos más populares y que han servido de inspiración para la creación de otros modelos dentro de esta arquitectura.

Estos modelos se hicieron simultáneamente en equipos de desarrollo diferentes, por eso son bastante similares conceptualmente [88]. La principal diferencia, como se muestra en la Figura 6.11 es que en las ESN, como se ha visto anteriormente, el reservorio esta formado por neuronas recurrentes, mientras que el reservorio de las LSM (denominado 'liquido') está formado por neuronas de impulsos (siendo su correspondencia anglosajona el término 'spiking neurons'). Se abordará este concepto en más profundidad en los siguientes apartados.

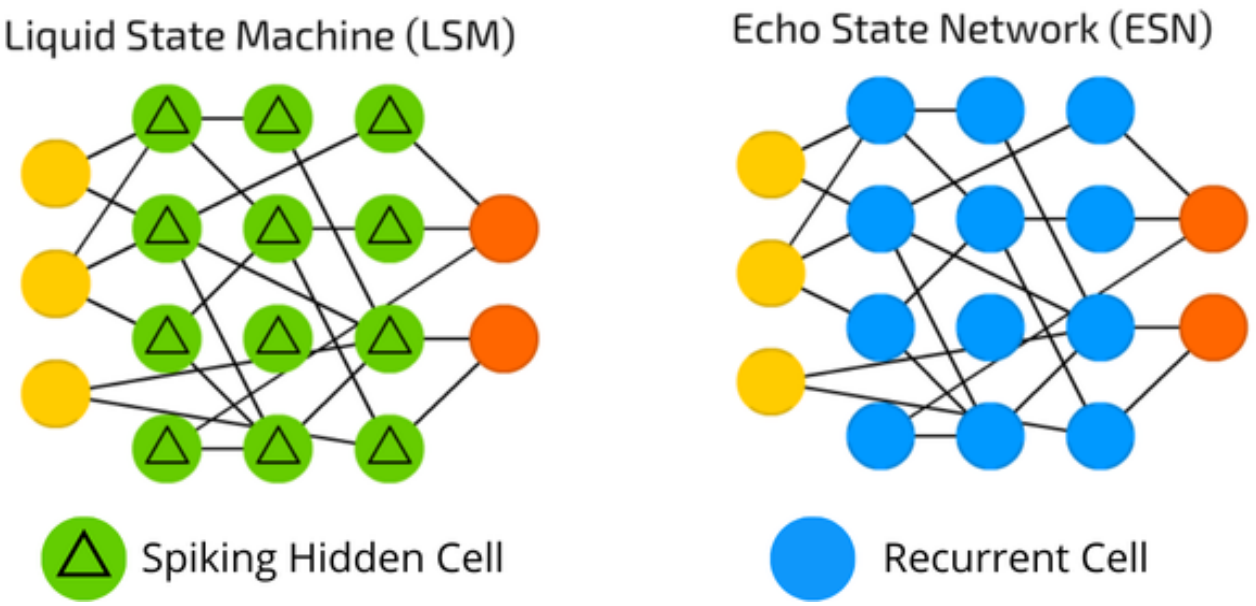

Figura 6.11: Esquema comparativo de la arquitectura de una LSM y ESN [71].

Base teórica

Una LSM es un tipo de red neuronal constituida por neuronas de impulsos inicializados aleatoriamente que se encargan de procesar información en tiempo real continuo.

Este tipo de neuronas está inspirada en las propiedades dinámicas de los cerebros biológicos, pues intentan emular el comportamiento real de las neuronas a la hora de ejecutar el modelo [89] de forma más realista que las tradicionales (ver la comparación en la Figura 6.12).

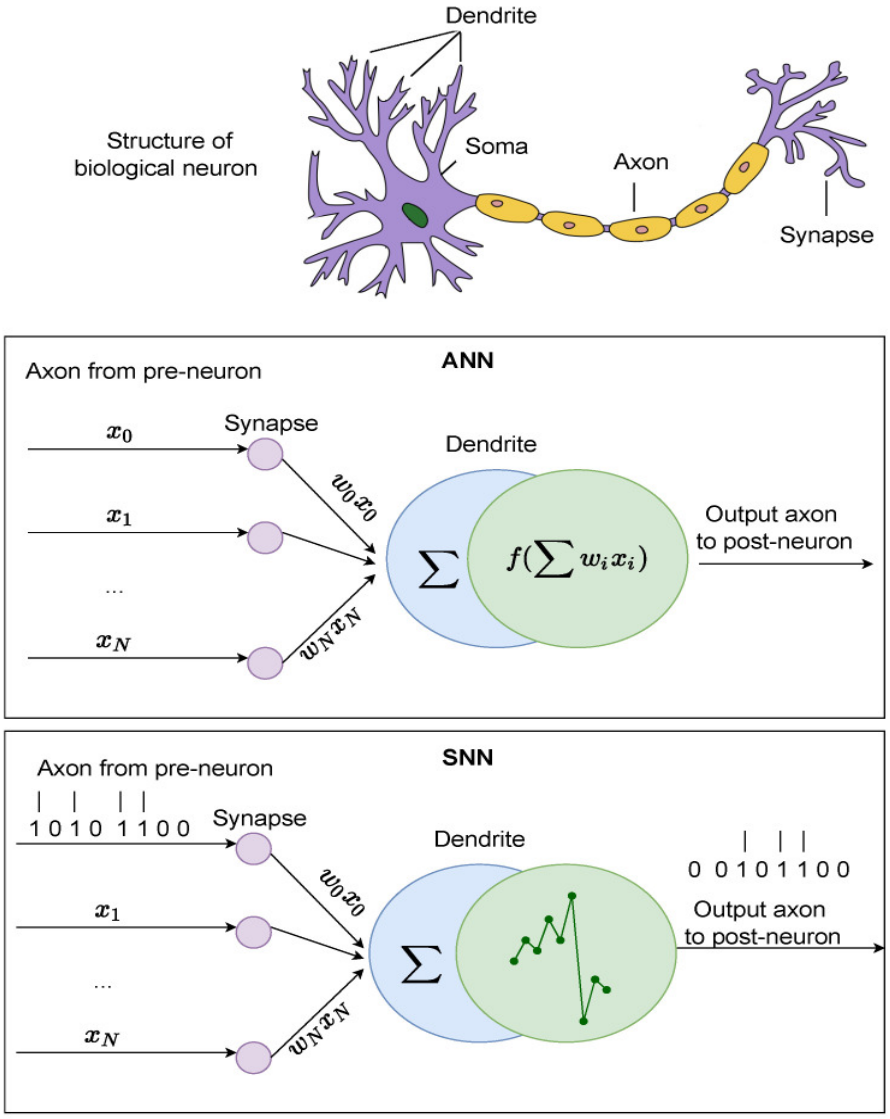

Figura 6.12: Esquema comparativo de una neurona natural, una neurona tradicional y una neurona de impulsos en sentido descendiente [90].

Estas neuronas se comunican entre sí mediante pulsos eléctricos cortos llamados potenciales de acción, también conocidos como *spikes*. Las neuronas imitan este comportamiento de las neuronas que encontramos en la biología generando pulsos de voltaje en lugar de valores continuos [91].

Además, como las neuronas actúan dinámicamente, la información se procesa a través de neuronas interconectadas en vez de neuronas secuenciales, permitiendo de esta forma cálculos complejos y no lineales mientras se mantiene una estructura de red flexible y adaptable [63]. Con esto se consigue que sean útiles a la hora de procesar series temporales, y por eso se ha tenido en cuenta en este TFG.

La arquitectura de la LSM está formada por los siguientes componentes principales cuyo funcionamiento es muy similar a la ESN:

- **Capa líquida**: está compuesta por las neuronas mencionadas anteriormente. Recibe la entrada y se transforma internamente a una dimensión entendible por la capa y la interpreta. La palabra "líquida" en el nombre proviene de la analogía que se hace al dejar caer una piedra en un cuerpo de agua u otro líquido en reposo. La piedra que cae generará ondulaciones en el líquido. La entrada (movimiento de la piedra que cae) se ha convertido en un patrón espacio-temporal de desplazamiento del líquido (ondulaciones) [92].
- **Capa de salida**: esta capa interpreta la actividad en la capa líquida y proporciona la salida, revirtiendo la transformación que se hizo al inicio.

Esta arquitectura hace que el aprendizaje sea robusto y rápido, ya que se delega nuevamente para el aprendizaje en la capa de salida. La primera razón para justificar esto es que al aprender de una neurona de la capa de salida, hace el procedimiento rápido y que no se quede en un mínimo local (como podía pasar con el algoritmo de BPTT. Otra ventaja descrita de este modelo es su capacidad de generalización de los flujos de entrada a pesar de que el entrenamiento sea tan sencillo [82].

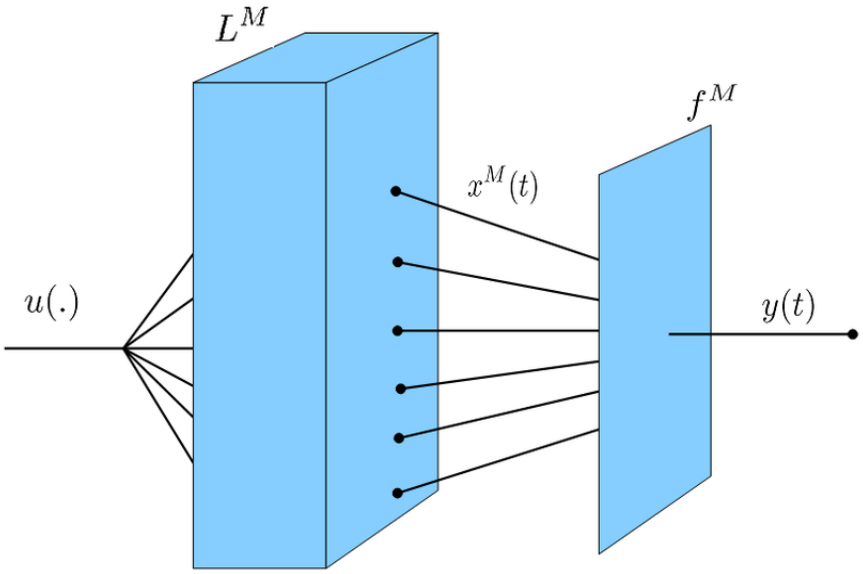

Figura 6.13: Arquitectura básica de una LSM [93].

Observando la Figura 6.13, se puede describir el funcionamiento de la LSM formalmente para un único flujo de entrada y de salida. Siendo $u(\cdot)$ un flujo de datos de entrada, L^M la capa líquida y $x^M(t)$ es la entrada a la capa de salida f^M, en el instante de tiempo t, que en verdad es el estado interno de la capa líquida que se hace visible para la capa de salida [82].

Teniendo esto en cuenta, $x^M(t)$, también denominado estado líquido de la LSM se puede definir como:

$$x^M(t) = (L^M u)(t) \tag{6.9}$$

Y con ello obtenemos la salida del modelo $y(t)$:

$$y(t) = f^M(x^M(t)) \tag{6.10}$$

Implementación

Como se ha explicado anteriormente, la LSM es capaz de interpretar flujos de datos. Para ver cómo sería el flujo de datos con señales, se puede visualizar la Figura 6.14

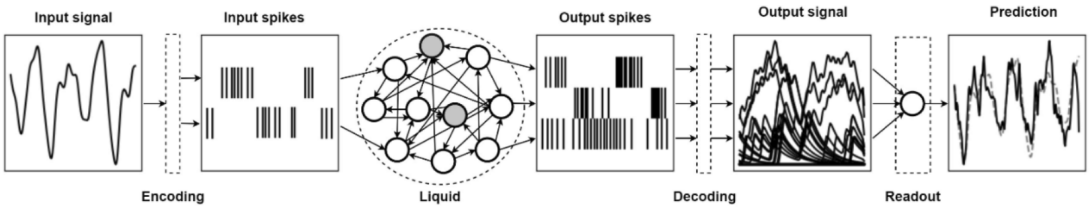

Figura 6.14: Flujo de datos de una señal en una LSM [63].

Mediante el código que implementa la LSM disponible en GitHub de Johannes Maurin [94], se ha implementado de forma que admitiese las señales MEA.

Se ha elegido este repositorio debido a que se ha querido probar las ventajas de la cuantización en un modelo.

La cuantización es el proceso mediante el cual se reduce la precisión de un modelo, que normalmente está representado mediante números punto flotante de 32 bits, para conseguir una representación muchísimo más pequeña [95].

Este método ayuda a reducir el tamaño de la memoria y sus respectivos accesos durante el entrenamiento, además al sustituir las operaciones elementales por operaciones bit a bit, se consigue un ahorro de energía considerable al compararlo con la versión normal de 32 bits en punto flotante, sin afectar apenas la cantidad de aciertos lograda en la inferencia del modelo [96].

Johannes Maurin [63] en su trabajo prueba con 2, 3, 6 y 16 bits. En este TFG se ha probado con representaciones de 3, 6, 8 y 16 bits.

Se han sustituido los 2 bits que emplea Johannes Maurin en su artículo por los 8 debido a que la diferencia presentada en 2 y 3 bits con las señales MEA era despreciable. Otro motivo por el cual se eligieron 8 bits es porque esta es la cuantización que emplean los dispositivos hardware de Google Coral y, de esta forma, se conseguía comprender los conceptos clave para poder realizar el Capítulo 7.

Posteriormente, se buscarán los hiperparámetros que maximicen el rendimiento del modelo cuantizado. Para ello, se han seleccionado el número de neuronas de la capa líquida N, el tamaño de las señales MEA que se introducirán al modelo l y el tamaño de la señal generada k.

Limitaciones

La implementación práctica de las LSM presenta diversos desafíos que limitan su rendimiento y efectividad. En este TFG se han detectado los siguientes:

- Tiene un grado de complejidad interna que hace complicado controlar el comportamiento de la red.
- Hay poco control sobre el proceso, ya que no hay una forma garantizada de diseccionar una red en funcionamiento y averiguar cómo o qué cálculos se están realizando.

6.4. Next Generation Reservoir Computing (NGRC)

Este apartado esta basado en su totalidad en el artículo [62], ya que al ser un enfoque relativamente reciente, aún no hay demasiada bibliografía completa al respecto.

Hasta ahora todo lo que se ha mencionado en este documento es cómo el paradigma RC soluciona ciertos problemas de los modelos de RNN para realizar tareas de predicción. Incluso esta aproximación es capaz de modelar sistemas dinámicos caóticos complejos de manera eficaz. Sin embargo, los modelos de este enfoque sufren de algunas limitaciones, como se ha mencionado en sus respectivos apartados (6.3.2 y 6.3.3).

Para intentar solucionar dichos problemas, se han establecido algunas métricas y requisitos de optimización para intentar que este enfoque aproveche todo lo posible los datos de entrada, en este caso las señales MEA, pero depende de los datos con los que se trabaje, por lo que esta optimización no resulta efectiva en términos de generalización.

Por ello, se ha intentado buscar un enfoque universal, basando su implementación en un RC y otras características que se discutirán en los próximos apartados.

6.4.1. Base teórica

La idea surgió de intentar combinar un RC con nodos que se activan de forma lineal con un vector de características que sume funciones no lineales de los valores de los nodos del reservorio. Teniendo en cuenta que un RC de estas características es idéntico a una máquina NVAR, se puede sustituir el reservorio por la NVAR y de esta forma conseguir el modelo que se va a estudiar en este TFG: NGRC.

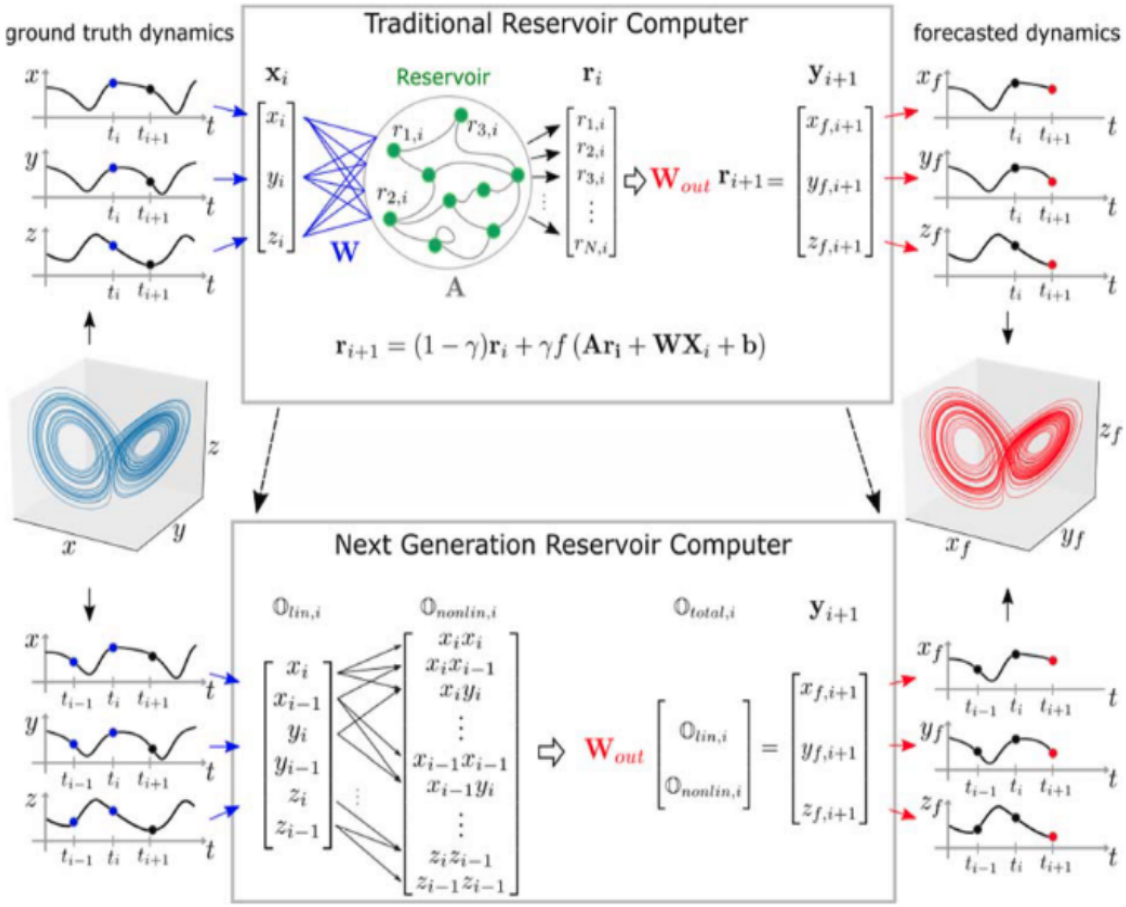

Figura 6.15: Comparación entre el enfoque NGRC y RC [62].

Como se puede ver en la imagen, se crea un vector de características a partir de los datos de entrada. Para computar la salida, se obtiene del entrenamiento de la siguiente ecuación:

$$y(i+1) = W_{out}\mathbb{O}_{total} \tag{6.11}$$

Donde W_{out} es la salida de la matriz de pesos y el vector de características $\mathbb{O}_{total}$ representa que puede estar compuesta por constantes o términos lineales o no lineales.

Las partes lineales se obtienen de la observación actual y pasada de los datos de entrada, mientras que las no lineales se obtienen mediante la transformación del contenido de las partes lineales, capturando de esta forma las relaciones no lineales existentes, obteniendo como resultado lo que se ve en la Ecuación 6.12:

$$\mathbb{O}_{total} = c \bigoplus \mathbb{O}_{lin} \bigoplus \mathbb{O}_{nonlin} \tag{6.12}$$

De esta forma, para obtener la salida simplemente hay que entrenar las características obtenidas de 6.11 y usar la regularización Ridge como se hacía en los modelos de RC.

6.4.2. Implementación

La implementación del modelo se ha hecho siguiendo y modificando el código de Daniel J. Gauthier mencionado en [62], que se puede encontrar en el repositorio de GitHub mencionado en dicho artículo [97].

La parte lineal del modelo se mantiene igual que lo descrito en el artículo, pero para la parte no lineal se ha añadido un nuevo hiperparámetro que permite ampliar el orden de las funciones no lineales.

Como en el modelo presentado por Daniel J.Gauthier solo permitía como máximo funciones de orden 2, se ha planteado una nueva mejora a este modelo mediante la implementación de un nuevo hiperparámetro que permita funciones de mayor orden.

Para ello, se generan todas las combinaciones posibles de índices (con reemplazo) que puede tomar del vector de características lineales en función al orden de no linealidad especificado. De esta forma, se consigue dotar al modelo de una mayor expresividad, permitiendo capturar de una mejor manera las relaciones no lineales del sistema.

Para finalizar con la implementación, cabe mencionar que se hizo la búsqueda de hiperparámetros como con los anteriores modelos. Los hiperparámetros a buscar son el valor de la regularización Ridge (*ridge*), el número de observaciones pasadas (k) que representan el estado del sistema hasta entonces y el nuevo hiperparámetro creado en este TFG que representa el máximo orden que pueden tener las funciones no lineales.

6.4.3. Limitaciones

Siguiendo con el artículo [62], algunas de las ventajas que se presentan con este enfoque es que debido a que el tamaño del vector de características es mucho menor que en RC se han obtenido ejecuciones satisfactorias en menor tiempo. Otra ventaja frente a RC es que hay menos parámetros que ajustar e hiperparámetros que optimizar, haciendo que el modelo sea computablemente más rápido, ahorrando cierto tiempo en esta etapa.

Sin embargo, como se mencionan en las investigaciones más recientes [98], se ha descubierto que este enfoque

puede presentar algunas complicaciones si los datos están recolectados con poca frecuencia de muestreo, ya que se pierde información relevante para el modelo, o el tipo de no linealidades que se presentan en los datos.

6.5. Long Short-Term Memory

El último modelo del que se han realizado inferencias es la LSTM. Este modelo se eligió como comparación a los anteriores modelos, ya que como se ha mencionado, RC buscaba mejorar los aspectos en los que RNN no era del todo competente. Por ello, la LSTM se ha elegido como representante de la arquitectura de RNN en este TFG ya que también se han explorado tareas de predicción y generación con este modelo [99].

6.5.1. Base teórica

La LSTM, creada en los años 90, sigue siendo de las RNN más utilizadas. Se creó con la idea de conseguir solucionar el problema de la evanescencia del gradiente que sucedía en las RNN simples [100].

Su arquitectura se muestra en la siguiente figura:

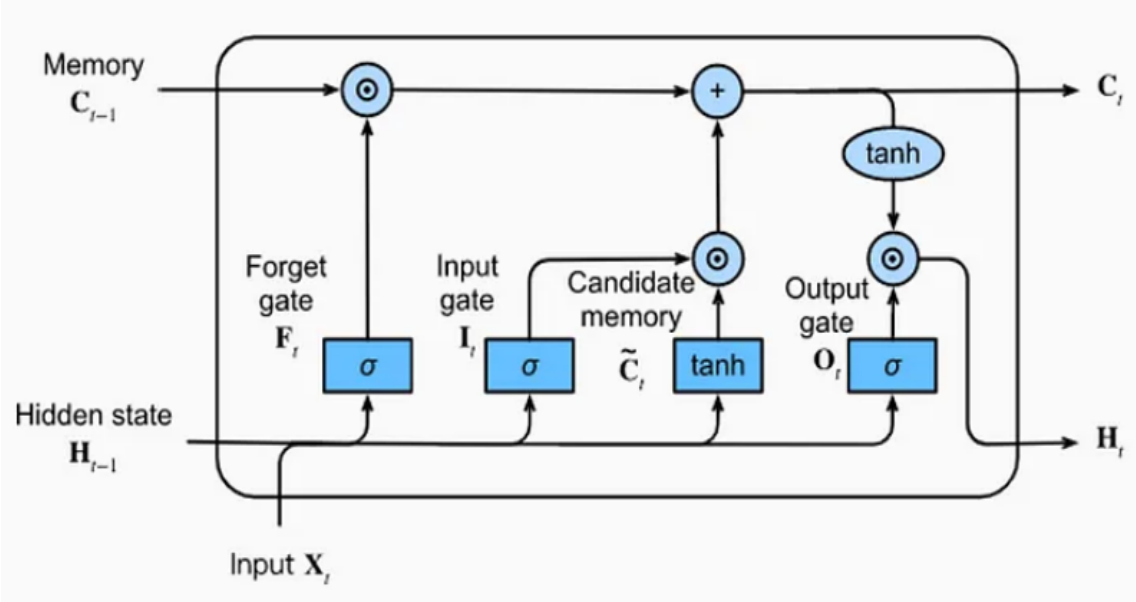

Figura 6.16: Arquitectura de una LSTM [101].

A partir de la Figura 6.16, se va a explicar su funcionamiento [102].

En la celda de la LSTM entran, en el instante *t*, los datos de entrada *X*, y las salidas en instantes previos, tanto las almacenadas en la memoria representada por *C* como las del instante *t-1* contenidas en *H*.

Posteriormente, esta información se trata mediante tres bloques de funciones denominadas puertas. Cada una de ellas se encarga de una tarea, como si de una memoria de verdad se tratase:

- La puerta del olvido (F_t) se encarga de eliminar la información que ya no merece la pena que sea recordada. Para ello, usa la activación de la función sigmoide (σ) con la entrada actual y la del instante *t - 1*. Con el resultado obtenido, lo multiplica con el contenido que se encuentra disponible dentro de la memoria y realiza la eliminación de los datos correspondientes.

- La siguiente puerta, la de entrada (I_t), se encarga de añadir nuevos contenidos a la memoria. Para ello, usa σ para saber qué contenidos de la entrada merece la pena guardar, y después lo normaliza mediante la activación de la tangente hiperbólica ($tanh$). Esto genera el candidato a entrar a la memoria ($\widetilde{C}_t$).
- Por último, la puerta de salida se encarga de predecir el siguiente valor a partir del actual, generando lo que se va a enviar a la siguiente celda de la LSTM mediante σ y $tanh$.

De esta forma consigue recordar entradas anteriores, a pesar de que estén alejadas en el tiempo. Por ello, se ha elegido para realizar la tarea y predicción de las señales MEA.

6.5.2. Limitaciones

Las limitaciones de este modelo contienen las ya vistas en el apartado de las limitaciones de una RNN (6.2.2). A pesar de intentar solucionar estos problemas, aun así pueden seguir sufriendo de evanescencia del gradiente [100].

Otras limitaciones propias de las LSTM son su tendencia al sobreajuste cuando se dispone de pocos datos de entrenamiento y los recursos computacionales necesarios para entrenar las LSTM, haciéndolas lentas en algunos experimentos [103].

Finalmente, para la implementación de la LSTM se ha optado por desarrollar el código de forma que pudiera ejecutarse en un dispositivo hardware de Google Coral. Por este motivo, la implementación de este modelo se verá en el siguiente capítulo.

Capítulo 7

Implementación en hardware

En este capítulo se va a presentar la implementación exitosa de una LSTM en un dispositivo Google Coral, específicamente en la placa de desarrollo Coral Dev Board.

Para ello, se hablará de la motivación de este capítulo, seguido de la definición de Google Coral y de las prestaciones que ofrece que puedan ser de utilidad para esta investigación. Después, se mencionará la implementación en Coral Dev Board con las señales MEA disponibles y se debatirán los aspectos relevantes de esta implementación.

7.1. Google Coral

Como se ha podido observar con las otras implementaciones, implementar modelos de IA con conjuntos grandes de datos en ordenadores convencionales requiere de un gran coste computacional.

En este caso, para tratar series temporales, que requieren de memorizar dependencias para inferir el siguiente dato o generar secuencias, el coste se incrementa.

Google Coral [104] es una iniciativa de Google que ofrece hardware y software para la implementación de aplicaciones de IA en dispositivos de borde, es decir, dispositivos que procesan y analizan datos localmente en lugar de enviarlos a la nube.

La ventaja de utilizar dispositivos de borde frente a un ordenador convencional o su ejecución en la nube [105] para este proyecto reside principalmente en que ejecuta las señales de forma local sin necesidad de hacerlo en un servidor externo.

Esto reduce los tiempos de espera drásticamente, aporta una mayor privacidad y seguridad al no compartir los datos con terceros y genera una mayor fiabilidad y robustez al no depender de una conexión a internet estable.

A mayores, estos dispositivos están diseñados para realizar tareas específicas con un consumo energético reducido, siendo más eficientes energéticamente que los ordenadores convencionales.

Google Coral ofrece una gran variedad de dispositivos que pueden realizar tareas de predicción y generación de series temporales, ya que cuentan con un procesador dedicado para ML y una arquitectura de hardware optimizada para la ejecución de modelos de IA.

En este TFG se ha decidido realizar la implementación en el dispositivo Google Dev Board, el cual se describirá a continuación.

7.2. Google Dev Board

Para llevar a cabo la generación y predicción de señales MEA se ha elegido este dispositivo de la gama de Google Coral. Se puede observar una imagen del dispositivo en la Figura 7.1 y esquemas en las Figuras 7.2 y 7.3.

Google Dev Board [106] es una computadora de placa única que cuenta con un procesador TPU Edge de Google, diseñado específicamente para la aceleración de inferencia de ML.

La Dev Board también incluye una variedad de otros componentes, que se discutirán a continuación.

7.2.1. Especificaciones del sistema

Las especificaciones completas del dispositivo se pueden encontrar en [107].

Tabla 7.1: Tabla de especificaciones de Dev Board [106].

COMPONENTE	**ESPECIFICACIÓN**
CPU	NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F)
GPU	Gráficos integrados GC7000 Lite
Acelerador ML	Coprocesador Google Edge TPU: 4 TOPS (int8); 2 TOPS por vatio
RAM	4 GB LPDDR4
Memoria flash	8 GB eMMC, ranura MicroSD
Inalámbrico	Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) y Bluetooth 4.2
USB	Type-C OTG; Type-C para alimentación; Type-A 3.0 host; Consola serie Micro-B
LAN	Puerto Ethernet Gigabit
Audio	Conector de audio de 3,5 mm (compatible con CTIA); Micrófono digital PDM (x2); Terminal de 4 pines de 2,54 mm para altavoces estéreo
Vídeo	HDMI 2.0a (tamaño completo); Conector FFC de 39 pines para pantalla MIPI-DSI (4 carriles); Conector FFC de 24 pines para cámara MIPI-CSI2 (4 carriles)
GPIO	Riel de alimentación de 3,3 V; Impedancia programable de 40 a 255 ohmios; Corriente máxima de 82 mA
Potencia	5V DC (USB Type-C)

Gracias a las especificaciones vistas en 7.1, se puede aplicar a esta investigación por algunas características clave que se pueden extraer [106]:

- El coprocesador Edge TPU, que es un tipo de circuito integrado diseñado por Google, permite acelerar el entrenamiento y la inferencia de los modelos, ya que están optimizados de forma que son capaces de realizar tareas concretas mediante operaciones tensoriales. Esto resulta útil para la generación y predicción de las señales MEA en tiempo real, pues resulta ser bastante eficiente en lo que a rendimiento y consumo energético se refiere.
- Al ser compatible con Tensorflow Lite, permite usar modelos de este marco de software, los cuales han sido fáciles de implementar gracias a lo instruido en asignaturas a lo largo de esta carrera universitaria.
- Para ejecutar en este dispositivo, es necesario cuantizar los modelos, lo cual presenta una serie de ventajas ya debatidas en la implementación de la LSM (apartado 6.3.3).

Figura 7.1: Imagen de un dispositivo Google Dev Board [106].

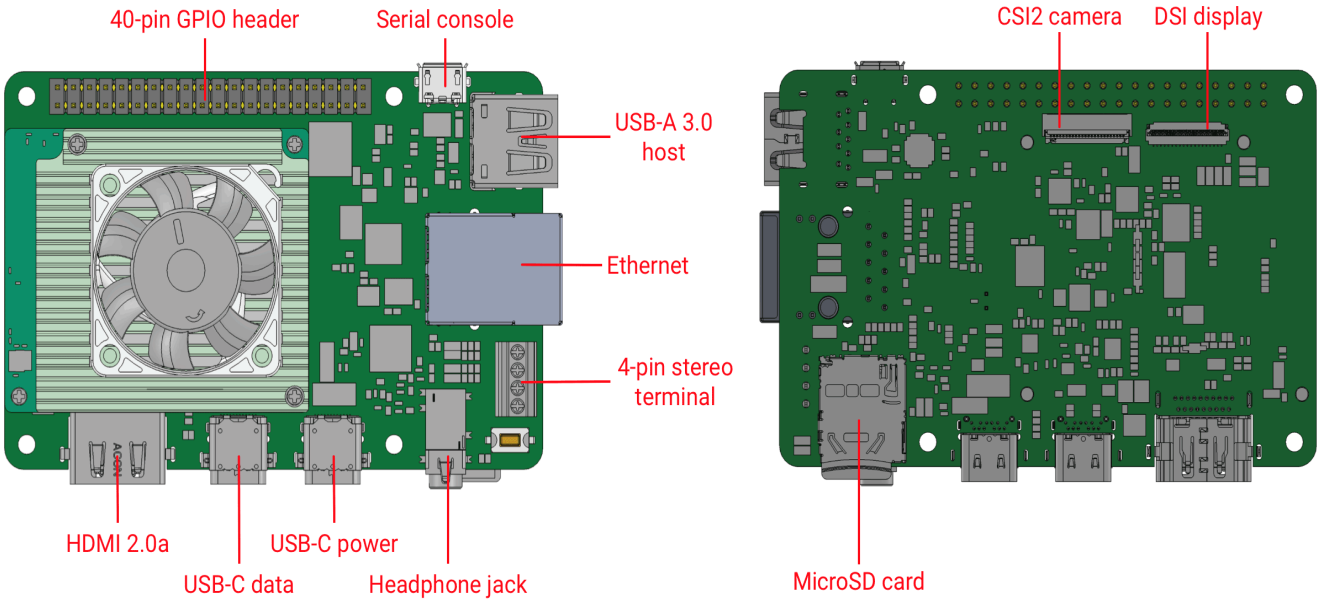

Figura 7.2: Conectores de Google Dev Board [107].

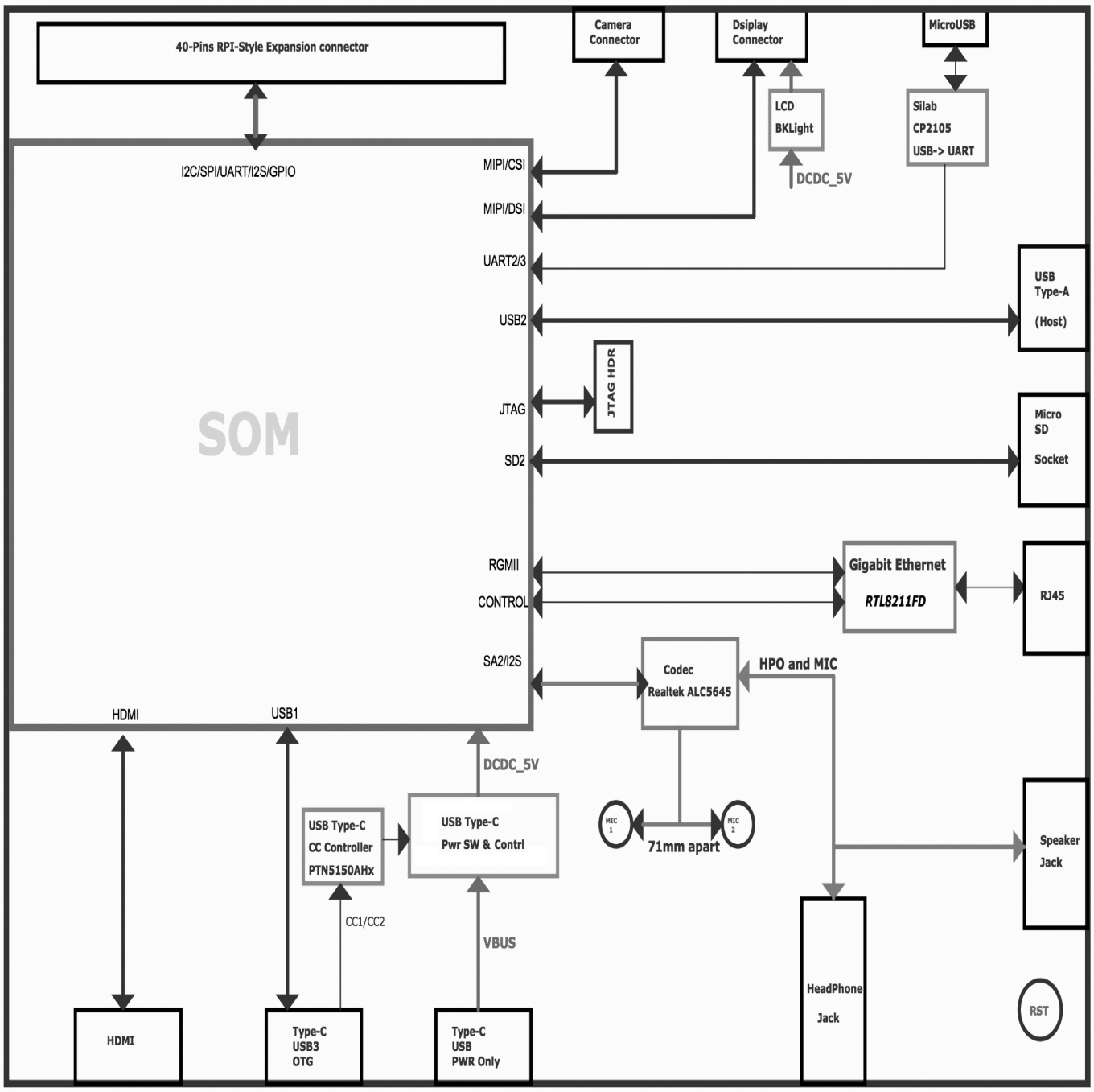

Figura 7.3: Diagrama de componentes de Google Dev Board [107].

7.3. Implementación

La implementación se llevará a cabo realizando un código de una LSTM en Google Colab, ya que es el tipo de RNN elegida para este TFG como se explicó en 6.5. Para ello, se implementará una LSTM en este dispositivo.

Por consiguiente, se siguieron varios tutoriales (los más útiles fueron [108] y [109]), y otros proporcionados por la misma plataforma de Google Coral [110], en específico [111].

Una vez realizados los modelos en Google Colab, se almacenan en un archivo con extensión .tflite, ya que es el tipo de archivos que admite el Google Dev Board.

Estos archivos tienen que estar cuantizados, entre otra serie de pasos, para que sean admitidos por el dispositivo, siguiendo el esquema mostrado en la Figura 7.4.

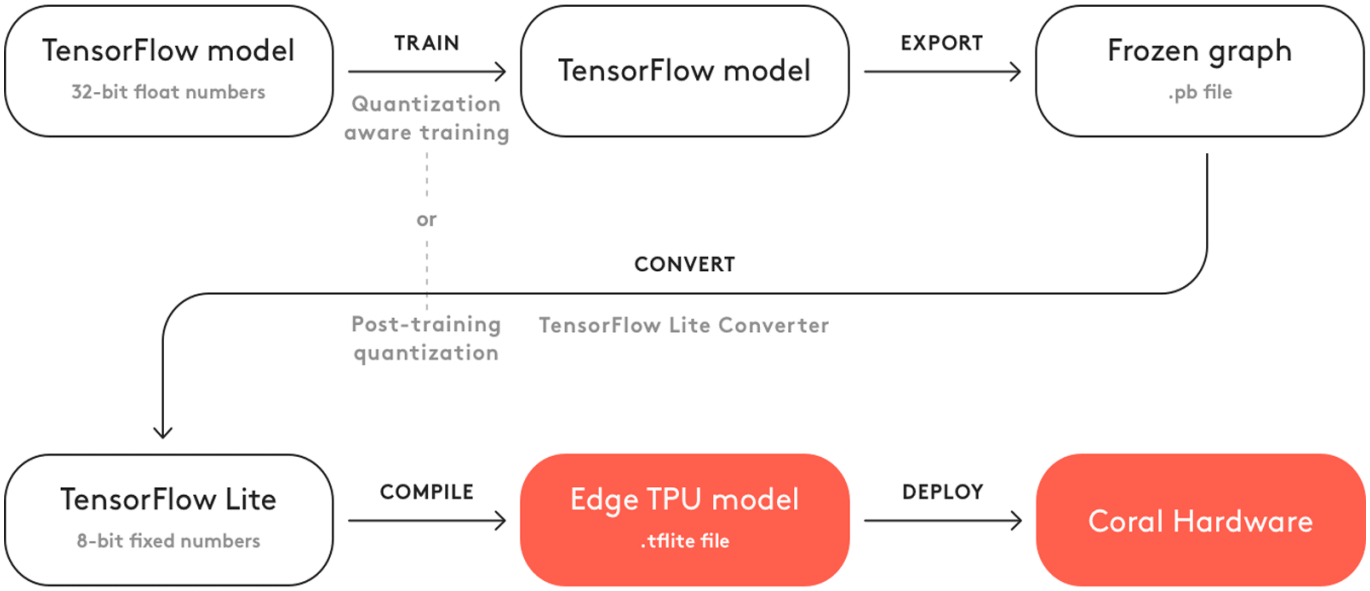

Figura 7.4: Flujo de trabajo para conseguir un modelo aceptable por un dispositivo de la gama Google Coral [112].

Como se ve en la figura anterior, la cuantización se puede hacer antes del entrenamiento o después. En este TFG, la cuantización se ha hecho después del entrenamiento, debido a su sencillez, aunque sería recomendable hacerlo antes del entrenamiento para evitar perder precisión.

Antes de cargar estos archivos, se necesitan hacer una serie de pasos para realizar el montaje del dispositivo Dev Board.

Para conectar el dispositivo, es necesario emplear dos cables tipo C, y conectar uno de ellos al puerto OTG y el otro al puerto de energía, situados en la parte inferior izquierda de la Figura 7.3.

Ya que varias de las placas de desarrollo de Google Coral (incluido Dev Board) funcionan con un derivado de Debian Linux llamado Mendel Linux, es necesario instalar Mendel Development Tools (MDT) para conseguir la comunicación con el dispositivo [113].

Una vez cargados los modelos y el conjunto de datos mediante MDT, se compilan los modelos.

A modo de ejemplo, se va a emplear el fichero de generación de señales MEA *LSTM_gen_MEA.tflite*.

```
Edge TPU Compiler version 16.0.384591198
Started a compilation timeout timer of 180 seconds.
Model compiled successfully in 113 ms.
Input model: LSTM_gen_MEA.tflite
Input size: 28.55KiB
Output model: LSTM_gen_MEA_edgetpu.tflite
Output size: 144.81KiB
On-chip memory used for caching model parameters: 62.25KiB
On-chip memory remaining for caching model parameters: 7.60MiB
Off-chip memory used for streaming uncached model parameters: 640.00B
Number of Edge TPU subgraphs: 1
Total number of operations: 6
Operation log: LSTM_gen_MEA_edgetpu.log
See the operation log file for individual operation details.
Compilation child process completed within timeout period.
Compilation succeeded!
```

Al compilar, se genera un .log que indica qué partes del modelo se asignan a la TPU.

```
Edge TPU Compiler version 16.0.384591198
Input: LSTM_gen_MEA.tflite
Output: LSTM_gen_MEA_edgetpu.tflite
Operator                       Count      Status
FULLY_CONNECTED                2          Mapped to Edge TPU
UNIDIRECTIONAL_SEQUENCE_LSTM   1          Mapped to Edge TPU
QUANTIZE                       2          Mapped to Edge TPU
STRIDED_SLICE                  1          Mapped to Edge TPU
```

Una vez compilados, se crearán unos nuevos archivos con la siguiente estructura:

nombredelarchivo_edgetpu.tflite

Este archivo, junto a los datos, se envían al dispositivo mediante el comando *push* de MDT.

Una vez cargado el modelo en el dispositivo, está preparado para realizar las inferencias necesarias.

Los resultados obtenidos se describirán en el siguiente capítulo.

7.3.1. Limitaciones del hardware

A pesar de las ventajas de las que disponen los modelos de Gooogle Coral, como puede ser su portabilidad o su gran capacidad de cómputo en comparación al gasto energético al tratarse de un dispositivo muy bien optimizado, durante la implementación de los modelos han surgido alguna serie de inconvenientes.

El primero de ellos, y el más notorio, es la cantidad de requisitos y limitaciones para poder compilar un modelo en un dispositivo de Google Coral [112].

Entre estas limitaciones se encuentra la versión de Python, pues solo se puede utilizar con la versión 2.7. Además, las operaciones que se pueden utilizar de Tensorflow son muy limitadas, demorando la elaboración del código.

Otro de los problemas ligado a lo anterior es la falta de soporte por parte de la comunidad para la resolución de problemas, pues hay poca documentación que resuelva correctamente los problemas que surgen durante la implementación.

Con una mejor red de soporte por parte de los desarrolladores y de la comunidad, la implementación de los modelos se haría muchísimo más fluida, permitiendo a los usuarios gozar de todas las ventajas que proporcionan este tipo de dispositivos.

Capítulo 8

Comparación y discusión de los resultados

En este capítulo se tratarán los distintos resultados obtenidos en cada modelo, además de una comparación entre ellos.

Cabe destacar que todas las gráficas de este capítulo se van a representar en el dominio del tiempo, por lo que el eje X de todas las gráficas corresponderá al tiempo en milisegundos y el eje Y mostrará la amplitud de la señal MEA en microvoltios, como se ha visto en anteriores representaciones (por ejemplo, en la Figura 5.5).

8.1. Resultados ESN

El primer modelo desarrollado en este proyecto es la ESN, como se indicó en la Figura 2.2 en la planificación del modelo.

8.1.1. Resultados de la búsqueda de hiperparámetros

Como se ha mencionado en la implementación, los hiperparámetros a optimizar mediante Optuna han sido el radio espectral *sr*, el escalado de la entrada *iss* y la tasa de fuga *lr* para un determinado número de neuronas *N*.

Es conveniente que la búsqueda se centre en estos hiperparámetros, como se menciona en [114]. Además, la función de pérdida a minimizar es la raíz del error cuadrático medio normalizado (NRMSE) [115].

Posteriormente, el mejor conjunto de hiperparámetros (es decir, los que menor valor tengan en la función de pérdida) se utilizará para ejecutar el modelo, y obtener una serie de resultados de predicción y generación que se añadirán en este TFG.

Cabe resaltar que la búsqueda de hiperparámetros no tiene por qué devolver el mejor conjunto de hiperparámetros, pero sí facilita la búsqueda con respecto a la búsqueda manual, dando de esta forma un conjunto de hiperparámetros que den en general buenos resultados [114].

A la hora de encontrar los hiperparámetros idóneos para la tarea a realizar, se ha seleccionado unos rangos entre los que buscar, teniendo en cuenta la teoría de cada hiperparámetro descrita en [22]:

- **Radio espectral**: es el máximo valor de los autovalores de la matriz de pesos internos W. Cuanto menor sea el valor, más estables serán las dinámicas del sistema. Por el contrario, cuanto más grande sea su valor más caóticas serán.

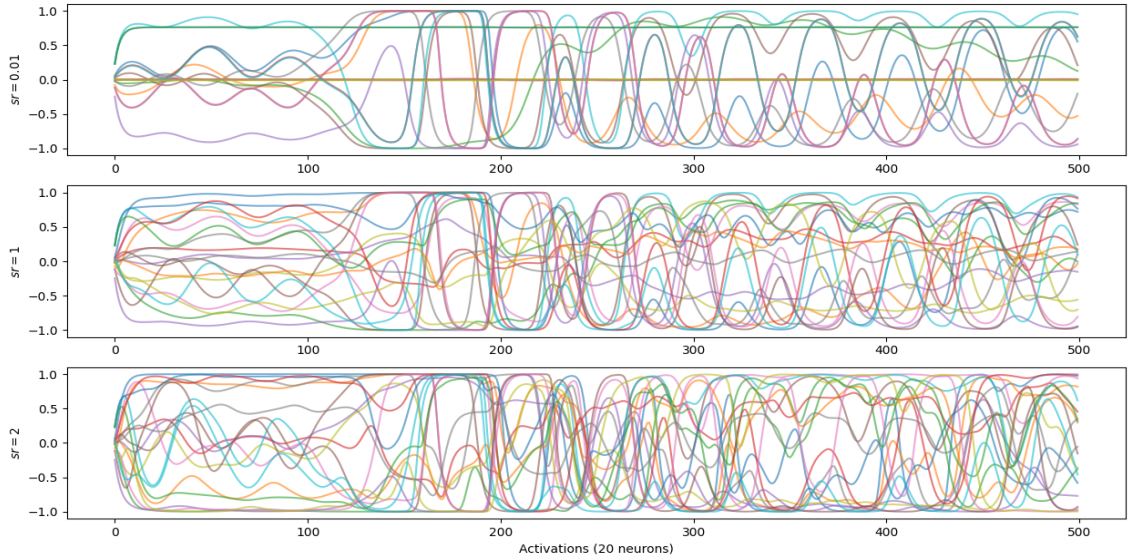

Figura 8.1: Fluctuación del radio espectral en función de las señales MEA. Elaboración propia, basada en [22].

Teóricamente, un radio espectral cercano a 1 permite a los estados internos del reservorio ser más independientes de las condiciones iniciales; se ha realizado la búsqueda entre los valores $1e^{-2}$ y 2.

- **Escalado de entrada**: aplicando este coeficiente a la matriz de pesos de entrada W^{in} se puede variar el grado de influencia de las entradas en el modelo. Cuanto mayor sea este valor, mayor influencia tendrán las entradas hasta llegar a un punto de saturación, que varía en función del modelo. Por el contrario, cuanto menor sea el valor, más independencia tendrá el modelo de las entradas y consecuentemente un comportamiento más libre.

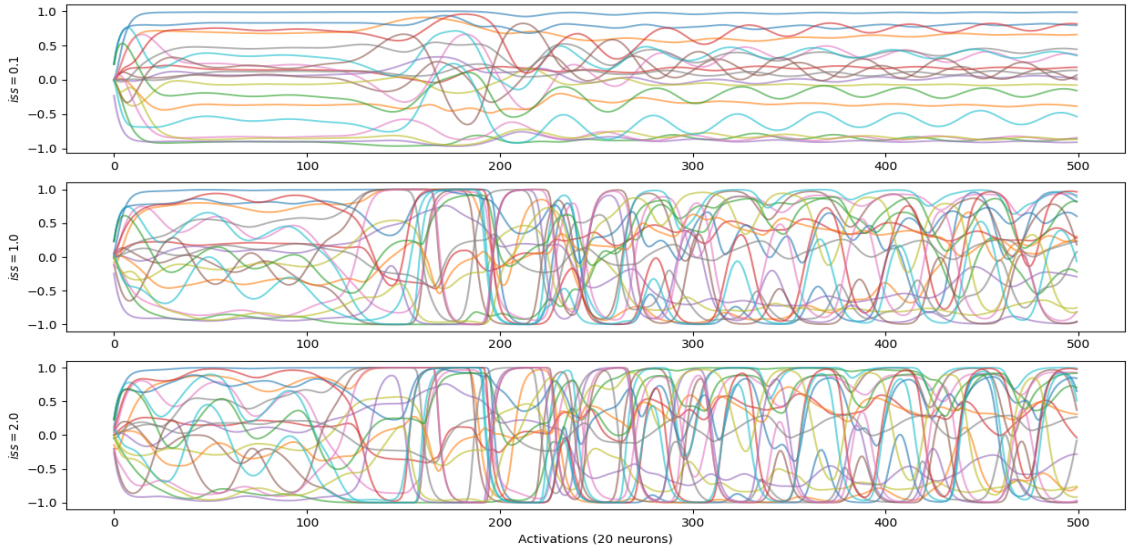

Figura 8.2: Fluctuación del escalado de entrada en función de las señales MEA. Elaboración propia, basada en [22].

Se ha explorado este hiperparámetro en el rango de 0.1 a 2.

- **Tasa de fuga**: controla la influencia que tiene la información pasada en el estado actual mediante la fórmula:

$$x(t+1) = (1-\alpha)x(t) + \alpha f(u(t+1), x(t)) \tag{8.1}$$

donde α es la tasa de fuga, $x(t)$ es el estado actual y las nuevas entradas vienen definidas por:

$$f(u, x) = tanh(W^{in} \cdot u + W \cdot x) \tag{8.2}$$

Cuanto mayor sea la tasa de fuga α, menos influencia tendrán los estados anteriores, 'olvidando' la información pasada. Cuanto más pequeña sea la tasa de fuga, se centra más en los estados anteriores, creando mayores dependencias a largo plazo.

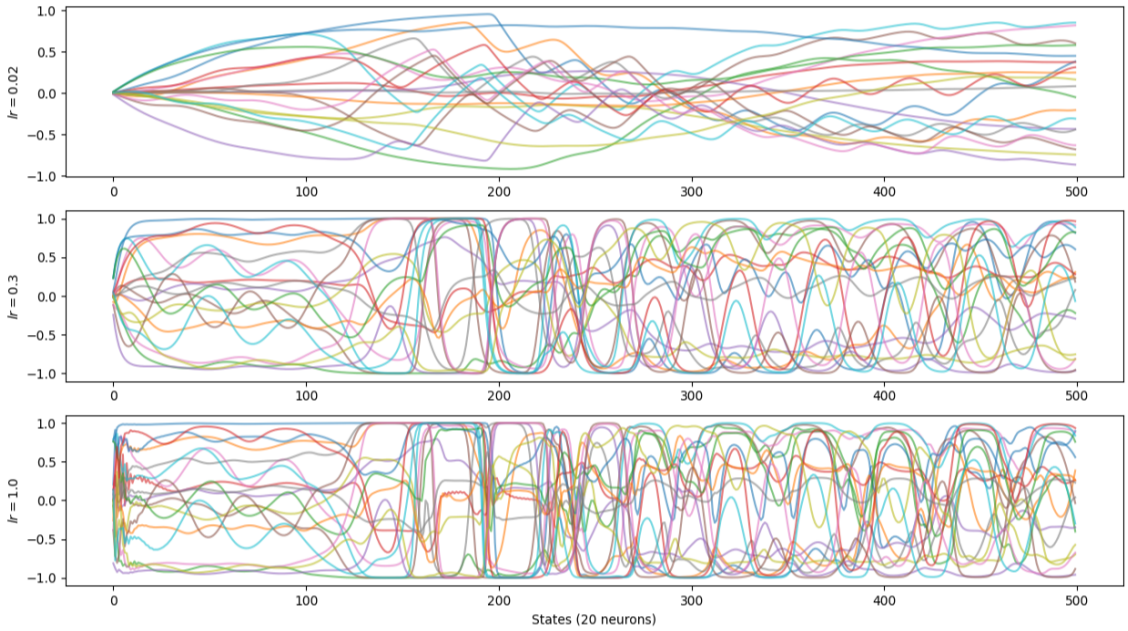

Figura 8.3: Fluctuación de la tasa de fuga en función de las señales MEA. Elaboración propia, basada en [22].

Se ha centrado la búsqueda de este hiperparámetro entre los valores $1e^{-3}$ y 1.

Las Figuras 8.1, 8.2 y 8.3 se obtienen de la muestra de datos de la Figura 8.4.

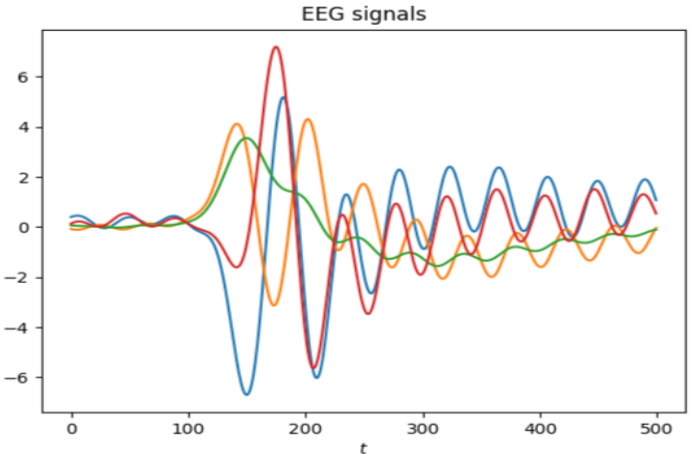

Figura 8.4: Muestra de las señales MEA. Elaboración propia.

Los mejores resultados se observan en la Tabla 8.1.

Tabla 8.1: Resultados de la búsqueda de hiperparámetros en ESN con menor pérdida. Elaboración propia.

N	**sr**	**lr**	**iss**	**loss**
100	0.063120	0.816191	0.123739	0.000103
500	0.430137	0.730453	0.113715	0.000022
1000	0.076380	0.359258	0.109564	0.000076
1500	0.071024	0.990785	0.126851	0.000088
2000	**0.023699**	**0.225728**	**0.124702**	**0.000002**

8.1.2. Resultados del modelo

Como se puede ver en la tabla anterior, el mejor conjunto de hiperparámetros es el correspondiente para el valor N = 2000 neuronas, por lo que se realizarán diferentes pruebas de predicción y generación para este conjunto.

Predicción de las señales MEA

La predicción se ha realizado con 1, 10 y 100 puntos de desfase, como se puede ver en las Figuras 8.5, 8.6 y 8.7, respectivamente.

En el caso de 1 y 10 puntos de desfase, la predicción es realmente buena, logrando unos buenos valores en las métricas evaluadas, como se puede ver a modo de resumen en la Tabla 8.2.

Para 100 puntos de desfase, los resultados son peores, consiguiendo peores resultados debido a la diferencia de los puntos.

Tabla 8.2: Resultados de las métricas en la predicción de señales MEA con ESN.

Puntos de Desfase	MSE	RMSE	NRMSE	R^2
1	0.000055	0.007439	0.005157	0.999957
10	0.008412	0.091721	0.006359	0.993487
100	0.610770	0.781520	0.070010	0.356664

Generación de las señales MEA

En cuanto a la generación, se han realizado las siguientes gráficas con los mejores resultados obtenidos, en las cuales se muestra la generación durante (Figura 8.8), antes (Figura 8.9), y después (Figura 8.10) de un ataque epiléptico registrado en las señales MEA.

El resumen de las evaluaciones de las métricas en la generación de las señales MEA se puede ver a continuación, en la Tabla 8.3. En esta tabla, solo se van a mostrar las medias de las medidas de los cuatro electrodos que aparecen en las Figuras ya mencionadas:

Tabla 8.3: Resultados de las métricas en la generación de señales MEA con ESN.

Puntos de Calentamiento	Puntos generados	NRMSE	R^2
100	100	0.081200	0.870300
175	225	0.060200	0.883250
200	500	0.047175	0.934100

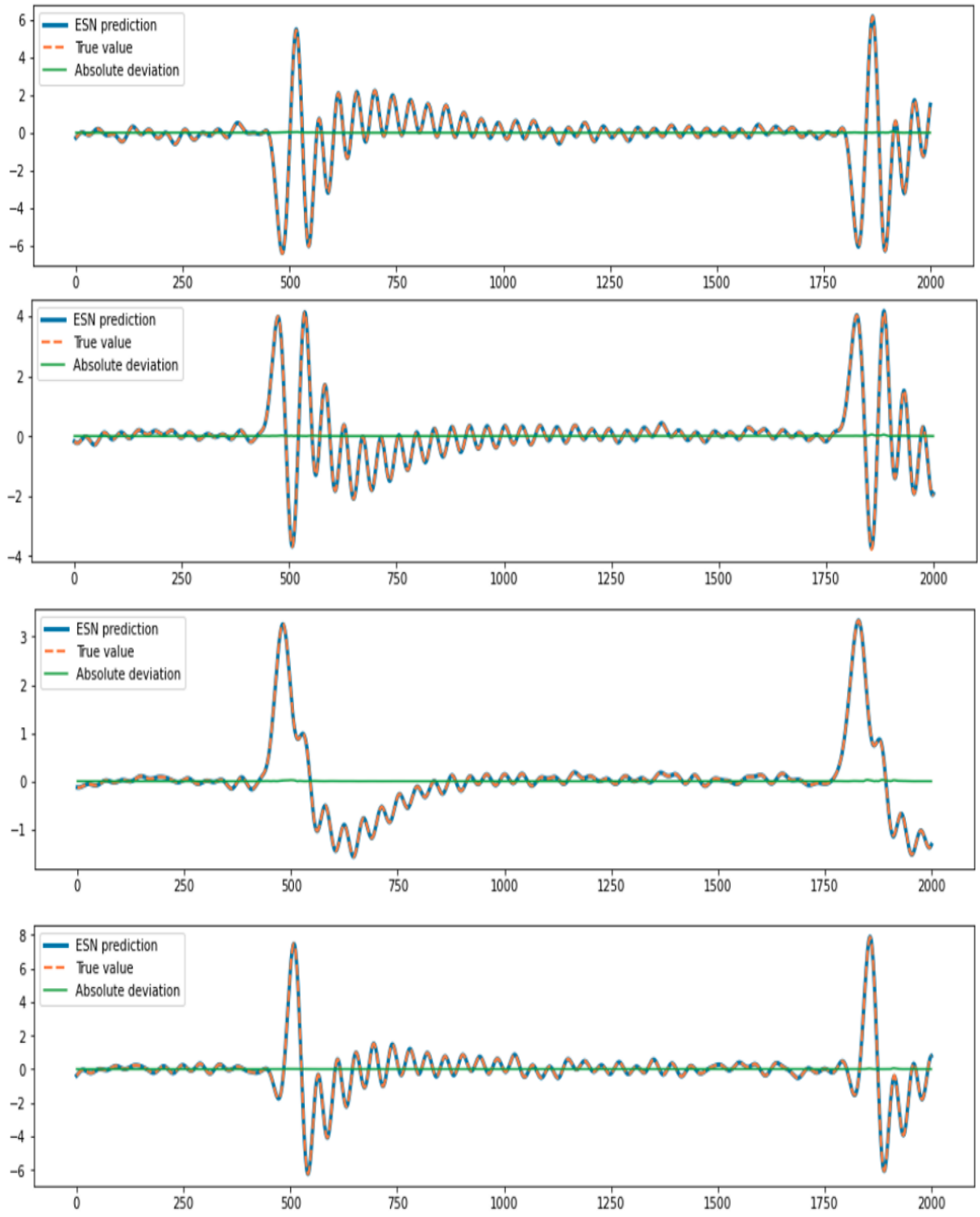

Figura 8.5: Predicción con desfase de 1 punto. Elaboración propia.

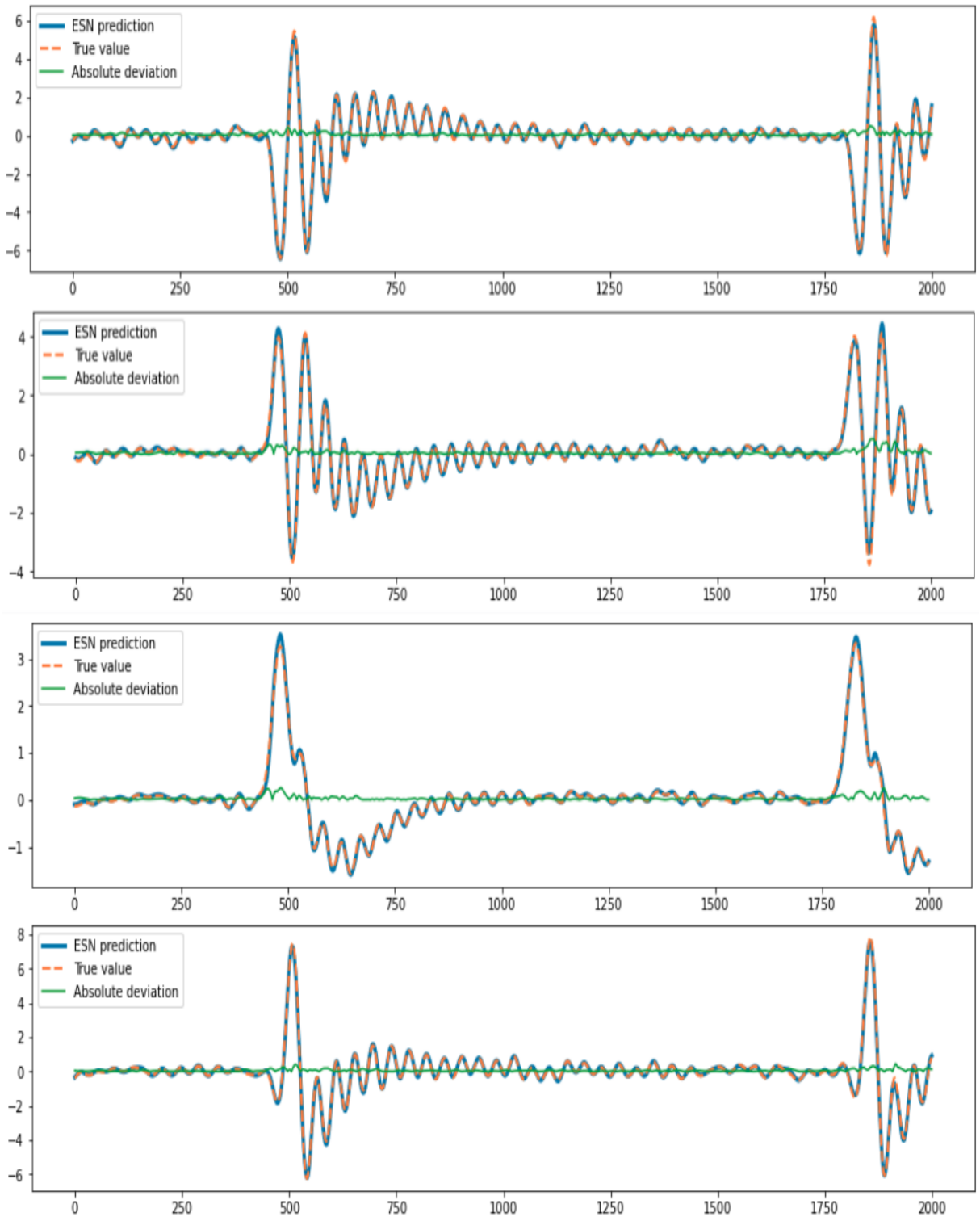

Figura 8.6: Predicción con desfase de 10 puntos. Elaboracion propia.

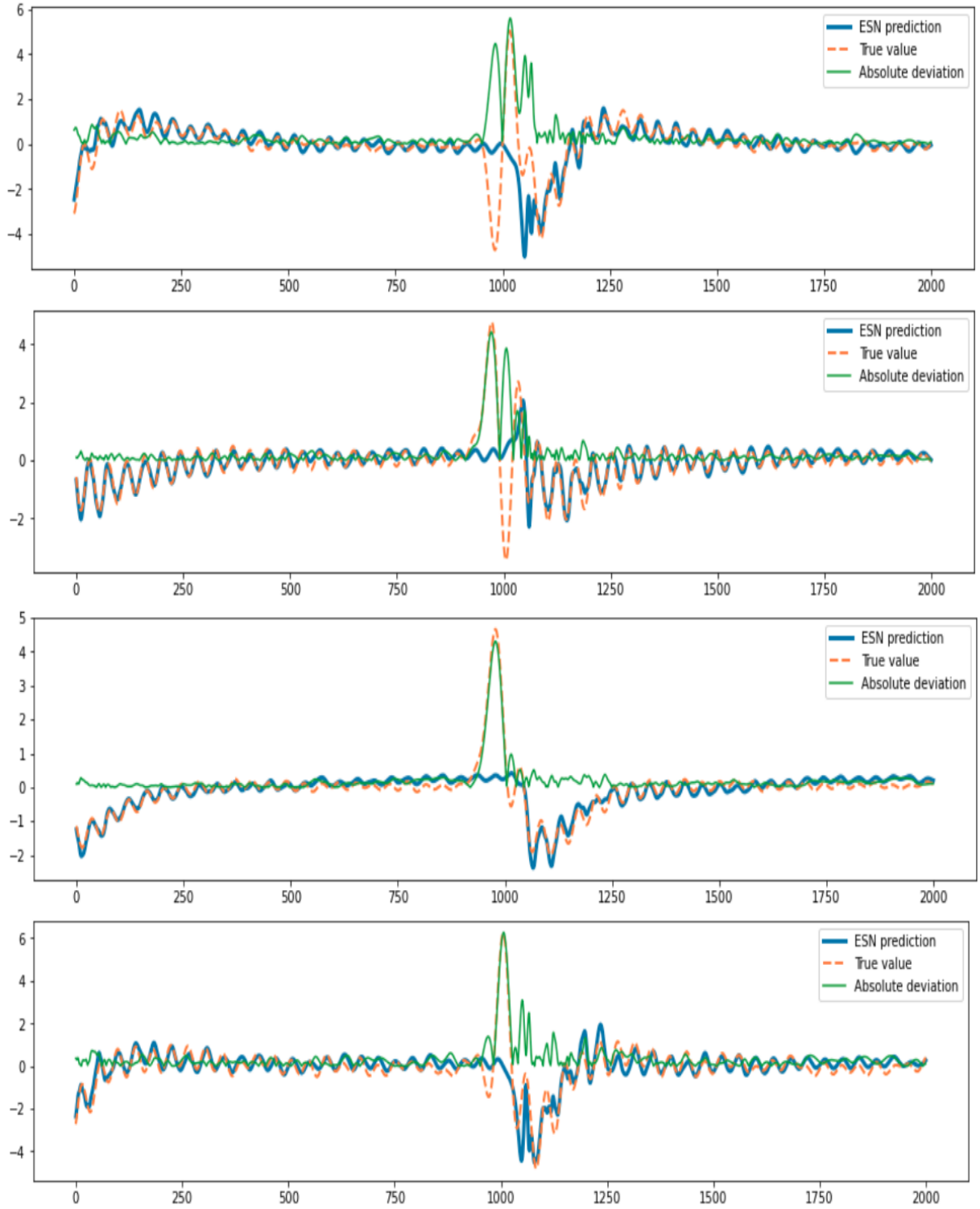

Figura 8.7: Predicción con desfase de 100 puntos. Elaboración propia.

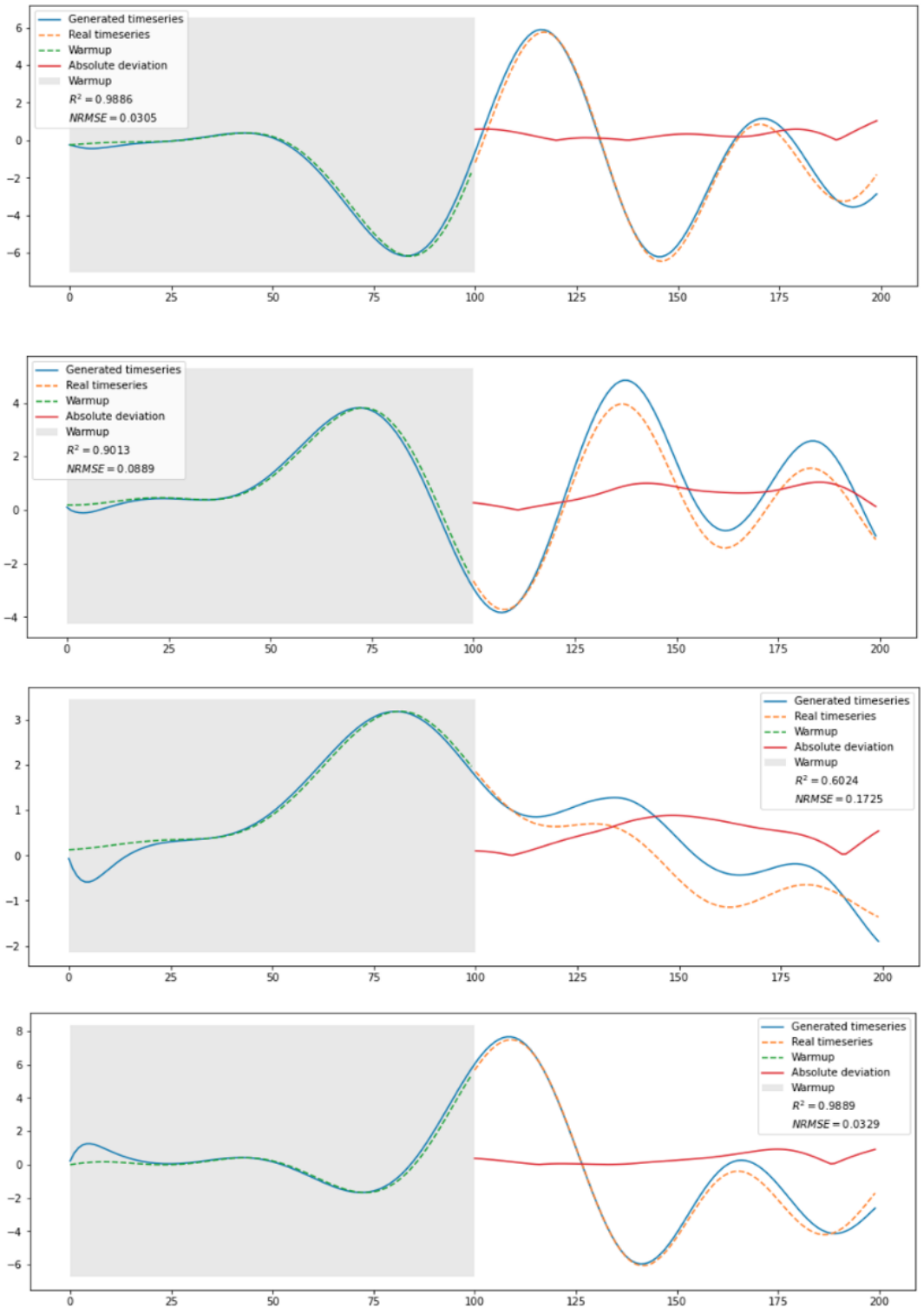

Figura 8.8: Generación de una señal MEA durante un ataque epiléptico. Elaboración propia.

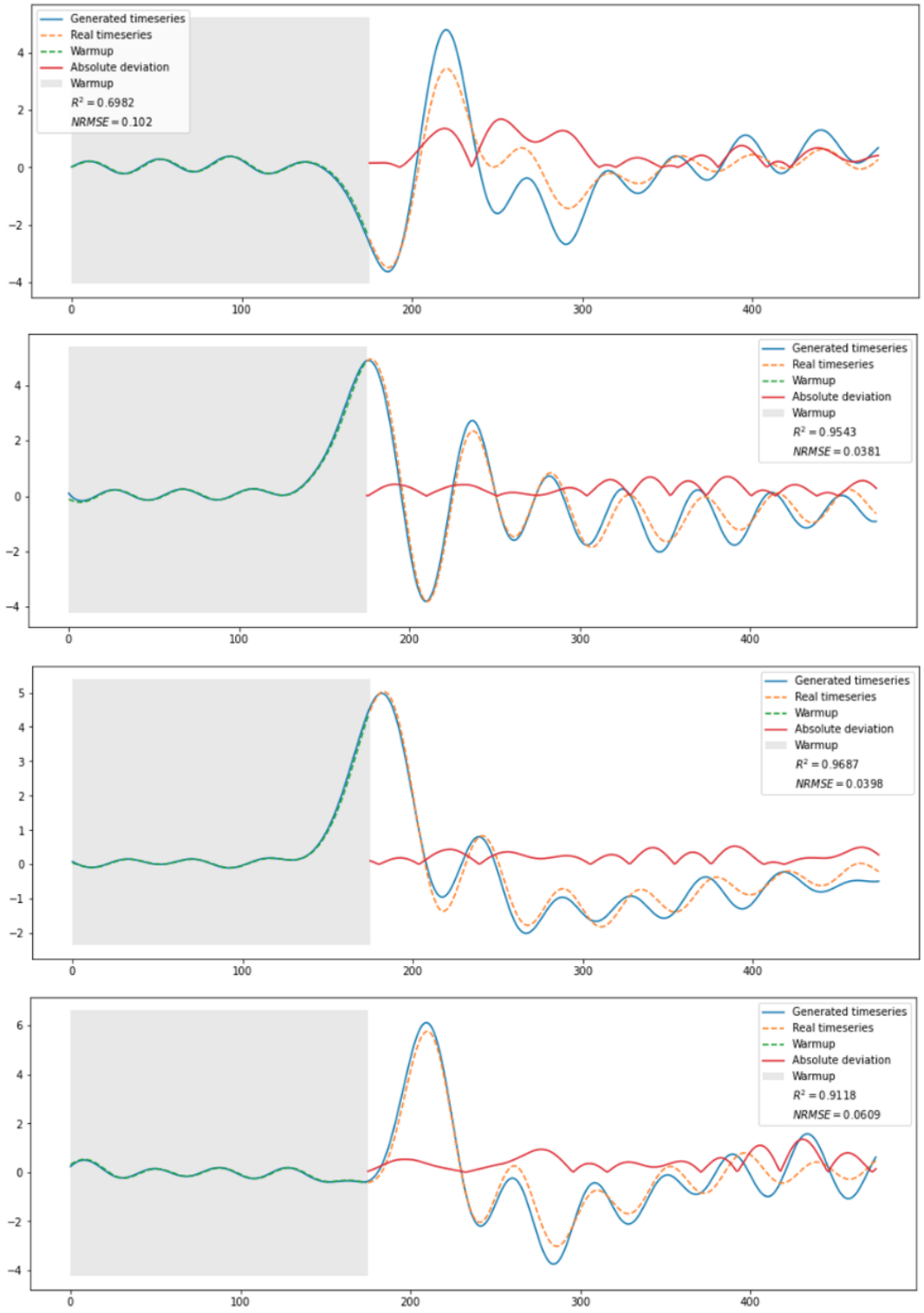

Figura 8.9: Generación de una señal MEA antes un ataque epiléptico. Elaboración propia.

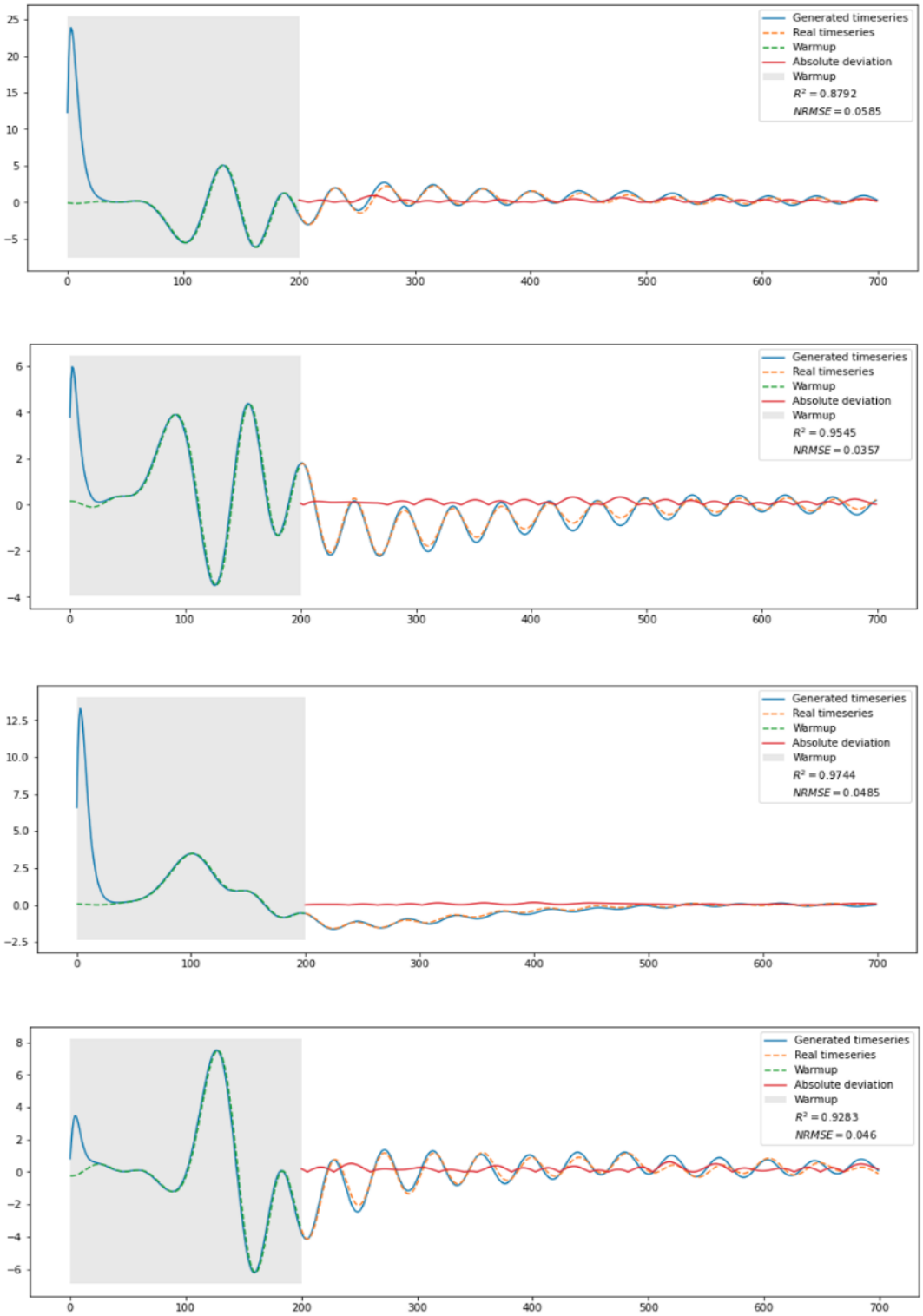

Figura 8.10: Generación de una señal MEA después de un ataque epiléptico. Elaboración propia.

8.1.3. Discusión de los resultados obtenidos

Para poder analizar los datos profundamente, es importante tener en cuenta algunos conceptos teóricos de la teoría del caos [116].

Como las señales MEA se pueden interpretar como un sistema con dinámicas caóticas, pequeñas variaciones en las condiciones iniciales del sistema pueden implicar grandes cambios en el futuro, por lo que a la hora de predecir a largo plazo, las predicciones se vuelven menos precisas [117].

Esto se puede ver reflejado en la Tabla 8.2 y en las Figuras 8.5, 8.6 y 8.7, ya que a medida que las observaciones se desplazan mucho en el tiempo, las predicciones empeoran considerablemente.

En la generación también se puede ver reflejado, pues cuanto más cercano se está a la zona de calentamiento (que está formada por una pequeña muestra de la señal que se proporciona al modelo para que genere los siguientes puntos), más precisas son las predicciones ya que acumulan menor error. Por el contrario, cuánto más se aleja, si una predicción se hace con algo de error en comparación a la real, este error se va acumulando, llegando a una mayor divergencia de los resultados.

Esto sucede en diversas imágenes de las Figuras 8.8, 8.9 y 8.10, y hay veces en las que el modelo consigue ajustarse mejor y otras peor.

En la generación a veces pasa otro efecto, y es que, al ser tan sensible en las condiciones iniciales, a veces el modelo falla incluso en las mismas generaciones, con los mismos parámetros. Esto se debe a que al dejar al modelo generar las señales libremente debido a su configuración, a veces no se ajusta adecuadamente, llegando a divergir en muchas ocasiones.

A pesar de que el segundo mejor valor de NRMSE se corresponde a la configuración de hiperparámetros de N = 500 neuronas, al variar tanto el valor de algunos hiperparámetros como el tamaño del reservorio, se consiguen resultados no deseados, como se puede observar en la Figura 8.11. Por ello, el modelo es dependiente de un buen ajuste de hiperparámetros.

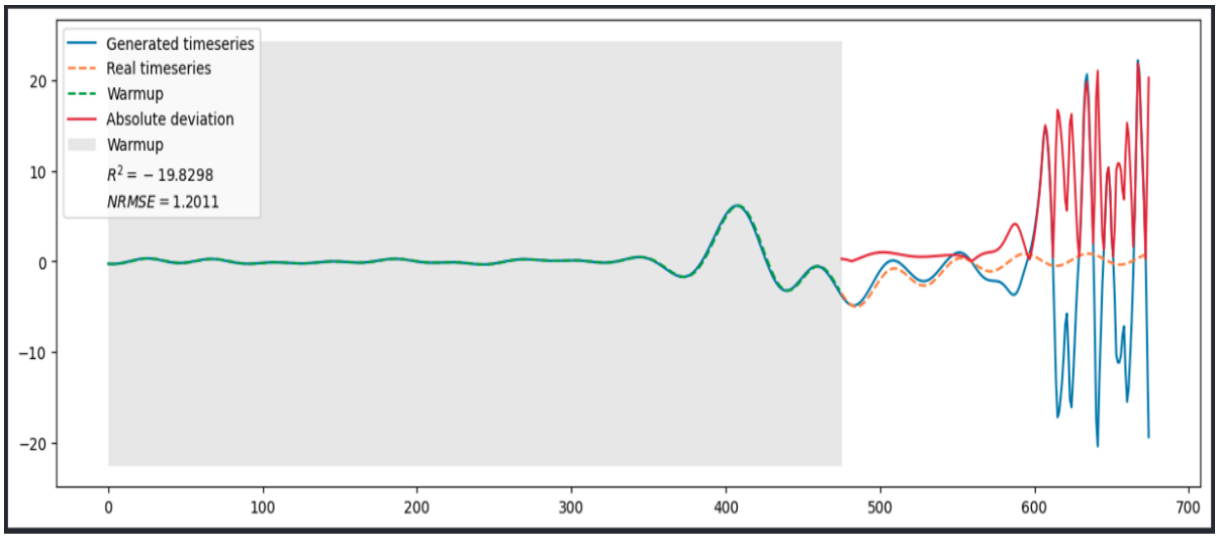

Figura 8.11: Generación errónea de una señal MEA en el electrodo 27 con el modelo ESN mal optimizado. Elaboración propia.

8.2. Resultados LSM

Siguiendo con la planificación propuesta, el siguiente modelo del que se van a presentar resultados es la LSM cuantificada, en la cual se han utilizado los datos normalizados.

Como se ha explicado previamente en el apartado de implementación de LSM del Capítulo 6, este modelo es una adaptación de lo aportado en [94] para probar si se podía conseguir representar las señales MEA disponibles en modelos con cuantizaciones pequeñas de 3, 6, 8 y 16.

8.2.1. Resultados de la búsqueda de los hiperparámetros

Utilizando Optuna, se ha hecho una búsqueda de los hiperparámetros representativos para este proyecto teniendo en cuenta la información extraída del artículo [63].

Por consecuencia, los hiperparámetros que se han ajustado para obtener mejores resultados han sido el tamaño de la capa líquida (*liquid_size*), el tamaño de las señales de entrada (*ts_size*) y el tamaño de las señales generadas a la salida del modelo (*gen_size*).

Como únicamente se han optimizado estos hiperparámetros, se ha intentado buscar el mejor resultado proveniente del menor número de neuronas de la capa líquida, obteniendo los siguientes resultados de la búsqueda en Optuna.

Primero, se ha buscado entre las siguientes muestras:

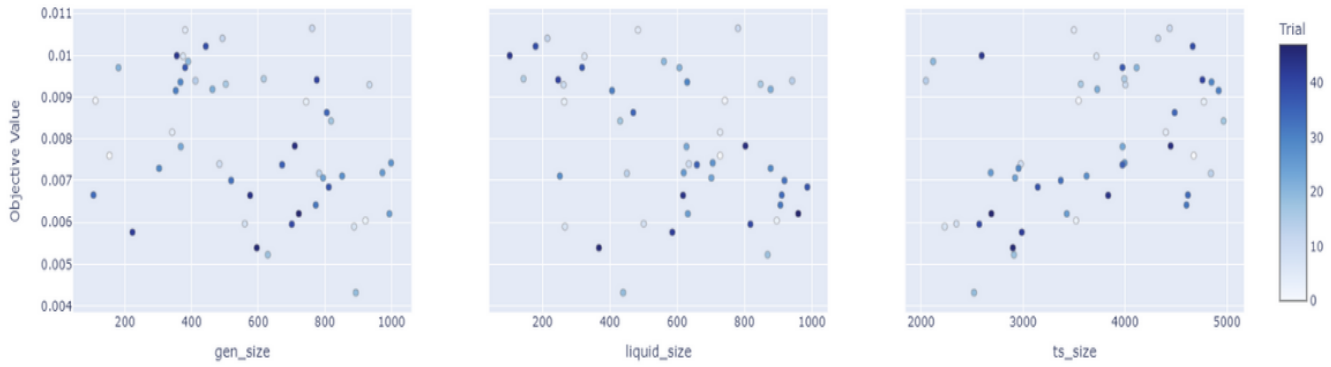

Figura 8.12: Gráfico de puntos en Optuna que representa las muestras en función a su valor y el valor que adquiere la función de pérdida MSE. Elaboración propia.

Los rangos establecidos para cada hiperparámetro son: entre 100 y 1000 para *gen_size* y *liquid_size*; y entre 2000 y 5000 para *ts_size*.

Como se puede ver en la Figura 8.13, entre todos los hiperparámetros buscados, los que consiguen reducir la función de pérdida (que en este caso se ha optado por error cuadrático medio (MSE)) con mejores resultados se muestran en la Tabla 8.4.

Tabla 8.4: Mejores resultados de la búsqueda de hiperparámetros en LSM.

liquid_size	gen_size	ts_size	MSE
440	2520	893	0.00431662

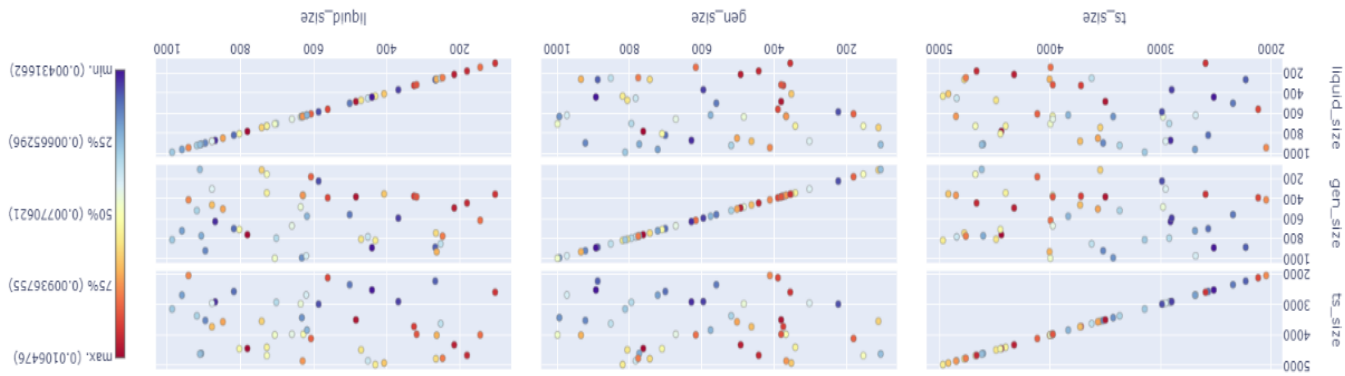

Figura 8.13: Gráfico de la distribución de las muestras entre los diferentes hiperparámetros con colores que representan un orden en función del valor de la función de pérdida MSE. Elaboración propia.

Con los hiperparámetros de la Tabla 8.4 se ha seguido con el entrenamiento de la LSM.

Para simplificar las gráficas de los resultados, se ha decidido realizar las ejecuciones con las señales del electrodo 27 exclusivamente (Figura 8.14), pero se puede aplicar para el resto de los electrodos.

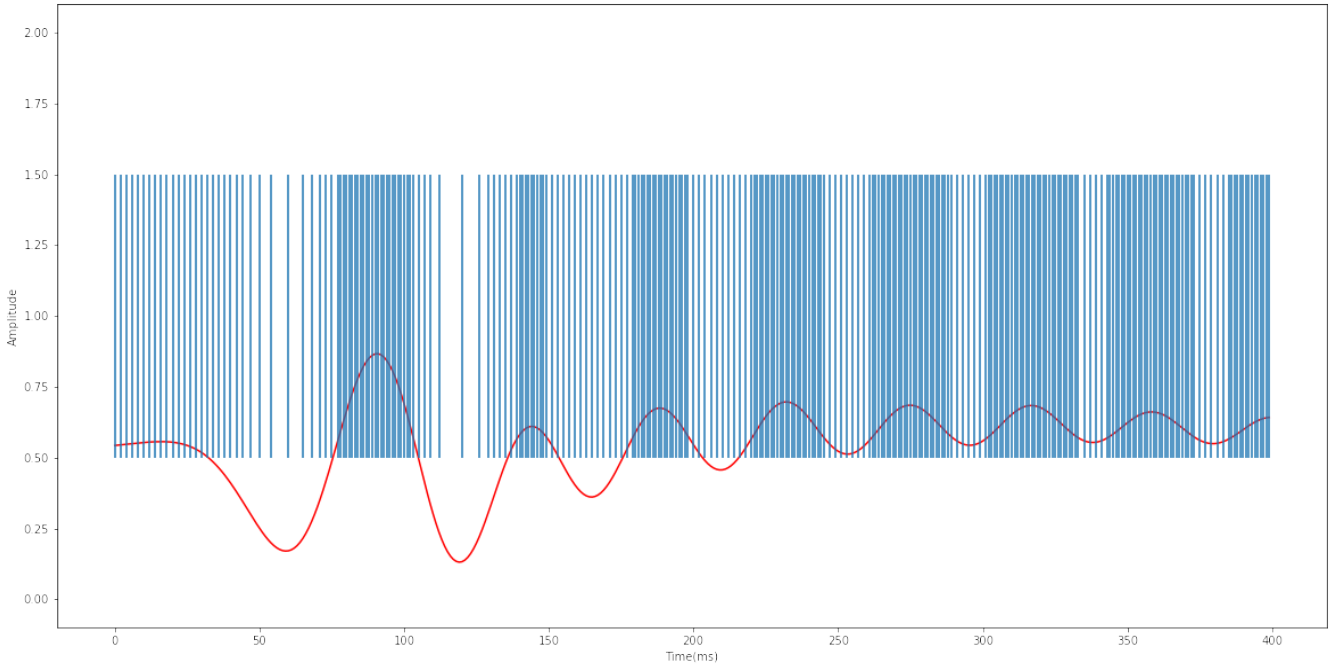

Figura 8.14: Muestra de los datos de la señal MEA del electrodo 27 con los *spikes* representados. Elaboración propia, basada en [94].

Como se puede visualizar, se representa una muestra de la señal indicada con la línea roja, junto a la activación de los *spikes* que va a ser enviada como entrada al modelo de la LSM señalada con líneas verticales de color azul.

Cuando existe una elevación en la señal, más neuronas de este estilo se activan, y por tanto, más impulsos eléctricos se mandan al modelo, haciendo que las líneas verticales de color azul estén más concentradas. Cuando la señal desciende, se agrupan menos neuronas y se producen menos impulsos eléctricos que sirvan como entrada al modelo, haciendo que las líneas de color azul estén más separadas.

Durante el entrenamiento, se han obtenido los siguientes resultados para las representaciones de 3,4,8 y 16 bits, representados en la Figura 8.15.

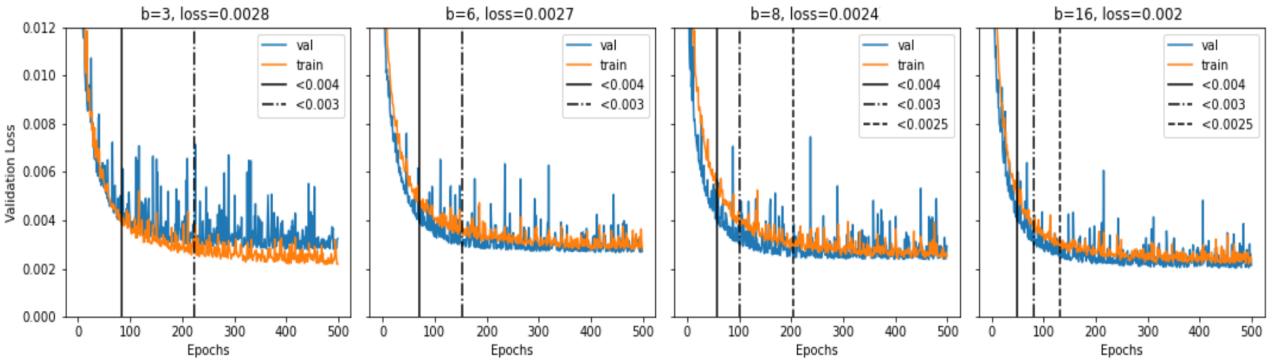

Figura 8.15: Resultado del entrenamiento en las diferentes representaciones de bits en el modelo LSM. Elaboración propia, basada en [94].

8.2.2. Resultados del modelo

Despúes del entrenamiento, se han obtenido las siguientes representaciones de las señales. En este caso no se ha tratado la tarea de predicción, pues lo que se buscaba con este modelo era adaptarlo a la tarea de generación usando la cuantificación del modelo.

Generación de las señales MEA

El conjunto de imágenes mostrado en la Figura 8.16 muestra diferente información respecto a la salida del modelo para los diferentes bits.

Empezando por arriba, observamos una representación de la energía y la actividad del modelo. Esto puede indicar cómo funcionan los impulsos eléctricos dentro de la capa líquida. Esto se ha tenido en cuenta, ya que a menor actividad, implica menor consumo de energía del modelo en caso de una implementación física.

Las Ecuaciones de la actividad y de la energía vienen representadas por 8.3 y 8.4 respectivamente [63]:

$$A^M(t) = \frac{1}{|N|} \cdot s(t) \cdot 1 \tag{8.3}$$

$$E^M(t) = \frac{1}{\delta_b |N|} \cdot v(t) \cdot 1 \tag{8.4}$$

Donde $s(t)$ es la suma total de los impulsos eléctricos en el instante t y $v(t)$ es la potencia de la membrana de todas las neuronas en el instante t.

En el siguiente conjunto de imágenes se puede ver qué conexiones han sido excitadoras, es decir, qué neurona ha estimulado a otra neurona [63].

Posteriormente, se puede ver la decodificación de los *spikes* de 50 neuronas, viendo de esta forma el flujo de impulsos eléctricos dentro del modelo.

Por último, se puede ver la generación de la señal devuelta por el modelo.

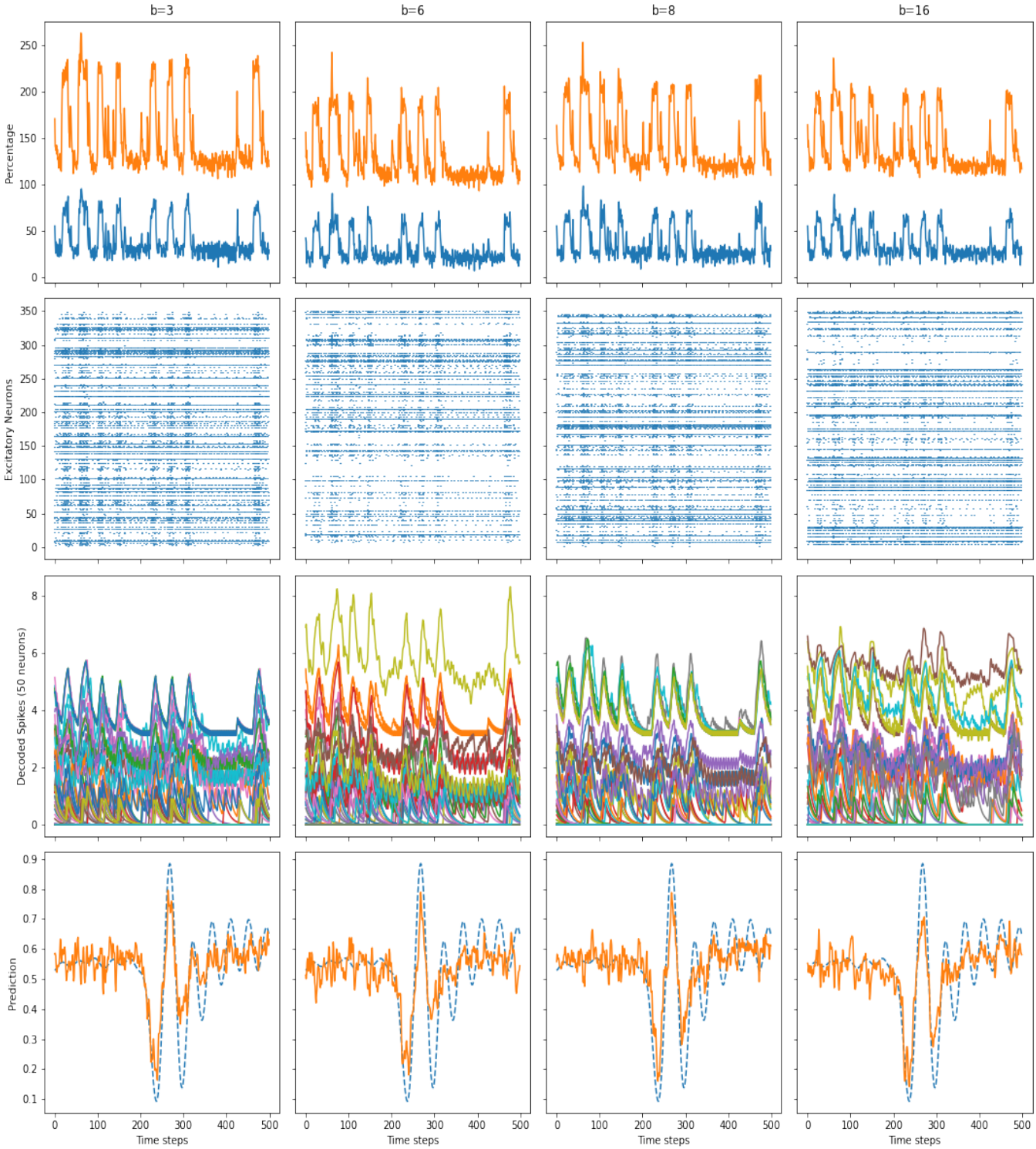

Figura 8.16: Resultados de la predicción en las diferentes representaciones de bits en el modelo LSM. Elaboración propia, basada en [94].

En la Figura 8.17 se muestran las señales generadas por el modelo mediante la línea verde. Para suavizar los picos y entender mejor la señal generada, se ha aplicado un suavizado y se ha representado con la línea roja.

Después de las gráficas, se va a representar en la Tabla 8.5 las métricas obtenidas en la representación de la señal predicha y la suavizada para cada representación de bits.

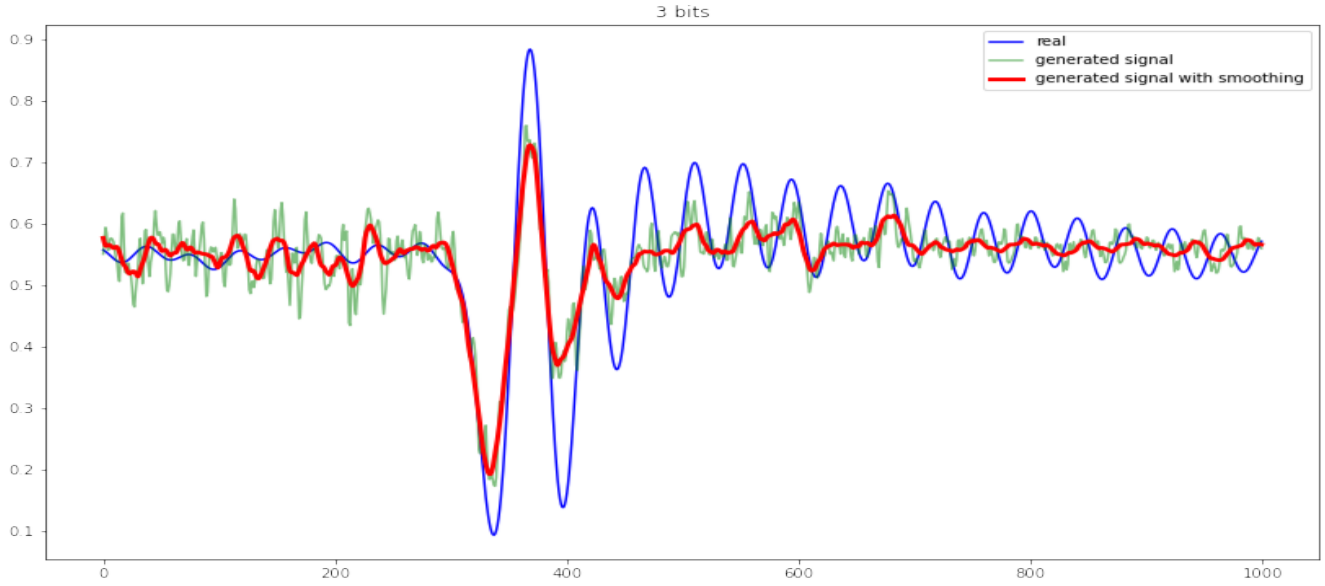

Figura 8.17: Resultados de la generación con suavizado en el modelo LSM con 3 bits. Elaboración propia.

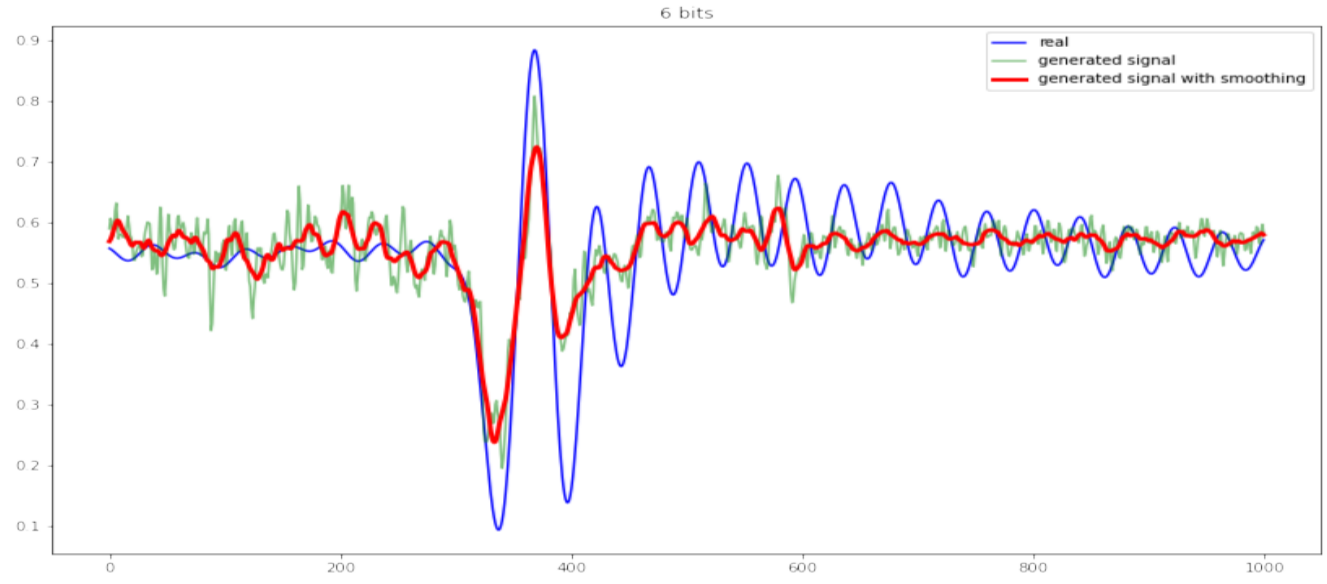

Figura 8.18: Resultados de la generación con suavizado en el modelo LSM con 6 bits. Elaboración propia.

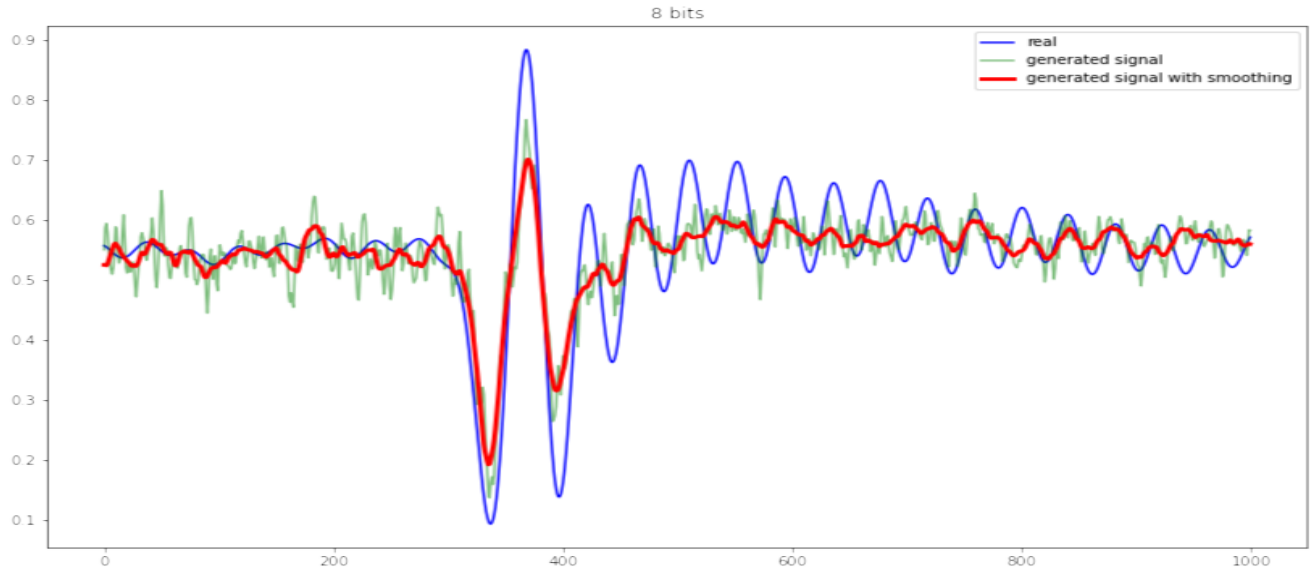

Figura 8.19: Resultados de la generación con suavizado en el modelo LSM con 8 bits. Elaboración propia.

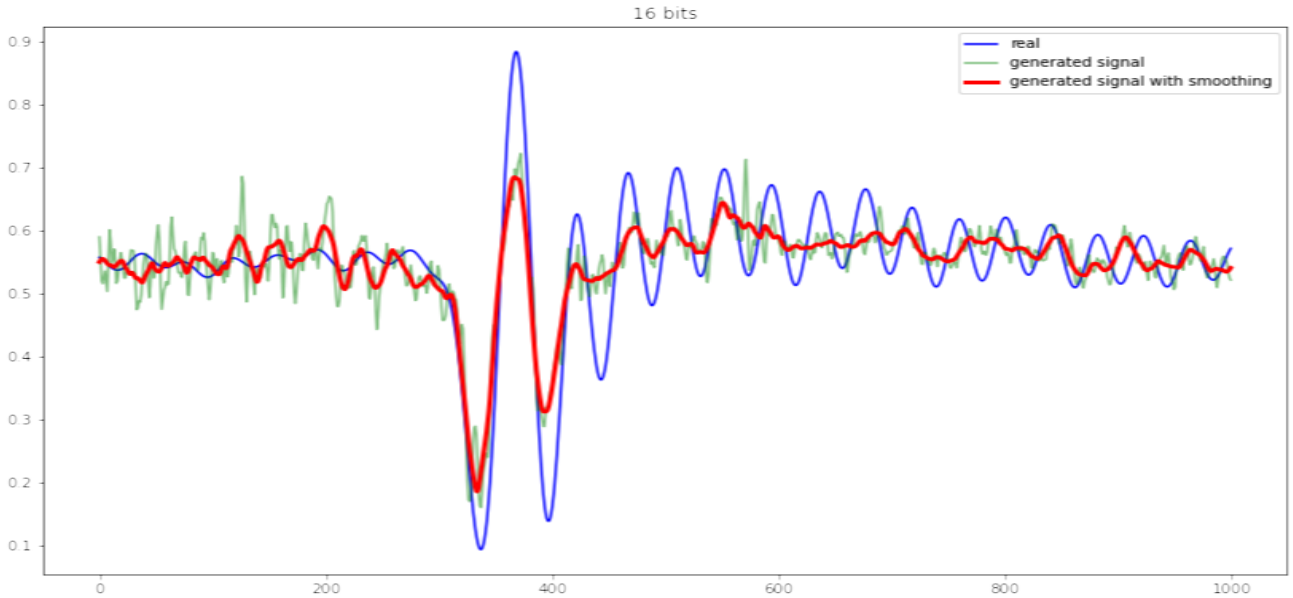

Figura 8.20: Resultados de la generación con suavizado en el modelo LSM con 16 bits. Elaboración propia.

Tabla 8.5: Resultados de las métricas en la generación de señales MEA con LSM cuantizada para 3, 6, 8 y 16 bits.

	MSE	RMSE	NRMSE	R^2
Normal (3 bits)	0.003310	0.057531	0.234089	0.277848
Suavizada (3 bits)	0.002562	0.050615	0.205949	0.277848
Normal (6 bits)	0.003605	0.060046	0.244319	0.213350
Suavizada (6 bits)	0.002882	0.053681	0.218424	0.213350
Normal (8 bits)	0.003259	0.057087	0.232285	0.288939
Suavizada (8 bits)	0.002440	0.049392	0.200970	0.288939
Normal (16 bits)	0.002627	0.051251	0.208535	0.426910
Suavizada (16 bits)	0.002012	0.044853	0.182504	0.426910

8.2.3. Discusión de los resultados obtenidos

Teniendo en cuenta todos los resultados debatidos en la generación, podemos deducir el comportamiento de este modelo.

Si observamos las métricas, todos los tipos de errores cuadráticos medios obtienen buenos resultados. Sin embargo, si contemplamos el R^2, es muy bajo, obteniendo por tanto que la señal observada no es similar a la generada ni a la suavizada (ya que ambas señales tienen el mismo valor en esta métrica).

Aún así, los resultados se pueden considerar buenos, puesto que el modelo es capaz de detectar los picos pronunciados en las señales MEA del electrodo 27.

Por ello, aunque no sea capaz de representar con exactitud la señal, es capaz de intuir en qué momento se van a producir las alteraciones en la señal, logrando así resultados válidos para la investigación.

Además, los resultados obtenidos en las diferentes representaciones de bits son similares, permitiendo de esta forma representar estas señales en el modelo cuantizado con los bits que se requieran sin alterar la información relevante de los datos.

8.3. Resultados NGRC

El siguiente modelo del que se van a discutir los resultados es de NGRC.

8.3.1. Resultados de la búsqueda de hiperparámetros

Los hiperparámetros elegidos para optimizar fueron el valor de la función Ridge (*ridge*), las dimensiones de la parte lineal (k) y el máximo orden de las funciones de la parte no lineal (*nolineal-order*).

El hiperparámetro *ridge* se ha buscado entre los valores $1e^{-9}$ y $1e^{-2}$, mientras que k y *nolinear-order* se han buscado entre 2 y 5.

Teniendo en cuenta que la función de pérdida a minimizar vuelve a ser NMRSE, los mejores resultados obtenidos en esta búsqueda se encuentran en la Tabla 8.6.

Tabla 8.6: Resultados de la búsqueda de hiperparámetros en NGRC con menor pérdida. Elaboración propia.

k	**ridge**	**nolinear-order**	**loss**
4	0.000001	2	0.172768
4	**0.002517**	**4**	**0.171178**
2	0.000009	4	0.171350
2	0.000004	4	0.173145

8.3.2. Resultados del modelo

Tras realizar diferentes pruebas con los diferentes datasets, el que mejor se ajusta es el dataset que no incluye ni normalización ni estandarización.

Teniendo en cuenta el mejor conjunto de hiperparámetros y que en el artículo [62] no se incluye la parte de predicción, se han obtenido los siguientes resultados.

Resultados de la generación

Las figuras 8.21, 8.22, 8.23, 8.24 muestran las señales MEA generadas por el modelo NGRC con el conjunto de hiperparámetros propuesto.

El resumen de las evaluaciones de las métricas en la generación de las señales MEA se puede ver a continuación, en la Tabla 8.7. En esta tabla, solo se van a mostrar las medidas de los cuatro electrodos que aparecen en las Figuras ya mencionadas:

Tabla 8.7: Resultados de las métricas en la generación de señales MEA con NGRC.

Electrodo	NRMSE	R^2
27	0.162485	-1.219535
65	0.186350	-1.505742
75	0.168164	-1.528703
77	0.155121	-1.001604

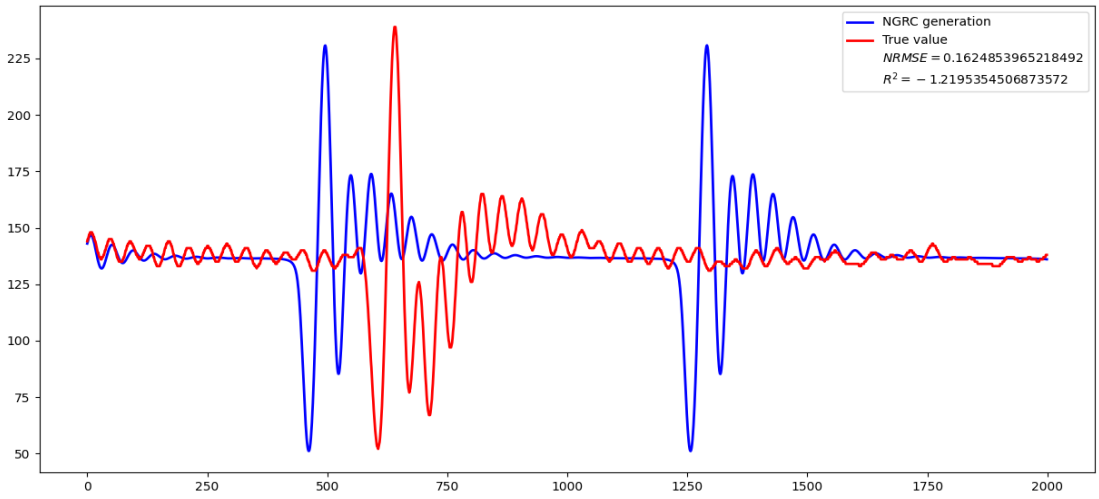

Figura 8.21: Generación de una señal MEA en el electrodo 27 mediante el modelo NGRC . Elaboración propia.

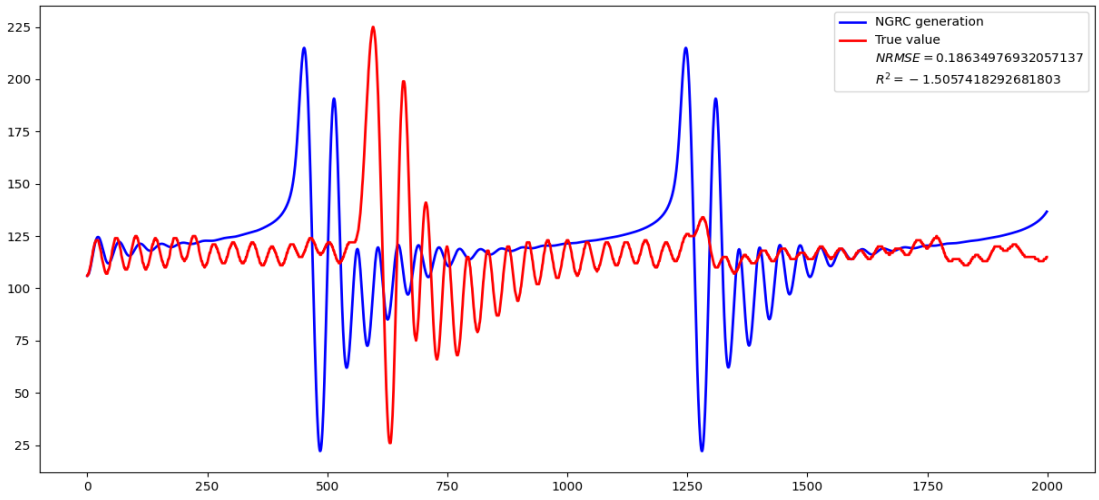

Figura 8.22: Generación de una señal MEA en el electrodo 65 mediante el modelo NGRC . Elaboración propia.

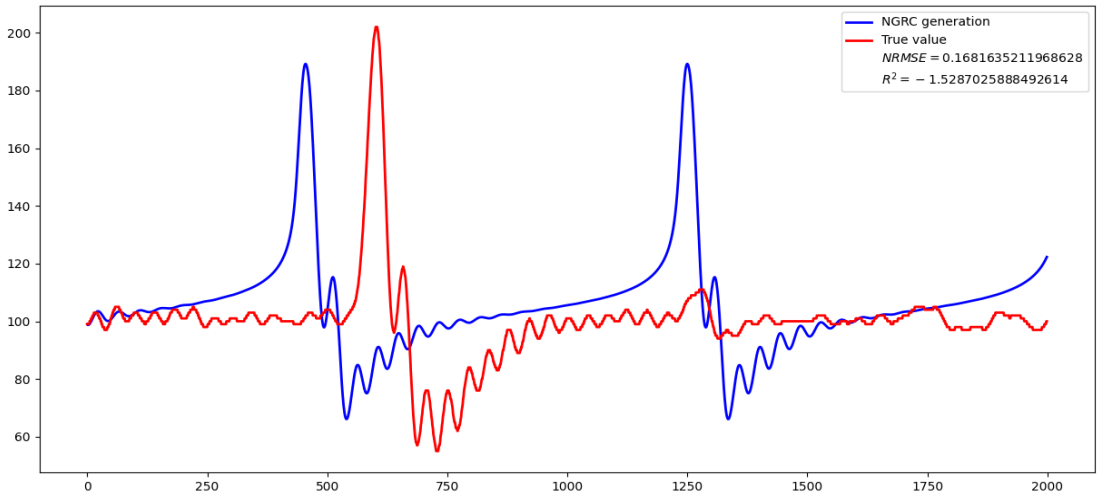

Figura 8.23: Generación de una señal MEA en el electrodo 75 mediante el modelo NGRC . Elaboración propia.

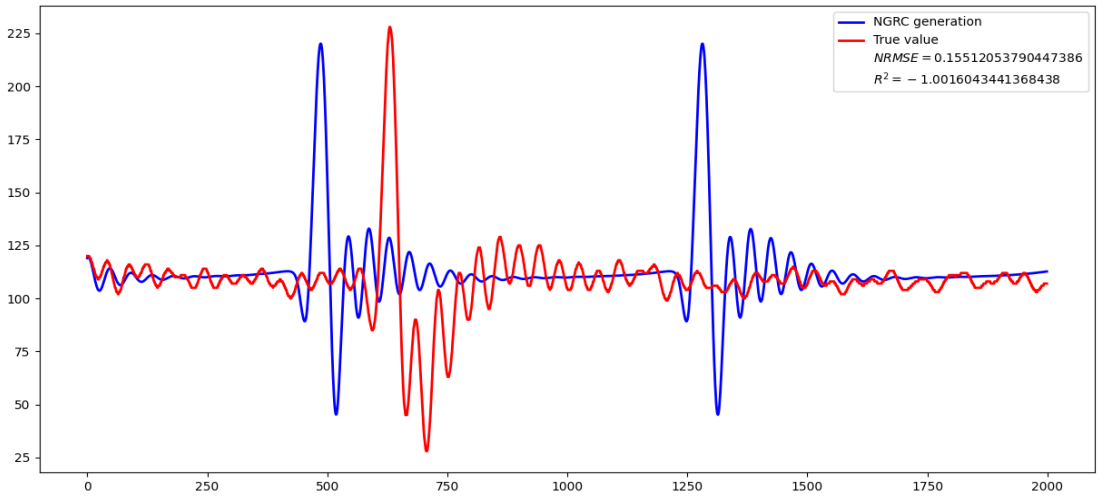

Figura 8.24: Generación de una señal MEA en el electrodo 77 mediante el modelo NGRC . Elaboración propia.

8.3.3. Discusión de los resultados obtenidos

A pesar de que el valor del NRMSE es bajo, lo cual sería indicador de que el modelo haría buenas predicciones, el valor de R^2 indica que el modelo no se ajusta bien a los datos. Esto se puede ver reflejado en la tabla 8.7.

En las gráficas podemos observar cómo el modelo recuerda las características relevantes de la señal justo antes de que suceda la crisis epiléptica, pero que haya una periodicidad en la amplitud de la onda de la señal MEA, repitiendo de esta forma la onda característica de la crisis epiléptica entorno al valor 1250 en todas las gráficas, indica que simplemente el modelo no es capaz de adaptarse correctamente a los datos.

Sería adecuado revisar el modelo, modificar los hiperparámetros y realizar más búsquedas, o incluso cambiar la función de pérdida a optimizar, pero debido a las limitaciones de tiempo de este TFG, se deja como recomendación a posteriores investigaciones.

8.4. Resultados LSTM

Los resultados obtenidos para la predicción y la generación en el modelo de la LSTM se discutirán a continuación.

8.4.1. Resultados de la búsqueda de hiperparámetros

En este modelo, debido a limitaciones de tiempo, no se ha hecho una búsqueda exhaustiva de los hiperparámetros óptimos, por lo que este apartado se deja para líneas futuras de trabajo.

Sin embargo, se hicieron algunos experimentos, en los cuales se buscó optimizar el número de neuronas de la LSTM y de la capa densa, buscando en un rango de 0 a 100 en ambas, obteniendo lo siguiente.

La función a minimizar, en este caso, fue la MSE.

Tabla 8.8: Resultados de las métricas de la búsquda de hiperparámetros en el modelo LSTM.

units(lstm)	units(dense)	loss
60	50	0.001000

8.4.2. Resultados del modelo

En este modelo, se han empleado los datos de la señal MEA del electrodo 27 sin normalizar ni estandarizar, haciendo la siguiente división. Todo el procedimiento es aplicable al resto.

Esta muestra se ha dividido en datos de entrenamiento, prueba y validación, como se ve en la Figura 8.25.

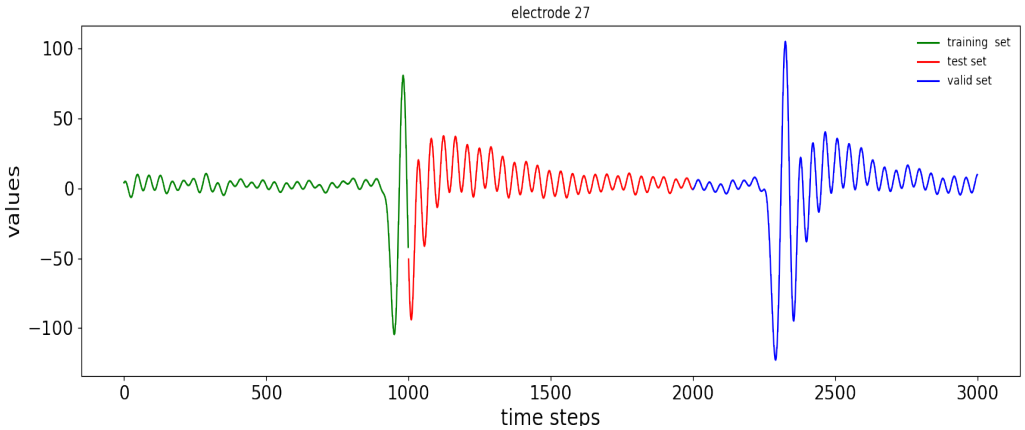

Figura 8.25: Muestra de los datos para el modelo LSTM. Elaboración propia.

Predicción de las señales MEA

En el caso de la predicción, se ha probado cómo de bien es capaz de ajustarse a los datos, y se ha obtenido lo siguiente con el conjunto de test:

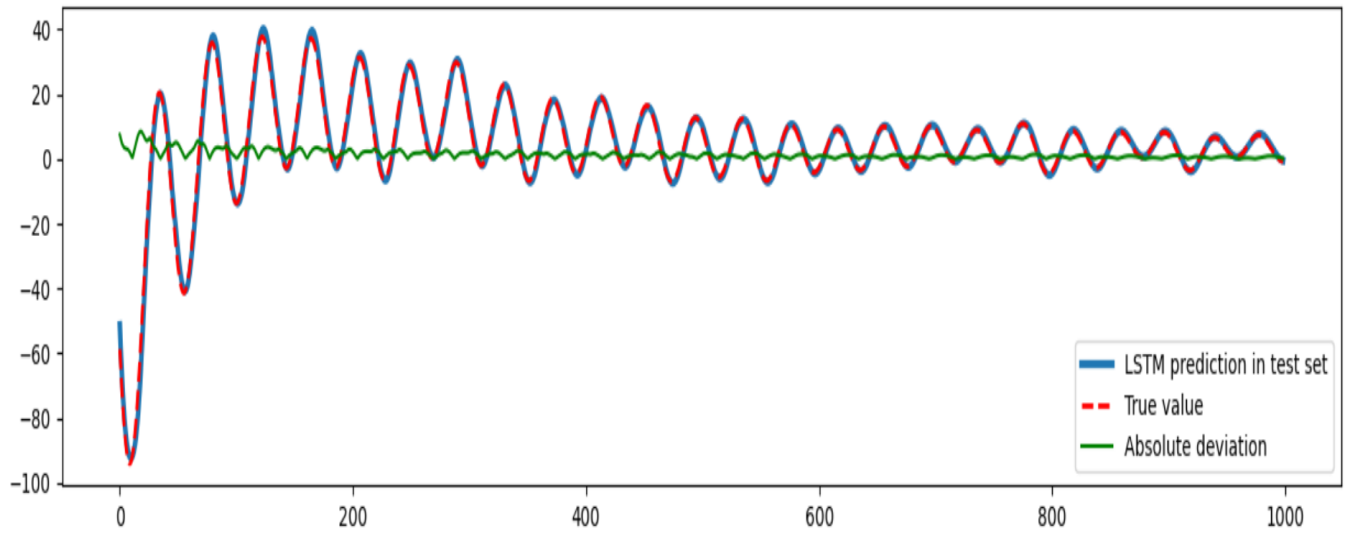

Figura 8.26: Resultado de la LSTM en el conjunto de prueba en la predicción con desfase de 10 puntos. Elaboración propia.

Y en la validación, se han conseguido los siguientes resultados para 1, 10 y 100 puntos de desfase:

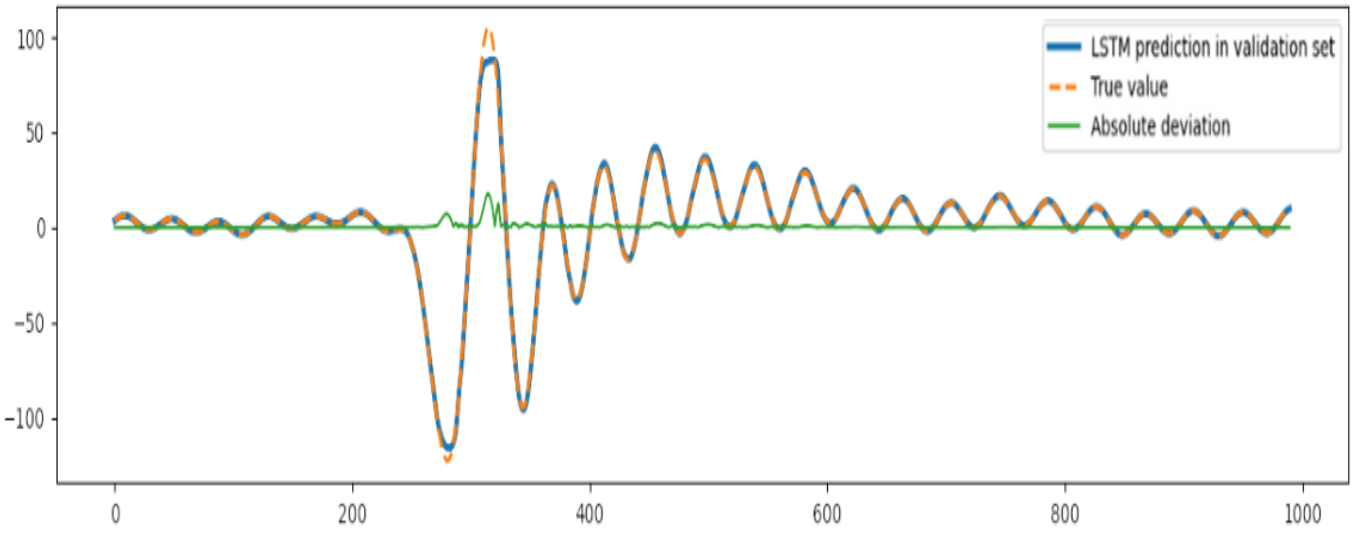

Figura 8.27: Resultado de la LSTM en el conjunto de validación con desfase de 1 puntos. Elaboración propia.

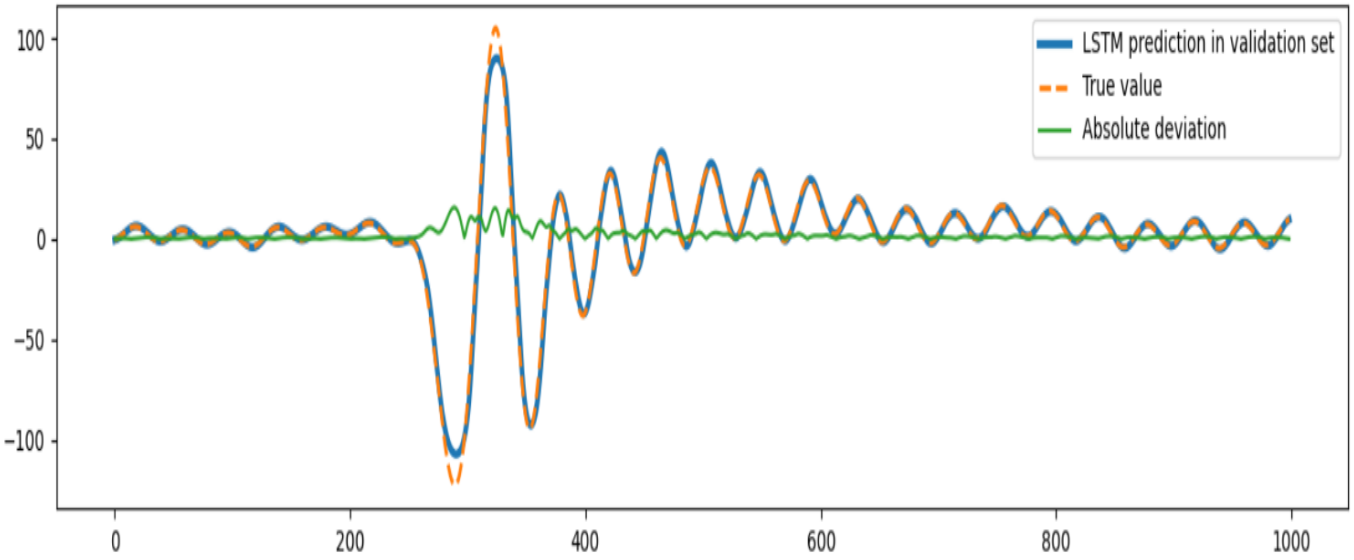

Figura 8.28: Resultado de la LSTM en el conjunto de validación con desfase de 10 puntos. Elaboración propia.

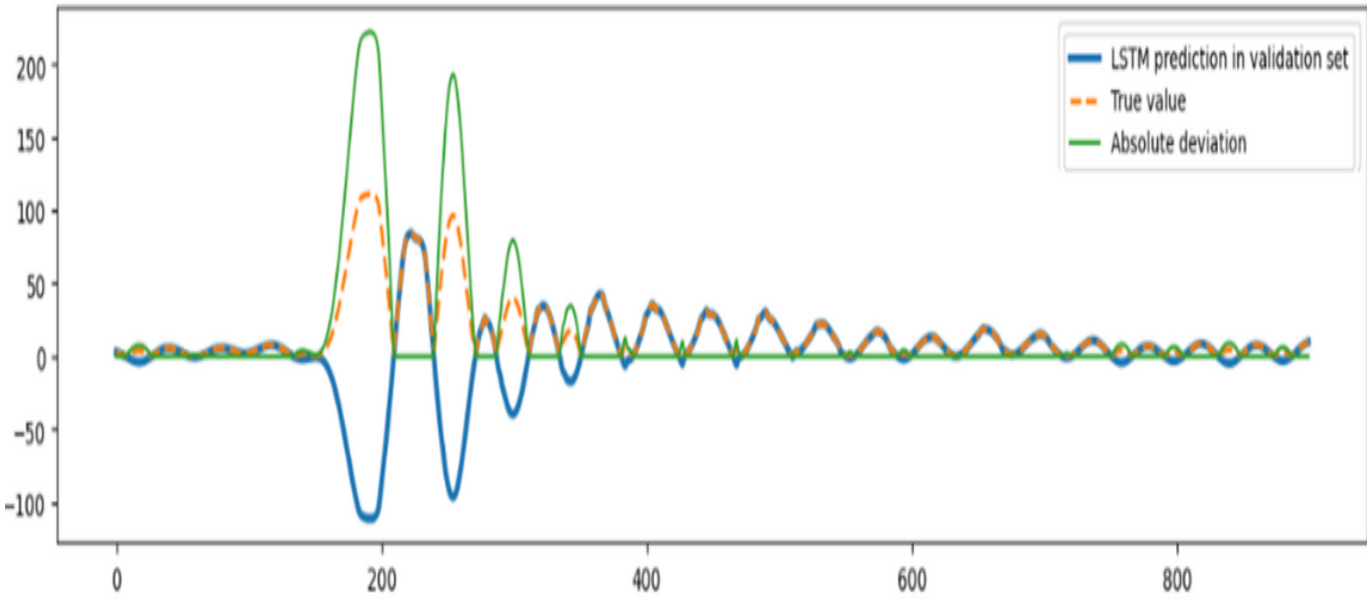

Figura 8.29: Resultado de la LSTM en el conjunto de validación con desfase de 100 puntos. Elaboración propia.

Las métricas correspondientes a estas gráficas se resumen en la siguiente tabla:

Tabla 8.9: Resultados de las métricas en la predicción de señales MEA con ESN.

Puntos de Desfase	MSE	RMSE	NRMSE	R^2
1	0.113490	0.033688	0.037693	0.984814
10	0.468954	0.216448	0.388992	0.973865
100	0.672534	0.259333	0.402023	0.288390

Predicción en Google Dev Board

Una vez observado que el modelo obtiene buenos resultados, se ha hecho que la LSTM prediga solamente el valor siguiente, sin hacer una secuencia de predicciones.

Esta predicción es la que se realizará dentro de Google Dev Board.

Como se puede observar en la Figura 8.30, se ha obtenido buenos resultados a la hora de predecir con este modelo.

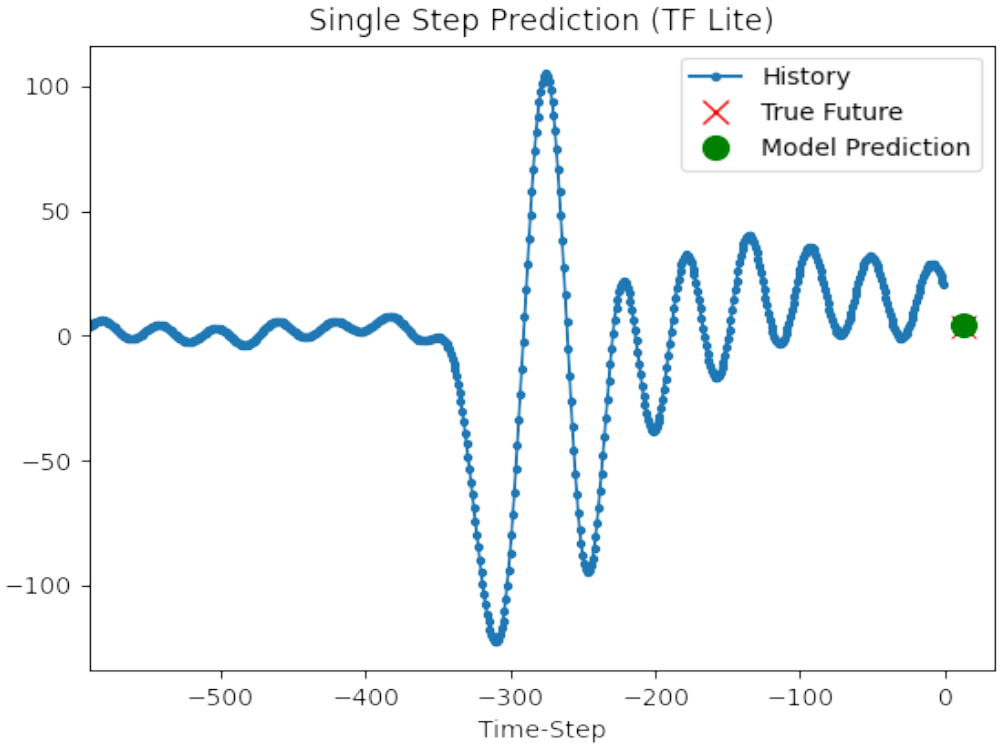

Figura 8.30: Resultado de la LSTM en predecir el punto siguiente en la versión cuantizada del modelo. Elaboración propia, basada en [111].

En la ejecución en Google Dev Board se ha obtenido lo siguiente (Figura 8.31), teniendo en cuenta que la secuencia de datos predichas en uint8 es lo que se muestra en la Figura 8.32.

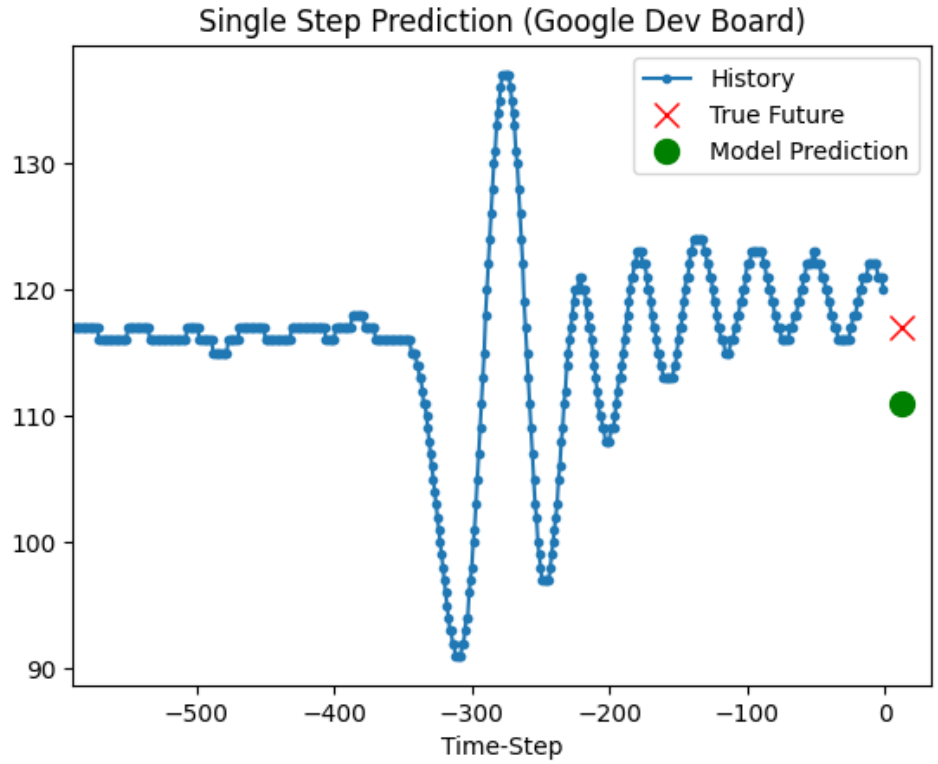

Figura 8.31: Predicción del punto siguiente mediante Google Dev Board. Elaboración propia, basada en [111].

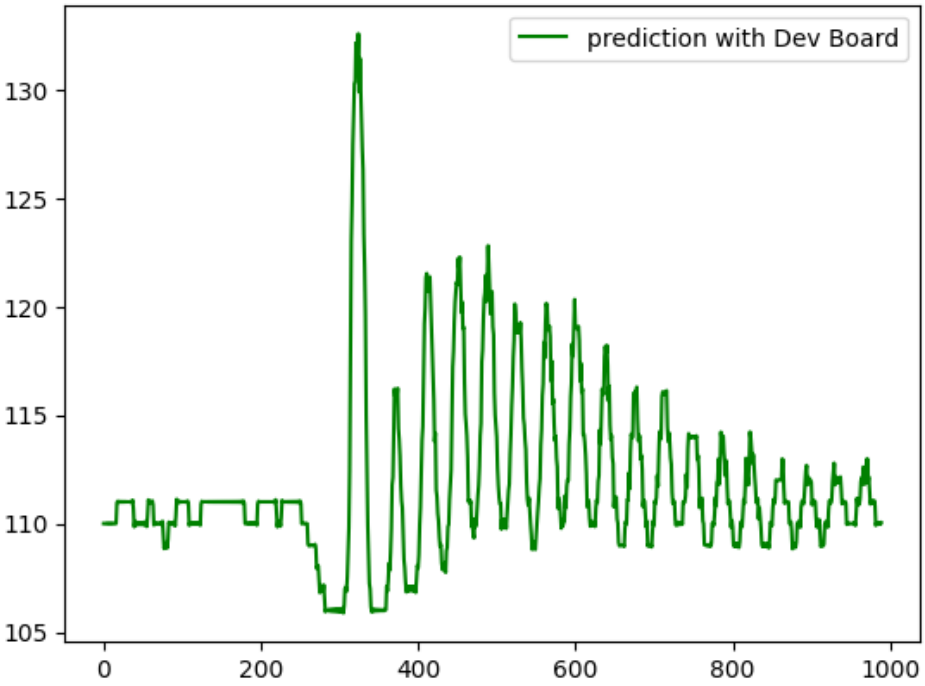

Figura 8.32: Muestra de la secuencia predicha en uint8 por Google Dev Board. Elaboración propia.

Generación en las señales MEA

En el caso de la generación, se han obtenido los siguientes resultados

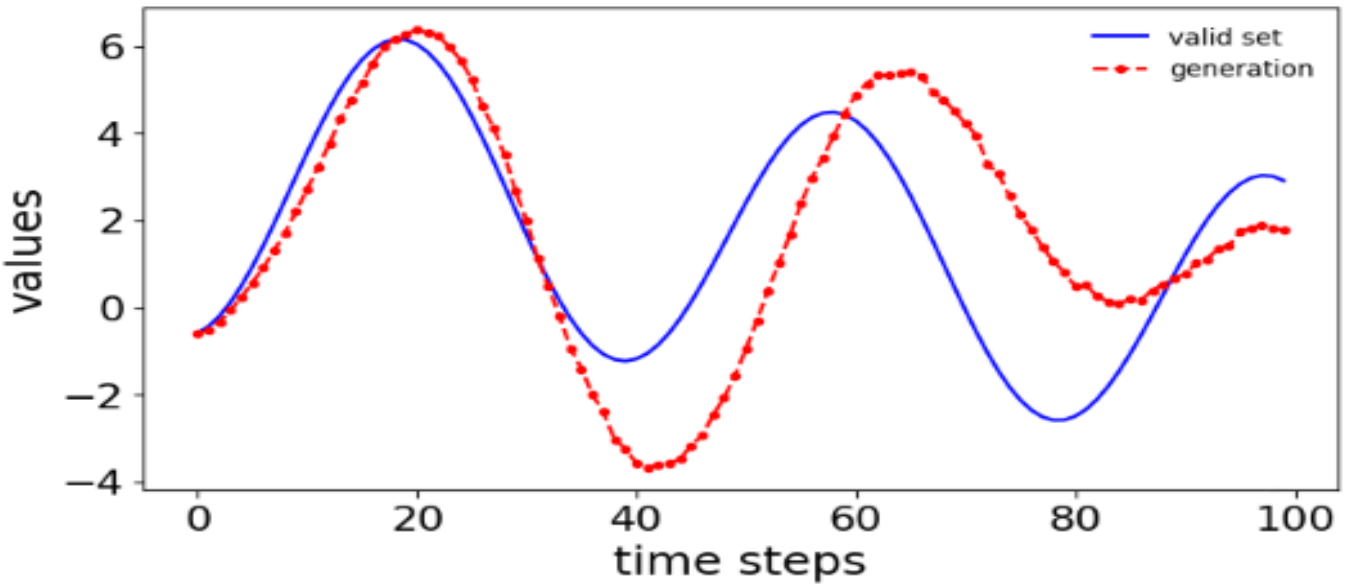

Figura 8.33: Resultado de la LSTM en la generación de 100 puntos observando los 10 previos. Elaboración propia.

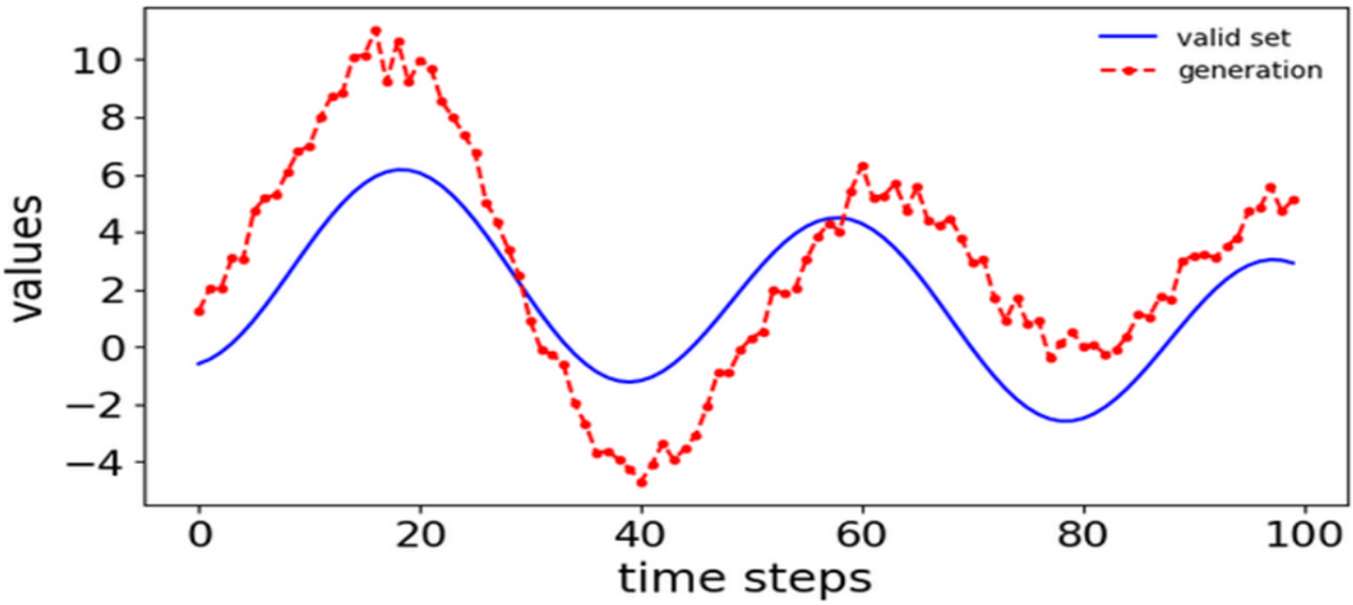

Figura 8.34: Resultado de la LSTM en la generación de 100 puntos observando los 100 previos. Elaboración propia.

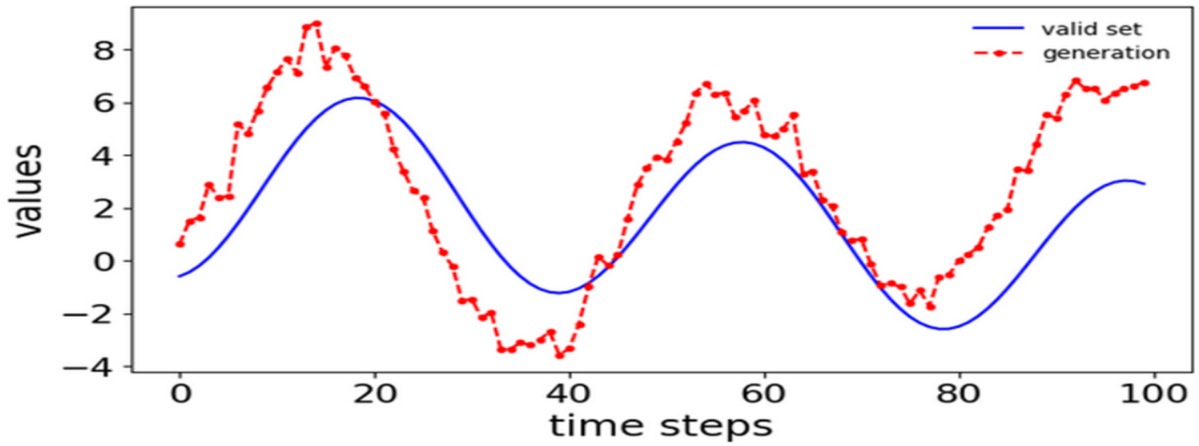

Figura 8.35: Resultado de la LSTM en la generación de 100 puntos observando los 1000 previos. Elaboración propia.

Los resultados de las métricas asociadas a las Figuras 8.33, 8.34 y 8.35 son los siguientes.

Tabla 8.10: Resultados de las métricas en la predicción de señales MEA con LSTM.

Puntos de Calentamiento	Puntos generados	NRMSE	R^2
10	100	0.081200	0.870300
100	100	0.45672	0.105920
1000	100	0.547175	0.0734100

Generación en Google Dev Board

Como se puede observar en la Tabla 8.10, el que mejor resultados ha obtenido es el de 10-100, por lo que esa va a ser la generación que se ejecute en Google Dev Board.

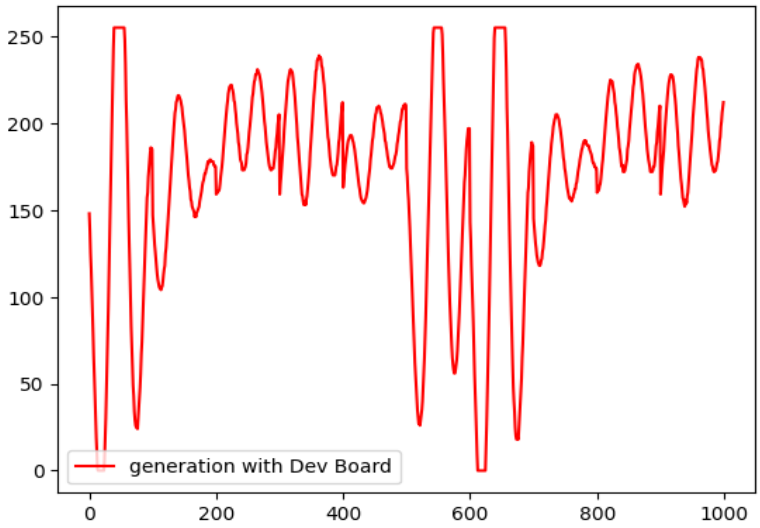

Figura 8.36: Generación en uint8 de la señal MEA mediante Dev Board. Elaboración propia.

8.4.3. Discusión de los resultados obtenidos

Como se puede observar, el modelo durante la predicción consigue unos resultados buenos, pero en la generación deja mucho que desear, tanto en local como en Google Dev Board.

Esto se puede deber a varios factores, pero principalmente se puede encuadrar en la dificultad de las LSTM para capturar información de series temporales y otra serie de limitaciones e inconvenientes discutidos anteriormente.

Otro motivo puede ser que, con una mejor optimización de hiperparámetros y con mayor capacidad de cómputo, el modelo consiga generalizar mejor, pero eso se puede adaptar solamente a secuencias temporales demasiado breves debido a la poca capacidad de memoria de la que disponen este tipo de modelos.

Como se puede ver en la Figura 8.37, a medida que se va exigiendo una generación de una señal MEA más larga, el modelo empieza a inferir perores resultados.

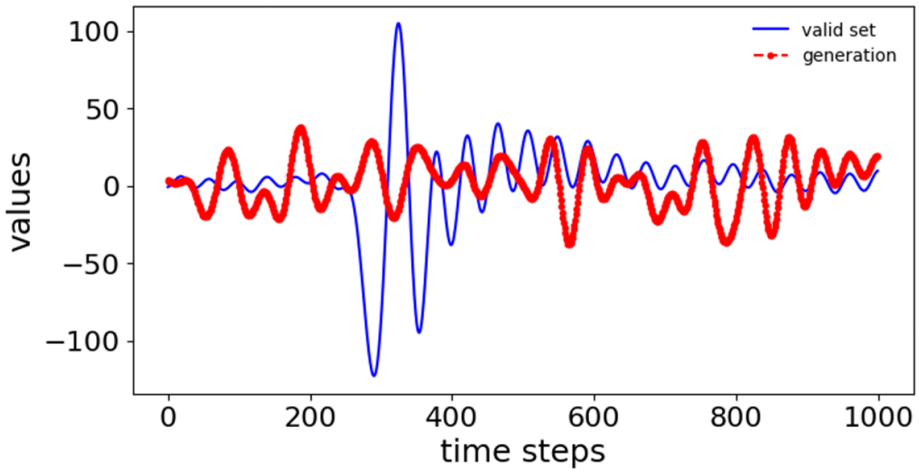

Figura 8.37: Generación errónea de 1000 puntos en una señal MEA con el modelo LSTM visualizando los 100 previos. Elaboración propia.

Además, la precisión de la inferencia de Google Dev Board es menor que el modelo ejecutado en el equipo de trabajo, pues a pesar de las mejoras que trae consigo misma la cuantización, no hay que perder de vista que al disminuir la cantidad de bits para representar, se pierde precisión. Es por ello que en la predicción en local se han obtenido mejores resultados que en el dispositivo.

8.5. Comparación de los modelos

Una vez analizados los diferentes modelos, se va a realizar una comparativa entre ellos.

En la predicción, los mejores resultados para cada modelo se engloban en la Tabla 8.11, mientras que para la generación se encuentran en la Tabla 8.12.

Tabla 8.11: Resultados de la predicción de señales MEA en los diferentes modelos.

Modelo	Puntos de desfase	MSE	RMSE	NRMSE	R^2
ESN	1	0.000055	0.007439	0.005157	0.999957
LSM	-	-	-	-	-
NGRC	-	-	-	-	-
LSTM	1	0.113490	0.033688	0.037693	0.984814

Como se puede observar, teniendo en cuenta que solo se han analizado en la predicción los modelos de ESN y LSTM, los mejores resultados se han obtenido para un único punto de desfase.

Esto, como se ha mencionado anteriormente, guarda relación con las variaciones dentro de un sistema dinámico caótico a la hora de realizar predicciones a largo plazo, por lo que cuanto menor sea la distancia de la predicción, mejores resultados se va a obtener, reflejándose también en este experimento.

Como resultado, el modelo que mejor se ha ajustado a las señales MEA para desempeñar esta tarea ha sido la ESN.

Tabla 8.12: Resultados de la generación de señales MEA en los diferentes modelos.

Modelo	Puntos de calentamiento	Puntos generados	NRMSE	R^2
ESN	200	500	0.047175	0.934100
LSM (16 bits)	893	2520	0.208535	0.426910
NGRC	1000	2000	0.155121	-1.001604
LSTM	10	100	0.081200	0.870300

En la generación, el modelo que mejores resultados ha obtenido ha sido el modelo ESN, seguida de LSTM y LSM. NGRC ha sido el modelo con peores resultados.

Los modelos ESN y LSTM (en su versión en un ordenador convencional) han obtenido muy buenas métricas, con la inconveniencia de que las señales generadas son de poca longitud.

En cambio, con la LSM cuantizada se ha conseguido reducir la representación en bits y conseguir unas generaciones más largas aunque las métricas sean algo peores.

Si lo que se quiere es un mejor ajuste de la señal, se recomienda el uso de la ESN, pero si lo que se quiere es una mayor longitud de secuencia que sea capaz de detectar cuándo se produce una crisis epiléptica restando importancia al ajuste, en este caso, de la amplitud de la onda, el uso de la LSM es más recomendable.

En conclusión, los modelos dentro del marco RC han obtenido mejores resultados frente a los modelos derivados de la arquitectura RNN y de la aproximación NGRC.

Capítulo 9

Conclusiones y líneas futuras

Para finalizar esta memoria, a continuación se van a extraer una serie de conclusiones del trabajo realizado, así como unas pautas para trabajos posteriores que quieran seguir con la línea del proyecto.

9.1. Conclusiones

En este TFG, se han estudiado y analizado diversos modelos para la generación y predicción de señales cerebrales con la intención de obtener una serie de resultados y conclusiones.

Para ello, se ha realizado una planificación que permitiera conseguir una serie de objetivos y, de esta forma, obtener una serie de resultados. Para ello, se ha recolectado información teórica de cada uno de los modelos, para posteriormente implementarlos y, mediante el ajuste de hiperparámetros, conseguir un mejor rendimiento de los modelos.

El primer y segundo modelo se engloban dentro del marco RC. Estos dos modelos son conocidos como ESN y LSM respectivamente, y son los más destacables dentro del marco. Cada modelo ha conseguido adaptar las señales cerebrales a su manera, obteniendo una serie de resultados y conclusiones satisfactorias acorde a la documentación teórica disponible.

Consecuentemente, se ha procedido a analizar el modelo NGRC, el primer modelo que da el nombre al marco NGRC cuyo objetivo principal es mejorar las limitaciones de RC. Aun así, no se ha conseguido ajustar bien a las señales cerebrales disponibles, aunque con más estudio dentro de la arquitectura y mayores capacidades de cómputo pueden surgir modelos englobados en este marco que si lo consigan.

Por último, ya que el marco RC surge con el propósito de solucionar los problemas del desvanecimiento y explosión del gradiente y la memoria limitada de la que disponen los modelos que presentan una arquitectura RNN, se ha decidido probar si un modelo de esta arquitectura bastante reconocido como es la LSTM era capaz de lograr predecir y generar las señales cerebrales con éxito.

En este modelo, debido a que su estructura comparada con el resto de modelos analizados es más sencilla, se ha decidido implementar en hardware y realizar una comparación tanto en un ordenador convencional como en un dispositivo físico destinado a la inferencia de modelos de ML.

El modelo en local conseguía una representación exitosa, con el inconveniente de que las secuencias temporales generadas y predichas necesitaban ser muy cortas. Por otro lado, en su implementación en hardware ha logrado unos resultados mejorables, planteando unas alternativas a seguir en futuras investigaciones.

9.2. Líneas futuras

Este TFG ha servido como introducción a las capacidades que presentan algunos modelos de ML para predecir y generar las señales MEA de las que se disponía. Teniendo en cuenta los resultados obtenidos, se recomiendan las siguientes pautas de trabajo antes de proseguir con esta investigación en otros proyectos.

Para comenzar, una recomendación es probar con equipos de trabajo con más capacidad para analizar secuencias temporales más largas y obtener más resultados que los que se proporcionan en esta memoria, además de intentar mejorar las predicciones y generaciones de los modelos.

Otra recomendación es conseguir que todos los modelos estuvieran cuantizados y comprobar su funcionamiento en Google Dev Board o en otros dispositivos similares, como podrían ser la Raspberry Pi [118] o la gama de Nvidia Jetson [119], y de esta forma comparar los resultados obtenidos, teniendo así una comprensión global de qué dispositivo es más recomendable teniendo en cuenta la tarea a desarollar.

Una vez realizado este trabajo, sería recomendable tratar con otro tipo de series temporales y, en el área de investigación en bioinformática, realizar pruebas con señales capturadas de otros sistemas de registros de actividad cerebral, como por ejemplo señales obtenidas de un electroencefalograma.

Teniendo en cuenta estas sugerencias, se obtendría una amplia gama de resultados que permitirían explorar más a fondo estos modelos, para de esta forma adaptar cada uno de ellos en función de la tarea que se quiera realizar.

Bibliografía

[1] Hassan Sajjad, Nadir Durrani y Fahim Dalvi. «Neuron-level Interpretation of Deep NLP Models: A Survey». En: *Transactions of the Association for Computational Linguistics* 10 (nov. de 2022), págs. 1285-1303. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00519. eprint: https://direct.mit.edu/tacl/article-pdf/doi/10.1162/tacl_a_00519/2060745/tacl_a_00519.pdf. URL: https://doi.org/10.1162/tacl%5C_a%5C_00519.

[2] Philippe Faure y Henri Korn. «Is there chaos in the brain? I. Concepts of nonlinear dynamics and methods of investigation». En: *Comptes rendus de l'Académie des sciences. Série III, Sciences de la vie* 324 (oct. de 2001), págs. 773-93. DOI: 10.1016/S0764-4469(01)01377-4.

[3] Germán Rodríguez-Bermúdez y Pedro García Laencina. «Analysis of EEG signals using nonlinear dynamics and chaos: A review». En: *Applied Mathematics and Information Sciences* 9 (ene. de 2015), págs. 1-13. DOI: 10.12785/amis/090512.

[4] B. Hughes. *Software Project Management 5e*. McGraw-Hill Education, 2009. ISBN: 978-0-07-716954-1. URL: https://books.google.es/books?id=IMsvEAAAQBAJ.

[5] Peter Chapman. *CRISP-DM 1.0: Step-by-step data mining guide*. 2000. URL: https://api.semanticscholar.org/CorpusID:59777418.

[6] *Manifiesto Por El Desarrollo Ágil de Software*. https://agilemanifesto.org/iso/es/manifesto.html. (Visitado 15-05-2024).

[7] Nick Hotz. *What Is CRISP DM?* Sep. de 2018. (Visitado 15-05-2024).

[8] Alex Souza. *Metodologia CRISP-DM: Uma Abordagem Abrangente para Projetos de Dados*. en. Jul. de 2023. URL: https://medium.com/blog-do-zouza/metodologia-crisp-dm-uma-abordagem-abrangente-para-projetos-de-dados-d7e7135b907e (visitado 16-05-2024).

[9] *PMBOK® Guide*. en. URL: https://www.pmi.org/pmbok-guide-standards/foundational/pmbok (visitado 16-05-2024).

[10] *Sueldo: Machine Learning Engineer en España en 2024*. es. URL: https://www.glassdoor.es/Sueldos/machine-learning-engineer-sueldo-SRCH_KO0,25.htm (visitado 16-05-2024).

[11] *Sueldo: Ingeniero Biomedico en España 2024*. es. URL: https://www.glassdoor.es/Sueldos/ingeniero-biomedico-sueldo-SRCH_KO0,19.htm (visitado 29-05-2024).

[12] Roger Dobaño-CEO Quipu. *¿Cuánto cuesta contratar a un trabajador? (Actualizado 2024)*. es. Dic. de 2021. URL: https://getquipu.com/blog/cuanto-cuesta-contratar-un-trabajador/ (visitado 17-05-2024).

[13] *Procesador Intel® Core™ i5-11400 (caché de 12 MB; hasta 4,40 GHz) - Especificaciones de productos*. es. URL: https://www.intel.la/content/www/xl/es/products/sku/212270/intel-core-i511400-processor-12m-cache-up-to-4-40-ghz/specifications.html (visitado 17-05-2024).

[14] *Zone Evil Office Plus 22AH510I557 Intel Core i5-11400/16GB/500GB SSD — PcComponentes.com*. URL: https://www.pccomponentes.com/zone-evil-office-plus-22ah510i557-intel-core-i5-11400-16gb-500gb-ssd (visitado 17-05-2024).

[15] *Precios de Los Servicios de Pago de Colab*. https://colab.research.google.com/signup. (Visitado 04-06-2024).

[16] *Google Colaboratory*. es. URL: https://colab.research.google.com/github/google-coral/tutorials/blob/master/train_lstm_timeseries_ptq_tf2.ipynb (visitado 11-04-2024).

[17] *Dev Board*. en-us. URL: https://coral.ai/products/dev-board/ (visitado 17-05-2024).

[18] *TensorFlow*. es. Page Version ID: 158188987. Feb. de 2024. URL: https://es.wikipedia.org/w/index.php?title=TensorFlow&oldid=158188987 (visitado 14-05-2024).

[19] *Keras*. es. Page Version ID: 159645409. Abr. de 2024. URL: https://es.wikipedia.org/w/index.php?title=Keras&oldid=159645409 (visitado 14-05-2024).

[20] *TensorFlow Lite*. en. URL: https://www.tensorflow.org/lite/guide (visitado 14-05-2024).

[21] *pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visitado 15-05-2024).

[22] *ReservoirPy — ReservoirPy 0.3.11 documentation*. URL: https://reservoirpy.readthedocs.io/en/latest/index.html (visitado 02-04-2024).

[23] *SciPy*. es. Page Version ID: 157628165. Ene. de 2024. URL: https://es.wikipedia.org/w/index.php?title=SciPy&oldid=157628165 (visitado 14-05-2024).

[24] *NumPy documentation — NumPy v1.26 Manual*. URL: https://numpy.org/doc/stable/ (visitado 14-05-2024).

[25] *Matplotlib — Visualization with Python*. URL: https://matplotlib.org/ (visitado 14-05-2024).

[26] *scikit-learn: machine learning in Python — scikit-learn 1.4.2 documentation*. URL: https://scikit-learn.org/stable/ (visitado 15-05-2024).

[27] Takuya Akiba et al. «Optuna: A Next-generation Hyperparameter Optimization Framework». en. En: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. Anchorage AK USA: ACM, jul. de 2019, págs. 2623-2631. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330701. URL: https://dl.acm.org/doi/10.1145/3292500.3330701 (visitado 14-05-2024).

[28] *Joblib: running Python functions as pipeline jobs — joblib 1.4.2 documentation*. URL: https://joblib.readthedocs.io/en/stable/ (visitado 15-05-2024).

[29] T. M. McKenna, T. A. McMullen y M. F. Shlesinger. «The brain as a dynamic physical system». eng. En: *Neuroscience* 60.3 (jun. de 1994), págs. 587-605. ISSN: 0306-4522. DOI: 10.1016/0306-4522(94)90489-8.

[30] Rainer Klages. «Introduction to Dynamical Systems». en. En: ().

[31] Manuel Benjamin Ortiz Moctezuma. *Sistemas dinámicos en tiempo continuo: Modelado y simulación*. es. 1st. OmniaScience, dic. de 2015. ISBN: 978-84-944673-2-5. DOI: 10.3926/oss.25. URL: https://www.omniascience.com/books/index.php/scholar/catalog/book/38 (visitado 21-05-2024).

[32] Steven Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. en. Second edition, first issued in hardback. A Chapman & Hall book. Boca Raton London New York: CRC Press, 2019.

[33] *Dynamical systems theory*. en. Page Version ID: 1220918822. Abr. de 2024. URL: https://en.wikipedia.org/w/index.php?title=Dynamical_systems_theory&oldid=1220918822 (visitado 21-05-2024).

[34] Daniel Durstewitz, Georgia Koppe y Max Ingo Thurm. *Reconstructing Computational Dynamics from Neural Measurements with Recurrent Neural Networks*. en. Nov. de 2022. DOI: 10.1101/2022.10.31.514408. URL: http://biorxiv.org/lookup/doi/10.1101/2022.10.31.514408 (visitado 21-05-2024).

[35] Lequan Lin et al. *Diffusion Models for Time Series Applications: A Survey*. en. arXiv:2305.00624 [cs]. Abr. de 2023. URL: http://arxiv.org/abs/2305.00624 (visitado 22-05-2024).

[36] Marin Biloš et al. *Modeling Temporal Data as Continuous Functions with Stochastic Process Diffusion*. en. arXiv:2211.02590 [cs]. Mayo de 2023. URL: http://arxiv.org/abs/2211.02590 (visitado 22-05-2024).

[37] Yan Li et al. *Generative Time Series Forecasting with Diffusion, Denoise, and Disentanglement*. en. arXiv:2301.03028 [cs]. Ene. de 2023. URL: http://arxiv.org/abs/2301.03028 (visitado 22-05-2024).

[38] Aaron van den Oord et al. *WaveNet: A Generative Model for Raw Audio*. en. arXiv:1609.03499 [cs]. Sep. de 2016. URL: http://arxiv.org/abs/1609.03499 (visitado 22-05-2024).

[39] Kashif Rasul et al. *Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting*. en. arXiv:2101.12072 [cs]. Feb. de 2021. URL: http://arxiv.org/abs/2101.12072 (visitado 22-05-2024).

[40] Kashif Rasul et al. *Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows*. en. arXiv:2002.06103 [cs, stat]. Ene. de 2021. URL: http://arxiv.org/abs/2002.06103 (visitado 22-05-2024).

[41] Tijin Yan et al. *ScoreGrad: Multivariate Probabilistic Time Series Forecasting with Continuous Energy-based Generative Models*. en. arXiv:2106.10121 [cs, stat]. Jun. de 2021. URL: http://arxiv.org/abs/2106.10121 (visitado 22-05-2024).

[42] Ruikun Li et al. «Graph Convolution Recurrent Denoising Diffusion Model for Multivariate Probabilistic Temporal Forecasting». En: nov. de 2023, págs. 661-676. ISBN: 978-3-031-46660-1. DOI: 10.1007/978-3-031-46661-8_44.

[43] Haomin Wen et al. *DiffSTG: Probabilistic Spatio-Temporal Graph Forecasting with Denoising Diffusion Models*. en. arXiv:2301.13629 [cs]. Mar. de 2024. URL: http://arxiv.org/abs/2301.13629 (visitado 22-05-2024).

[44] Juan Miguel Lopez Alcaraz y Nils Strodthoff. *Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models*. en. arXiv:2208.09399 [cs, stat]. Mayo de 2023. URL: http://arxiv.org/abs/2208.09399 (visitado 22-05-2024).

[45] Wei Cao et al. «BRITS: Bidirectional Recurrent Imputation for Time Series». En: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://papers.nips.cc/paper_files/paper/2018/hash/734e6bfcd358e25ac1db0a4241b95651-Abstract.html (visitado 22-05-2024).

[46] Zhengping Che et al. «Recurrent Neural Networks for Multivariate Time Series with Missing Values». en. En: *Scientific Reports* 8.1 (abr. de 2018). Publisher: Nature Publishing Group, pág. 6085. ISSN: 2045-2322. DOI: 10.1038/s41598-018-24271-9. URL: https://www.nature.com/articles/s41598-018-24271-9 (visitado 22-05-2024).

[47] Yonghong Luo et al. «Multivariate Time Series Imputation with Generative Adversarial Networks». En: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://papers.nips.cc/paper_files/paper/2018/hash/96b9bff013acedfb1d140579e2fbeb63-Abstract.html (visitado 22-05-2024).

[48] Yusuke Tashiro et al. *CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation*. en. arXiv:2107.03502 [cs, stat]. Oct. de 2021. URL: http://arxiv.org/abs/2107.03502 (visitado 22-05-2024).

[49] Mingzhe Liu et al. *PriSTI: A Conditional Diffusion Framework for Spatiotemporal Imputation*. en. arXiv:2302.09746 [cs]. Feb. de 2023. URL: http://arxiv.org/abs/2302.09746 (visitado 22-05-2024).

[50] Casey Chu, Kentaro Minami y Kenji Fukumizu. *Smoothness and Stability in GANs*. en. arXiv:2002.04185 [cs, stat]. Feb. de 2020. URL: http://arxiv.org/abs/2002.04185 (visitado 22-05-2024).

[51] Abhyuday Desai et al. *TimeVAE: A Variational Auto-Encoder for Multivariate Time Series Generation*. en. arXiv:2111.08095 [cs]. Dic. de 2021. URL: http://arxiv.org/abs/2111.08095 (visitado 22-05-2024).

[52] Chris Donahue, Julian McAuley y Miller Puckette. *Adversarial Audio Synthesis*. arXiv:1802.04208 [cs]. Feb. de 2019. DOI: 10.48550/arXiv.1802.04208. URL: http://arxiv.org/abs/1802.04208 (visitado 22-05-2024).

[53] Haksoo Lim et al. *Regular Time-series Generation using SGM*. en. arXiv:2301.08518 [cs]. Ene. de 2023. URL: http://arxiv.org/abs/2301.08518 (visitado 22-05-2024).

[54] Mantas Lukoševičius y Herbert Jaeger. «Reservoir computing approaches to recurrent neural network training». En: *Computer Science Review* 3.3 (2009), págs. 127-149. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2009.03.005. URL: https://www.sciencedirect.com/science/article/pii/S1574013709000173.

[55] Dominik Walther et al. «A systematic comparison of deep learning methods for EEG time series analysis». English. En: *Frontiers in Neuroinformatics* 17 (feb. de 2023). Publisher: Frontiers. ISSN: 1662-5196. DOI: 10.3389/fninf.2023.1067095. URL: https://www.frontiersin.org/articles/10.3389/fninf.2023.1067095 (visitado 22-05-2024).

[56] Farhad Mortezapour Shiri et al. «A Comprehensive Overview and Comparative Analysis on Deep Learning Models: CNN, RNN, LSTM, GRU». en. En: ().

[57] Sambeet Mishra et al. «Comparison of deep learning models for multivariate prediction of time series wind power generation and temperature». En: *Energy Reports*. Technologies and Materials for Renewable Energy, Environment and Sustainability 6 (feb. de 2020), págs. 273-286. ISSN: 2352-4847. DOI: 10.1016/j.egyr.2019.11.009. URL: https://www.sciencedirect.com/science/article/pii/S2352484719308546 (visitado 22-05-2024).

[58] Benjamin Schrauwen, David Verstraeten y Jan Campenhout. «An overview of reservoir computing: Theory, applications and implementations». En: ene. de 2007, págs. 471-482.

[59] Yaqing Zhang et al. «An Investigation of Deep Learning Models for EEG-Based Emotion Recognition». English. En: *Frontiers in Neuroscience* 14 (dic. de 2020). Publisher: Frontiers. ISSN: 1662-453X. DOI: 10.3389/fnins.2020.622759. URL: https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2020.622759/full (visitado 22-05-2024).

[60] Alireza Goudarzi et al. *A Comparative Study of Reservoir Computing for Temporal Signal Processing*. en. arXiv:1401.2224 [cs]. Ene. de 2014. URL: http://arxiv.org/abs/1401.2224 (visitado 22-05-2024).

[61] Shahrokh Shahi, Flavio H. Fenton y Elizabeth M. Cherry. «Prediction of chaotic time series using recurrent neural networks and reservoir computing techniques: A comparative study». En: *Machine Learning with Applications* 8 (jun. de 2022), pág. 100300. ISSN: 2666-8270. DOI: 10.1016/j.mlwa.2022.100300. URL: https://www.sciencedirect.com/science/article/pii/S2666827022000275 (visitado 22-05-2024).

[62] Daniel J Gauthier et al. «Next Generation Reservoir Computing». en. En: ().

[63] Johannes Maurin Voshol. «Quantized Liquid State Machines: Design and Implementation». en. En: ().

[64] Gabriella Panuccio, Davide Caron y Angel Canal-Alonso. *Mimicking CA3 Temporal Dynamics Controls Limbic Ictogenesis*. Feb. de 2022. DOI: 10.5281/zenodo.6278046. (Visitado 07-06-2024).

[65] Terri Roberts et al. «Encoding Temporal Regularities and Information Copying in Hippocampal Circuits». En: *Scientific Reports* 9 (dic. de 2019). DOI: 10.1038/s41598-019-55395-1.

[66] Davide Caron, Ángel Canal-Alonso y Gabriella Panuccio. «Mimicking CA3 Temporal Dynamics Controls Limbic Ictogenesis». en. En: *Biology* 11.3 (mar. de 2022). Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, pág. 371. ISSN: 2079-7737. DOI: 10.3390/biology11030371. URL: https://www.mdpi.com/2079-7737/11/3/371 (visitado 29-05-2024).

[67] Siyan Wang et al. «CA3 principal cell activation triggers hypersynchronous-onset seizures in a mouse model of mesial temporal lobe epilepsy». eng. En: *Journal of Neurophysiology* 130.4 (oct. de 2023), págs. 1041-1052. ISSN: 1522-1598. DOI: 10.1152/jn.00244.2023.

[68] Alvaro Tejero-Cantero. «Of memories and ripples: functional and mechanistic aspects of memory sequences during hippocampal ripples». Tesis doct. Jul. de 2012.

[69] *Inteligencia artificial*. es. Page Version ID: 160001682. Mayo de 2024. URL: https://es.wikipedia.org/w/index.php?title=Inteligencia_artificial&oldid=160001682#Aprendizaje_automatizado_y_aprendizaje_profundo (visitado 09-05-2024).

[70] Altaf Hussain, Ijaz Ullah y Tariq Hussain. «The Approach of Data Mining: A Performance-based Perspective of Segregated Data Estimation to Classify Distinction by Applying Diverse Data Mining Classifiers». En: *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal* 10 (feb. de 2022), págs. 339-359. DOI: 10.14201/ADCAIJ2021104339359.

[71] Aridio Silva. *aridiosilva/My-TensorFlow-tutorials*. original-date: 2019-02-15T19:25:45Z. Mar. de 2023. URL: https://github.com/aridiosilva/My-TensorFlow-tutorials (visitado 11-04-2024).

[72] *Redes neuronales recurrentes*. es. Page Version ID: 159512598. Abr. de 2024. URL: https://es.wikipedia.org/w/index.php?title=Redes_neuronales_recurrentes&oldid=159512598#Funcionamiento_de_las_redes_neuronales_recurrentes (visitado 20-05-2024).

[73] Ye Jiexia et al. *How to Build a Graph-Based Deep Learning Architecture in Traffic Domain: A Survey*. Mayo de 2020.

[74] Robin M. Schmidt. «Recurrent Neural Networks (RNNs): A gentle Introduction and Overview». en. En: (nov. de 2019). arXiv:1912.05911 [cs, stat]. URL: http://arxiv.org/abs/1912.05911 (visitado 21-05-2024).

[75] Rubén Rodríguez Abril. *Redes Neuronales Recurrentes • Artículo de LMO*. es. Mayo de 2021. URL: https://lamaquinaoraculo.com/deep-learning/redes-neuronales-recurrentes/ (visitado 20-05-2024).

[76] *The Unreasonable Effectiveness of Recurrent Neural Networks*. URL: http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (visitado 20-05-2024).

[77] Corentin Tallec y Yann Ollivier. *Unbiasing Truncated Backpropagation Through Time*. arXiv:1705.08209 [cs]. Mayo de 2017. DOI: 10.48550/arXiv.1705.08209. URL: http://arxiv.org/abs/1705.08209 (visitado 21-05-2024).

[78] Jingzhao Zhang et al. *Why gradient clipping accelerates training: A theoretical justification for adaptivity*. arXiv:1905.11881 [cs, math]. Feb. de 2020. DOI: 10.48550/arXiv.1905.11881. URL: http://arxiv.org/abs/1905.11881 (visitado 21-05-2024).

[79] IFISC IT TEam + APSL- www.apsl.net. *Reservoir Computing: Theory, Physical Implementations, and Applications*. URL: https://ifisc.uib-csic.es/es/publications/reservoir-computing-theory-physical-implementation/ (visitado 23-05-2024).

[80] Matthew Stewart PhD. *Predicting Stock Prices with Echo State Networks*. en. Feb. de 2023. URL: https://towardsdatascience.com/predicting-stock-prices-with-echo-state-networks-f910809d23d4 (visitado 30-05-2024).

[81] Herbert Jaeger. «The "Echo State" Approach to Analysing and Training Recurrent Neural Networks». En: *GMD-Report 148, German National Research Institute for Computer Science* (ene. de 2001).

[82] Wolfgang Maass. «Liquid State Machines: Motivation, Theory, and Applications». En: (ene. de 2010). DOI: 10.1142/9781848162778_0008.

[83] Gang Li et al. «Echo State Network with Bayesian Regularization for Forecasting Short-Term Power Production of Small Hydropower Plants». En: *Energies* 8 (oct. de 2015), págs. 12228-12241. DOI: 10.3390/en81012228.

[84] Claudio Gallicchio y Alessio Micheli. *Deep Echo State Network (DeepESN): A Brief Survey*. Sep. de 2020. DOI: 10.48550/arXiv.1712.04323. arXiv: 1712.04323 [cs, stat]. (Visitado 03-06-2024).

[85] Samar Bouazizi, Emna Benmohamed y Hela Ltifi. «Enhancing EEG-based Emotion Recognition Using PSD-Grouped Deep Echo State Network». En: *JUCS - Journal of Universal Computer Science* 29.10 (oct. de 2023), págs. 1116-1138. ISSN: 0948-6968. DOI: 10.3897/jucs.98789. (Visitado 03-06-2024).

[86] Soumya Shrivastava. *Cross Validation in Time Series*. Ene. de 2020. (Visitado 26-05-2024).

[87] Qiuyi Wu, Ernest Fokoue y Dhireesha Kudithipudi. *On the Statistical Challenges of Echo State Networks and Some Potential Remedies*. 2018. arXiv: 1802.07369 [stat.ML].

[88] Nicholas Soures y Dhireesha Kudithipudi. «Deep Liquid State Machines With Neural Plasticity for Video Activity Recognition». En: *Frontiers in Neuroscience* 13 (jul. de 2019), pág. 686. ISSN: 1662-4548. DOI: 10.3389/fnins.2019.00686. (Visitado 28-05-2024).

[89] Kashu Yamazaki et al. «Spiking Neural Networks and Their Applications: A Review». En: *Brain Sciences* 12.7 (jun. de 2022), pág. 863. ISSN: 2076-3425. DOI: 10.3390/brainsci12070863. (Visitado 28-05-2024).

[90] Chander Prakash et al. «Computing of neuromorphic materials: an emerging approach for bioengineering solutions». en. En: *Materials Advances* 4.23 (2023). Publisher: Royal Society of Chemistry, págs. 5882-5919. DOI: 10.1039/D3MA00449J. URL: https://pubs.rsc.org/en/content/articlelanding/2023/ma/d3ma00449j (visitado 30-05-2024).

[91] *Spiking neural network*. en. Page Version ID: 1214850196. Mar. de 2024. URL: https://en.wikipedia.org/w/index.php?title=Spiking_neural_network&oldid=1214850196 (visitado 11-04-2024).

[92] «Liquid State Machine». En: *Wikipedia* (mayo de 2023). (Visitado 28-05-2024).

[93] Önder Gürcan. «Exploration of Biological Neural Wiring Using Self-Organizing Agents». Tesis doct. Sep. de 2013.

[94] *M4urin/Quantized-Liquid-State-Machines: A Liquid State Machine Using Quantized Neurons That Are Operating on Lower-Bit Representations and Fixed Point Computations. It Provides a next Step towards the Implementation of Efficient Accelerators That Can Be Used in the Field of Neuromorphic Computing*. https://github.com/m4urin/quantized-liquid-state-machines/tree/main. (Visitado 01-06-2024).

[95] *Neural Network Quantization*. https://www.allaboutcircuits.com/ technical-articles/neural-network-quantization-what-is-it-and-how-does-it-relate-to-tiny-machine-learning/. (Visitado 01-06-2024).

[96] Itay Hubara et al. «Quantized neural networks: Training neural networks with low precision weights and activations». En: *Journal of Machine Learning Research* 18.187 (2018), págs. 1-30.

[97] *quantinfo/ng-rc-paper-code*. original-date: 2021-05-25T15:33:00Z. Abr. de 2024. URL: https://github.com/quantinfo/ng-rc-paper-code (visitado 16-04-2024).

[98] Ravi Chepuri et al. *Hybridizing Traditional and Next-Generation Reservoir Computing to Accurately and Efficiently Forecast Dynamical Systems*. en. arXiv:2403.18953 [cs]. Mar. de 2024. URL: http://arxiv.org/abs/2403.18953 (visitado 16-04-2024).

[99] *LSTM weather forecasting model for Coral Edge TPU*. URL: https://colab.research.google.com/github/%20google-coral/tutorials/blob/master/train_lstm_timeseries_ptq_tf2.ipynb (visitado 29-05-2024).

[100] Ottavio Calzone. *An Intuitive Explanation of LSTM*. en. Abr. de 2022. URL: https://medium.com/@ottaviocalzone/an-intuitive-explanation-of-lstm-a035eb6ab42c (visitado 30-05-2024).

[101] *10.1. Long Short-Term Memory (LSTM) — Dive into Deep Learning 1.0.3 documentation*. URL: https://d2l.ai/chapter_recurrent-modern/lstm.html (visitado 30-05-2024).

[102] *machine-learning-articles/build-an-lstm-model-with-tensorflow-and-keras.md at main · christianversloot/machine-learning-articles*. en. URL: https://github.com/christianversloot/machine-learning-articles/blob/main/build-an-lstm-model-with-tensorflow-and-keras.md (visitado 03-05-2024).

[103] Prudhviraju Srivatsavaya. *LSTM — Implementation, Advantages and Diadvantages*. en. Oct. de 2023. URL: https://medium.com/@prudhviraju.srivatsavaya/lstm-implementation-advantages-and-diadvantages-914a96fa0acb (visitado 29-05-2024).

[104] *Coral*. https://coral.ai/. (Visitado 09-06-2024).

[105] Raghubir Singh y Sukhpal Singh Gill. «Edge AI: A survey». En: *Internet of Things and Cyber-Physical Systems* 3 (2023), págs. 71-92. ISSN: 2667-3452. DOI: https://doi.org/10.1016/j.iotcps.2023.02.004. URL: https://www.sciencedirect.com/science/article/pii/S2667345223000196.

[106] *Dev Board*. https://coral.ai/products/dev-board/. (Visitado 17-05-2024).

[107] *Dev Board Datasheet*. https://coral.ai/docs/dev-board/datasheet/#features. (Visitado 09-06-2024).

[108] Santi Iglesias. *Google Edge TPU Coral with a Keras Custom Model (All You Need to Know, a Real Deploy Case)*. Jun. de 2020. (Visitado 11-04-2024).

[109] *Build an Lstm Model with Tensorflow and Keras*. https://github.com/christianversloot /machine-learning-articles/blob/main/build-an-lstm-model-with-tensorflow-and-keras.md. (Visitado 03-05-2024).

[110] *Examples — Coral*. https://coral.ai/examples/. (Visitado 09-06-2024).

[111] *LSTM Weather Forecasting Model for Coral Edge TPU*. https://colab.research.google.com/github/google-coral/tutorials/blob/master/train_lstm_timeseries_ptq_tf2.ipynb. (Visitado 29-05-2024).

[112] *TensorFlow Models on the Edge TPU*. https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview. (Visitado 11-06-2024).

[113] *Mendel Development Tool (Mdt)*. https://coral.ai/docs/dev-board/mdt/#install-mdt. (Visitado 11-06-2024).

[114] Mantas Lukoševičius. «A Practical Guide to Applying Echo State Networks». en. En: *Neural Networks: Tricks of the Trade*. Ed. por Grégoire Montavon, Geneviève B. Orr y Klaus-Robert Müller. Vol. 7700. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, págs. 659-686. DOI: 10.1007/978-3-642-35289-8_36. URL: http://link.springer.com/10.1007/978-3-642-35289-8_36 (visitado 05-04-2024).

[115] «Raíz del error cuadrático medio». En: *Wikipedia, la enciclopedia libre* (mayo de 2020). (Visitado 03-06-2024).

[116] *Teoría del caos*. es. Page Version ID: 160126174. Mayo de 2024. URL: https://es.wikipedia.org/w/index.php?title=Teor%C3%ADa_del_caos&oldid=160126174#En_medicina (visitado 04-06-2024).

[117] *Efecto mariposa*. es. Page Version ID: 158625459. Mar. de 2024. URL: https://es.wikipedia.org/w/index.php?title=Efecto_mariposa&oldid=158625459 (visitado 04-06-2024).

[118] Raspberry Pi Ltd. *Raspberry Pi*. https://www.raspberrypi.com/. (Visitado 11-06-2024).

[119] *Sistemas integrados NVIDIA para las máquinas autónomas de la próxima generación*. https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/. (Visitado 11-06-2024).

[120] *La Asamblea General adopta la Agenda 2030 para el Desarrollo Sostenible*. Sep. de 2015. (Visitado 06-06-2024).

[121] *Ministerio de Derechos Sociales, Consumo y Agenda 2030 - Agenda 2030*. https://www.mdsocialesa2030.gob.es/index.htm. (Visitado 06-06-2024).

[122] Robert S. Sloviter. «Hippocampal Epileptogenesis in Animal Models of Mesial Temporal Lobe Epilepsy with Hippocampal Sclerosis: The Importance of the "Latent Period" and Other Concepts». En: *Epilepsia* 49.s9 (2008), págs. 85-92. ISSN: 1528-1167. DOI: 10.1111/j.1528-1167.2008.01931.x. (Visitado 06-06-2024).

Apéndice A

Contemplación de los Objetivos de Desarrollo Sostenible

Actualmente, muchos de los gobiernos están actuando en conjunto para lograr un futuro accesible y sostenible.

Es por ello que en 2015 se adoptó por la Organización de las Naciones Unidas (ONU) un plan de acción que consiguiera satisfacer una serie de necesidades, creando así la Agenda 2030 [120].

Este plan engloba 17 objetivos generales, llamados Objetivos de Desarrollo Sostenible (ODS), como se puede observar en la Figura A.1.

Este TFG se puede englobar principalmente dentro del **objetivo 3: Salud y Bienestar**, que busca garantizar una vida sana y promover el bienestar para todos en todas las edades, asegurando de esta forma el desarrollo sostenible [121].

Esto se debe a que la epilepsia, en concreto la que se ha tratado en este TFG que es la denominada epilepsia del lóbulo temporal (ELT), afecta a millones de personas en todo el mundo. Entorno al 30 % de los casos de crisis epilépticas corresponden a este tipo, siendo la más frecuente entre los adultos [122].

Usando este TFG con fines académicos y de investigación se podría emplear para la predicción de convulsiones. Empleando los diferentes modelos propuestos y analizados se podría ser capaz de identificar los patrones y, mediante la predicción temprana de las convulsiones, poner en alerta a los pacientes, facilitando la toma de medidas preventivas. Si se consiguiese avisar al paciente con el tiempo suficiente, este se podría ubicar en un lugar seguro antes de la crisis y facilitar la asistencia médica.

También se podrían emplear los modelos descritos para la generación de señales MEA sintéticas para que sirvan de estudio o como nuevos datos para entrenar modelos más potentes que se puedan usar para crear diferentes prótesis o implantes cerebrales.

De forma secundaria, también se logran cumplimentar el **objetivo 9: Industria, innovación e infraestructura**, ya que si se continúa investigando esta área se podría impulsar el crecimiento y la innovación del sistema sanitario global; y el **objetivo 10: Reducción de las desigualdades**, pues si se llegase a comercializar ciertos avances podrían crearse herramientas de diagnóstico y prótesis capaces de ser accesibles en países tercermundistas.

Gracias al trabajo desempeñado en este proyecto, se espera que se conciencie a los lectores de este TFG de la importancia que tiene la investigación de esta enfermedad, y que diversos sectores y empresas sigan destinando parte de sus fondos en ayudar a las personas que sufren de esta patología para conseguir un futuro más saludable y sostenible.

Figura A.1: Objetivos de desarrollo sostenible planteados en la Agenda 2030 por la ONU [120].

Apéndice B

Tiempos de ejecución en la búsqueda de hiperparámetros

En los próximos párrafos se van a discutir los mejores tiempos de ejecución en la búsqueda de hiperparámetros según el numero de procesadores de los que se disponga.

Cabe destacar que estos tiempos corresponden a las ejecuciones realizadas en el equipo de trabajo descrito con anterioridad, por lo que los resultados pueden variar al ejecutarse en otros dispositivos con otras prestaciones diferentes.

Debido a que la búsqueda de hiperparámetros es lo que mayor impacto genera a la hora de hacer que un modelo funcione correctamente, como se ha visto en el Capítulo 8, es importante intentar sacar el mejor conjunto posible.

No obstante, estas búsquedas pueden tardar demasiado tiempo en comparación con otras partes, es por eso que se han tomado ciertas medidas para intentar hacer el proceso de búsqueda lo más eficaz posible.

La que se va a tratar en este anexo es el conseguir paralelizar la búsqueda en diferentes procesos en CPU.

Para ello, se han hecho diferentes pruebas en cada uno de los modelos con 4, 8 y 12 procesos usando Joblib (ver 3.9) bajo el mismo número de intentos. Se han elegido estos números ya que el equipo contaba con un total de 12 procesos utilizables.

Cabe destacar que, como el modelo de LSTM se ha ejecutado en otro entorno distinto, prescindiendo de la librería de Joblib, se ha excluido de esta comparación ya que no estaría realizada bajo las mismas condiciones de partida.

Después, se ha elaborado la media de los tiempos medios, respresentándolo en una gráfica, y se ha establecido una comparación con el tiempo medio de lo que se tardaría con un solo procesador.

Una vez obtenido el mejor número de procesos para la tarea, se ha realizado el resto de búsquedas de hiperparámetros dentro del modelo correspondiente con ese número de procesos, consiguiendo de esta forma ahorrar tiempo en dicha búsqueda.

B.1. Tiempos de ejecución en la búsqueda de ESN

Para la ESN se lograron los siguientes tiempos de ejecución en segundos.

Primero, se realizaron 5 experimentos con 4, 8 y 12 procesos, que se compararon a su vez usando tan solo un proceso, obteniendo los siguientes resultados

1 Proceso	4 Procesos	8 Procesos	12 Procesos
400	348	335	320
420	345	340	315
410	350	336	318
405	346	339	312
415	346	340	315

Tabla B.1: Valores para diferentes números de procesos en ESN en segundos.

Obteniendo los siguientes tiempos medios:

Procesos	Promedio
1	410
4	347
8	338
12	316

Tabla B.2: Promedios para diferentes números de procesos en ESN en segundos.

De los valores obtenidos extraemos que el mejor tiempo medio se obtiene usando 12 procesos, reduciendo en casi 100 segundos el tiempo medio que se obtendría usando tan solo un proceso.

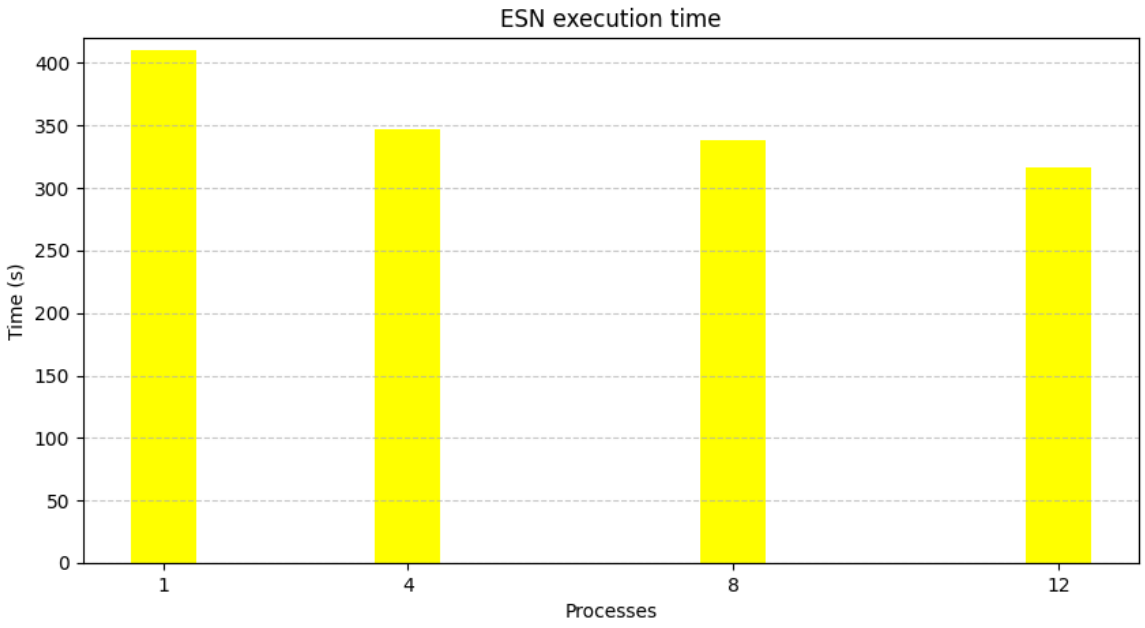

Figura B.1: Tiempo de ejecución medio en segundos en la búsqueda de hiperparámetros de ESN con distintos procesos.

B.2. Tiempos de ejecución en la búsqueda de LSM

A continuación, se van a mostrar los tiempos de ejecución en la búsqueda de hiperparámetros de la LSM para 4, 8 y 12 procesos.

Primero, se realizaron 5 experimentos, en los cuales se compararon el tiempo de 4, 8 y 12 procesos con el valor usando tan solo 1 proceso. Los resultados son los siguientes:

1 Proceso	4 Procesos	8 Procesos	12 Procesos
90	72	30	55
85	68	40	45
95	65	20	60
88	75	45	50
92	70	45	40

Tabla B.3: Valores para diferentes números de procesos en LSM en segundos.

Se hallaron los tiempos medios de estos 5 experimentos:

Procesos	Promedio
1	90
4	70
8	35
12	50

Tabla B.4: Promedios para diferentes números de procesos en LSM en segundos.

Como se puede observar, el mejor valor se obtiene con 8 procesos, reduciendo en aproximadamente dos tercios el tiempo medio en caso de que se realizase tan solo con 1.

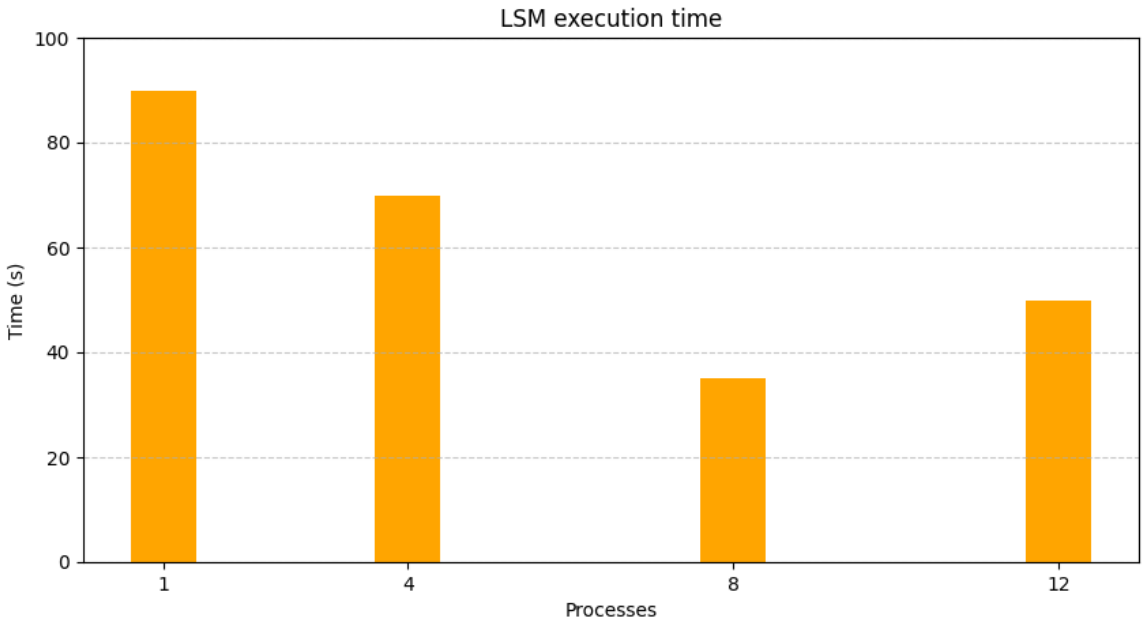

Figura B.2: Tiempo de ejecución medio en segundos en la búsqueda de hiperparámetros de LSM con distintos procesos.

B.3. Tiempos de ejecución en la búsqueda de NGRC

Por último, evaluaremos los resultados de los tiempos de ejecución para NGRC.

Primero, se realizaron 5 experimentos con 4, 8 y 12 procesos, que se compararon a su vez usando tan solo un proceso, obteniendo los siguientes resultados

1 Proceso	4 Procesos	8 Procesos	12 Procesos
850	780	700	650
840	775	710	640
860	785	690	660
845	780	705	655
855	780	695	645

Tabla B.5: Valores para diferentes números de procesos en NGRC en segundos.

Obteniendo los siguientes tiempos medios:

Procesos	Promedio
1	850
4	780
8	700
12	650

Tabla B.6: Promedios para diferentes números de procesos en NGRC en segundos.

De los valores obtenidos extraemos que el mejor tiempo medio se obtiene usando 12 procesos, reduciendo en casi 200 segundos el tiempo medio que se obtendría usando tan solo un proceso.

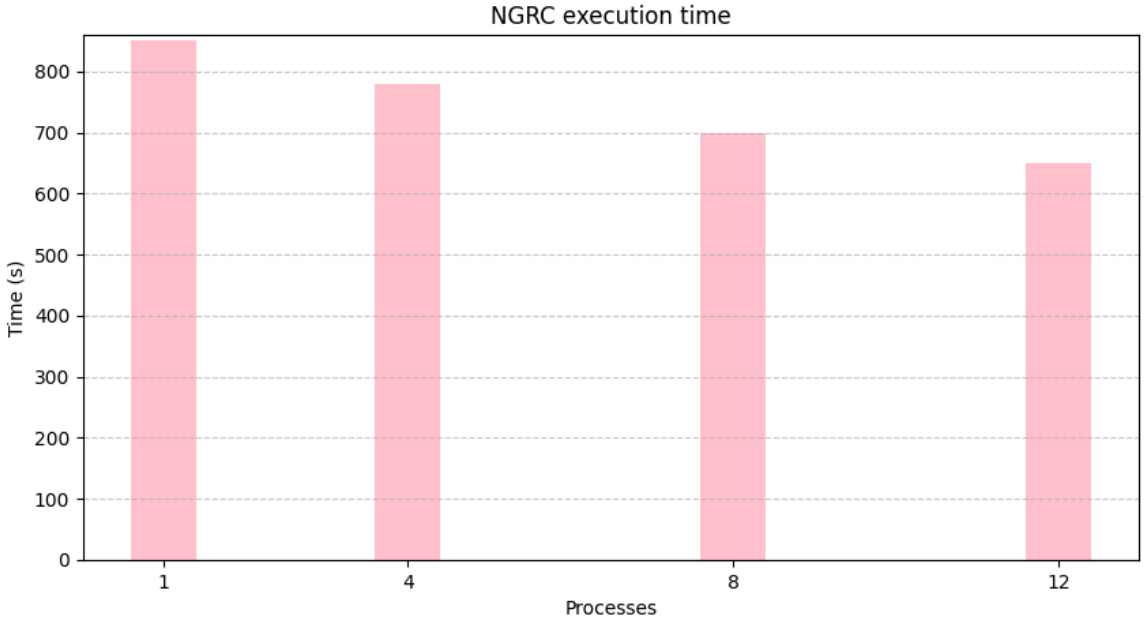

Figura B.3: Tiempo de ejecución medio en segundos en la búsqueda de hiperparámetros de NGRC con distintos procesos.

Apéndice C

Convenio

En la siguiente página se adjunta el documento firmado para la compartición de titularidad de derechos de propiedad intelectual de este TFG con la empresa AIR Institute.

Universidad de Valladolid

AIR INSTITUTE

ANEXO

DOCUMENTO PARA LA COMPARTICIÓN DE TITULARIDAD DE DERECHOS DE PROPIEDAD INTELECTUAL DE TFG/TFM EN CONVENIO

D./Dña. Míriam Arconada Pedriza, alumna de la Universidad de Valladolid y autor del TFG/TFM en convenio con la Fundación instituto internacional de investigación en inteligencia artificial y ciencias de la computación titulado *"Generación de señales cerebrales mediante técnicas de Reservoir Computing"*, expresa su decisión de compartir la titularidad de los derechos de propiedad intelectual de su TFG/TFM con la Fundación instituto internacional de investigación en inteligencia artificial y ciencias de la computación para realizar las siguientes acciones:

- Realizar publicaciones de los resultados del TFG/TFM siempre y cuando el alumno esté informado y, además, figure su nombre en dichas publicaciones.
- Realizar nuevos proyectos que impliquen modificaciones o ampliaciones del TFG/TFM.
- Reutilizar partes del TFG/TFM en otros proyectos que puedan tener fines comerciales, siempre y cuando lo permitan las licencias bajo las que se ha desarrollado el TFG/TFM.

En Valladolid, a 10 de mayo de 2024

Fdo.: