



UNIVERSIDAD DE GRANADA

PREDICCIÓN DE SERIES TEMPORALES APLICADO A MODELOS DE DISCIPLINADO DE RELOJES

DANIEL CÁRDENAS CASTRO

Trabajo Fin de Grado

Doble Grado en Ingeniería Informática y Matemáticas

Tutor

Eduardo Ros Vidal

FACULTAD DE CIENCIAS

E.T.S. DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, a 26 de junio de 2023

Predicción de Series Temporales aplicado a modelos de Disciplinado de Relojes

Daniel Cárdenas Castro

*A mis padres, a mi hermana y a María,
a quienes se lo debo todo.*

Agradecimientos

Un viaje está formado por varias etapas y todas deben concluir. Quería dar las gracias tanto a mis compañeros, como a mi pareja, los únicos que saben lo mucho que hemos sufrido en todos estos años. A veces, el esfuerzo sí tiene recompensa, y haber cursado este doble grado con vosotros ha sido el mayor aprendizaje que me podía llevar.

También, quiero agradecer a D. Eduardo Ros Vidal todo el empeño y las ganas que ha puesto en que este trabajo salga adelante, algo que empezó allá por junio de 2022 y que hoy llega a su fin. Gracias.

Predicción de Series Temporales aplicado a modelos de Disciplinado de Relojes

Daniel Cárdenas Castro

Resumen

En un mundo globalizado completamente interconectado cobran una gran importancia los algoritmos de disciplinado de relojes atómicos, que son precisos instrumentos usados para la sincronización de procesos de cualquier tipo. Al mismo tiempo, la inteligencia artificial se muestra como un campo en continuo desarrollo que demuestra ser aplicable para muchas tareas de diversa naturaleza. Ambos conceptos convergen cuando realizamos predicciones con series temporales y las aplicamos a datos que se obtienen de experimentos previos con algoritmos de disciplinado de relojes.

En este trabajo se presenta un estudio sobre las bases matemáticas necesarias para el desarrollo de modelos predictivos, como son los procesos estocásticos aplicados a series temporales o el algoritmo de descenso del gradiente. Además, se estudiarán relojes atómicos y el desarrollo de redes neuronales recurrentes con bloques LSTM (*long-short term memory*) desde cero, comenzando por la noción de neurona y pasando por conceptos clave como pueden ser el algoritmo de propagación hacia atrás a lo largo del tiempo o la optimización del entrenamiento de las redes neuronales mediante técnicas para la mejora de los resultados.

Finalmente, se construirán modelos predictivos basados en los datos de los relojes atómicos propiedad de la empresa Safran - Navigation & Timing, obtenidos mediante un acuerdo de confidencialidad entre ambas partes. Se tratará de potenciar el rendimiento de los modelos y se analizarán los resultados obtenidos.

Todo el código desarrollado durante el transcurso de este trabajo puede encontrarse en el [repositorio del proyecto](#); no siendo así para los datos, que mantienen su carácter privado.

Palabras Clave

Series Temporales, Aprendizaje Automático, LSTM, Relojes Atómicos, Algoritmo de Disciplinado

Time Series Prediction applied to Clock Discipline Modeling

Daniel Cárdenas Castro

Abstract

In a fully globalized interconnected world, clock discipline algorithms play a crucial role as they are precise instruments used for the synchronization of processes. In addition, artificial intelligence emerges as a continuously evolving field with multiple practical applications. The meeting point of these concepts occurs when making predictions using time series data previously obtained from experiments involving clock discipline algorithms.

This work presents a study on the mathematical foundations required for the development of predictive models, such as stochastic processes applied to time series or the popular gradient descent algorithm. Additionally, atomic oscillators and the construction of recurrent neural networks with long-short term memory (LSTM) blocks will be thoroughly examined, starting from the basic notion of neurons and exploring key concepts such as backpropagation through time or the optimization of neural network training through different techniques for better results.

Furthermore, predictive models will be built based on data obtained from atomic clocks owned by the company Safran - Navigation & Timing, under a NDA between both. The goal will be to enhance the model's performance and analyze the obtained results.

All the code developed throughout this work can be found in the [project repository](#), although the data itself will remain private.

Keywords

Time Series, Machine Learning, LSTM, Atomic Clocks, Discipline Algorithm

DECLARACIÓN DE ORIGINALIDAD

Yo, **Daniel Cárdenas Castro**, con DNI 77689216T, declaro que el presente Trabajo de Fin de Grado es original, no habiéndose utilizado fuentes sin ser citadas debidamente. De no cumplir con este compromiso, soy consciente de que, de acuerdo con la Normativa de Evaluación y de Calificación de los estudiantes de la Universidad de Granada, de 20 de mayo de 2013, esto conllevará automáticamente la calificación numérica de cero [...] independientemente del resto de las calificaciones que el estudiante hubiera obtenido. Esta consecuencia debe entenderse sin perjuicio de las responsabilidades disciplinarias en las que pudieran incurrir los estudiantes que plagien.



Fdo: Daniel Cárdenas Castro

Granada, a 26 de junio de 2023.

Yo, **Daniel Cárdenas Castro**, alumno del Doble Grado en Ingeniería Informática y Matemáticas de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación y de la Facultad de Ciencias de la Universidad de Granada**, con DNI 77689216T, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

A handwritten signature in black ink, appearing to read "Daniel Cárdenas Castro".

Fdo: Daniel Cárdenas Castro

Granada, a 26 de junio de 2023.

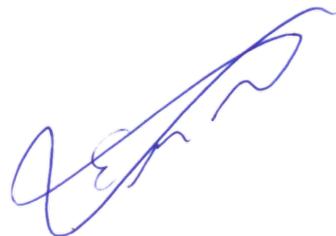
D. **Eduardo Ros Vidal**, profesor del Departamento de Ingeniería de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Predicción de Series Temporales aplicado a modelos de Disciplinado de Relojes*, ha sido realizado bajo su supervisión por **Daniel Cárdenas Castro**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada, a 26 de junio de 2023.

El director:



Fdo: Eduardo Ros Vidal

ÍNDICE GENERAL

1. ESTADO DEL ARTE	13
2. OBJETIVOS DEL TRABAJO	16
3. PLANIFICACIÓN	18
3.1. Planificación Temporal	18
3.2. Coste del Proyecto	20
 I. BASES MATEMÁTICAS	 21
4. SERIES TEMPORALES	22
4.1. Representación de Series Temporales	23
5. PROCESOS ESTOCÁSTICOS	24
5.1. Fundamentos de Estadística	24
5.1.1. Noción de Medibilidad	25
5.2. Procesos Estocásticos	29
5.3. Propiedades	31
6. DIFERENCIABILIDAD	34
6.1. Conceptos Básicos	35
6.2. Funciones Diferenciables	36
6.3. Regla de la Cadena	38
6.4. Derivadas Direccionales y Derivadas Parciales	41
6.5. Vector Gradiente	42
6.5.1. Interpretación del Gradiente	43
6.5.2. Algoritmo de Descenso del Gradiente	45
 II. TEORÍA DE INFORMÁTICA	 48
7. ALGORITMOS DE DISCIPLINADO DE RELOJES	49
7.1. El Internet de las Cosas	49
7.2. Reloj Atómico	50
7.3. Algoritmo de Disciplinado	51
7.3.1. Varianza de Allan	52
8. APRENDIZAJE AUTOMÁTICO Y REDES NEURONALES	55
8.1. Conceptos Básicos	57
8.1.1. Clasificación de Algoritmos	57
8.1.2. Medida del Error	60
8.1.3. Optimizadores de la Función de Coste	62

8.2.	Creación de un Modelo	63
8.3.	Aprendizaje Profundo	65
8.3.1.	Neuronas y Redes Neuronales	65
8.3.2.	Arquitectura de una red de Aprendizaje Profundo	68
8.3.3.	Propagación hacia atrás (Backpropagation)	69
9.	REDES NEURONALES RECURRENTES	74
9.1.	Introducción a las Redes Recurrentes	74
9.1.1.	Despliegue de Redes Recurrentes	75
9.1.2.	Ventana Temporal	76
9.2.	Propagación hacia atrás a través del tiempo (BPTT)	77
9.2.1.	Propagación hacia delante en una RNN	78
9.2.2.	Algoritmo de BPTT	79
9.3.	Arquitectura Long-Short Term Memory (LSTM)	81
9.3.1.	Introducción a las redes LSTMs	81
9.3.2.	Idea Principal de las redes LSTMs	82
9.3.3.	Funcionamiento de una celda LSTM	83
III. APLICACIÓN		89
10.	PRESENTACIÓN DE LOS DATOS	90
10.1.	Lectura y Procesamiento de los Datos	92
10.2.	Etapas de disciplinado	94
10.3.	Análisis Exploratorio de los Datos	97
11.	ESTUDIO DE CORRELACIÓN	103
11.1.	Correlación entre Atributos	104
11.2.	Autocorrelación, Autocorrelación Parcial y Correlación Cruzada . .	106
12.	MODELOS PREDICTIVOS CON REDES LSTM	112
12.1.	Diferenciación entre Etapas	112
12.2.	Técnicas de Entrenamiento para RNN	113
12.2.1.	Técnicas de Ajuste Dinámico	114
12.2.2.	Ajuste de Hiperparámetros	115
12.2.3.	Validación Cruzada leave one/two out	119
12.3.	Predicción del Disciplinado de una Etapa Completa	119
12.3.1.	Explicación del Problema	120
12.3.2.	Desarrollo de los Modelos para Etapas Completas	121
12.4.	Predicciones con Ventana Móvil por Etapas	125
12.4.1.	Explicación del Problema de Predicción de Errores	125
12.4.2.	Desarrollo de los Modelos con Ventana Temporal Móvil . . .	126
12.5.	Evaluación de los Resultados	135
13.	CONCLUSIONES Y TRABAJO FUTURO	138
13.1.	Repaso de los Objetivos del Proyecto	138

13.2. Trabajo Futuro	139
A. APÉNDICE	144
A.1. Otros Relojes Atómicos	144

ESTADO DEL ARTE

El estado del arte de este trabajo se divide en 4 partes, que son las 4 áreas de trabajo que actúan como base del proyecto.

APRENDIZAJE AUTOMÁTICO Y REDES NEURONALES

En los últimos años, el mundo de la inteligencia artificial y del aprendizaje automático, como parte fundamental de esta, han experimentado un crecimiento sin precedentes. Asimismo, las técnicas de aprendizaje automático desempeñan un papel crucial en la predicción de series temporales, donde podemos encontrar aplicaciones relevantes en el campo del disciplinado de relojes (osciladores) atómicos. Estos algoritmos se ha utilizado ampliamente para la detección y la corrección de variaciones y errores temporales en los relojes atómicos basados en datos históricos.

Existen numerosos libros fundamentales sobre el aprendizaje automático, pero destacan 2 entre el resto, como son el libro "Deep Learning" [1] de Ian Goodfellow, Yoshua Bengio y Aaron Courville, que proporciona una base teórica sólida en técnicas de predicción de series temporales; y el libro "Machine Learning" [2] de Tom Michael Mitchell, que nos da una visión extensa sobre las técnicas de aprendizaje automático más conocidas. Por otro lado, existen artículos como "Time Series Forecasting with Deep Learning: A Survey" [3] de Sheraz Ahmed, que ayudan a tener una visión general sobre el aprendizaje automático aplicado al estudio de las series temporales.

REDES NEURONALES RECURRENTES

Las redes neuronales recurrentes han demostrado ser altamente efectivas en la predicción de series temporales debido a su capacidad para manejar dependencias a largo plazo y modelar secuencias de datos con patrones complejos. Existen múltiples tipos de redes recurrentes, pero entre ellas destacan las redes LSTM [4] (*Long Short-Term Memory*, o memoria a corto-largo plazo), que se han ganado su popularidad en base a su capacidad de aprender, ordenar y recordar información relevante a lo largo del tiempo, sabiendo en cada instante donde se debe poner el foco.

A día de hoy, tenemos multitud de trabajos que tratan sobre RNN y LSTM, entre los más disruptivos, destacan "Learning to Forget: Continual Prediction with LSTM" [5] de Felix A. Gers y "Deep Learning with Long Short-Term Memory for Time Series Prediction" [4] de Yuxiu Hua, Zhao Zhifeng, Rongpeng Li, Xianfu Chen, Zhiming Liu y Honggang Zhang. Ambos fueron pioneros en el desarrollo y mejora de los modelos con bloques de neuronas LSTM.

ALGORITMOS DE DISCIPLINADO DE RELOJES ATÓMICOS

El disciplinado de relojes atómicos y todos sus algoritmos relacionados tienen que ver con la corrección y mejora de la precisión de los osciladores. En estos algoritmos de disciplinado pueden incluirse técnicas como el control PID (*proportional-integral-derivative*) o la retroalimentación de fase, entre otros. De cara a la comprensión de estos algoritmos, existen diversos trabajos relacionados entre los que podemos destacar "Overview of Time Synchronization for IoT Deployments: Clock Discipline Algorithms and Protocols" [6] de Hüseyin Yiğitler, Behnam Badihi y Riku Jäntti, o por otro lado "Adaptive hybrid clock discipline algorithm for the network time protocol" de David L. Mills. Ambos proporcionan una visión global de los conceptos básicos sobre osciladores y su disciplinado.

PREDICCIONES EN ALGORITMOS DE DISCIPLINADO

Este es el campo más desconocido de entre los 4. La aplicación de redes recurrentes, en particular LSTM, a la predicción de series temporales específicas de disciplinado de relojes atómicos es un área de investigación todavía en crecimiento. Existen algunos estudios que ya han investigado el posible uso de redes LSTM de cara a predecir las variaciones en el disciplinado de relojes atómicos. Entre

ellos, destaca principalmente el siguiente: "An Artificial Neural Network Model for Timescale Atomic Clock Ensemble Algorithm" [7] de R. Nandita, Shikha Maharrana, Rajathilagam Bijoy, Subramanya Ganesh y Subhalaksh Krishnamoorthy, que aborda el tema de nuestro estudio: la predicción de series temporales mediante redes recurrentes aplicada a los relojes atómicos.

2

OBJETIVOS DEL TRABAJO

Nuestro proyecto se enmarca dentro de una línea de trabajo que, como se indica en el título, trata la predicción de series temporales con aplicación a un campo en específico. Los objetivos del trabajo que se plantean van en consonancia con ello, tratando de finalizar el proyecto habiendo desarrollado una serie de modelos predictivos para el disciplinado de relojes atómicos que cumplan ciertos requisitos de optimalidad.

Teniendo esto en cuenta, se han identificado los siguientes objetivos, en torno a los que girará el desarrollo del trabajo:

- OBJ-1. Estudio teórico de las bases matemáticas relacionadas con procesos estocásticos y el algoritmo del descenso del gradiente. Lo primero, enfocado al estudio de series temporales; lo segundo, para la comprensión del origen del algoritmo y en qué se fundamenta.
- OBJ-2. Búsqueda y estudio en referencia a los relojes atómicos y su relevante en el contexto actual de interconexión y plena sincronización. Finalizar el estudio con un análisis de las bases de los algoritmos de disciplinado.
- OBJ-3. Repaso de las bases teóricas y directrices básicas para la construcción de redes neuronales recurrentes aplicando bloques de neuronas LSTM. Asimismo, analizar diferentes técnicas de optimización para el entrenamiento de las redes.
- OBJ-4. Construir modelos que pueda predecir las observaciones restantes en una etapa de disciplinado de un reloj atómico. El objetivo es encontrar el comportamiento adecuado que debe llevar el algoritmo para finalizar correctamente la etapa. Estos modelos se pueden utilizar para detectar problemas en el disciplinado de futuros algoritmos, si los patrones que se esperan en la conclusión de un experimento no se cumplen.

OBJ-5. Obtener modelos predictivos con capacidad para deducir futuros valores del algoritmo de disciplinado en el muy corto plazo. Estos modelos deben ser capaces de predecir errores de disciplinado, que pueden haber sido causados en parte por errores puntuales de un reloj atómico. Se usarán para, cuando se detecte una posible errata en el algoritmo por causas externas o fallo humano, poder corroborar cuál debería haber sido el valor nominal correcto, funcionando como una alarma de mal funcionamiento.

Al final del trabajo, en la conclusión del mismo, se incluye un análisis acerca de la consecución, o no, de los 5 objetivos anteriores.

3

PLANIFICACIÓN

En este capítulo, trataremos los aspectos más organizativos de nuestro proyecto, como son la planificación a lo largo del tiempo y el presupuesto necesario.

3.1 PLANIFICACIÓN TEMPORAL

A continuación, vamos a presentar la planificación que hemos llevado para poder desarrollar correctamente todo el trabajo. Para ello, vamos a dividir la totalidad del proyecto en subtareas que posteriormente representaremos en un Diagrama de Gantt. Asimismo, cada una de estas subtareas debe estar relacionada con alguno de los objetivos del trabajo.

La notación que vamos a usar simplificará la representación de la planificación en el diagrama y es la siguiente:

- La tarea i-ésima que además se corresponda con el objetivo j-ésimo se representa como **T-i / OBJ-j**. Si una tarea está asociada con varios objetivos, se separan los objetivos con comas: **T-i / OBJ-j,k**
- Las tareas que sean comunes a todos los objetivos se escriben **T-i / OBJ-A**, refiriéndonos a todos (all) los objetivos del proyecto.

Ahora, en la tabla (1) se indican las tareas que hemos realizado, junto con las fechas de inicio y fin y el número de horas estimado.

Las tareas de la tabla (1) quedan representadas gráficamente en el diagrama de gantt de la imagen (1), obteniendo un total de 386 horas de trabajo repartidas a lo largo de 6 meses.

Código	Descripción de la Tarea	Fecha Inicio	Fecha Fin	Horas
T-1 / OBJ-A	Documentación sobre el contexto del trabajo	23-01-23	03-02-23	13
T-2 / OBJ-1	Estudio sobre procesos estocásticos	04-02-23	20-02-23	11
T-3 / OBJ-1	Análisis de la diferenciabilidad (descenso del gradiente)	20-02-23	12-03-23	18
T-4 / OBJ-2	Documentación sobre el disciplinado de relojes	12-03-23	30-03-23	22
T-5 / OBJ-3	Documentación sobre aprendizaje automático y creación de modelos	02-04-23	11-04-23	25
T-6 / OBJ-3	Documentación sobre las redes recurrentes	12-04-23	21-04-23	16
T-7 / OBJ-3	Documentación sobre el funcionamiento de los bloques LSTMs	21-04-23	10-05-23	10
T-8 / OBJ-3,4,5	Documentación sobre técnicas de optimización de RRNN	10-05-23	31-05-23	8
T-9 / OBJ-4,5	Preparación y preprocesado de los datos	15-03-23	06-04-23	26
T-10 / OBJ-4,5	Ánalysis exploratorio de datos	06-04-23	09-04-23	18
T-11 / OBJ-4,5	Estudio de correlación	09-04-23	12-04-23	10
T-12 / OBJ-4	Modelos de predicción para etapas completas	12-04-23	24-04-23	24
T-13 / OBJ-5	Modelos de predicción con ventanas móviles	24-04-23	17-05-23	82
T-14 / OBJ-4	Optimización con validación cruzada para etapas completas	18-05-23	27-05-23	6
T-15 / OBJ-5	Optimización con validación cruzada para ventanas móviles	28-05-23	13-06-23	21
T-16 / OBJ-4,5	Evaluación y Análisis de Resultados	13-06-23	18-06-23	12
T-17 / OBJ-A	Revisión final de la memoria	14-06-23	26-06-23	62

Cuadro 1: Información sobre las subtareas del proyecto

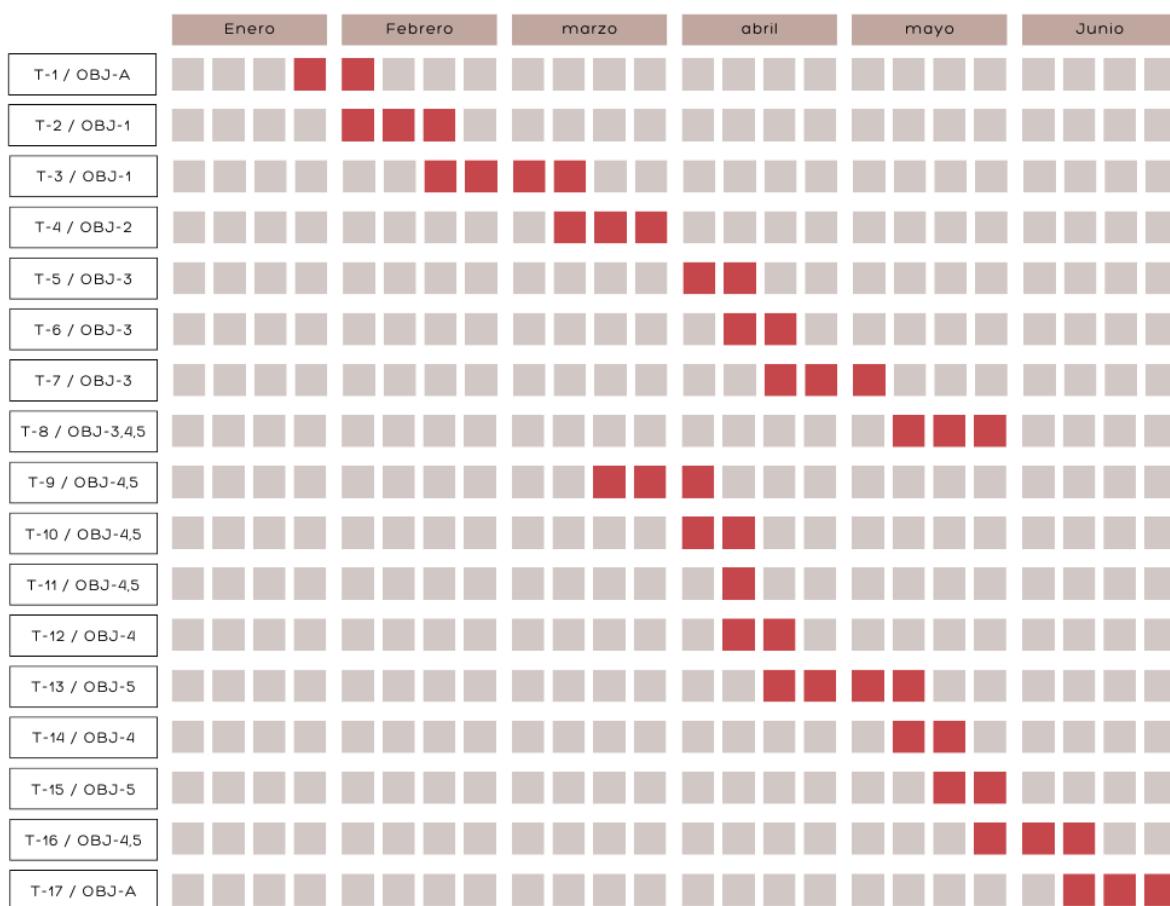


Figura 1: Diagrama de Gantt del proyecto

3.2 COSTE DEL PROYECTO

Nuestro proyecto no incluye ningún coste adicional de equipo, ni software, ni hardware de pago. Por ello, para estimar el coste únicamente debemos tener en cuenta el coste de personal. Para poder realizar el cálculo, suponemos un salario bruto de 24000€, que es el salario bruto medio de un analista de datos junior en nuestro país. Sabiendo este dato, podemos estimar que el salario bruto por hora es de unos 14,4€. De esta manera, el coste del proyecto es el siguiente:

$$14,4 \frac{\text{€}}{\text{hora}} \times 386 \text{ horas} = 5558,4\text{€}$$

Parte I

BASES MATEMÁTICAS

Se presenta la teoría matemática para su posterior uso en la parte de aplicación. Explicaremos la manera de representar series temporales y cómo los procesos estocásticos nos ayudan a entenderlas. Acabamos con breves nociones de diferenciabilidad, necesarias para la comprensión de los algoritmos para la optimización de redes neuronales.

4

SERIES TEMPORALES

En este capítulo de la sección de matemáticas, vamos a explicar algunas nociones básicas sobre series temporales, así como algunas cuestiones de notación, para que la lectura posterior pueda ser más fluida.

Definición 1. Se define una **Serie Temporal** como una secuencia de mediciones u observaciones tomadas en orden cronológico y equidistantes en el tiempo sobre una o varias características de una unidad observable.

En ocasiones, nos referiremos a las series temporales como **series**, por simplicidad.

Las observaciones mencionadas son los datos recogidos, por lo que una serie temporal es una secuencia de datos ordenados y equidistantes cronológicamente en el tiempo. Asimismo, el término unidad observable hace referencia a fenómenos o aparatos con características medibles en el tiempo. A continuación, enumeramos algunos ejemplos de series temporales.

- Volumen diario de lluvia de un territorio a lo largo de un año
- Consumo de electricidad de un edificio por horas
- Índice de paro en un país
- Índices bursátiles (IBEX35, DAX30, etc)

Diferenciamos dos tipos de series temporales. Para ello, se introduce la siguiente definición.

Definición 2. Una serie temporal se dice **univariante** (o escalar) si en cada observación se mide una única característica; mientras que una serie temporal es **multivariante** (o vectorial) si en cada observación se miden varias características.

4.1 REPRESENTACIÓN DE SERIES TEMPORALES

Ahora, mostramos de qué manera vamos a representar estos dos tipos de series matemáticamente.

Sea x_t la observación en el instante $t = 1, \dots, N$, representamos una **serie temporal univariante** de N observaciones, de cualesquiera de las siguientes maneras:

- x_1, x_2, \dots, x_N
- $\{x_t\}_{t=1,\dots,N}$
- $\{x_t\}_{t=1}^N$
- $\{x_t : t = 1, \dots, N\}$

Matricialmente, podría verse como

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Por otro lado, para una **serie temporal multivariante** de N observaciones con M características o atributos, esta puede representarse en forma de matriz, pues ahora cada observación es un vector (fila) de características. Es decir, para cada x_t con $t = 1, \dots, N$, se tiene que

$$x_t = \begin{bmatrix} x_{t1} & x_{t2} & \dots & x_{tM} \end{bmatrix}$$

Entonces, podemos construir la **matriz asociada** a dicha serie temporal, como la unión de todos los vectores fila correspondientes a sus N observaciones, obteniendo

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1M} \\ x_{21} & x_{22} & \dots & x_{2M} \\ \vdots & \vdots & & \vdots \\ x_{N1} & x_{N2} & \dots & x_{NM} \end{bmatrix}$$

donde el valor x_{ij} es la observación en el instante i de la variable u atributo j .

5

PROCESOS ESTOCÁSTICOS

En este capítulo, introduciremos el concepto de **proceso estocástico**, que posteriormente nos ayudará con el tratamiento de las características asociadas a las series temporales.

Antes, vamos a presentar algunas nociones básicas de estadística, que serán necesarias para comprender la naturaleza de un proceso estocástico.

5.1 FUNDAMENTOS DE ESTADÍSTICA

La estadística tiende a ir de la mano de la probabilidad, y para poder hablar de probabilidad, debemos situarnos en un espacio probabilístico.^[8] Aquí podremos definir variables aleatorias (habiendo trabajando previamente la noción de medibilidad), que son la base de los procesos estocásticos.

Definición 3. Un Espacio Probabilístico es una tripleta (Ω, \mathcal{F}, P) de forma que

- Ω es un conjunto de sucesos elementales no vacío, denominado espacio muestral.
- \mathcal{F} es una σ -álgebra formada por sucesos aleatorios. \mathcal{F} está compuesta por subconjuntos del espacio muestral; por tanto, para todo $A \in \mathcal{F}$, se cumple que $A \subset \Omega$. Para ser σ -álgebra, debe cumplir:
 1. $\emptyset, \Omega \in \mathcal{F}$
 2. \mathcal{F} es cerrada con respecto al complementario; esto es,

$$\forall A \in \mathcal{F}, A' = \Omega - A \in \mathcal{F}$$

3. \mathcal{F} es cerrada con respecto a uniones numerables; esto es,

$$\forall \{A_n\}_{n \in \mathbb{N}}, \bigcup_{n \in \mathbb{N}} A_n \in \mathcal{F}$$

- P es una función de probabilidad; es decir, $P: \mathcal{F} \rightarrow [0, 1]$, donde para cualquier $A \in \mathcal{F}$, $P(A)$ es la probabilidad del suceso $A \in \mathcal{F}$. Además, P debe cumplir:

1. $P(\emptyset) = 0$
2. $P(\Omega) = 1$
3. P cumple la propiedad de la σ -aditividad; esto es, para una familia de elementos $\{A_n\}_{n \in \mathbb{N}} \subset \mathcal{F}$ disjuntos dos a dos, se tiene que

$$P\left(\bigcup_{n \in \mathbb{N}} A_n\right) = \sum_{n \in \mathbb{N}} P(A_n)$$

En las siguientes páginas, vamos a trabajar siempre sobre un espacio de probabilidad. Este será nuestro marco de partida, que nos servirá para ir construyendo las bases de los siguientes apartados.

5.1.1 Noción de Medibilidad

Ahora, debemos dar un pequeño paso atrás para presentar lo que denominamos **espacio medible**, que es una pareja (Ω, \mathcal{F}) tal y como se han definido anteriormente; esto es, Ω es nuestro espacio muestral y \mathcal{F} su σ -álgebra asociada. Sobre un espacio medible arbitrario, se introduce la siguiente definición

Definición 4. Sean dos espacios medibles $(\Omega_1, \mathcal{F}_1)$ y $(\Omega_2, \mathcal{F}_2)$. Diremos que una aplicación $f: \Omega_1 \rightarrow \Omega_2$ es medible, si para cualquier elemento $A \in \mathcal{F}_2$, se cumple que $f^{-1}(A) \in \mathcal{F}_1$

En general, la aplicación f anterior puede escribirse como $f: (\Omega_1, \mathcal{F}_1) \rightarrow (\Omega_2, \mathcal{F}_2)$. En tal caso, podemos decir que f es **medible** si, y solamente si, $f^{-1}(\mathcal{F}_2) \in \mathcal{F}_1$.

Para abordar los siguientes conceptos y poder concretar la generalización anterior de aplicación medible, debemos tomar intuición sobre el concepto de σ -álgebra. Una σ -álgebra sobre un conjunto establece una manera de medir los

elementos (o subconjuntos de elementos, mejor dicho) de ese conjunto. Gracias a esta manera de medir, podemos posteriormente asignar una probabilidad mediante una función de probabilidad adecuada.

Una vez familiarizados con la utilidad real de definir una σ -álgebra, vamos a presentar la probablemente más usada del mundo: la **σ -álgebra de Borel¹**. Esta está definida sobre los números reales y se suele representar como $\mathcal{B}(\mathbb{R})$. Por definición, es la mínima σ -álgebra de subconjuntos de \mathbb{R} que contiene todos los abiertos² de \mathbb{R} .[8]

La σ -álgebra de Borel nos proporciona un conjunto de elementos de la recta real que podemos medir de manera más o menos asequible. Sin embargo, la siguiente pregunta que intuitivamente podemos cuestionarnos es qué hay exactamente en ese conjunto. Veámoslo.

Ejemplo 1. Sabemos que cualquier σ -álgebra es cerrada para uniones numerables y complementarios, entonces basta tener en cuenta las Leyes de Morgan³ para deducir que también lo es para intersecciones numerables y diferencias.[8]

i. Todos los números reales están en $\mathcal{B}(\mathbb{R})$, pues para cualquier $a \in \mathbb{R}$,

$$\{a\} = \bigcap_{n \in \mathbb{N}} \left(a - \frac{1}{n}, a + \frac{1}{n}\right) \in \mathcal{B}(\mathbb{R})$$

ii. Cualquier intervalo cerrado $[a, b]$ está en $\mathcal{B}(\mathbb{R})$, puesto que puede expresarse como

$$[a, b] = \{a\} \cup (a, b) \cup \{b\} \in \mathcal{B}(\mathbb{R})$$

iii. Los intervalos semiabiertos también pertenecen a $\mathcal{B}(\mathbb{R})$ pues

$$[a, b) = \{a\} \cup (a, b) \in \mathcal{B}(\mathbb{R})$$

$$(a, b] = (a, b) \cup \{b\} \in \mathcal{B}(\mathbb{R})$$

¹ Émile Borel (Saint-Affrique, 07/01/1871 - París, 03/02/1956) fue un político y matemático de origen francés. Es considerado uno de los pioneros de la llamada Teoría de la medida y de todas las aplicaciones de esta al campo probabilístico

² El conjunto de todos los abiertos de \mathbb{R} puede representarse como $\bigcup_{n \in \mathbb{N}} (a_n, b_n)$ con $a_n, b_n \in \mathbb{R}$ para cualquier n natural

³ Augustus de Morgan (Madurai, India, 27/06/1806 - Londres, 18/03/1871) fue un lógico y matemático de origen indio y nacionalidad británica. Se dio a conocer como profesor en Londres, en el University College, entre los años 1828 y 1866; además, fue el primer presidente de la Sociedad Matemática de Londres.

iv. El conjunto de los números racionales \mathbb{Q} pertenece a $\mathcal{B}(\mathbb{R})$. Teniendo en cuenta que \mathbb{Q} es numerable, se tiene que

$$\mathbb{Q} = \{w_n / n = 1, 2, \dots\} = \bigcup_{n \in \mathbb{N}} \{w_n\} \in \mathcal{B}(\mathbb{R})$$

v. El conjunto de los números racionales \mathbb{Q} pertenece a $\mathcal{B}(\mathbb{R})$, puesto que puede expresarse como diferencia de elementos de $\mathcal{B}(\mathbb{R})$. En este caso sería

$$I = \mathbb{R} - \mathbb{Q} \in \mathcal{B}(\mathbb{R})$$

Ahora que hemos manejado brevemente esta σ -álgebra, veamos qué es una función medible, definición que nos derivará inmediatamente al concepto de **variable aleatoria**.

Definición 5. Una función $X : (\Omega, \mathcal{F}) \rightarrow (\mathbb{R}, \mathcal{B}(\mathbb{R}))$ se dice medible si satisface alguna de las condiciones siguientes:

$$1. \text{ Para cualquier } a \in \mathbb{R}, \{\omega \in \Omega / X(\omega) < a\} = X^{-1}((-\infty, a)) \in \mathcal{F} \quad (5.1)$$

$$2. \text{ Para cualquier } a \in \mathbb{R}, \{\omega \in \Omega / X(\omega) \leq a\} = X^{-1}((-\infty, a]) \in \mathcal{F} \quad (5.2)$$

$$3. \text{ Para cualquier } a \in \mathbb{R}, \{\omega \in \Omega / X(\omega) > a\} = X^{-1}((a, +\infty)) \in \mathcal{F} \quad (5.3)$$

$$4. \text{ Para cualquier } a \in \mathbb{R}, \{\omega \in \Omega / X(\omega) \geq a\} = X^{-1}([a, +\infty)) \in \mathcal{F} \quad (5.4)$$

$$5. \text{ Para cualquier } B \in \mathcal{B}(\mathbb{R}), X^{-1}(B) \in \mathcal{F} \quad (5.5)$$

Una función medible sobre un espacio probabilístico (Ω, \mathcal{F}, P) como en la definición anterior, se denomina **Variable Aleatoria (o estocástica) Real**.

Proposición 1. Las condiciones (5.1), (5.2), (5.3), (5.4) y (5.5) son equivalentes.

Demostración. Las equivalencias entre las primeras condiciones (5.1), (5.2), (5.3) y (5.4) son triviales y pueden deducirse al realizar el complementario de tales intervalos en \mathbb{R} . De forma que por ejemplo $(-\infty, a)' = \mathbb{R} - (-\infty, a) = [a, \infty)$ o también podemos hacer $(-\infty, a]' = \mathbb{R} - (-\infty, a] = (a, \infty)$.

La equivalencia que suscita mayor interés es la (5.5) con respecto a las demás. Probémosla.

Se puede probar la expresión (5.1) a partir de (5.5). Supongamos (5.5) y tomemos un elemento de $\mathcal{B}(\mathbb{R})$ de la forma $(-\infty, a)$. Entonces se cumple que $X^{-1}((-\infty, a)) \in \mathcal{F}$ para cualquier valor $a \in \mathbb{R}$.

Ahora, probamos (5.5) a partir de la expresión (5.1). En primer lugar, definimos $\mathcal{A} = \{E \subset \mathbb{R} / X^{-1}(E) \in \mathcal{F}\}$. Veamos ahora que \mathcal{A} tiene estructura de σ -álgebra sobre elementos de \mathbb{R} :

1. Como $\emptyset \in \mathbb{R}$, tomando $X^{-1}(\emptyset) = \emptyset$ y sabiendo que $\emptyset \in \mathcal{F}$ por ser \mathcal{F} σ -álgebra, entonces llegamos a que $\emptyset \in \mathcal{A}$.

Análogamente, tomando el conjunto total \mathbb{R} y haciendo $X^{-1}(\mathbb{R}) = \Omega$, como $\Omega \in \mathcal{F}$, entonces se obtiene que $\mathbb{R} \in \mathcal{A}$.

2. Tomamos un elemento cualquiera $E \in \mathcal{A}$ y calculamos la preimagen de su complementario por X . Por la continuidad de la función inversa, obtenemos que $X^{-1}(E') = (X^{-1}(E))'$ y como $X^{-1}(E) \in \mathcal{F}$ su complementario también; es decir, $(X^{-1}(E))' \in \mathcal{F}$.

Esto último implica directamente que $E' \in \mathcal{A}$ y como $E \in \mathcal{A}$ era arbitrario, entonces \mathcal{A} es cerrada para la formación del complementario.

3. Tomamos un conjunto numerable de elementos $E_n \in \mathcal{A}$ y de nuevo por la continuidad de la función inversa, obtenemos que

$$X^{-1} \left(\bigcup_{n \in \mathbb{N}} E_n \right) = \bigcup_{n \in \mathbb{N}} X^{-1}(E_n) \in \mathcal{F}$$

por ser \mathcal{F} cerrada para las uniones numerables. Esto implica que $\bigcup_{n \in \mathbb{N}} E_n \in \mathcal{A}$ y por tanto, que \mathcal{A} es cerrada para uniones numerables.

Así, los puntos anteriores (1), (2) y (3) prueban que \mathcal{A} es una σ -álgebra sobre \mathbb{R} .

Supongamos que para cualquier $a \in \mathbb{R}$, se cumple que $X^{-1}((-\infty, a)) \in \mathcal{F}$. Esto resulta en que todos los intervalos de la forma $(-\infty, a) \in \mathcal{A}$. Entonces, teniendo en cuenta que la σ -álgebra $\mathcal{B}(\mathbb{R})$ es la "más pequeña" de todas las que contienen los intervalos del tipo $(-\infty, a)$, entonces se deduce que $\mathcal{B}(\mathbb{R}) \subset \mathcal{A}$.

Para concluir, basta tomar un elemento arbitrario $B \in \mathcal{B}(\mathbb{R})$ y entonces, como $B \in \mathcal{A}$, se cumple que $X^{-1}(B) \in \mathcal{F}$. □

5.2 PROCESOS ESTOCÁSTICOS

Ahora que conocemos el concepto de Variable Aleatoria y tenemos algo de noción sobre lo que representan más allá del plano abstracto, podemos comenzar a trabajar con los procesos estocásticos propiamente dichos. Comenzamos con la definición. [9]

Definición 6. Sea la tripleta (Ω, \mathcal{F}, P) un espacio de probabiñístico, se define un **proceso estocástico** como una colección $X = \{X_t / t \in T\} = \{X_t\}_{t \in T}$ de variables aleatorias definidas en dicho espacio probabilístico.

Algunos pequeños apuntes sobre la definición anterior:

- El conjunto T se denomina **espacio paramétrico** y suele representar instantes de tiempo; o bien, discretos, o bien, continuos. Algunos ejemplos podrían ser: $T \subseteq \mathbb{N}$, $T = \mathbb{R}$, $T = [0, \infty)$ ó $T = [a, b]$.
- Fijado un elemento $\omega \in \Omega$, la función $X(\omega) : T \rightarrow \mathbb{R}$ que asigna un número real $X_t(\omega)$ a cada valor $t \in T$ se denomina **realización o trayectoria muestral del proceso** asociada al elemento $\omega \in \Omega$.

Notación 1. Nos podemos referir a un proceso estocástico de cualquiera de las siguientes maneras.

- $\{X_t / t \in T\}$, donde cada X_t es una variable aleatoria
- $\{X(t) / t \in T\}$, donde las funciones X son funciones de dos variables que vienen definidas como $X : T \times \Omega \rightarrow \mathbb{R}$ que a cada par (t, ω) le asigna el valor de la variable aleatoria X_t evaluada en ω ; esto es, el valor $X_t(\omega) \in \mathbb{R}$

En este segundo caso, como tenemos una función de dos variables sobre $T \times \Omega$, tenemos 2 interpretaciones interesantes:

$$\forall t \in T, X_t : (\Omega, \mathcal{F}) \rightarrow (\mathbb{R}, \mathcal{B}(\mathbb{R})) \text{ son variables aleatorias}$$

o bien

$$\forall \omega \in \Omega, X(\omega) : T \rightarrow \mathbb{R} \text{ son trayectorias del proceso}$$

Existen diferentes maneras de clasificar procesos estocásticos. En primer lugar, podemos hacerlo en función de la condición de numerabilidad del conjunto T . Si T es numerable (o contable), entonces existe una biyección entre T y \mathbb{N} de forma que un proceso estocástico $X = \{X_t\}_{t \in T}$ puede expresarse como $\{X_n / n \in \mathbb{N}\} = (X_1, X_2, \dots)$. En tal caso, diremos que X es un proceso (estocástico) **discreto**

o discreto en el tiempo (discrete-time process). Por el contrario, si el conjunto T no es numerable, entonces diremos que X es un **proceso continuo o continuo en el tiempo**. [10]

En este trabajo, nos ocuparemos de los procesos discretos, puesto que estos son los que, por su naturaleza, van ligados al estudio de series temporales. De esta forma, en posteriores capítulos veremos que para cualquier $t \in \mathbb{N}$ la variable aleatoria X_t representa la observación del fenómeno que estemos tratando, en el instante t . Así, la idea es que tomando un proceso estocástico cualquiera $X = \{X_t / t \in T\}$ siendo $T = \mathbb{Z}$, podamos tomar un subconjunto de N variables aleatorias de forma que podamos definir una serie temporal como $(X_{t+1}, X_{t+2}, \dots, X_{t+N})$ para algún $t \in \mathbb{Z}$.

A continuación, vamos a definir una serie de conceptos relacionados con procesos estocásticos que serán de gran ayuda a la hora de estudiar, entre otras cosas, la correlación entre series temporales.[11]

Definición 7. Sea un proceso estocástico $X = \{X_t / t \in T\}$, definimos:

- Su **Función Media**, como la aplicación μ que asocia a cada instante $t \in T$, la esperanza de la variable aleatoria X_t correspondiente

$$\begin{aligned}\mu : T &\rightarrow \mathbb{R} \\ t &\rightarrow \mu(t) = E[X_t]\end{aligned}$$

- Los **Momentos no centrados de orden k** , con $k > 1$, como la aplicación μ_k que asocia a cada instante $t \in T$, la esperanza de la variable aleatoria X_t^k correspondiente

$$\begin{aligned}\mu_k : T &\rightarrow \mathbb{R} \\ t &\rightarrow \mu_k(t) = E[X_t^k]\end{aligned}$$

- Su **Función Varianza**, como la aplicación σ^2 que asocia a cada instante $t \in T$, la diferencia entre el momento de orden 2 y la función media al cuadrado

$$\begin{aligned}\sigma^2 : T &\rightarrow \mathbb{R} \\ t &\rightarrow \sigma^2(t) = \mu_2(t) - (\mu(t))^2\end{aligned}$$

- La **Función Correlación**, como la aplicación R que asocia a cada par de instantes $t_1, t_2 \in T$, la esperanza de la variable aleatoria producto $X_{t_1}X_{t_2}$

$$R : T \times T \rightarrow \mathbb{R}$$

$$(t_1, t_2) \rightarrow R(t_1, t_2) = E[X_{t_1}X_{t_2}]$$

- La **Función Covarianza**, como la aplicación C que asocia a cada par de instantes $t_1, t_2 \in T$, la esperanza de la variable aleatoria producto de las diferencias de cada variable menos su función media

$$C : T \times T \rightarrow \mathbb{R}$$

$$(t_1, t_2) \rightarrow C(t_1, t_2) = E[(X_{t_1} - \mu(t_1))(X_{t_2} - \mu(t_2))]$$

5.3 PROPIEDADES

En la sección anterior, hemos introducido nociones básicas sobre procesos estocásticos, que serán de gran utilidad como base matemática de las construcciones que veremos a continuación, ya más enfocadas en el estudio de series temporales.

En primer lugar y a raíz de lo explicado anteriormente, podemos retomar el concepto de serie temporal y adaptarlo con esta nueva definición:

Una **Serie Temporal** es una sucesión de observaciones sobre una variable, que han sido recogidas secuencialmente en el tiempo. Dichas observaciones se suelen tomar en equiespaciados instantes de tiempo de manera uniforme; de esta manera, los datos se consideran dependientes entre sí si proceden del mismo proceso, puesto que el orden de observación es relevante.

Ahora que conocemos la estrecha **relación entre series temporales y procesos estocásticos**, de ahora en adelante, denotaremos indistintamente a ambos, teniendo en cuenta que siempre nos vamos a referir a procesos estocásticos discretos en el tiempo, pues las series que trataremos en posteriores apartados serán de este tipo.

Comenzamos a continuación introduciendo algunas definiciones propias de series temporales, que debemos tener en cuenta.

Definición 8. Una serie temporal $\{X_t / t \in \mathbb{Z}\}$ es **Estacional** cuando su valor esperado varía con una pauta cíclica; esto es,

$$\forall t \in T, \mu(t) = E[X_t] = E[X_{t+S}] = \mu(t + S)$$

En tal caso, diremos que X es estacional con periodo S .

Ejemplo 2.

1. Una serie temporal que mida la luz solar recibida por un sensor a lo largo del día, es susceptible de tener un periodo de 24 horas.
2. La serie que registre observaciones del aforo de un centro comercial, es probable que tenga un periodo de 7 días, pues los fines de semana registra de media más ocupación que durante la semana.
3. Una serie que mida la ocupación de los pasillos de una facultad de universidad, podrá tener un periodo de 1 hora, por ser las horas punta cuando más estudiantes se encuentran fuera de las aulas.

Nótese que una misma serie temporal puede ser estacional con varios períodos distintos. Por ejemplo, el caso (1) anterior, también podría tener un periodo de 1 año. Además, en la naturaleza no es normal que se cumpla la definición de forma rigurosa. Debido a factores externos, consideramos una serie estacional si sus valores son similares con periodo S .

Las definiciones siguientes son muy parecidas, pero no deben confundirse. Ambas tratan el concepto de estacionariedad de una serie temporal.

Definición 9. Una serie temporal es **Estrictamente Estacionaria** si su distribución de probabilidad es independiente del tiempo.

Definición 10. Una serie temporal $\{X_t / t \in \mathbb{Z}\}$ es **Débilmente Estacionaria** si $\mu(t)$ y $\sigma^2(t)$ son constantes para todo $t \in \mathbb{Z}$ y además la covarianza $C(t_1, t_2)$ solo depende de la diferencia $t_2 - t_1$.

El concepto de estacionariedad está relacionado con la **tendencia de una serie temporal**. Diremos que esta es estacionaria cuando las observaciones son estables durante el tiempo. Esto se puede ver reflejado gráficamente, pues los valores de la serie suelen oscilar alrededor de una media que se mantiene constante. Asimismo, la variabilidad con respecto a tal media (el significado práctico de la varianza) también se mantiene constante en el tiempo.

Los conceptos anteriores de estacionalidad y estacionariedad relacionados con una serie temporal nos hablan de las características que puede tener dicha serie. La valoración de estas características, entre otras, nos permitirá saber la bondad de las posibles predicciones que llevemos a cabo sobre futuros valores de una serie.

Además de la estacionalidad y la estacionariedad, introducimos un nuevo concepto que se refiere a la parte más impredecible de una serie temporal".

Definición 11. Una serie temporal $\{X_t / t \in \mathbb{Z}\}$ es de **Ruido Blanco** si verifica:

1. $\mu(t) = 0$, para todo $t \in \mathbb{Z}$
2. $\sigma^2(t) = k$ constante, para todo $t \in \mathbb{Z}$
3. $C(t_1, t_2) = 0$, para todo $t_1, t_2 \in \mathbb{Z}, t_1 \neq t_2$

Notación 2. Llamaremos a las series de ruido blanco por $\varepsilon = \{\varepsilon_t / t \in \mathbb{Z}\}$

Nótese que esta clase de series temporales son un caso particular de serie estacionaria, es más, son el caso más simple.

6

DIFERENCIABILIDAD

En este capítulo, trataremos algunos conceptos clave en el campo de la diferenciabilidad, pues el estudio del cálculo de **derivadas** y **gradientes** ha sido la base matemática para el desarrollo de potentes técnicas de optimización. Entre ellas, se encuentran el descenso del gradiente (*gradient descent*, en inglés) y la retropropagación (*backpropagation*, en inglés), que son dos herramientas indispensables en multitud de algoritmos de aprendizaje automático y redes neuronales artificiales. Ambas serán tratadas más adelante cuando trabajemos con la teoría del entrenamiento de redes neuronales.

Grosso modo, la idea que desencadenó el estudio de la **diferenciabilidad** es la capacidad de medir el cambio de una función en un instante dado. El inicio de este campo de la matemática se remonta a los primeros pasos del cálculo y las matemáticas modernas. Uno de los primeros matemáticos investigadores en desarrollar la noción de derivada (herramienta clave de la diferenciabilidad), fue Isaac Newton¹ en el siglo XVII, fundando el cálculo infinitesimal. A raíz de ahí, otros muchos matemáticos notables, como Gottfried Leibniz², Augustin-Louis Cauchy³ y Bernhard Riemann⁴, entre otros, continuaron desarrollando y refinando la teoría de la diferenciabilidad.[12]

¹ Isaac Newton (Lincolnshire, 25/12/1642 - Londres, 20/03/1727) fue un matemático, físico y teólogo de origen inglés. Es el autor de los libros *Principia* y co-autor del mayor desarrollo histórico del cálculo integral y del cálculo diferencial; que además, empleó para la formulación de sus leyes de la física y astronomía.

² Gottfried Leibniz (Leipzig, 01/07/1646 - Hannover, 14/11/1716) fue un matemático, teólogo, filósofo y político de origen alemán, que es conocido como el último genio universal. Fue el primero en introducir el término de *función*, además de atribuirse el descubrimiento del cálculo infinitesimal.

³ Augustin-Louis Cauchy (París, 21/08/1789 - Lyon, 23/05/1857) fue un matemático de origen francés, miembro de la prestigiosa Academia de Ciencias de Francia. Se le atribuye descubrimientos en el campo del análisis matemático, tales como la introducción de las funciones holomorfas en análisis complejo o los criterios de convergencia para series y series de potencias.

⁴ Bernhard Riemann (Breselenz, Alemania, 17/09/1826 - Verbania, Italia, 20/07/1866) fue un matemático de origen alemán que contribuyó ampliamente al análisis y geometría diferencial. Algunos de sus avances fueron claves de cara al desarrollo más avanzado de la relatividad general.

En términos de la teoría de redes neuronales, la sensibilidad de las salidas de la red en relación a los cambios en sus entradas, están basadas en la diferenciabilidad. Además, el algoritmo de **descenso del gradiente** permite actualizar los pesos de las neuronas durante el entrenamiento, ajustándolos de forma que se busca minimizar el error o la función de pérdida de la red. Este algoritmo, unido a alguno de sus optimizadores (que veremos posteriormente) ayuda a evitar que nos quedemos estancados en mínimos locales de la función de coste durante el proceso de optimización de la red.

Por otro lado, al llevar a cabo el algoritmo de **retropropagación** o *backpropagation*, la diferenciabilidad se usa para propagar el error a través de la red; esto es, desde la salida hasta la entrada de la red, por todas las neuronas, actualizando los pesos para que la salida de la red se acerque al máximo al resultado óptimo esperado.

Estas dos herramientas son las más conocidas para que una red neuronal pueda adaptarse a los datos de entrada, maximizando su **capacidad de predicción y clasificación** ante nuevos datos con los que no haya sido entrenada.

6.1 CONCEPTOS BÁSICOS

Nuestro estudio del cálculo diferencial en funciones multivariable lo iniciamos en un contexto genérico, a partir de la definición de diferenciabilidad de funciones entre espacios normados arbitrarios. Para ello y a lo largo de toda esta sección de nuestro trabajo, haremos uso conjunto de [12], [13] y [14].

Comenzamos recordando, de forma muy básica, el concepto de derivada utilizado en funciones reales con variable real. Si tomamos A como un subconjunto de \mathbb{R} no vacío y f una función $f : A \rightarrow \mathbb{R}$; entonces diremos que f es derivable en $a \in A \cap A'$ ⁵ cuando exista un número real $\lambda \in \mathbb{R}$ que verifique

$$\lim_{x \rightarrow a} \frac{f(x) - f(a)}{x - a} = \lambda \quad (6.1)$$

En tal caso, llamaremos $\lambda = f'(a)$ a la **derivada de f en a** .

Por otro lado, si f está definida en un subconjunto de un espacio normado arbitrario X , el cociente anterior carece de sentido. Podemos resolver esto de manera sencilla reescribiendo la ecuación (6.1) de cualquiera de las siguientes de formas, todas ellas equivalentes.

⁵ A' es el conjunto de puntos de acumulación de A

$$\lim_{x \rightarrow a} \frac{f(x) - f(a) - \lambda(x - a)}{x - a} = 0$$

$$\lim_{x \rightarrow a} \frac{|f(x) - f(a) - \lambda(x - a)|}{|x - a|} = 0$$

$$\lim_{x \rightarrow a} \frac{f(x) - f(a) - \lambda(x - a)}{|x - a|} = 0$$

Notación 3. Si X, Y son espacios normados, entonces llamaremos $L(X, Y)$ al conjunto de las funciones lineales y continuas de X a Y ; es decir

$$L(X, Y) = \{f : X \rightarrow Y / f \text{ lineal y continua}\}$$

6.2 FUNCIONES DIFERENCIABLES

Veamos ahora algunas propiedades de las Funciones Diferenciables, que nos servirán *a posteriori* para poder probar la Regla de la Cadena [13], entre otros. Comenzamos con la definición más general posible.

Definición 12. Sean X, Y espacios normados, $A \neq \emptyset$ un subconjunto de X y $f : A \rightarrow Y$ una función. Entonces, diremos que la función f es **diferenciable** en un punto $a \in A$ si existe una aplicación $T \in L(X, Y)$ lineal y continua verificando una de las 2 condiciones equivalentes siguientes; o bien,

$$1. \lim_{x \rightarrow a} \frac{\|f(x) - f(a) - T(x - a)\|}{\|x - a\|} = 0$$

o bien,

$$2. \lim_{x \rightarrow a} \frac{f(x) - f(a) - T(x - a)}{\|x - a\|} = 0$$

Teniendo en mente la definición (12) sobre diferenciabilidad en un punto, vamos a probar 2 proposiciones que ampliarán el sentido y la utilidad del concepto anterior. En todas ellas, se entiende el mismo contexto que en la definición (12) y por tanto, se omite.

Proposición 2 (Unicidad de la Diferencial). *Si f es diferenciable en $a \in A$, entonces la aplicación $T \in L(X, Y)$ anterior es única.*

Demostración. Como $A \subseteq \mathbb{A}$, entonces $\exists \delta > 0$ verificando $B(a, \delta) \subset A$. Tomando un elemento arbitrario $v \in X \setminus \{0\}$, aplicamos la definición inicial (12) con el cambio $x = a + tv$, con $t \in \mathbb{R}$ y con la restricción $|t| < \frac{\delta}{\|v\|}$, obteniendo

$$\lim_{t \rightarrow 0} \frac{\|f(a + tv) - f(a) - tT(v)\|}{|t| \cdot \|v\|} = 0 \quad (6.2)$$

En la ecuación (6.2) se ha cambiado el límite, pues cuando $x \rightarrow a$, se cumple que $t \rightarrow 0$.

Ahora, como T es una función lineal, llegamos a

$$\frac{1}{\|v\|} \lim_{t \rightarrow 0} \left\| \frac{f(a + tv) - f(a)}{t} - T(v) \right\| = 0$$

y por ser $v \neq 0$, podemos despejar T y obtenemos

$$T(v) = \lim_{t \rightarrow 0} \frac{f(a + tv) - f(a)}{t} \quad (6.3)$$

Si tenemos dos funciones lineales $T_1, T_2 : X \rightarrow Y$ que verifican la definición (12), entonces tanto T_1 como T_2 cumplen (6.3) y por tanto, $T_1 = T_2$, lo que prueba la unicidad de la diferencial \square

Ahora que hemos probado la **unicidad de la diferencial**, es buen momento para introducir una nueva nomenclatura para este tipo de funciones, que nos facilite el trabajo en futuras demostraciones.

Definición 13. Si f es diferenciable en un punto $a \in A$, denominaremos **diferencial de f en a** a la función $T \in L(X, Y)$ que satisface la definición (12). Además, como está función es única, la denotaremos por $Df(a)$, de manera que $Df(a) \in L(X, Y)$ verifica

$$\lim_{x \rightarrow a} \frac{f(x) - f(a) - Df(a)(x - a)}{\|x - a\|} = 0 \quad (6.4)$$

Se incluye ahora una proposición *a priori* intuitiva que nos habla de la relación de la diferenciabilidad con la continuidad de una función.

Proposición 3. Si f es diferenciable en un punto $a \in A$, entonces f es continua en a .

Demostración. Fijamos un elemento $x \in A$ con $x \neq a$ y desarrollamos el valor de f en x de la siguiente manera

$$f(x) = f(a) + Df(a)(x - a) + ||x - a|| \frac{f(x) - f(a) - Df(a)(x - a)}{||x - a||} \quad (6.5)$$

donde podemos tomar límite cuando $x \rightarrow a$ a ambos lados y sabiendo que

- $\lim_{x \rightarrow a} Df(a)(x - a) = 0$ por ser $Df(a)$ una función continua.
- $\lim_{x \rightarrow a} ||x - a|| \frac{f(x) - f(a) - Df(a)(x - a)}{||x - a||} = 0$, por (6.4).

entonces, como 2 de los 3 sumandos a la derecha de la ecuación (6.5), concluimos que f es continua en a ; esto es, $\lim_{x \rightarrow a} f(x) = f(a)$. \square

6.3 REGLA DE LA CADENA

La regla de la cadena establece cómo calcular la derivada de una función compuesta. En el contexto del entrenamiento de redes neuronales, resultará esencial para calcular las derivadas parciales necesarias para ajustar los pesos y sesgos de la red mediante el algoritmo de retropropagación (backpropagation), donde el objetivo es minimizar una función de pérdida que mide la diferencia entre la salida de la red y la salida deseada.

La regla de la cadena se usa para propagar el error desde la capa de salida hacia las capas anteriores de la red y estas derivadas se utilizan en el algoritmo de optimización de descenso del gradiente.

Teorema 1 (Regla de la Cadena). *Sean X, Y, Z espacios normados, $U, V \neq \emptyset$ subconjuntos abiertos de X e Y respectivamente y f, g dos funciones tales que $f : U \rightarrow V$ y $g : V \rightarrow Z$. Si f es diferenciable en $a \in U$ y a su vez g lo es $b = f(a) \in V$, entonces la composición $g \circ f$ también es diferenciable en $a \in U$, verificando*

$$D(g \circ f)(a) = Dg(b) \circ Df(a) = Dg(f(a)) \circ Df(a) \quad (6.6)$$

Demostración. A lo largo de esta demostración, definimos una serie de funciones que ayudarán a simplificar la notación del problema. Primero, sea $\Phi : U \rightarrow \mathbb{R}_0^+$ tal que $\Phi(a) = 0$ y

$$\Phi(x) = \frac{\|f(x) - f(a) - Df(a)(x - a)\|}{\|x - a\|}, \text{ con } x \in U \setminus \{a\} \quad (6.7)$$

Como f es diferenciable en a , entonces por la proposición anterior, Φ es continua en a y cumple que

$$\|f(x) - f(a) - Df(a)(x - a)\| = \Phi(x)\|x - a\|, \text{ con } x \in U \quad (6.8)$$

En segundo lugar, vamos a construir una función análoga a Φ salvo que con dominio en V . Entonces, sea $\Psi : V \rightarrow \mathbb{R}_0^+$ tal que $\Psi(b) = 0$ y

$$\Psi(y) = \frac{\|g(y) - g(b) - Dg(b)(y - b)\|}{\|y - b\|}, \text{ con } y \in V \setminus \{b\} \quad (6.9)$$

además, al igual que sucede con Φ en la ecuación (6.8), ocurre lo análogo para Ψ , cumpliendo

$$\|g(y) - g(b) - Dg(b)(y - b)\| = \Psi(y)\|y - b\|, \text{ con } y \in V$$

En tercer lugar, construimos la función que representa la composición de la f y la g como $\Lambda : U \rightarrow \mathbb{R}_0^+$ tal que

$$\Lambda(x) = \|(g \circ f)(x) - (g \circ f)(a) - (Dg(b) \circ Df(a))(x - a)\| \text{ con } x \in U$$

Como tenemos que $(Dg(b) \circ Df(a)) \in L(X, Z)$, entonces solo tenemos que ver que $\lim_{x \rightarrow a} \frac{\Lambda(x)}{\|x - a\|} = 0$. Por otro lado, tomando un $x \in U$ arbitrario y $f(x) = y \in V$, aplicamos la desigualdad triangular a Λ

$$0 \leq \Lambda(x) \leq \|g(y) - g(b) - Dg(b)(y - b)\| + \|Dg(b)(y - b - Df(a)(x - a))\| \quad (6.10)$$

Ahora tomamos tanto la definición (6.7) de Φ , como la de una norma en el espacio $L(Y, Z)$ y operamos con el segundo sumando de (6.10)

$$\begin{aligned} \|Dg(b)(y - b - Df(a)(x - a))\| &\leq \|Dg(b)\| \cdot \|y - b - Df(a)(x - a)\| \\ &= \|Dg(b)\| \cdot \|f(x) - f(a) - Df(a)(x - a)\| \\ &= \|Dg(b)\| \cdot \Phi(x) \cdot \|x - a\| \end{aligned} \quad (6.11)$$

Análogamente, operamos con el primer sumando de (6.10) usando la definición (6.9) de Ψ ,

$$\|g(y) - g(b) - Dg(b)(y - b)\| = \Psi(y) \|y - b\| = \Psi(f(x)) \|f(x) - f(a)\| \quad (6.12)$$

y podemos aplicar la desigualdad triangular en el último término de la igualdad en (6.12), quedando

$$\begin{aligned} \|f(x) - f(a)\| &\leq \|f(x) - f(a) - Df(a)(x - a)\| + \|Df(a)(x - a)\| \\ &\leq (\Phi(x) + \|Df(a)\|) \|x - a\| \end{aligned}$$

por lo que el segundo sumando de (6.10) queda acotado por

$$\|g(y) - g(b) - Dg(b)(y - b)\| \leq \Psi(f(x)) \cdot (\Phi(x) + \|Df(a)\|) \|x - a\| \quad (6.13)$$

Tomamos las cotas (6.11) y (6.13) de los 2 sumandos y obtenemos que

$$0 \leq \Lambda(x) \leq \Psi(f(x)) \cdot (\Phi(x) + \|Df(a)\|) \cdot \|x - a\| + \|Dg(b)\| \cdot \Phi(x) \cdot \|x - a\| \quad (6.14)$$

donde basta considerar $x \in U \setminus \{a\}$ para llegar a la siguiente expresión, en la que tenemos que ver que la parte derecha de la desigualdad (6.14) tiene límite 0 cuando $x \rightarrow a$

$$0 \leq \frac{\Lambda(x)}{\|x - a\|} \leq \Psi(f(x)) \cdot (\Phi(x) + \|Df(a)\|) + \|Dg(b)\| \cdot \Phi(x) \quad (6.15)$$

Sabemos que f es diferenciable y por tanto, continua, en a . Además, Ψ es continua en $b = f(a)$, Φ es continua en a y la composición $(\Psi \circ f)$ también es continua en a . Esto implica que

- $\lim_{x \rightarrow a} \Phi(x) = \Phi(a) = 0$, por definición de Φ .
- $\lim_{x \rightarrow a} \Psi(f(x)) = \Psi(f(a)) = \Psi(b) = 0$, por definición de Ψ .

Por lo tanto, podemos tomar límite cuando $x \rightarrow a$ en el término de la derecha de la desigualdad (6.15), y obtenemos

$$\lim_{x \rightarrow a} \left(\Psi(f(x)) \cdot (\Phi(x) + ||Df(a)||) + ||Dg(b)|| \cdot \Phi(x) \right) = 0$$

con lo que llegamos al límite buscado, quedando $\lim_{x \rightarrow a} \frac{\Lambda(x)}{||x - a||} = 0$, que prueba que $(g \circ f)$ es diferenciable en a , verificando así la ecuación (6.6), como queríamos.

□

6.4 DERIVADAS DIRECCIONALES Y DERIVADAS PARCIALES

Seguiremos estudiando la diferenciabilidad, pero la restringiremos a los vectores del dominio X que nos interesen en cada caso. Para caracterizar estos vectores, se introduce la siguiente definición.

Definición 14. Llamaremos **dirección** en un espacio normado X a todo elemento $u \in X$ que verifique $||u|| = 1$. Además, denotaremos como S_X al conjunto de todas las direcciones del espacio, siendo $S_X = \{u \in X / ||u|| = 1\}$

Ahora, debemos considerar una función Φ_u para cada dirección $u \in S_X$, de forma que fijado un radio $r > 0$ que cumpla que $B(a, r) \subset U \subset X$, tenemos $\Phi_u :]-r, r[\rightarrow Y$, y para todo $t \in]-r, r[$

$$\Phi_u(t) = f(a + tu)$$

La función Φ_u describe cómo se comporta f cerca del punto a , en aquella recta que tiene u como vector de dirección y pasa por el punto a .

Dada Φ_u , podemos introducir la definición que relaciona las direcciones con las derivadas.

Definición 15. Sea $u \in S_X$, diremos que f es **derivable en a en la dirección u** si Φ_u es derivable en 0. En tal caso,

$$\Phi'_u(0) = \lim_{t \rightarrow 0} \frac{\Phi_u(t) - \Phi_u(0)}{t} = \frac{f(a + tu) - f(a)}{t}$$

Al vector derivada $\Phi'_u(0)$ lo llamaremos **derivada direccional de f en a** , según la dirección u y escribiremos $f'_u(s)$ para referirnos a él.

La definición de dirección y por tanto, la de derivada direccional van ligadas a los ejes de coordenadas con lo que se esté trabajando, pues en ellos está la referencia de la que parten dichas direcciones. En nuestro caso, las direcciones de los ejes de coordenadas se corresponden con las de la base usual, cuyos vectores serán $\{e_1, e_2, \dots, e_N\}$ ⁶.

De cara a las siguientes definiciones, vamos a llamar Δ_N al conjunto de los N índices de las variables con las que estamos trabajando. Sabiendo lo anterior, se introduce la siguiente definición.

Definición 16. Fijado un índice $k \in \Delta_N$, si una función f es derivable en a en la dirección e_k , entonces diremos que f es parcialmente derivable en a con respecto a la k -ésima variable y llamaremos $\frac{\partial f}{\partial x_k}(a)$ a la **derivada parcial de f con respecto a la k -ésima variable en a** . De esta forma, se escribe

$$\frac{\partial f}{\partial x_k}(a) = f'_{e_k}(a) = \lim_{t \rightarrow 0} \frac{f(a + te_k) - f(a)}{t}$$

o simplemente $f'_{x_k}(a)$, en su notación más abreviada.

6.5 VECTOR GRADIENTE

Ahora que hemos conocemos la manera que construir derivadas con respecto a cada variable, podemos definir el vector gradiente asociado a una función, para posteriormente realizar un análisis sobre sus utilidades y aplicaciones, especialmente a la hora de optimizar los resultados de una red neuronal.

Definición 17. Sea f una función parcialmente derivable en a ; esto es, parcialmente derivable con respecto a todas las variables, entonces se define el **vector gradiente** $\nabla f(a) \in \mathbb{R}^N$ de la siguiente manera,

$$\nabla f(a) = \left(\frac{\partial f}{\partial x_1}(a), \frac{\partial f}{\partial x_2}(a), \dots, \frac{\partial f}{\partial x_N}(a) \right) = \sum_{k=1}^N \frac{\partial f}{\partial x_k}(a) e_k$$

donde la derivada parcial de f con respecto a la k -ésima variable viene representada como la k -ésima componente del vector.

6 Base canónica usual, donde para cada $k = 1, \dots, N$ el vector e_k es aquel que tiene la forma $e_k = (0, \dots, 0, 1, 0, \dots, 0)$, con $k - 1$ ceros al principio y $N - k$ ceros al final; esto es, el que apunta a la dirección k -ésima

6.5.1 Interpretación del Gradiente

A continuación, veremos algunas aplicaciones prácticas del vector gradiente; puesto que, a pesar de ser una herramienta ampliamente utilizada en la teoría de matemáticas y física, su relevancia trasciende este ámbito. A partir de algunos ejemplos, conoceremos la utilidad real de este vector, acabando con aquel que nos concierne más, como es el estudio de los extremos de un campo escalar.

Interpretación Física

El vector gradiente puede ser interpretado desde el punto de vista de la física, de manera que fijada una dirección $u \in S$, una distancia t y un punto a , diremos que el campo escalar f experimenta una variación de $f(a + t \cdot u) - f(a)$ si nos desplazamos desde a , una distancia t en el sentido y dirección dados por u . Por ello, se suele decir que la derivada direccional equivale a la tasa de variación del campo en el punto a con dirección u ; puesto que el campo exactamente $f'_u(a)$ "unidades de campo" por unidad de longitud.

Podemos decir que el vector gradiente viene a indicar la dirección y sentido en la cual esta función aumenta con mayor rapidez desde un punto en específico, en este caso, a . Asimismo, la magnitud del vector gradiente nos indica tal rapidez. En otras palabras, en términos físicos, podemos considerarlo como una medida de la pendiente de dicha función en la dirección dada.

Plano Tangente a una Superficie

Vamos a conocer de qué manera la construcción del vector gradiente nos permitirá hallar los vectores tangentes a cualquier superficie de \mathbb{R}^3 .

En primer lugar, sea $\Omega \subset \mathbb{R}^2$ un dominio y $f : \Omega \rightarrow \mathbb{R}$ una función continua, diremos que

$$S = \left\{ (x, y, f(x, y)) / (x, y) \in \Omega \right\} \subset \mathbb{R}^3$$

es una superficie en \mathbb{R}^3 .

De esta manera, si f es diferenciable en $(x_0, y_0) \in \Omega$, entonces llamaremos $z_0 = f((x_0, y_0))$ y el vector gradiente sería

$$\nabla f(x_0, y_0) = \left(\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0) \right) = (\alpha_0, \beta_0)$$

Ahora, podemos definir un plano afín en \mathbb{R}^3 a partir de la ecuación

$$z - z_0 = \alpha_0(x - x_0) + \beta_0(y - y_0)$$

de forma que la función asociada a esta superficie puede obtenerse despejando la variable z , quedando

$$\begin{aligned} z = g(x, y) &= z_0 + \alpha_0(x - x_0) + \beta_0(y - y_0) \\ &= f(x_0, y_0) + Df(x_0, y_0)((x, y) - (x_0, y_0)) \end{aligned}$$

Entonces, cerca del punto (x_0, y_0) , podemos decir que g es la función afín que mejor aproxima a la función f . Así, diremos que el plano Π de ecuación

$$z - z_0 = (x - x_0) \cdot \frac{\partial f}{\partial x}(x_0, y_0) + (y - y_0) \cdot \frac{\partial f}{\partial y}(x_0, y_0)$$

es el **plano tangente a S** en el punto (x_0, y_0, z_0) ; y además, el vector

$$\left(\frac{\partial f}{\partial x}(x_0, y_0), \frac{\partial f}{\partial y}(x_0, y_0), -1 \right)$$

un **vector normal a S** en el punto (x_0, y_0, z_0) , por ser ortogonal a aquellos vectores del plano tangente a S definidos como $v_1 - v_2$, para cualesquiera $v_1, v_2 \in \Pi$; esto es, todas las rectas contenidas en el plano tangente.

Extremos Relativos de un Campo Escalar

Se conocen como problemas de optimización aquellos que consisten en averiguar si una función f tiene un máximo o un mínimo relativo. Para resolverlos, se emplea lo que denominamos la **condición necesaria de extremo relativo**.

Proposición 4. *Sea $A \in \mathbb{R}^N$ un subconjunto cualquiera no vacío y sea f una función definida como $f : A \rightarrow \mathbb{R}$, tal que f tiene un extremo relativo en algún punto $a \in A$ y es parcialmente derivable en tal punto. Entonces, se tiene que $\nabla f(a) = 0$.*

Demostración. Sin pérdida de generalidad, podemos suponer que el extremo relativo es un máximo. En tal caso, existe $r > 0$ de manera que $f(x) \leq f(a)$ para cualquier $x \in B(a, r)$.

Ahora, debemos fijar un índice $k \in \Delta_N$ de forma que se tiene

$$\frac{f(a + t \cdot e_k) - f(a)}{t} \leq 0 \text{ para todo } t \in]0, r[\quad (6.16)$$

$$\frac{f(a + t \cdot e_k) - f(a)}{t} \geq 0 \text{ para todo } t \in]-r, 0[\quad (6.17)$$

De la primera ecuación (6.16), se desprende que $\frac{\partial f}{\partial x_k}(a) \leq 0$; mientras que de la segunda, (6.17), obtenemos $\frac{\partial f}{\partial x_k}(a) \geq 0$, por lo que ha de ser $\frac{\partial f}{\partial x_k}(a) = 0$ para cualquier índice k , o lo que es lo mismo, $\nabla f(a) = 0$. \square

Esta última proposición muestra la importancia de la función del vector gradiente de cara a resolver problemas de minimización u optimización, que nos sirve de introducción para el apartado siguiente.

6.5.2 Algoritmo de Descenso del Gradiente

Como se ha comentado antes, el entrenamiento de modelos de aprendizaje automático se entiende matemáticamente como un problema de optimización, donde el objetivo es encontrar un extremo relativo; en particular, un mínimo de la función a optimizar.

De cara a resolver este problema, podemos volver a la definición de derivada, pues sabemos que esta nos proporciona información sobre la manera en que una función varía con respecto a las variables de entrada. Más concretamente, el gradiente en cierto punto nos indica la dirección en la que la función a minimizar crece con mayor rapidez con respecto a pequeños cambios.

Intuitivamente, podemos inferir que si nos movemos en dirección contraria al vector gradiente, nos estaremos aproximando de la forma más rápida posible a un mínimo de nuestra función. Esto se puede observar en la figura (2), donde la sucesión de aproximaciones a la solución se mueve por el camino más corto hacia la optimalidad.

De manera formal, este método se encuadra dentro de los algoritmos iterativos que buscan la optimización o minimización de una función; esto es, dada cierta $\mathcal{F} : \mathbb{R}^N \rightarrow \mathbb{R}$ que sea convexa⁷ y diferenciable y dado cierto punto x_0 del

⁷ \mathcal{F} se dice convexa si para cualesquiera dos elementos x_1, x_2 del dominio, se cumple que $\mathcal{F}(\alpha x_1 + (1 - \alpha)x_2) < \alpha\mathcal{F}(x_1) + (1 - \alpha)\mathcal{F}(x_2)$ para todo $\alpha \in [0, 1]$

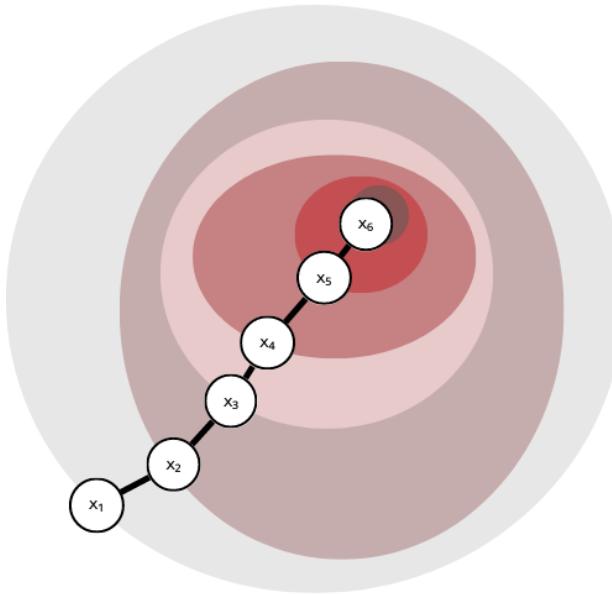


Figura 2: Explicación gráfica del algoritmo de descenso del gradiente

dominio, diremos que las siguientes iteraciones del algoritmo de descenso del gradiente se calcula como

$$x_{n+1} = x_n - \lambda \cdot \nabla \mathcal{F}(x_n) \quad (6.18)$$

donde

- λ es un valor real positivo dado que representa la **tasa de aprendizaje**. Indica cuánto tenemos que actualizar la solución, o en otras palabras, la velocidad a la que nos acercamos al valor óptimo. Si este valor es muy pequeño, el algoritmo tardará en converger; mientras que si es muy grande, existe la posibilidad de que no converja, pues no encontraría el camino hacia la solución.
- El signo menos representa que las aproximaciones se realizan en sentido contrario al gradiente.

Conocida la fórmula para el cálculo iterativo de un punto dado el anterior, el algoritmo final quedaría de la siguiente manera:

1. Aleatoriamente, se escoge un punto a que actúa como valor inicial x_0 .
2. Se halla el valor del vector gradiente en $a = x_0$, donde

$$\nabla \mathcal{F}(a) = \left(\frac{\partial \mathcal{F}}{\partial x_1}(a), \dots, \frac{\partial \mathcal{F}}{\partial x_N}(a) \right)$$

3. Se calcula el siguiente punto de la sucesión por medio de (6.18), dado cierto valor para la tasa de aprendizaje, no necesariamente igual en todas las iteraciones.

Estos pasos se repiten cierto número de veces. Este número puede venir pre-determinado o puede verse modificado en función de ciertas condiciones, como podrían ser que o bien, ya se haya alcanzado el nivel de precisión deseado, o que la variación de los siguientes puntos sea menor que cierto umbral, entre otros métodos.

Parte II

TEORÍA DE INFORMÁTICA

Bases de informática para la comprensión de la parte de aplicación. Comienza con la descripción de los algoritmos de disciplinado de relojes atomicos u osciladores, para seguir con los conceptos básicos de aprendizaje automático y en particular, redes neuronales. Se finaliza con la concreción de lo anterior aplicado a modelos de redes neuronales recurrentes para predicción de algoritmos de disciplinado.

7

ALGORITMOS DE DISCIPLINADO DE RELOJES

En este primer capítulo dedicado a la teoría informática nos dedicaremos a presentar, de manera breve, en qué consiste el disciplinado de relojes (u osciladores); veremos su utilidad en la sociedad y algunas nociones más rigurosas sobre estos algoritmos.

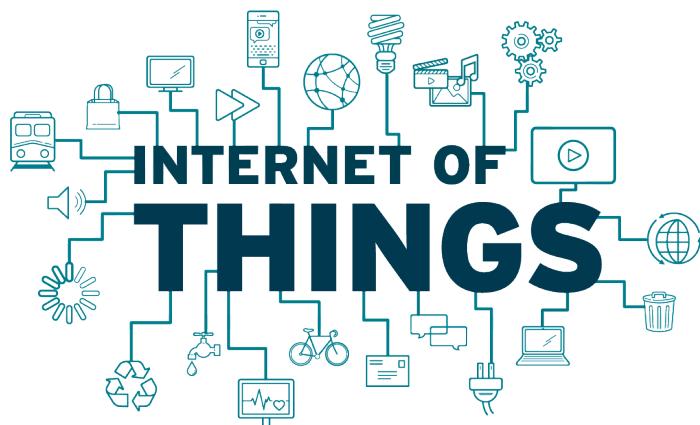
En primer lugar, vamos a ganar algo de perspectiva conociendo en qué campo de la informática nos estamos moviendo al hablar de algoritmos de disciplinado de relojes.

7.1 EL INTERNET DE LAS COSAS

El **Internet de las cosas** (IoT, por sus siglas en inglés) contempla la comunicación y conexión de dispositivos utilizando sensores, actuadores y tecnologías inalámbricas como pueden ser Wi-Fi, Bluetooth y NFC. Todos estos dispositivos están conectados a Internet y así se encargan de **recopilar y compartir datos entre sí**, lo que permite la realización de una gran variedad de tareas automatizadas y optimizadas.

El campo IoT se encuentra en plena **expansión y crecimiento** al verse impulsado por la continua reducción de los costos derivados de la producción de sensores, la expansión a escala mundial de la conectividad inalámbrica y el aumento de la velocidad de procesamiento en pequeños dispositivos. El IoT tiene una gran aplicabilidad, al poder ser utilizado en diversos campos, desde el monitoreo de la salud y el hogar inteligente, hasta la optimización e incluso automatización de ciertos procesos en las industrias.

En cuanto a su lugar dentro del mundo de la informática, IoT es un campo altamente interdisciplinar, que combina electrónica e ingeniería. Por un parte, los dispositivos IoT requieren de componentes electrónicos para funcionar correcta-



mente como pueden ser sensores y actuadores. Por otra parte, los datos que recopilan estos dispositivos son procesados y posteriormente analizados por potentes sistemas informáticos, para ser utilizados en futuras aplicaciones.

7.2 RELOJES ATÓMICOS

Este trabajo está dedicado a los relojes atómicos; pero antes de entender qué son exactamente y para qué se utilizan estos dispositivos, tenemos que conocer la definición de oscilador.

En electrónica, un **oscilador** [15] se define como un circuito que produce una señal continua, periódica y alterna sin ningún *input*. Los relojes atómicos y algunos sistemas de comunicación son ejemplos comunes de dispositivos que utilizan osciladores para generar señales precisas y estables.

Debido a factores ambientales y de diseño, la frecuencia del oscilador puede llegar a desviarse de su valor nominal, lo que causará errores y fallos en el sistema. Estos desvíos son totalmente normales, pero deben ser corregidos y ese es el objetivo del trabajo que nos concierne.

El **disciplinado de relojes** es el proceso de ajustar la frecuencia (señal periódica) de un oscilador con el objetivo de mantenerla tan cerca como sea posible de su valor nominal. Dicho valor puede venir de otro oscilador de referencia, denominado oscilador *leader*. Para lograr esto, empleamos una señal que provee una referencia de tiempo de los relojes atómicos en los satélites de la constelación que se esté utilizando. Estas señales provienen de los denominados sistemas globales de navegación por satélite (GNSS, por sus siglas en inglés) [16].

Entre los GNSS más conocidos, se encuentran los siguientes:

- GPS: El *global positioning system* es un sistema desarrollado y ahora empleado por los Estados Unidos; en particular, por el Departamento de Defensa. Actualmente es propiedad de la Fuerza Espacial del país y es capaz de determinar la posición de cualquier objeto en el globo con una precisión de centímetros.
- GLONASS: El *Global'naya Navigatsionnaya Sputnikovaya Sistema* es un sistema creado por la Unión Soviética como alternativa al sistema estadounidense. Actualmente lo utiliza la Federación Rusa y consta de unos 31 satélites en órbita para llevar a cabo las tareas de geolocalización.
- Galileo: Sistema desarrollado entre la Agencia Espacial Europea y la Unión Europea, con el objetivo de no depender de otros países y mejorar los proyectos ya existentes de Estados Unidos y la Federación Rusa. Actualmente cuenta con 26 satélites en órbita, aunque se planean lanzar al menos 4 más.
- BeiDou: GNSS desarrollado por el gobierno chino como alternativa al GPS. Su nombre significa Osa Mayor en chino y cuenta con unos 30 satélites en órbita.

A partir de la señal proporcionada por un GNSS, diremos que disciplinamos un oscilador con respecto a tal referencia cuando ajustamos la frecuencia de dicho reloj para siga de la mejor manera posible a la señal de referencia.

El disciplinado suele utilizarse en sistemas de control de procesos, de posicionamiento global y de comunicación, además de en otros muchos campos donde se requiere una sincronización precisa del tiempo. Más concretamente, estos algoritmos pueden aplicarse para la sincronización en distintos puntos de acceso a las bolsas de valores, para la sincronización de relojes en órbita terrestre o para el correcto funcionamiento de aceleradores de partículas. En general, cualquier instalación de instrumentos distribuidos donde sea necesario la coordinación de acciones de sensorización o de control, requiere de disciplinado de relojes. Todos ellos tienen en común que un error de nanosegundos puede ser crítico para estos procesos.

7.3 ALGORITMO DE DISCIPLINADO

El algoritmo de disciplinado de relojes es la base de la especificación Network Time Protocol (mayormente conocido por sus siglas, NTP) y su implementación de referencia. Este conocido algoritmo está compuesto por una combinación de bucles de retroalimentación híbridos de bloqueo de fase y de frecuencia, que ade-

más es adaptable y se ajusta automáticamente. Su objetivo es ajustar el reloj del sistema de la forma más óptima posible a la vez que se minimiza la sobrecarga de la red. []

7.3.1 Varianza de Allan

El algoritmo opera en 2 modos y para poder entenderlos debemos presentar la definición de Varianza de Allan¹. Esta viene definida como la mitad de la media temporal de las diferencias elevadas al cuadrado entre observaciones sucesivas de la desviación de la frecuencia observada del oscilador en todo el periodo de sondeo. Por tanto, al depender del periodo, podemos decir que la varianza de Allan es una función de este.[17]

Se considera que una Varianza de Allan baja es una buena característica para determinar que un oscilador tiene un comportamiento estable a lo largo del periodo en el que estemos trabajando.

Ahora, necesitamos conocer una medida más genérica que nos permitirá definir la Varianza de Allan posteriormente. Esta es la Varianza de M muestras, que viene dada por

$$\sigma_y^2(M, T, \tau) = \frac{1}{M-1} \left\{ \sum_{i=0}^{M-1} \left[\frac{x(iT + \tau) - x(iT)}{\tau} \right]^2 - \frac{1}{M} \left[\sum_{i=0}^{M-1} \frac{x(iT + \tau) - x(iT)}{\tau} \right]^2 \right\}$$

donde

- $x(t)$: observación del oscilador en el instante t .
- M : número de muestras de frecuencia empleadas.
- T : periodo de muestreo (tiempo entre cada observación de la frecuencia).
- τ : duración de cada una de las estimaciones de la frecuencia.

A partir de la Varianza de M muestras, definimos la **Varianza de Allan** como

$$\sigma_y^2(\tau) = \langle \sigma_y^2(2, \tau, \tau) \rangle$$

¹ El físico de relojes atómicos David Wayne Allan da nombre a esta medida (comúnmente conocida como varianza de dos muestras) para la estabilidad de la frecuencia en osciladores y relojes. Se expresa matemáticamente como $\sigma_y^2(\tau)$

donde el símbolo $\langle \dots \rangle$ sirve para indicar el valor esperado del operando al que se refiera.

Cabe destacar que la fórmula indica que las observaciones están tomadas sin tiempo entre ellas. Esto viene reflejado cuando determinamos $T = \tau$.

Ahora ya podemos introducir los 2 modos que existen en los algoritmos de disciplinado:

- Bucle de Bloqueo de Fase (Phase-Lock Loop) o PLL: se utiliza cuando los intervalos de sondeo son inferiores al punto de Varianza de Allan, que suele tener un valor por defecto de 2048 segundos
- Bucle de Bloqueo de Frecuencia (Frequency-Lock Loop) o FLL: cuando los intervalos de sondeo son superiores al punto de Varianza de Allan.

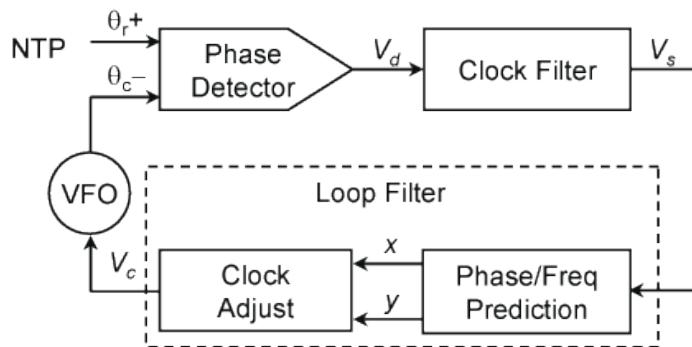


Figura 3: Representación del Algoritmo de Disciplinado [18]

La figura (3) muestra un diagrama a partir de bloques que representa el proceso de disciplinado. La idea es comparar el timestamp (marca de tiempo) de una fuente de referencia (ya sea un reloj de referencia o un servidor remoto) con el timestamp del reloj del sistema, que se representa como un oscilador de frecuencia variable (Variable Frequency Oscillator, o VFO), para producir una muestra de *offset* bruto que llamaremos V_d .

Esta muestra V_d se procesa por un filtro de reloj con el objetivo de producir una actualización V_s . Este filtro implementa un controlador proporcional integral (PIC) que tiene la función de minimizar errores en tiempo y en frecuencia empleando para ello predictores x e y respectivamente, como se puede apreciar en la figura. El proceso de ajuste del reloj se puede llevar a cabo mediante dos disciplinados diferentes. [6]

- Para **disciplina demonio** (daemon), se realiza el muestreo de estos predictores una vez por segundo.
- Para **disciplina núcleo** (kernel), se realiza el muestreo de estos predictores una vez por cada interrupción del *tick*.

Al final del proceso, se produce la actualización del reloj del sistema, V_c .

Veamos ahora cómo funcionan los distintos modos. [19]

- i. En el modo **PLL**, por un lado, el predictor de la frecuencia se calcula como una integral del offset sobre las actualizaciones anteriores. Por otro lado, el predictor de fase es el offset compensado a lo largo del tiempo, con el objetivo de evitar el retraso del reloj.
- ii. En el modo **FLL** no se utiliza el predictor de la fase. En cuanto al predictor de frecuencia, este es similar al algoritmo *lockclock* presentado por el NIST² (National Institute of Standards and Technology) [20]. En este algoritmo, el predictor de frecuencia se calcula como una fracción del actual offset dividido por el tiempo que ha transcurrido desde la última actualización, para así minimizar el offset de la siguiente actualización.

La respuesta del algoritmo de disciplinado en el modo PLL viene determinada por la constante de tiempo, lo que resulta en una rigidez que depende de la fluctuación de las fuentes disponibles y del offset del reloj (oscilador) del sistema. Además, la constante de tiempo ajustada también se emplea como un intervalo de sondeo. Sin embargo, cuando estamos en modo simétrico NTP, cada fuente gestiona su propio intervalo de sondeo y estos pueden no coincidir. En tal caso, cada fuente utiliza el valor mínimo de su intervalo de sondeo y el de la otra fuente, que además se incluye en la cabecera del paquete NTP utilizado.[21]

² El Instituto Nacional de Estándares y Tecnología, también conocido como Oficina Nacional de Normas, es una agencia del departamento de Comercio de EEUU. Tiene el objetivo de promover la competencia y la innovación industrial en el país a través de mejoras en, principalmente, tecnología, metrología y normas.

8

APRENDIZAJE AUTOMÁTICO Y REDES NEURONALES

En este capítulo comenzamos el estudio sobre Inteligencia Artificial y todo lo que esta engloba en nuestro trabajo. Por lo que antes de empezar, vamos a dar la primera definición general:

Llamamos **Inteligencia Artificial** al campo de la informática que tiene como función la creación de sistemas que pueden realizar tareas que requieren inteligencia humana, como podrían ser el reconocimiento de imágenes o voz, el procesamiento del lenguaje natural, la toma de decisiones o la resolución de problemas

La inteligencia artificial tiene como base la idea de que una máquina puede imitar hasta cierto punto el comportamiento humano, e incluso para ciertas tareas, puede hacerlo de una manera más rápida, precisa y automática.

Por otro lado, el **Aprendizaje Automático** es una rama de la IA centrada en el desarrollo de algoritmos y modelos que permiten a las máquinas aprender a partir de datos, siendo esto un enfoque puramente empírico, basado en la observación y el análisis de patrones en los datos.

Finalmente, al hablar de redes neuronales que dispongan de varias capas ocultas, estaremos entrando en el campo del **Aprendizaje Profundo**; esto es, una técnica de aprendizaje automático que emplea estas redes multicapa para aprender a partir de los datos. Las redes neuronales son modelos matemáticos que permiten capturar cierto conocimiento a partir de una base de datos y su comportamiento se asemeja al funcionamiento de las neuronas del cerebro humano. Estas se utilizan mayoritariamente para realizar tareas complejas como la visión por computadora, el procesamiento del lenguaje natural y la predicción de series temporales. Por otro lado, una red neuronal con una única capa oculta, no entra dentro del dominio del aprendizaje profundo; sin embargo, debe ser capaz de aprender patrones más complejos que un algoritmo de aprendizaje automático

En la definición anterior cabe destacar un matiz que ampliaremos en capítulos posteriores. Las redes neuronales están compuestas por capas de neuronas y atendiendo a la cantidad de capas situadas entre la de entrada y la de salida (llamadas intuitivamente capas profundas), diremos que una red neuronal es profunda (deep neural network, en inglés) si tiene 2 o más capas profundas. En tal caso, diremos que estamos en el campo de estudio del aprendizaje profundo. Este tipo de redes son capaces de modelar relaciones no lineales complejas. Por otro lado, una red con una sola capa oculta, también denominada perceptrón, solo puede modelar relaciones simples, funcionando de manera similar a puertas lógicas a partir de los datos de entrada.

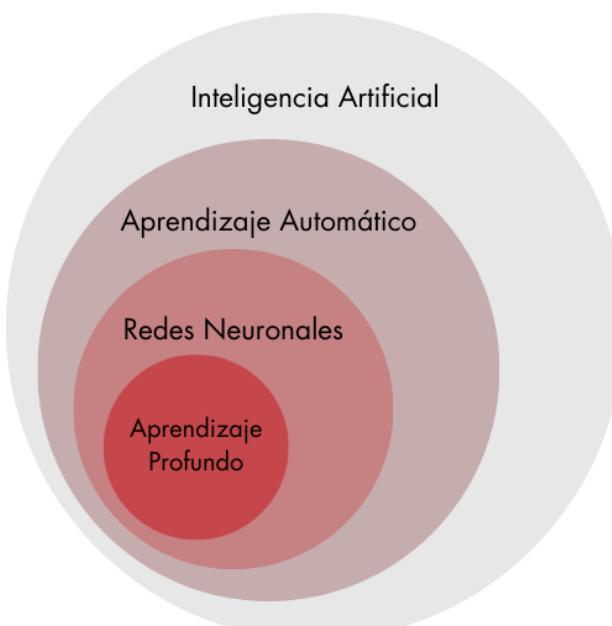


Figura 4: Representación Genérica de los ámbitos de la IA, AA, RRNN y AP

Las definiciones anteriores pueden verse visualmente en la figura (4), que marca las relaciones entre todos conceptos vistos.

Volviendo al tema del que trata nuestro trabajo, para la predicción de series temporales se pueden utilizar técnicas de redes neuronales para analizar los valores registrados y realizar predicciones sobre los valores futuros de la serie. Entre las técnicas más comunes se encuentran los modelos ARIMA (AutoRegressive Integrated Moving Average, o media móvil autoregresiva integrada) y los modelos de suavizado exponencial. Por otro lado, la técnica de aprendizaje profundo más empleada son las redes neuronales recurrentes, que se emplean para tratar de

obtener predicciones mucho más precisas ante series temporales complejas, especialmente en datos de alta dimensionalidad.

8.1 CONCEPTOS BÁSICOS

En esta sección, trataremos algunos conceptos que usaremos de forma habitual en la parte de aplicación.

Primero, tenemos que conocer más acerca de los algoritmos de aprendizaje automático; esto es, aquellos que deben ser capaces de aprender de los datos proporcionados. Ante esta afirmación, debe surgir la siguiente pregunta: ¿qué se entiende por aprendizaje? El informático Tom Mitchell¹ responde a esta cuestión en su libro [2] con esta popular cita.

"Se dice que un programa informático aprende de la experiencia con respecto a algún conjunto de tareas y medida de rendimiento, si su rendimiento en dichas tareas en función de tal medida de rendimiento, mejora con la experiencia"

Veamos ahora distintas formas de clasificar algoritmos de aprendizaje automático.

8.1.1 Clasificación de Algoritmos

En función del problema y de cómo se nos presenten los datos disponibles, podemos establecer la siguiente clasificación atendiendo al tipo de aprendizaje empleado para poder resolver dicho problema. En este caso, diferenciamos tres tipos: supervisado, no supervisado y por refuerzo. [22]

- **Aprendizaje supervisado:** En este tipo de algoritmos, el objetivo es generar un modelo predictivo cuyo entrenamiento se ha basado tanto en datos de entrada como en datos de salida. Se emplea intuitivamente el concepto de supervisión, puesto que el conjunto de datos dado ha sido etiquetado y clasificado previamente; esto es, sabemos con antelación o bien la categoría/grupo al que pertenece, o bien el valor al que corresponde cada uno de los ejemplos que forman nuestro conjunto de muestra. Estos datos de entrenamiento se emplean para poder ajustar el modelo inicial que se plantee.

¹ Tom Michael Mitchell es un informático estadounidense, fundador y ex-presidente del departamento de aprendizaje automático de la Carnegie Mellon University, además de ser miembro de la Academia Nacional de Ingeniería de los Estados Unidos. Es ampliamente conocido por sus contribuciones al desarrollo del aprendizaje automático y la neurociencia cognitiva.

A través de ellos, el algoritmo *aprende* a clasificar datos de entrada (inputs) comparando el resultado de su predicción con la etiqueta real (outputs) de la muestra, calibrando sus parámetros internos de acuerdo al error en la estimación. Existen dos problemas que pueden ser resueltos con este tipo de aprendizaje, son problemas de regresión y de clasificación, que definiremos posteriormente. Algunos algoritmos o métodos que implementan aprendizaje supervisado son:

- I. Redes neuronales
- II. Árboles de decisión
- III. Regresión logística
- IV. Support vector machine
- V. Clasificador de Naïve-Bayes
- VI. K-vecinos más cercanos

- **Aprendizaje no supervisado:** Los algoritmos de aprendizaje no supervisado siguen una dinámica similar a los anteriores, salvo que éstos únicamente ajustan sus parámetros internos teniendo en cuenta solo los datos de entrada, por lo que desaparece este factor de *supervisión*. En estos algoritmos, se presentan unos datos de entrada que no están clasificados ni etiquetados de ninguna forma, puesto que esto no es necesario para entrenar el modelo. El objetivo es obtener información valiosa de los datos sin tener un conocimiento previo de las variables de salida, simplemente explorando la estructura de los datos. El algoritmo más conocido es el agrupamiento (clustering, en inglés), cuya misión es buscar particiones de los datos en distintos grupos de forma que los elementos de cada grupo comparten características similares. Algunos algoritmos o métodos que implementan este tipo de aprendizaje son:

- I. Modelo de mezcla gaussiana
 - II. K-medias
 - III. Agrupamiento jerárquico
- **Aprendizaje por refuerzo** o reinforcement learning: Estos algoritmos buscan definir modelos y funciones enfocados en maximizar una serie de recompensas, basadas en las acciones y en el ambiente en el que un agente inteligente vaya a desempeñar ciertas funciones. En lugar de disponer de unos datos

de entrada y salida, se busca describir la situación del sistema, además de especificar un objetivo y una serie de acciones posibles dependiendo de cómo esté restringido el entorno. Este algoritmo consiste en un modelo acción recompensa a través de la experiencia y el error. Algunos de los algoritmos más utilizados son:

- I. Programación dinámica
- II. Q-learning
- III. SARSA (state - action - reward - state - action)

Como ya hemos mencionado en párrafos anteriores, dentro del aprendizaje supervisado existen dos tipos de problemas a resolver, dependiendo de la naturaleza de los da. Vamos a presentar ambos para así situar después nuestro trabajo dentro del marco teórico más adecuado que corresponda.

- **Clasificación:** Este tipo de problemas buscan crear un modelo que especifique a qué categoría pertenece un dato de entrada; es decir, asignar una clase a cada objeto. Para poder resolver esta tarea formalmente, dado un elemento x en un dominio X y un conjunto de clases $\{1, 2, \dots, k\} = Y$ se busca encontrar una función $f : X \rightarrow \{1, 2, \dots, k\}$ que asigne una clase a cada elemento del dominio, quedando $f(x) \in Y$. La idea principal de este problema es que las clases (el codominio de f) es un conjunto **discreto**.

Existen otras variantes de este problema, por ejemplo, podemos buscar que la salida sea una distribución de probabilidad sobre las clases; de manera que, para cada objeto, obtenemos la probabilidad de que este pertenezca a cada una de las clases. Este tipo de tarea en particular, suele emplearse para clasificación de imágenes.

- **Regresión:** Para esta clase de problemas, se busca crear un modelo que prediga un valor numérico dado cierto dato de entrada. Para resolver esta tarea formalmente, dado un elemento x en un dominio X y un rango de valores continuos Y , se busca encontrar una función $f : X \rightarrow Y$ que asigne un valor a cada elemento del dominio, quedando $f(x) \in Y$. Este problema es parecido al de clasificación (puesto que ambos se resuelven con algoritmos de aprendizaje supervisado), salvo que en este caso, la salida es continua.

Nuestro trabajo trata de predecir valores futuros asociados a una serie temporal correspondiente a las mediciones del algoritmo de disciplinado de un oscilador (reloj) atómico. Por lo que nuestros datos vendrán adecuadamente etiquetados: para cada serie, se asocia un valor numérico. A partir de esta información, es simple

detectar que nuestro problema debe ser resuelto mediante algoritmos de aprendizaje supervisado; en particular, como las predicciones se mueven en rangos continuos, es una tarea de regresión.

A partir de aquí, nos centraremos en el marco teórico de los problemas de **regresión con aprendizaje supervisado**.

8.1.2 Medida del Error

A la hora de crear un modelo, surgen diferentes problemáticas. Podemos considerar que la primera de todas ellas es la manera de saber que un modelo es bueno; o en otras palabras, ¿cómo podemos medir el rendimiento de un modelo de aprendizaje?

Para dar respuesta a esta pregunta, surge el concepto de **función de coste** (o función de pérdida, *loss function* en inglés), que se define como la medida del error entre el valor estimado por el modelo de aprendizaje y el valor real esperado. Esta función se representa como $\mathcal{L}(\hat{Y}, Y)$, donde \hat{Y} son los valores predichos e Y son los valores esperados (las etiquetas) y existen distintas formas de calcularla para un modelo de regresión con aprendizaje supervisado. [22]

Veamos las más utilizadas y sus diversas características, suponiendo un conjunto de m muestras:

- i. **Error Absoluto Medio (MAE):** Medida de precisión más simple, calculada como la media de los errores en valor absoluto.

$$MAE = \sum_{i=1}^m \frac{|\hat{y}^{(i)} - y^{(i)}|}{m}$$

Se caracteriza por:

- No penaliza los grandes errores.
- Es de fácil interpretación.
- Dificulta la diferenciación y la convergencia del error.

II. Error Absoluto Medio Escalado (MASE): Medida del error similar al MAE, salvo que los resultados quedan normalizados.

$$MASE = \frac{\sum_{i=1}^m |\hat{y}^{(i)} - y^{(i)}|}{\frac{m}{m-1} \sum_{i=2}^m |\hat{y}^{(m)} - y^{(m-1)}|}$$

Se caracteriza por:

- La escala es univariante y simétrica.
- Es de fácil interpretación.
- Dificulta la diferenciación y la convergencia del error.

III. Error Medio Cuadrático (MSE): Medida de precisión que se obtiene como la media de los residuos².

$$MSE = \sum_{i=1}^m \frac{(\hat{y}^{(i)} - y^{(i)})^2}{m}$$

Se caracteriza por:

- Penaliza los valores grandes de error.
- No es de fácil interpretación, porque viene expresado en unidades cuadradas.

IV. Raíz Cuadrada Media (RMSE): Medida del error similar al MSE, salvo que aplicamos una raíz cuadrada para que el resultado venga representado en las mismas unidades que la variable predicha.

$$RMSE = \sqrt{\sum_{i=1}^m \frac{(\hat{y}^{(i)} - y^{(i)})^2}{m}}$$

Se caracteriza por:

- Penaliza los valores grandes de error.

² Un residuo en estadística se entiende como el fallo en un ajuste de regresión, utilizado mayoritariamente en los problemas de regresión por mínimos cuadrados. De forma que un residuo es el cuadrado de la diferencia entre el valor real y el valor estimado

- No es de fácil interpretación, a pesar de estar expresado en las unidades originales.
- Buen rendimiento general en la mayoría de los problemas.

En la práctica, dependiendo del problema, se podrá escoger entre estas funciones de coste para determinar el rendimiento real de nuestro ajuste; aunque generalmente resulta positivo elegir varias medidas para así contrastarlas.

8.1.3 Optimizadores de la Función de Coste

Las funciones de coste que hemos presentado son aquellas que el algoritmo de descenso del gradiente tiene la intención de optimizar, buscando el camino más corto hacia su mínimo. Ahora bien, siendo este algoritmo la base, se han creado diversos métodos optimizadores que tratan de agilizar el proceso de convergencia de la función de coste hacia el mínimo correspondiente. [1]

Descenso del Gradiente Estocástico (SGD)

Este método se crea debido a la inviabilidad que puede suponer, en algunos casos, el calcular las derivadas parciales de la función de coste con respecto a todos los diferentes parámetros de la red.

La idea es usar una componente aleatoria (o estocástica) dentro del cálculo del gradiente, como podría ser la limitación de los cálculos a una única instancia de los datos (un batch), de manera que se reduce drásticamente la complejidad computacional del algoritmo.

Descenso del Gradiente Adaptativo (AdaGrad)

Se basa en la idea de que no todos los pesos deben variar de manera uniforme a la tasa de aprendizaje; sino que cada uno de ellos debe ser más óptimo con respecto a cierto valor de esta tasa.

De nuevo, como el coste computacional que supondría hallar este valor específico para cada variable y para cada iteración no resulta factible, este optimizador adapta y escala, a partir de un valor inicial, con respecto al gradiente acumulado para cada una de las variables.

Estimación Adaptativa del Momento (Adam) [1]

Se denomina Adam por sus siglas en inglés y es el más utilizado en la actualidad, por ser una combinación de las ideas de los mejores optimizadores existentes.

En este caso, modifica el gradiente y la tasa de aprendizaje para cada variable, pero además escala este valor, dividiéndolo por la raíz cuadrada media de la decadencia exponencial de los gradientes que se calcularon anteriormente.

Por último, este optimizador toma en cuenta el momentum³ medio del gradiente para hallar las tasas de aprendizaje correspondientes.

8.2 CREACIÓN DE UN MODELO

En esta sección, buscamos explicar de manera muy genérica el proceso de construcción, desde cero, de un modelo de aprendizaje automático con los matices del aprendizaje supervisado para una tarea de regresión. [23]

Cabe recordar, en primer lugar, que en nuestro problema buscamos la mejor estimación posible de una función $f : X \rightarrow Y$ que asigne a cada objeto $x \in X$ un valor numérico $y \in Y$. Además, en la mayor parte de los casos, el dominio X está formado por un conjunto de variables, de manera que cada dato queda compuesto por distintas componentes, como sigue:

$$x = (x_1, x_2, \dots, x_n) \in (X_1, X_2, \dots, X_n) = X$$

Este conjunto D de datos (o instancias) que servirá para que el modelo aprenda, podemos representarlo de la siguiente manera, en el caso de un conjunto total de m instancias

$$D = \left\{ [(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}), y^{(i)}] / i = 1, \dots, m \right\} \subset X \times Y$$

donde supondremos que estas instancias son independientes entre sí, además de idénticamente distribuidas, siguiendo una distribución de probabilidad conjunta, denotada como $P_{X \times Y}$.

Este conjunto D es nuestra referencia para obtener una buena aproximación de la función de ajuste f . Llamaremos $m_D : X \rightarrow Y$ a las funciones candidatas a ser

³ El momentum trata de agilizar el algoritmo de descenso del gradiente en direcciones que no varíen demasiado con respecto a las anteriores, siendo una manera de mantener la consistencia y robustez

aproximaciones óptimas de f a partir de D ; de forma que se busca $m_D(x) \approx f(x)$. Por tanto, diremos que dada una muestra $[x^{(i)}, y^{(i)}] = [(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)}), y^{(i)}]$, la función m_D proporciona una predicción

$$m_D(x^{(i)}) = m_D((x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})) = \hat{y}^{(i)} \approx y^{(i)} \in Y$$

En este punto, llega de forma natural la definición formal de función de coste, tal y como la describimos en [Medida del Error](#). Aquí, incluiremos que la función \mathcal{L} escogida, se define como $\mathcal{L} : Y \times Y \rightarrow \mathbb{R}_+$ para cada instancia. Así, para calcular las diferencias instancia por instancia, se emplean distintas funciones dependiendo de la medida del error que se vaya a utilizar.

- Para el MAE y el MASE, $\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = L_1(\hat{y}^{(i)}, y^{(i)}) = |\hat{y}^{(i)} - y^{(i)}|$.
- Para el MSE y el RMSE, $\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = L_2(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})^2$.

A partir de esta definición de la función de pérdida, debemos lidiar con la posibilidad de que nuestro modelo únicamente obtenga buenos resultados para los valores de entrada con los que ha sido entrenado; mientras que será incapaz de reconocer datos nuevos. Esto se conoce como overfitting o sobreajuste. La manera más usada de evitarlo consiste en usar un conjunto de datos externo con los que ir validando las hipótesis de función m_D obtenidas a partir de D . Análogamente, existe el concepto de underfitting o subajuste, que se refiere a un modelo al que le falta entrenamiento o que se ha entrenado de forma poco óptima. Así, estos modelos no son capaces de generalizar el conocimiento.

Normalmente, no es sencillo obtener estos datos externos, por lo que en la práctica, se divide el conjunto D en dos subconjuntos disjuntos, de forma que uno se usa para el entrenamiento del modelo y otro para su validación. El primero suele llamarse conjunto de entrenamiento o D_{train} , mientras que el segundo es conocido como conjunto de validación o D_{test} . De esta manera, se genera una hipótesis $m_{D_{train}}$ y posteriormente se valida, calculando el error mediante

$$\mathcal{L}(m_{D_{train}}(x^{(i)}), y^{(i)}), \text{ para } x^{(i)} \in D_{test}$$

No obstante, con esta manera de actuar, estaríamos condicionando la capacidad de generalizar del modelo únicamente a las instancias del conjunto de validación. Esto puede ser fatal en el caso de que este no sea representativo de todos los datos del conjunto D inicial. Para solucionar esto, se emplean métodos de validación cruzada o cross-validation. El más conocido es la **validación cruzada en K iteraciones** [2] o *k-fold cross validation* en inglés. Consiste en dividir D en un total

de K subconjuntos, cada uno de tamaño m/K y luego entrenar nuestro modelo en $K - 1$ subconjuntos; es decir todos menos en uno, que usaremos como conjunto de validación. Este proceso se repite K veces y el error final será la media de los errores en cada una de las iteraciones.

Un caso particular de validación cruzada en K iteraciones se da cuando se toma K igual a la longitud del conjunto D . Este mecanismo se denomina *leave one out* (dejar uno fuera) y como su nombre indica, consiste en entrenar el modelo tantas veces como instancias tengamos, con todas ellas salvo una distinta cada vez.

Finalmente, otro caso especial sería *leave two out*, donde dejamos una instancia fuera de todo el proceso de entrenamiento, y a partir de aquí repetimos el proceso de *leave one out*. De esta manera, tendremos una instancia que es nueva para todos los modelos y que podrá ser considerada el único elemento del conjunto de *test*.

8.3 APRENDIZAJE PROFUNDO

El cerebro del ser humano consta de unas 100 mil millones (o 100 billones americanos) de neuronas, que mantienen entre ellas una serie de interconexiones complejas, formando una gran red de memoria y procesamiento de señales. Estas neuronas son la unidad mínima funcional del sistema nervioso.

Una red neuronal artificial es una versión simplificada del sistema nervioso que compone cada ser humano, que ha servido de inspiración para su desarrollo.

8.3.1 Neuronas y Redes Neuronales

De manera general, una red neuronal está compuesta por nodos de cómputo o neuronas total o parcialmente conectadas entre sí y organizadas en capas con distintos niveles de profundidad.

En la imagen (5) se muestra el esquema de una neurona que tiene como entrada el vector $[x_1, x_2, \dots, x_m]$ y utiliza una serie de pesos $[\omega_1, \omega_2, \dots, \omega_m]$ y un sesgo (o bias) $b = \omega_0$, además de una función de activación ϕ para obtener la salida y correspondiente.

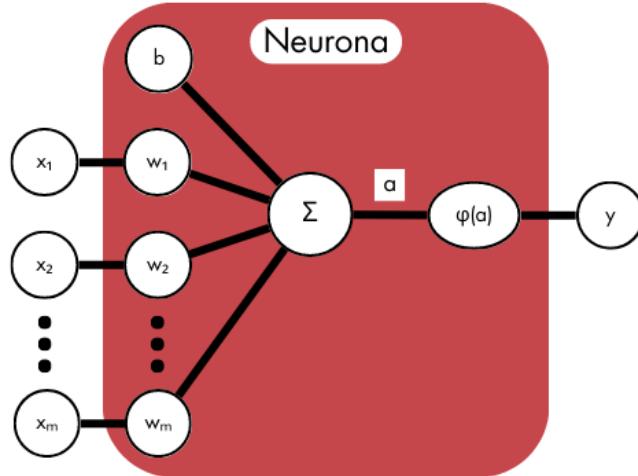


Figura 5: Estructura de una Neurona

Así, podríamos decir que una neurona es una aplicación $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$ tal que

$$\varphi(x) = \varphi([x_1, x_2, \dots, x_n]) = \varphi\left(\omega_0 + \sum_{i=1}^n \omega_i x_i\right) = y$$

es decir, computa una suma ponderada a partir de los valores de entrada (instancias u observaciones) y de los pesos. Estos pesos son los que se ajustan en el periodo de entrenamiento de la red, de manera que se realicen las ponderaciones adecuadas en función del problema que se quiera resolver.[1]

Una vez tenemos el resultado de esta suma ponderada, se debe aplicar una función de activación; es decir, una aplicación generalmente no lineal, que proporciona la salida definitiva de la neurona. Esta debe elegirse de antemano y habitualmente suelen usarse algunas de las siguientes:

- I. **Tangente Hiperbólica:** Esta función transforma los valores introducidos al intervalo $(-1, 1)$, donde los valores extremos nunca se alcanzan, pues son asíntotas de la función.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{2}{1 + e^{-2x}} - 1$$

Se caracteriza por:

- Tiene una convergencia lenta a la solución.
- Está centrada en 0 y acotada en $(-1, 1)$.

- Se utiliza para redes relacionadas con toma de decisiones.
- II. **Sigmoide:** De forma similar a la tangente hiperbólica, esta función transforma los valores introducidos al intervalo $(0, 1)$, donde, de nuevo, los valores extremos nunca se alcanzan, pues son asíntotas de la función.

$$\text{sigmoid}(x) = \frac{1}{1 - e^{-x}}$$

Se caracteriza por:

- Tiene una convergencia lenta a la solución.
- No está centrada en 0, pero sí está acotada en $(0, 1)$.
- Suele tener un buen rendimiento cuando se emplea en la última capa de la red, para problemas de clasificación binario.

- III. **Unidad Lineal Rectificada (ReLU):** Esta función definida a trozos tiene un comportamiento muy simple: transforma los valores negativos a 0 mientras que mantiene los valores positivos.

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$$

Se caracteriza por:

- Solo activa los valores positivos.
- Puede anular demasiadas neuronas si tienen valores negativos.
- Tiene un rendimiento bueno en redes convolucionales para el tratamiento de imágenes.

- IV. **Softmax:** Esta función transforma los valores de manera que asigna probabilidades, pues la suma de todas ellas siempre es igual 1.

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^m e^{x_j}}$$

Se caracteriza por:

- Se emplea con el objetivo de normalizar los resultados en una red para clasificación con varias clases.

- Está acotada en $(0, 1)$.
- Se utiliza para obtener un resultado involucrando probabilidades.
- Tiene un rendimiento bueno si se emplea en las últimas capas de una red.

8.3.2 Arquitectura de una red de Aprendizaje Profundo

Hasta este punto, hemos analizado las posibilidades de cómputo de una única neurona. Sin embargo, el verdadero potencial reside en la interconexión entre ellas.

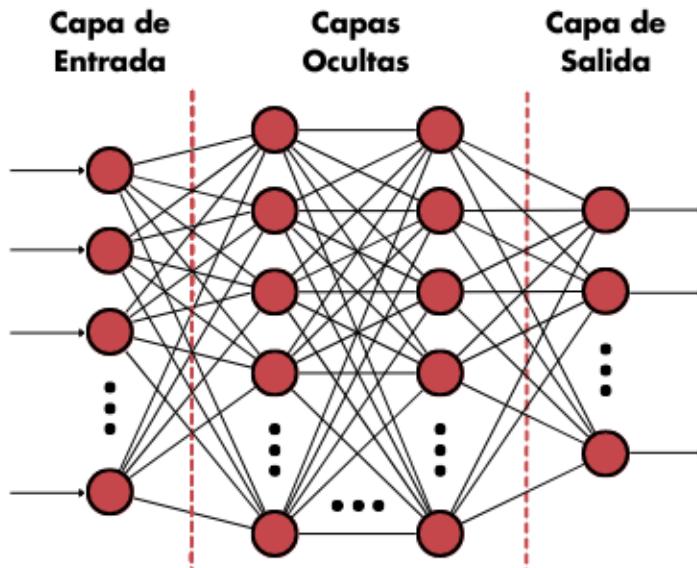


Figura 6: Arquitectura de una red neuronal con propagación hacia delante

Los modelos de redes neuronales estándar son aquellos donde la propagación del cómputo va hacia delante, del inicio al final de la red. Estas redes tienen tres capas completamente diferenciadas, como bien se ve reflejado en la imagen (6). Estas capas son las siguientes:

- Capa de entrada: Estas neuronas son las primeras de la red y reciben los datos de entrada que esta procesará posteriormente. En la mayoría de arquitecturas, el número de neuronas de la capa de entrada equivale a la dimensionalidad de los datos.

- Capas ocultas: Estas neuronas están aisladas tanto de los datos de entrada como de los de salida y pueden estar interconectadas de diversas formas, dando lugar a las distintas arquitecturas. El número de capas ocultas que tenga una red indica la profundidad de esta. Para que una red neuronal se considere como aprendizaje profundo, debe tener al menos 1.
- Capa de salida: Estas neuronas se encargan de transmitir los resultados del cómputo anterior, a través de los datos recibidos de las capas ocultas de la red.

Ahora que conocemos la manera en la que están construidas las redes neuronales, vamos a ver de qué forma se lleva a cabo el proceso de entrenamiento, de manera que converja a la solución del problema planteado.

8.3.3 Propagación hacia atrás (*Backpropagation*)

En este apartado, trataremos la forma en que una red actualiza sus pesos a la hora de encontrar una combinación de estos que proporcione un buen rendimiento para los datos de validación. Para ello, haremos uso del [Algoritmo de Descenso del Gradiente](#) aplicado a la función coste escogida.

Para poder explicar adecuadamente el algoritmo de *backpropagation*, debemos introducir la notación que vamos a usar:

- L es el número de capas (*layers*) de la red, sin contar la capa de entrada, que se entiende que es la capa 0. Además, esto implica que las capas ocultas siempre tienen índices $1, 2, \dots, L - 1$ y que la capa de salida tiene índice L .
- n^l indica el número de neuronas de la capa l , siendo n^0 el número de neuronas de entrada y n^L el número de neuronas de salida.
- ω_{jk}^l se refiere al peso que se dispone en la conexión entre la neurona k -ésima de la capa $(l - 1)$ hacia la neurona j -ésima de la capa l .
- b_j^l se refiere al sesgo de la capa l para la neurona j -ésima.
- a_j^l representa el valor de activación (una vez se ha realizado la computación con pesos y sesgo) de la neurona j -ésima de la capa l . Por ejemplo, a_1^L representa el último valor de activación para una red con una única neurona de

salida. De esta manera, podemos asociar los valores a_j^l y a_j^{l-1} con la siguiente fórmula,

$$a_j^l = \varphi \left(\sum_{k=1}^{n^{l-1}} \omega_{jk}^l a_k^{l-1} + b_j^l \right) \text{ para cada } j = 1, 2, \dots, n^l$$

- z_j^l se refiere a la entrada ponderada de la neurona; es decir, el valor de entrada de la función de activación de la neurona, por lo que $z_j^l = \omega_{jk}^l a_k^{l-1} + b_j^l$ para cada $j = 1, 2, \dots, n^l$

Podemos darnos cuenta de que los últimos 4 elementos dependen exclusivamente de la neurona en la que nos encontramos; por lo que podemos simplificar los cálculos tomándolos como vectores. Por ejemplo, en lugar de hallar a_1^l, a_2^l , etc, podemos decir que buscamos un vector A^l de dimensión n^l que verifica la siguiente ecuación:

$$A^l = \varphi \left(W^l A^{l-1} + B^l \right) = \varphi(Z^l)$$

donde hemos aplicado el mismo razonamiento para el vector B^l de dimensión n^l y para la matriz W^l de dimensión $n^l \times n^{l-1}$.

Algoritmo de Backpropagation [1]

Una vez comprendida la notación completa de una red neuronal, vamos a presentar una idea intuitiva de cómo llevar a cabo la retropropagación, propagación hacia atrás del error o *backpropagation* en una red.

Supongamos que escogemos una instancia del conjunto de entrenamiento y la introducimos en la red. Por simplicidad, supongamos también que únicamente disponemos de una neurona en la capa de salida ($n^L = 1$). Entonces, para esos datos de entrada, la red nos devuelve una salida; que al pasar por la función de coste, nos proporciona el error de la red.

Este error dado se debe en mayor o menor medida a todas las neuronas que han participado en el cómputo. Para saber la importancia de cada neurona, tenemos que propagar el error a cada una y usar el algoritmo de descenso del gradiente para modificar los pesos y sesgos de manera adecuada. Este error en cada una de las neuronas se representa como δ_j^l para la neurona j -ésima de la capa l y debe tratarse como un valor intermedio, pues lo que nos concierne es el cálculo del gradiente de los pesos y sesgos.

Intuitivamente, el error de las primeras neuronas de la red depende de todas las neuronas posteriores, mientras que las neuronas más cercanas a la capa de sa-

lida, dependerá de menos factores. Por ello, es natural pensar que para computar el error en las primeras capas, primero debemos hacerlo con las últimas, lo que implica un procedimiento iterativo que presentamos ahora:

- i. Para hallar el error en la última capa, debemos tomar tanto la variación del error con respecto a los valores de activación, como la variación de la función de activación con respecto a la entrada ponderada de la neurona, obteniendo que

$$\delta^L = \frac{\partial \mathcal{L}}{\partial A^L} \odot \varphi'(Z^L)$$

donde el símbolo \odot se refiere al producto Hadamard⁴, que lo usamos puesto que únicamente buscamos relacionar cada neurona con su elemento correspondiente, al contrario que en el producto de matrices usual.

- ii. A partir del error en la última capa, obtenemos el resto. Para ello, tenemos en cuenta los pesos de la capa posterior, a la vez que el error que estos "han producido", de manera que estamos moviendo el error hacia atrás en la red. Además, de esto, también implicamos a la derivada de la función de activación, que también mueve el error producido por esta, hacia atrás. La fórmula quedaría así:

$$\delta^l = ((W^{l+1})^T \cdot \delta^{l+1}) \odot \varphi'(Z^l) \text{ para } l = L-1, L-2, \dots, 1 \quad (8.1)$$

Cabe destacar que el orden de computación debe ser de atrás hacia el principio, empezando por δ^L , siguiendo con δ^{L-1} , δ^{L-2} , etc, hasta llegar a la primera capa.

- iii. En este punto, ya debemos conocer los vectores de error de todas las capas, y por ende, podemos obtener ya los gradientes de la función de coste tanto con respecto a los pesos como con respecto a los sesgos

$$\frac{\partial \mathcal{L}}{\partial W^l} = \delta^l \odot A^{l-1}$$

donde el valor de δ^l se obtiene de la ecuación (8.1). Si queremos hacerlo neurona a neurona, debemos adaptar las fórmulas anteriores de manera que

⁴ El producto Hadamard o producto elemento a elemento se define para matrices de igual dimensión $n \times m$ de forma que $(A \odot B)_{ij} = (A)_{ij}(B)_{ij}$ siendo el resultado una matriz de dimensión $n \times m$.

el producto Hadamard deja de ser necesario para el cálculo. Las fórmulas para cada peso y cada sesgo serían entonces

$$\frac{\partial \mathcal{L}}{\partial \omega_{jk}^l} = \delta_j^l \cdot a_k^{l-1} \text{ para cada } j = 1, 2, \dots, n^l \text{ y } k = 1, 2, \dots, n^{l-1}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \text{ para cada } j = 1, 2, \dots, n^l$$

Estos serían los datos que debemos tener para poder aplicar el descenso del gradiente con sus respectivos optimizadores a la función de coste.

Nótese que todas las ecuaciones anteriores están escritas en forma matricial, por simplicidad, por lo que para obtener los vectores gradiente de cada peso y cada sesgo, debemos tomar las columnas (o filas) correspondientes en cada caso.

Teniendo en cuenta todo lo anterior, el algoritmo se puede resumir como se presenta en (1)

Algorithm 1 Propagación hacia delante en redes recurrentes

```

1: Obtener la activación de la capa de entrada,  $A^0$ 
2: Propagar hacia delante:
3: for  $l$  in  $[1, \dots, L]$  do
4:    $Z^l \leftarrow \varphi(W^l A^{l-1} + B^l)$ 
5:    $A^l \leftarrow \varphi(Z^l)$ 
6: end for
7: Hallar el error de la capa de salida,  $\delta^L$ 
8: Propagar el error hacia atrás:
9: for  $l$  in  $[L - 1, L - 2, \dots, 1]$  do
10:    $\delta^l \leftarrow ((W^{l+1})^T \cdot \delta^{l+1}) \odot \varphi'(Z^l)$ 
11: end for
12: Computar el gradiente de la función de coste:
13: for  $l$  in  $[1, \dots, L]$  do
14:   for  $j$  in  $[1, \dots, n^l]$  do
15:      $\frac{\partial \mathcal{L}}{\partial b_j^l} \leftarrow \delta_j^l$ 
16:     for  $k$  in  $[1, \dots, n^{(l-1)}]$  do
17:        $\frac{\partial \mathcal{L}}{\partial \omega_{jk}^l} \leftarrow \delta_j^l \cdot a_k^{l-1}$ 
18:     end for
19:   end for
20: end for

```

REDES NEURONALES RECURRENTES

En esta parte, vamos a introducir un tipo de arquitectura de red neuronal diferente a la estándar que hemos visto anteriormente. Serán las que usaremos en nuestras predicciones, que vendrán explicadas más adelante. Ahora, nos centraremos en las bases de estas redes.

9.1 INTRODUCCIÓN A LAS REDES RECURRENTES

Las **redes neuronales recurrentes**, o recurrent neural networks (RNN) en inglés, son un tipo de red de neuronas que mantienen una arquitectura interconectada, representada como un grafo dirigido a lo largo de una secuencia temporal. Esto hace que las redes mantengan un comportamiento dinámico en el tiempo.[24]

Este tipo de redes pueden procesar secuencias de entrada de longitudes variables, a partir de su memoria interna. Esta característica hace que suelan emplearse para problemas tales como el reconocimiento de escritura o de voz o, en general, de cualquier clase de información que venga dada en **secuencias temporales**.

La idea detrás de las RNN es que mantienen un estado oculto a lo largo del tiempo de alta dimensión, de manera que pueden recordar posteriormente procesar información del pasado. Además, de cara al proceso de entrenamiento, se puede aplicar el descenso del gradiente, pues el cálculo de los vectores gradientes se realiza de forma eficiente.

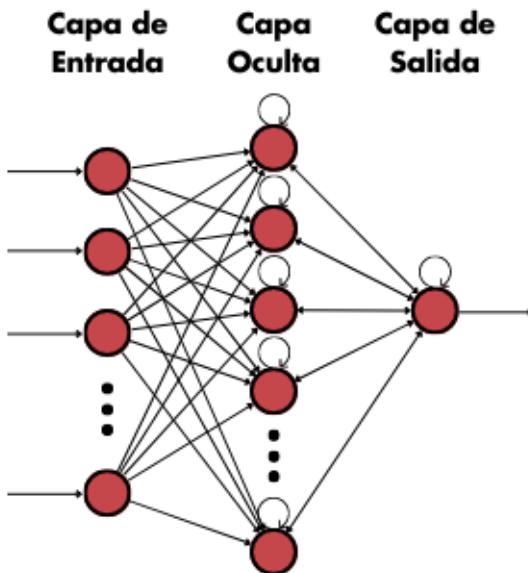


Figura 7: Arquitectura de una red neuronal recurrente

9.1.1 Despliegue de Redes Recurrentes

Existen diferentes formas de representación de redes neuronales recurrentes. Una de ellas es incluir más conexiones entre neuronas hacia atrás dentro de una red de propagación hacia delante con una única capa oculta. Otra forma, la más extendida, consiste en el despliegue de la red por capas de procesamiento a través del tiempo. Esta se adhiere adecuadamente al concepto de temporalidad que dichas redes representan.

Recordando la imagen (6), podemos observar que con respecto a la figura (7) que representa un esquema sencillo de red recurrente, existen nuevas conexiones entre neuronas. Estas son las relaciones que permiten a la red tratar información a través de secuencias. No obstante, la red desplegada es aquella de la figura (8), donde se observa claramente el carácter temporal de estas, de forma que cada neurona mezcla la computación de nueva información con información anterior.

Con respecto a esta última figura con la red desplegada, la cantidad de bloques denominados como S dependerá de la longitud de la secuencia de entrada, mientras que cada uno de estos bloques comparte tanto pesos como sesgos. De esta forma, cada salida z_t de la red, depende de toda la secuencia de información x_1, x_2, \dots, x_t que le precede.

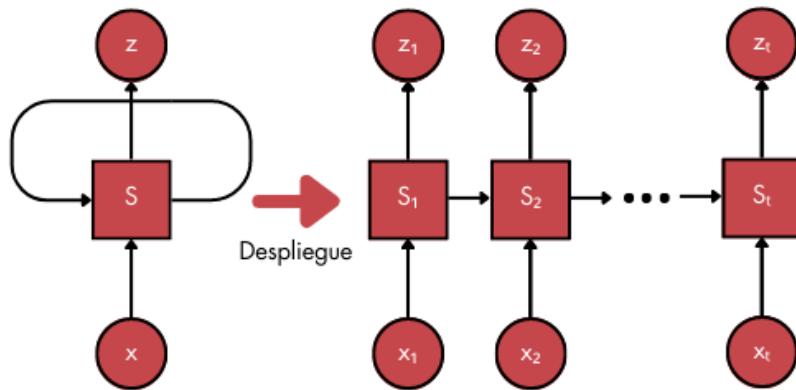


Figura 8: Arquitectura desplegada de una red neuronal recurrente

9.1.2 Ventana Temporal

Las redes recurrentes juegan un papel fundamental a la hora de predecir valores futuros en series temporales. No obstante, al tratar con series temporales, debemos tener claros ciertos parámetros críticos antes de comenzar. En este momento se introduce el concepto de **ventana temporal**, que se refiere a la cantidad de datos elegidos para hacer una predicción, la amplitud de esta y su situación en el tiempo. Su importancia radica en la gran diferencia que existe entre, por ejemplo, tener una secuencia de 20 observaciones, buscando averiguar la siguiente; o bien, tener una secuencia de 10 observaciones para averiguar las 10 siguientes; o incluso tener 20 observaciones y tratar de predecir dentro de 20 instantes de tiempo.[25]

En la figura (9), podemos observar 2 ejemplos como acabamos de decir. El primero de ellos es una secuencia con 6 datos de entrada, que busca predecir 2 observaciones pasados 5 instantes de tiempo. Este tipo de ventanas podrían ser útiles en las predicciones orientadas a la toma de decisiones, donde nos importa lo que va a pasar dentro de un tiempo (en este caso, 5 horas, días, o cualquiera que sea la medida del tiempo). El segundo ejemplo trata una secuencia más estándar, donde dado un cierto número de observaciones, tratamos de predecir la inmediatamente siguiente. En nuestro trabajo trataremos estos tipos, además de otros tantos.

La ventana temporal es una herramienta perfecta para lidiar con la correlación. Esto se debe a que en la mayoría de las series temporales, existe una **dependencia temporal**, de forma que los datos en ciertos instantes de tiempo están correlados con otras observaciones dadas en instantes anteriores (o lo estarán con posteriores). Si escogemos una ventana adecuada, podemos capturar de manera

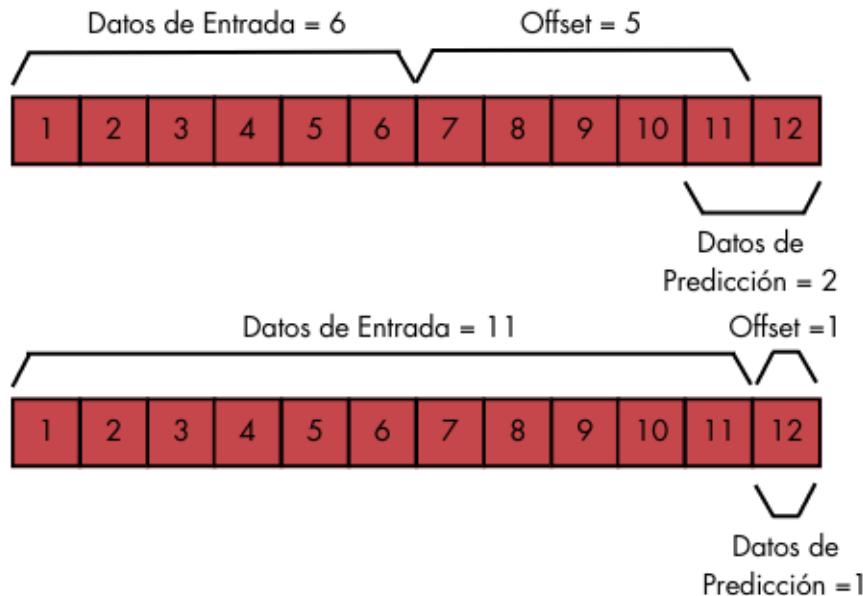


Figura 9: Ejemplos de tipos de Ventanas Temporales

más precisa los patrones de una serie y mejorar la calidad de nuestras predicciones.

En el momento de la elección, el equilibrio es fundamental. Una secuencia de entrada muy pequeña podría hacer que la red no llegue a aprender los patrones necesarios; mientras que una secuencia de entrada demasiado grande podría incluir información redundante, que a la larga sería confusa para el modelo y provocaría ejecuciones más lentas.

9.2 PROPAGACIÓN HACIA ATRÁS A TRAVÉS DEL TIEMPO (BPTT)

Cuando tratamos con redes neuronales más estándar, en el proceso de entrenamiento de los modelos se debe emplear el algoritmo de propagación hacia atrás. Esta es la forma en que el error llega a todas las neuronas, de manera que se pueden modificar los pesos y sesgos acorde a la cantidad de error que provoca cada uno. Existe una versión de este algoritmo cuando añadimos el factor temporal, siendo mucho más complejo que el anterior, pero a la vez de mucha utilidad para la propagación del error; no solo a través de la red, sino a través del tiempo. Este se denomina **propagación hacia atrás a través del tiempo** o *backpropagation through time* (BPTT) en inglés. [26]

En nuestro caso, vamos a explicar de manera breve cual sería el proceso hasta obtener los gradientes necesarios para actualizar los pesos y sesgos, puesto que

se podría inferir de manera intuitiva a partir del algoritmo de [Propagación hacia atrás \(Backpropagation\)](#) ya descrito anteriormente.

Supongamos una secuencia $\{x_t\}_{t=1}^T$ de T observaciones que actuará como secuencia de entrada, donde cada observación es un vector con los diferentes atributos. Entonces, la red recurrente obtiene a partir de esta una secuencia de estados ocultos (se emplean para el procesamiento interno de la red) $\{h_t\}_{t=1}^T$ y una secuencia de datos de salida $\{a_t\}_{t=1}^T$, ambos con un total de T observaciones. A partir de aquí, vamos a introducir una nueva notación con la que poder adaptarnos a la arquitectura de una red neuronal recurrente:

- W_x^h se refiere a la matriz donde se disponen los pesos para la conexión entre los nodos de entrada y la capa oculta.
- W_h^h se refiere a la matriz donde se disponen los pesos para la conexión entre dos capas ocultas de la red.
- W_h^y se refiere a la matriz donde se disponen los pesos para la conexión entre la (última) capa oculta y la capa de salida.
- B_h representa los sesgos de las neuronas de la capa oculta.
- B_y representa los sesgos de las neuronas de salida.

Tomando la información anterior, podemos deducir, por ejemplo, que los pesos W_h^y y los sesgos B_h son los pesos y sesgos compartidos a lo largo de toda la secuencia.

9.2.1 Propagación hacia delante en una RNN

Ahora, vamos a ver cuál es el funcionamiento habitual de una red recurrente, desde que se recibe una secuencia de datos de entrada $\{x_t\}_{t=1}^T$ hasta que se obtiene la salida correspondiente.

En el algoritmo (2) se entiende que el valor de h_0 es cero, pues no se ha calculado ningún estado oculto hasta ese momento.

Además, en este caso tenemos 2 funciones de activación dentro de la red, la primera ψ se encarga de la activación de los estados ocultos; mientras que la segunda, φ es la análoga a aquella de las redes que hemos visto anteriormente.

Algorithm 2 Propagación hacia delante en redes recurrentes

```

1: for  $t$  in  $[1, \dots, T]$  do
2:    $u_t \leftarrow W_x^h \cdot x_t + W_h^h \cdot h_{t-1} + B_h$ 
3:    $h_t \leftarrow \psi(u_t)$ 
4:    $z_t \leftarrow W_h^y \cdot h_t + B_y$ 
5:    $a_t \leftarrow \varphi(z_t)$ 
6: end for

```

9.2.2 Algoritmo de BPTT

Una vez comprendido de que manera se obtienen las salidas en una red recurrente, veamos la forma de propagar el error a lo largo de las neuronas y del tiempo. [26]

Antes de empezar, debemos saber que en las redes recurrentes el valor del error suele computarse como la suma de los errores a lo largo del tiempo; es decir, obteniendo un valor de error para cada instante de tiempo entre 1 y T y posteriormente sumarlos. Quedaría

$$\mathcal{L}(a, y) = \sum_{t=1}^T \mathcal{L}_t(a_t, y_t)$$

donde \mathcal{L}_t es la función de coste empleada en cada instante de tiempo y el vector y_t son los valores esperados en cada instante de tiempo.

En este caso, el algoritmo se debe realizar iterativamente para cada instante de tiempo, propagando el error para todo $t = 1, \dots, T$. Por lo que se entiende que todos los valores iniciales $\frac{\partial \mathcal{L}}{\partial W_h^y}, \frac{\partial \mathcal{L}}{\partial W_h^h}, \frac{\partial \mathcal{L}}{\partial W_x^h}, \frac{\partial \mathcal{L}}{\partial B_y}, \frac{\partial \mathcal{L}}{\partial B_h}$ y $\frac{\partial \mathcal{L}}{\partial h_0}$ comienzan siendo 0. Así, la propagación hacia atrás a través del tiempo quedaría de la siguiente manera:

De esta manera, en la T -ésima iteración (cuando $t = 1$), habremos conseguido los gradientes buscados, que son:

$$\frac{\partial \mathcal{L}}{\partial W_h^y}, \frac{\partial \mathcal{L}}{\partial W_h^h}, \frac{\partial \mathcal{L}}{\partial W_x^h}, \frac{\partial \mathcal{L}}{\partial B_y}, \frac{\partial \mathcal{L}}{\partial B_h} \text{ y } \frac{\partial \mathcal{L}}{\partial h_0}$$

con los que podemos aplicar el algoritmo de descenso del gradiente sin ninguna dificultad añadida. Por lo que, en definitiva, únicamente aumentan la cantidad de pasos a realizar, pero no la complejidad con respecto al algoritmo de *backpropagation* habitual.

Algorithm 3 Propagación hacia atrás a través del tiempo

```

1: for  $t$  in  $[T, \dots, 1]$  do
2:    $\frac{\partial \mathcal{L}}{\partial z_t} \leftarrow \frac{\partial \mathcal{L}_t}{\partial a_t} \cdot \varphi'(a_t)$ 
3:    $\frac{\partial \mathcal{L}}{\partial B_y} \leftarrow \frac{\partial \mathcal{L}_t}{\partial B_y} + \frac{\partial \mathcal{L}}{\partial z_t}$ 
4:    $\frac{\partial \mathcal{L}}{\partial W_h^y} \leftarrow \frac{\partial \mathcal{L}}{\partial W_h^y} + \frac{\partial \mathcal{L}}{\partial z_t} \cdot (h_t)^T$ 
5:    $\frac{\partial \mathcal{L}}{\partial h_t} \leftarrow \frac{\partial \mathcal{L}}{\partial h_t} + (W_h^y)^T \cdot \frac{\partial \mathcal{L}}{\partial z_t}$ 
6:    $\frac{\partial \mathcal{L}}{\partial a_t} \leftarrow \frac{\partial \mathcal{L}}{\partial h_t} \cdot \psi'(a_t)$ 
7:    $\frac{\partial \mathcal{L}}{\partial W_x^h} \leftarrow \frac{\partial \mathcal{L}}{\partial W_x^h} + \frac{\partial \mathcal{L}}{\partial a_t} \cdot (x_t)^T$ 
8:    $\frac{\partial \mathcal{L}}{\partial B_h} \leftarrow \frac{\partial \mathcal{L}_t}{\partial B_h} + \frac{\partial \mathcal{L}}{\partial a_t}$ 
9:    $\frac{\partial \mathcal{L}}{\partial W_h^h} \leftarrow \frac{\partial \mathcal{L}}{\partial W_h^h} + \frac{\partial \mathcal{L}}{\partial a_t} \cdot (h_{t-1})^T$ 
10:   $\frac{\partial \mathcal{L}}{\partial h_{t-1}} \leftarrow (W_h^h)^T \cdot \frac{\partial \mathcal{L}}{\partial a_t}$ 
11: end for
  
```

La única complicación extra se debe a la naturaleza iterativa del algoritmo, pues esto hace que la función de coste sea muy sensible a cambios. Por ello, una ligera modificación puede producir efectos que únicamente se lleguen a ver pasadas muchas iteraciones más tarde. Este fenómeno se conoce como efecto mariposa¹ y entre otras cosas, provoca que la función de coste se vuelva completamente discontinua. A su vez, esto provoca que el gradiente se desvanezca (*vanishing gradient*) o alcance valores muy grandes (*exploding gradient*) con el tiempo, obteniendo como consecuencia directa la imposibilidad de aplicar el algoritmo de descenso del gradiente de manera efectiva.[27]

Para resolver el problema de desvanecimiento o explosión del gradiente, se desarrollaron las redes neuronales recurrentes LSTMs.

¹ En teoría del caos, nos referimos a la inestabilidad de un sistema dinámico (una red neuronal recurrente, por ejemplo) como el efecto mariposa, de forma que cambiando las condiciones de entrada ligeramente, la función recurrente proporciona un resultado muy distinto

9.3 ARQUITECTURA LONG-SHORT TERM MEMORY (LSTM)

Como hemos visto anteriormente, las redes neuronales recurrentes son capaces de captar patrones del pasado para predecir hechos en el futuro. En algunos casos, basta con conocer poca información en un pasado muy reciente. Por ejemplo, en el caso de modelos de lenguaje, si tenemos una frase que comienza con "El mejor amigo del hombre es el" y queremos predecir la siguiente palabra, no necesitamos más contexto para averiguar la respuesta. Sin embargo, si al principio de un escrito se incluye la frase "Estamos en verano" y unas líneas después queremos predecir la próxima palabra después de "En esta época del año suele hacer mucho", necesitamos el contexto completo del texto para poder darle respuesta. En este último caso, el espacio entre la información relevante y el lugar donde la necesitamos puede volverse tan grande que las redes recurrentes estándar no sean capaces de proporcionar un rendimiento óptimo.

Este problema incluye lo que se denominan dependencias a largo plazo, y fueron el detonante para el descubrimiento de las redes LSTM cuyas siglas significan *Long-Short Term Memory*; es decir, redes neuronales de memoria a corto y largo plazo. La idea de estas es evitar los problemas de memoria con las dependencias a largo plazo, recordando información durante largos períodos de tiempo durante la ejecución.

Este tipo de redes pueden aplicarse a una amplia variedad de problemas, mejorando el rendimiento de una red recurrente estándar. Por ejemplo: reconocimiento de escritura a mano, reconocimiento de voz, detección y clasificación de spam, reconocimiento de géneros musicales o análisis de sentimientos, entre otros.

9.3.1 Introducción a las redes LSTMs

Para explicar al detalle cómo funcionan este tipo de redes, usaremos una serie de ilustraciones provenientes de [28].

Para comenzar, mostramos cómo se ve una red LSTM desplegada. Para ello, en la figura (10) se representan 3 bloques de esta, de forma que en el bloque central podemos observar las operaciones que se llevan a cabo dentro de uno:

- Las flechas representan el flujo de información.
- Los rectángulos amarillos representan la operación correspondiente a una capa de la red neuronal.

- Los círculos rojos representan la operación llevada a cabo para concatenar 2 flujos de datos.
- La operación $tanh$ en rojo representa una operación a realizar en un único flujo de información.

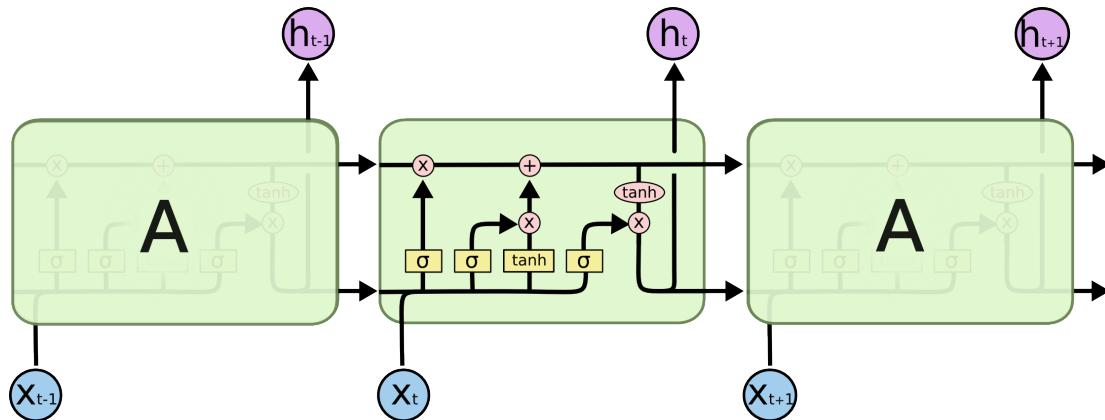


Figura 10: Red LSTM desplegada [28]

9.3.2 Idea Principal de las redes LSTMs

Uno de los puntos clave a la hora de entender la importancia de las redes LSTMs es lo que se denomina el **estado de la celda**. Este se refiere a la línea horizontal que recibe como entrada el estado de la celda anterior y tras 2 operaciones, obtiene el estado de la celda actual.[4]

El estado de la celda se puede ver en la figura (11) como una cinta transportadora que recorre el bloque y que solo sufre 2 interacciones (operaciones lineales). Posteriormente veremos que es habitual que los valores de entrada y de salida sean muy similares, sin sufrir modificaciones. Podemos tomar dichas modificaciones como la manera que tiene la celda de añadir o eliminar información del estado; esto se realiza mediante **compuertas**, que están compuestas por la operación sigmoide y

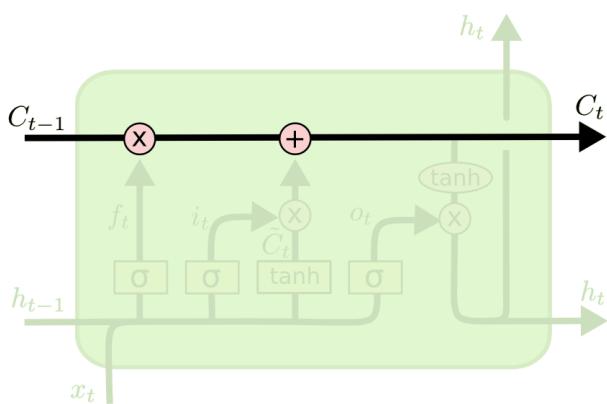


Figura 11: Representación del Estado de la Celda en un bloco LSTM [28]

por una operación de suma o producto. La idea es que valores cercanos a 1 modifican considerablemente el estado de la celda; mientras que valores cercanos a 0 no dejan pasar prácticamente nada de información al estado de la celda. Cada bloque dispone de 3 compuertas que veremos a continuación.

9.3.3 Funcionamiento de una celda LSTM

Cuando un bloque en un tiempo t recibe el estado oculto h_{t-1} de la celda anterior y el valor de entrada x_t , debe tomar una sucesión de decisiones.

Forget Gate

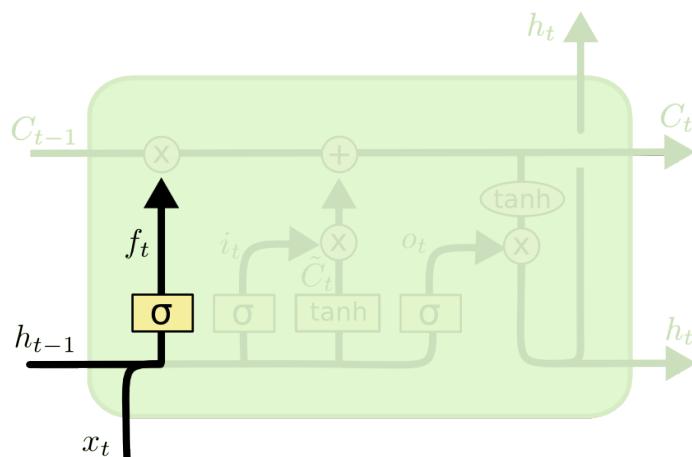


Figura 12: Representación de la Compuerta de Olvido en un bloque LSTM [28]

En primer lugar, debe decidir, de la información que ha recibido, qué información olvidar y qué información mantener para completar el estado de la celda. Esta decisión la toma una parte de la celda llamada **compuerta de olvido**, o *forget gate* en inglés, de forma que a partir de h_{t-1} y x_t , computa un número entre 0 y 1 que representa cuánta "cantidad de información" mantener.

Este valor se representa en la figura (12) como f_t , indicando el valor de la compuerta *forget* en el tiempo t y se calcula como

$$f_t = \sigma(W_{forget} \cdot [h_{t-1}, x_t] + B_{forget})$$

Finalmente, se multiplica este valor por el estado de la celda, actualizando este valor. Es una forma de decirle al estado, "olvida esto" y el cálculo es el siguiente:

$$C_{t_{forget}} \leftarrow f_t \cdot C_{t-1}$$

Input Gate

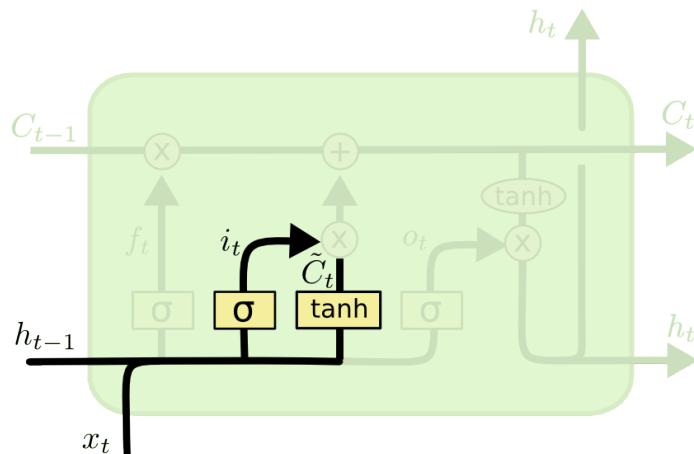


Figura 13: Representación de la Compuerta de Entrada en un bloque LSTM [28]

Mientras que en el paso anterior, se decidió qué información eliminar del estado de la celda, ahora veremos qué información nueva se va a añadir. Para computar este valor, se distinguen tres partes, que pueden verse gráficamente en la imagen (13). El proceso es el siguiente:

1. Una compuerta llamada **compuerta de entrada**, o *input gate* en inglés, que decide los valores a actualizar a partir de h_{t-1} y x_t , pasándolos por la función sigmoide con sus respectivos pesos y sesgos. Se calcula como

$$i_t = \sigma(W_{input} \cdot [h_{t-1}, x_t] + B_{input})$$

2. Una capa con la función de tangente hiperbólica que crea una serie de valores candidatos que podrían añadirse al estado de la celda. Se calcula como

$$\tilde{C}_t = \tanh(W_{\tilde{C}} \cdot [h_{t-1}, x_t] + B_{\tilde{C}})$$

3. Se calcula el producto de los valores i_t y \tilde{C}_t y el resultado se suma a lo que ya hubiera en el estado de la celda, de forma que:

$$C_{t_{input}} \leftarrow C_{t_{forget}} + i_t \cdot \tilde{C}_t$$

Esto es la manera en la que, dados unos candidatos a nuevo valor de estado, los escalamos para decidir cómo de importantes van a ser cada uno.

Este último cambio en el estado de la celda proporciona el valor definitivo de esta, quedando finalmente

$$C_t \leftarrow f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

Output Gate

Por último, una vez se ha decidido qué valores olvidar y cuáles añadir, debemos computar el valor de salida o estado oculto de la celda. Este se calcula a partir del estado de la celda (que ya no sufrirá más modificaciones) junto con los valores h_{t-1} y x_t . El valor de salida se computa por medio de los pesos y sesgos de la **compuerta de salida** a través de la función sigmoide, obteniendo

$$o_t = \sigma(W_{output} \cdot [h_{t-1}, x_t] + B_{output})$$

Finalmente, el estado oculto en tiempo t se obtiene escalando el estado de la celda al intervalo $[-1, 1]$ mediante la función tangente hiperbólica y multiplicando por el valor de salida de la compuerta de salida, quedando el estado oculto de la siguiente manera:

$$h_t = o_t \cdot \tanh(C_t) = o_t \cdot \tanh(f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t)$$

En la figura (14) pueden verse el orden de los cálculos explicados anteriormente, con los que hemos obtenido los valores resultantes de una celda LSTM, C_t y h_t .

Variantes de LSTMs

El procedimiento descrito es al que responden las celdas LSTM originales. No obstante, con el paso del tiempo y las nuevas necesidades, se han desarrollado diferentes versiones de estas. No vamos a desarrollarlas al detalle, pero se enumeran a continuación las más conocidas:

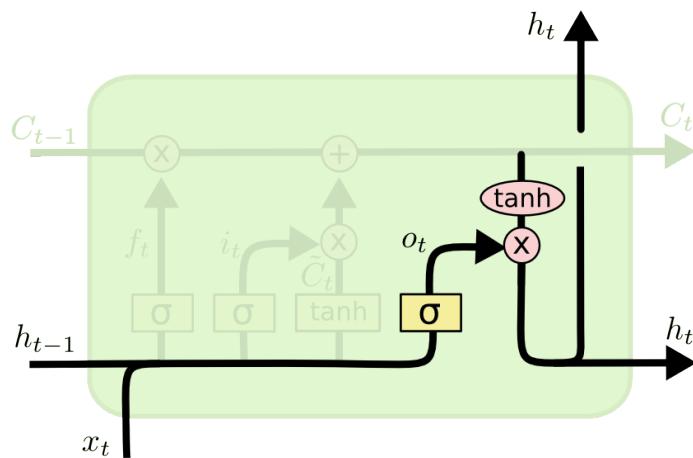


Figura 14: Representación de la Compuerta de Salida en un bloque LSTM [28]

- **LSTM con conexiones de mirilla:** Estas celdas se diferencian de las estándar en que las 3 compuertas tienen en cuenta el estado de la celda C_{t-1} a la hora de realizar sus cálculos correspondientes. [29]

Los bloques de estas redes tienen la estructura que se muestra en la figura (15).

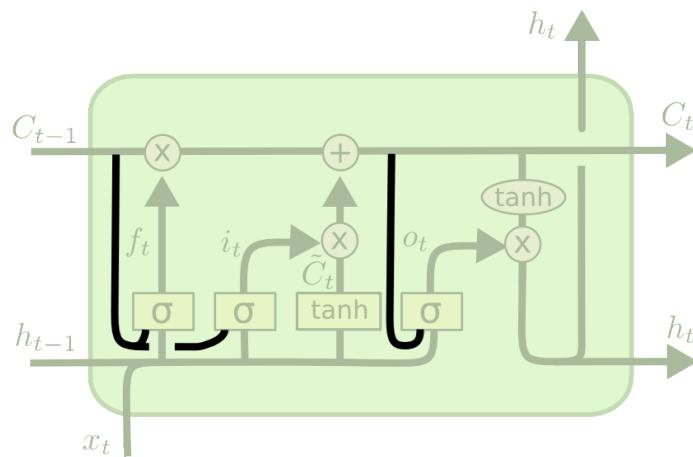


Figura 15: Representación de un bloque LSTM con conexiones de mirilla [28]

- **LSTM con duplicación de compuertas de olvido y entrada:** Estas se construyen con la idea de que las decisiones de qué eliminar o añadir en el estado de la celda deben tomarse conjuntamente. Por lo que este tipo de redes tienen unas celdas con estas compuertas interconectadas de manera inversamente proporcional. Así, "lo que se olvida, no se vuelve a añadir". [28]

Los bloques de estas redes tienen la estructura que se muestra en la figura (16).

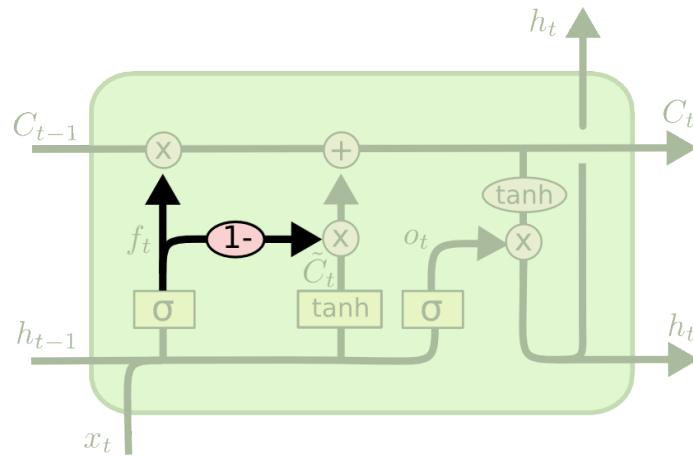


Figura 16: Representación de un bloque LSTM con duplicación de compuertas de olvido y entrada [28]

- **Unidad de Compuertas Recurrentes (GRU):** Esta es la variante más conocida, pues tiene resultados contrastados. La idea clave es combinar las compuertas de olvido y de entrada en una única compuerta llamada **compuerta de actualización**. Además, también combina el estado de la celda con el estado oculto, lo que resulta en una gran simplificación de los cálculos, reduciendo la complejidad de los modelos. [30]

Los bloques de estas redes tienen la estructura que se muestra en la figura (17).

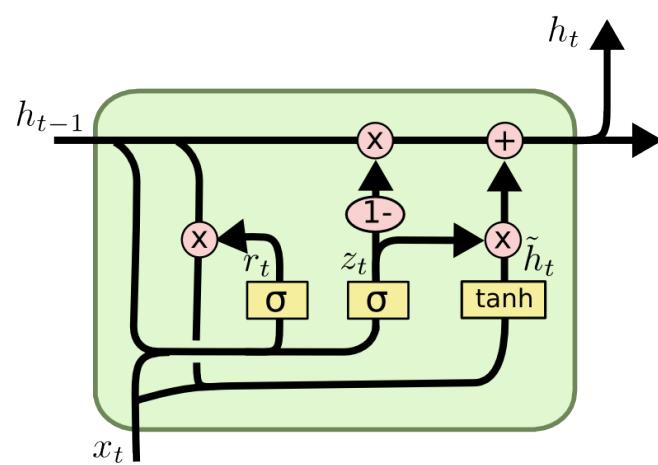


Figura 17: Representación de un bloque GRU [28]

Parte III

APLICACIÓN

Desarrollo de modelos predictivos basados en redes neuronales recurrentes con bloques LSTM y aplicación de estos a experimentos de algoritmos de disciplinado de relojes atómicos. Se aplican las técnicas desarrolladas en los apartados anteriores de matemáticas e informática y se hace uso de los conceptos relacionados más relevantes. Se acaba con una discusión de los resultados y una conclusión del trabajo.

PRESENTACIÓN DE LOS DATOS

Para poder emplear las técnicas de predicción de series temporales con redes neuronales que venimos de explicar, se va a utilizar un conjunto de datos proveniente de la empresa Safran - Navigation & Timing. Debido a la confidencialidad de estos, en el [repositorio del trabajo](#), solamente se encuentran disponibles los *notebooks* usados como parte del desarrollo, no así el conjunto de datos ni las salidas correspondientes del código, manteniendo así el carácter privado de la información.

Los datos iniciales provienen de diversos osciladores o relojes atómicos; en particular, de aquellos que tienen como número de dispositivo el valor 153, 154 y 155. Cada uno mantiene una configuración distinta, por lo que hemos escogido aquel cuyos datos presentan mayor suavidad, descartando aquellos que disponen de grandes picos y alta estacionariedad. Los datos de los osciladores descartados están dispuestos en el apéndice [Otros Relojes Atómicos](#).

De cada reloj, se realiza una observación de los siguientes atributos por cada segundo¹: algBsh, algMultiSrcErrStat, algMultiTimeComp, algScore, algSrcErrStat, gnssAgcValue, gnssAzim, gnssCno, gnssElev, gnssSvUsed, ntpOffset, rsPhaseData, srcStateDegraded, srcStateRefs, srcStateUsable, srcStateValid, ssHoldover, ssSync, sysOscTemp, xoDac, xoFreqError, xoPhaseError y xoState. Nos centraremos en los siguientes:

- **xoDac:** Valor DAC del control de disciplinado del reloj.
- **xoFreqError:** Error de frecuencia del disciplinado.
- **xoPhaseError:** Error de fase del disciplinado.
- **sysOscTemp:** Temperatura del sistema del reloj.

¹ Únicamente se describirán aquellos atributos que sean de utilidad para nuestro trabajo

- **xoState:** Estado (fase) de disciplinado.

Además, estas observaciones se presentan individualmente en formato csv, de manera existe un archivo para cada uno de los atributos anteriores, donde cada archivo dispone de 2 columnas; la primera, con los instantes de tiempo o *timestamps* y la segunda, con el valor observado correspondiente a tal instante.

Por último, los datos se disponen en distintas temporalidades, por **horas**, por **minutos** o por **segundos**. Los datos obtenidos por minuto son una media de los 60 segundos que forman tal minuto, ocurriendo lo análogo para la temporalidad horaria. Por la naturaleza de los relojes atómicos, la predicción de los valores por horas carece de utilidad práctica, por lo que debemos plantearnos qué temporalidad emplear, minutos o segundos.

No obstante, la pregunta tiene respuesta fácil cuando observamos la figura (18). Esta imagen muestra todos los datos que tenemos de disciplinado en segundos, donde se aprecia que apenas existen variaciones y estas son altamente bruscas, no dejando apenas margen para que un modelo pueda interpretarlos. Además de las irregularidad, tenemos que hablar de la escasez, porque tenemos muy pocos datos diferentes. Por tanto, descartamos esta temporalidad y en todo nuestro trabajo la medida del tiempo serán los **minutos**.



Figura 18: Variable DAC medida en Segundos

10.1 LECTURA Y PROCESAMIENTO DE LOS DATOS

La lectura y procesamiento de los datos se encuentra dispuesta en el archivo de nombre *data_preparation.ipynb*. Aquí, se hace uso de diversas funciones que nombraremos de manera breve.

En primer lugar, *get_file_path()* es la función auxiliar que nos permite obtener la ruta correspondiente a los archivos que necesitamos. Ahora, por un lado, para tratar con los estados de disciplinado, tenemos las funciones *get_raw_states()* y *delete_constant_lock_values()*, que procesan el archivo con los estados y eliminan las observaciones recogidas en un estado de disciplinado de bloqueo constante, respectivamente. Esto último se puede observar en la imagen (19), pues a partir de aproximadamente el instante de tiempo 6800, vemos que se mantiene constante el estado 4; esto es, el de bloqueo del oscilador.

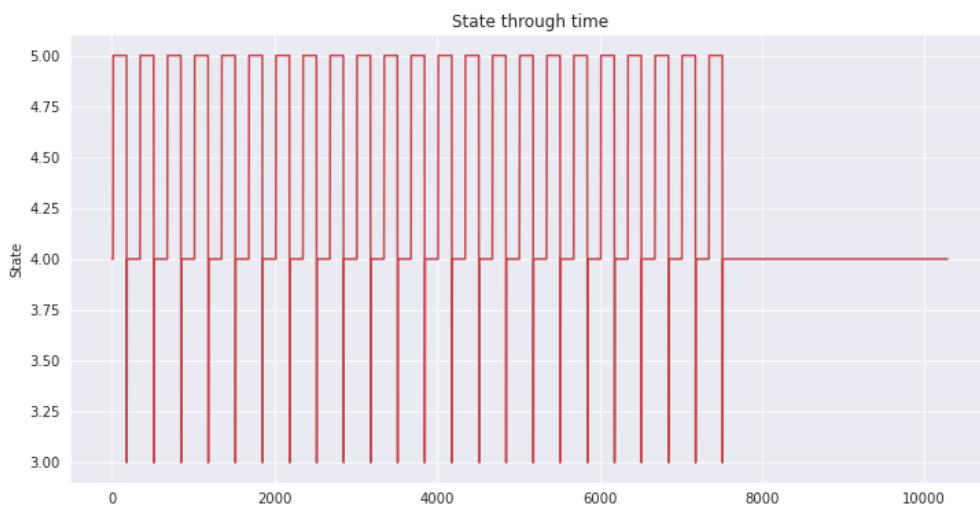


Figura 19: Fases de Disciplinado a lo largo del tiempo

Por otro lado, la función *get_lock_state_timestamps()* se encarga de eliminar todas las observaciones que no se corresponden con la fase de bloqueo², resultando en la figura (20), donde también podemos observar que el cambio entre estados no es instantáneo. Este fenómeno se debe a la temporalidad que manejamos, puesto

² La fase de bloqueo es aquella en que las observaciones se recogen de manera manual y a la vez precisa, para poder evaluar correctamente las métricas del reloj atómico. Es la única sobre la que tiene sentido llevar a cabo nuestro estudio

que para computar los datos en minutos, se ha hecho la media de los datos en segundos. Por lo que, a no ser que el cambio de estado se realice exactamente en el segundo 0 de cada minuto, siempre nos encontraremos con estos cambios progresivos midiendo en minutos.

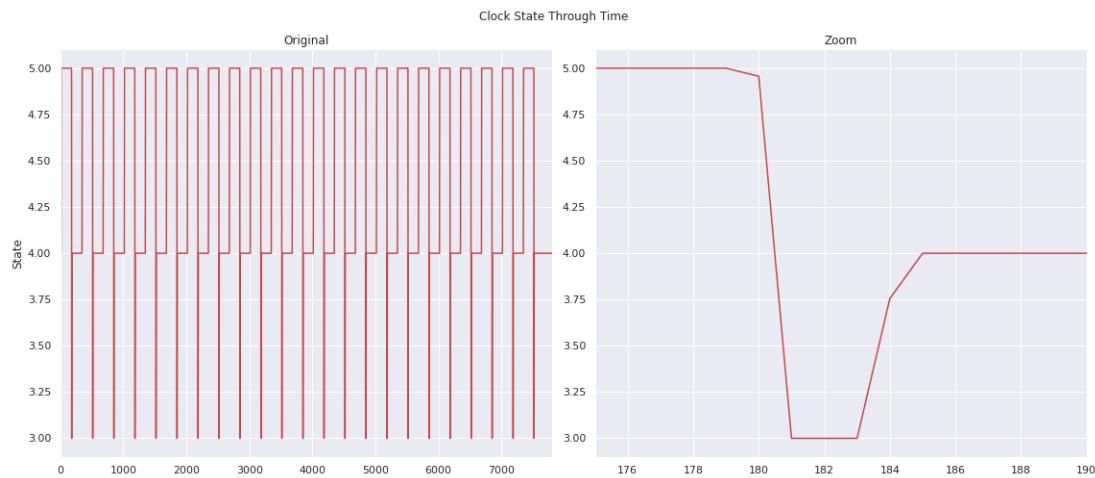


Figura 20: Modificación del estado del reloj con el tiempo

Finalmente, `join_features()` y `format_features()` concluyen el trabajo de lectura, creando el *data frame* que engloba las 4 variables que se emplean en este trabajo, únicamente observadas en estado de bloqueo.

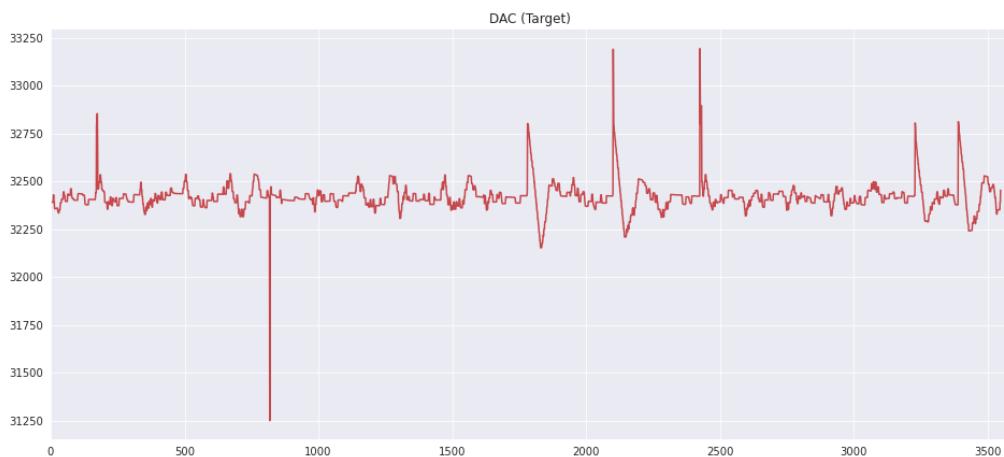


Figura 21: Valor del DAC sin procesado de Etapas de Disciplinado

En este punto, el valor de disciplinado a predecir es el de la figura (21), donde se aprecia numerosos picos y grandes irregularidades. Además, ampliando la imagen, podemos observar ciertos patrones que se repiten aproximadamente cada 150-160 instantes de tiempo. Esto se debe a que el disciplinado, como ya sabemos, se realiza por etapas que debemos tener en cuenta en esta parte de procesado de datos, antes de tratar con ellos.

10.2 ETAPAS DE DISCIPLINADO

Durante la lectura de las observaciones del reloj atómico, el cambio entre estados se produce manualmente, pues no existe la posibilidad de automatizar este proceso. Por tanto, el tiempo que se encuentra el reloj en cada estado no siempre es el mismo. En particular, en algunos casos, el reloj pasa 161 minutos en el estado de bloqueo; mientras que en otros casos, pasa 157. Incluso existe en nuestros datos un error (humano) donde el reloj pasa menos de 1 minuto en estado de bloqueo.

Este último caso es tan pequeño que ni se aprecia en la figura (20) como un cambio de estado, pero ha de tenerse en cuenta de cara al procesamiento posterior.

En definitiva, si tratamos de usar los datos "crudos" para llevar a cabo nuestras predicciones, nos encontraremos con una serie con una fuerte componente estacional. Es más, por la naturaleza de un reloj atómico de este tipo, realizar predicciones sin diferenciar el momento en el que se cambia de estado carece de sentido práctico.

Por todo lo anterior, se introduce el concepto de **etapa de disciplinado**. Diremos que una etapa de disciplinado comienza en el momento en el que se inicia el estado de bloqueo del reloj y acaba cuando el estado del reloj se modifica. En nuestros datos, encontramos un total de 22 etapas de disciplinado, cada una se representa con un color diferente en la figura (22) y como se puede observar, entre ellas guardan algún tipo de relación, por provenir del mismo reloj.

Inestabilidad de los datos

Como se puede apreciar en la imagen (22), al inicio de cada etapa existe cierta inestabilidad en el oscilador. Esto se debe a 2 factores:

- El algoritmo de disciplinado tiene cierto retardo desde que localiza el error de fase hasta que lo corrige. Luego, a pesar de oscilar alrededor de una media

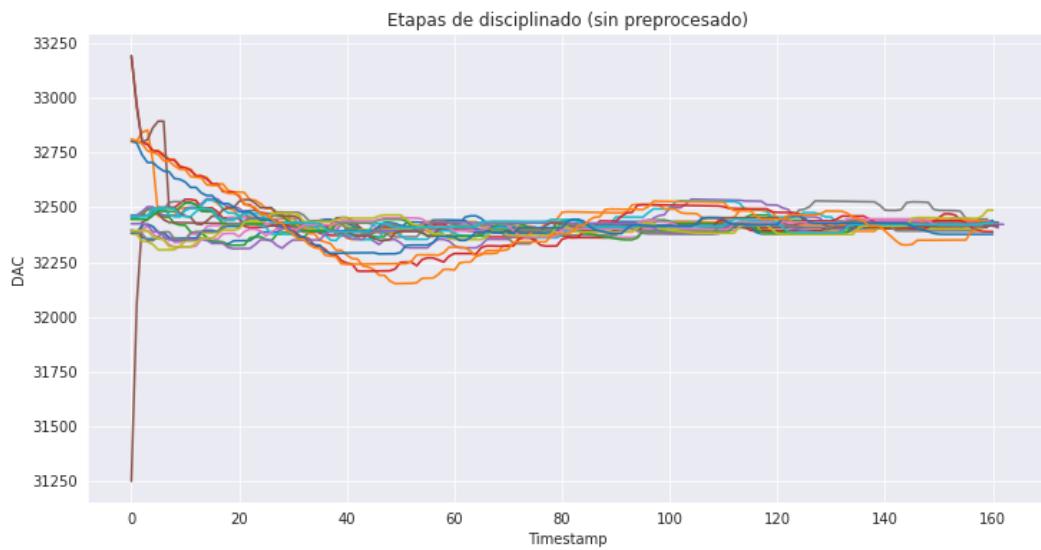


Figura 22: Valor del DAC por Etapas de Disciplinado

de 32400, al comenzar la etapa de bloqueo necesita una serie de instantes de tiempo para estabilizarse.

- Las observaciones realizadas se toman en cada segundo, por lo que al escalar esta medida a minutos (tomando la media de las observaciones en cada minuto) estamos perdiendo precisión a la hora de conocer cuál es el estado exacto del valor de disciplinado en cada momento.

Cabe destacar que el primer factor tiene un carácter impredecible, puesto que se debe al comportamiento del oscilador en otros estados de disciplinado. En otras palabras, cada una de las etapas comienza tras un periodo en el que el reloj ha operado libremente (estado de *free run*) sin disciplinado, donde muestra una deriva estocástica. Por ello, dentro de un rango de trabajo, los valores iniciales de disciplinado tienen una alta componente aleatoria.

Para tratar de corregir la inestabilidad mostrada al inicio de cada etapa, debemos filtrar estos primeros valores para lograr un resultado que posteriormente se pueda predecir. Para ello, hemos usado una función llamada *stabilize_stages()*, que elimina los primeros valores más oscilantes, además de conseguir que todas las etapas tengan la misma longitud. De esta forma, el tratamiento posterior se simplificará considerablemente.

El resultado puede verse al comparar la figura (21) inicial con la figura (23), donde podemos observar mayor número de patrones, además de un carácter estacional claro cada 150 instantes de tiempo. Por otro lado, si tomamos la gráfica

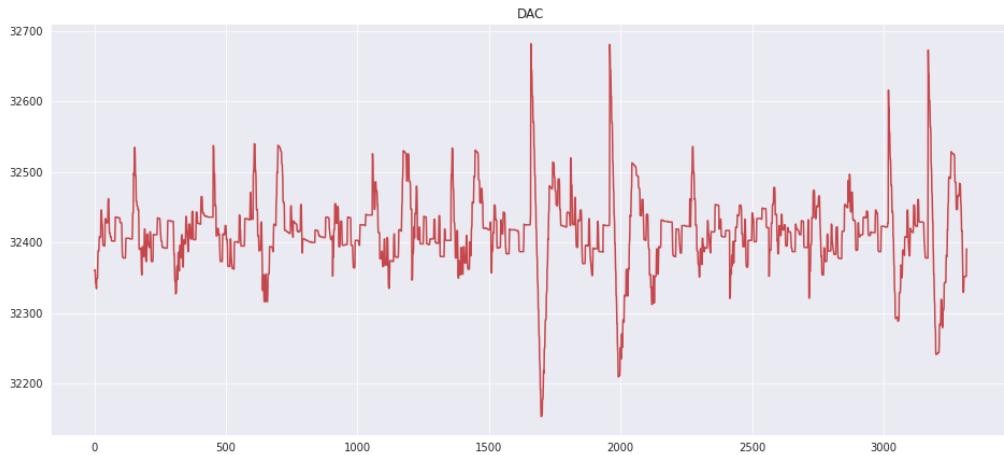


Figura 23: Valores de DAC tras filtrar los valores inestables

con todas las etapas superpuestas, también se observa una gran diferencia tras el filtrado de estos valores inestables propios del inicio de la fase de bloqueo. En este caso, se comparan las figuras (22) y (24).

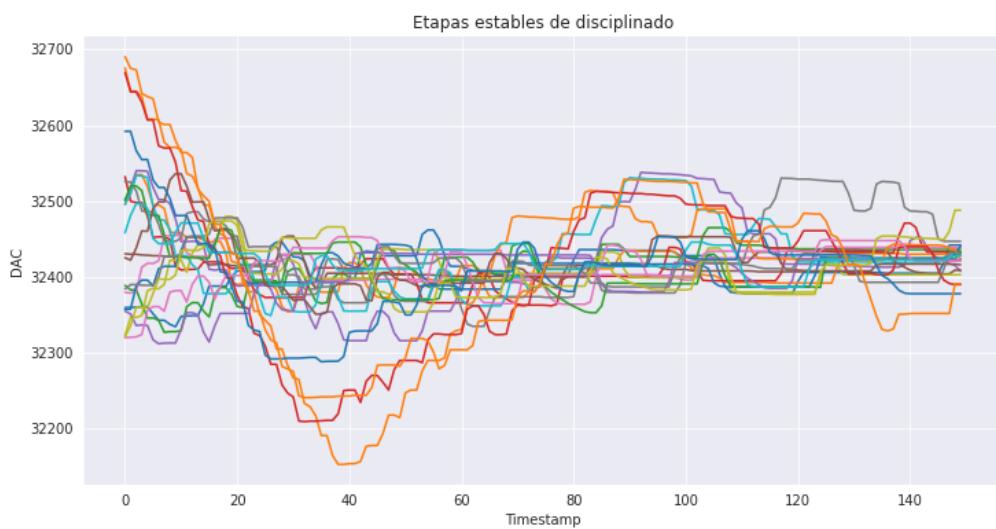


Figura 24: Valor del DAC por Etapas Estables de Disciplinado

Finalmente, hemos incluido una gráfica con los valores de todos los atributos separados por las 22 etapas de disciplinado y visualizados a través de los 150 minutos que dura cada etapa. Puede verse en la figura (25), que es análoga a

los otros 2 osciladores descartados, que se incluye en el apéndice [Otros Relojes Atómicos](#).

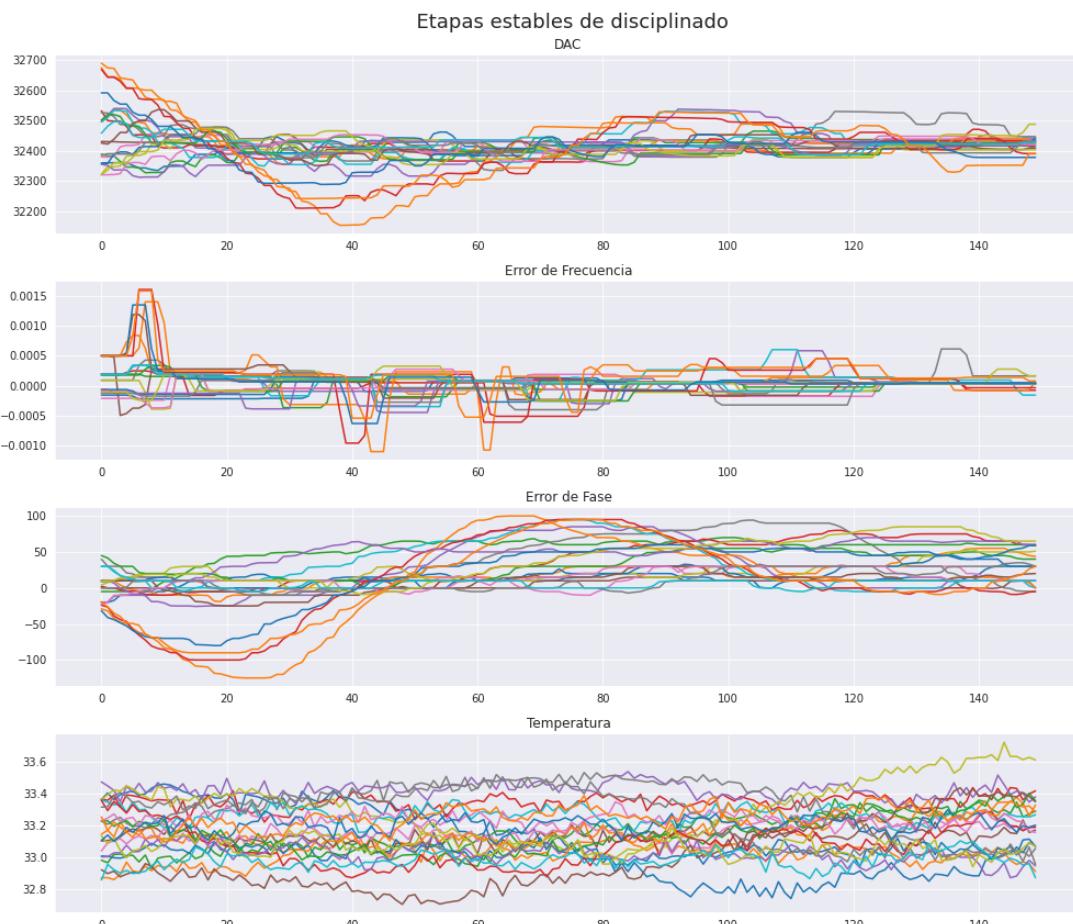


Figura 25: Valor de todos los atributos por Etapas Estables de Disciplinado

10.3 ANÁLISIS EXPLORATORIO DE LOS DATOS

Una vez que hemos logrado finalizar el preprocesado de los datos, es el momento de crear algunos gráficos para lograr una visión general de posibles patrones y características inherentes a nuestros datos. Este proceso se denomina análisis exploratorio de datos o *exploratory data analysis*, en inglés.

Una vez realizado este estudio, podremos extraer alguna conclusión preliminar que nos sirva de base para formular alguna hipótesis sobre los comportamientos observados en las gráficas. Dichas conclusiones, junto al posterior estudio de

la correlación, nos servirá como punto de parte para el desarrollo de modelos predictivos con RNN.

En primer lugar, mediante la función *pandas.DataFrame.describe()* aplicada al *dataframe* donde se encuentran dispuestos todos los datos preprocesados, obtenemos la tabla (2).

	DAC	Frequency Error	Phase Error	Temperature
count	3 300	3 300	3 300	3 300
mean	32 413.63	0.000038	21.41	33.1695
std	52.84	0.000206	34.16	0.16
min	32 153.00	-0.001105	-125.00	32.7083
25 %	32 391.88	-0.000063	0.90	33.0454
50 %	32 415.00	0.000045	15.00	33.1583
75 %	32 435.00	0.000117	45.02	33.2878
max	32 690.44	0.00610	100.00	33.7222

Cuadro 2: Características de las Variables

De esta tabla se desprende que no nos falta ningún dato en ningún instante de tiempo, además de que efectivamente tenemos 22 etapas de disciplinado, pues $\frac{3300 \text{ observaciones}}{150 \frac{\text{observaciones}}{\text{etapa}}} = 22$ etapas. Por otro lado, podemos observar que los valores de temperatura son muy estables, siendo la media un valor muy representativo, puesto que el coeficiente de variación [8] es

$$CV_{temp} = \frac{\sigma_{temp}}{\mu_{temp}} \cdot 100 = \frac{0,16}{33,1695} \cdot 100 = 0,4823 \%$$

donde para valores por debajo del 30 % se pueden considerar los datos como homogéneos. No obstante, esto último dependería de la sensibilidad de los relojes en cuanto a temperatura.

Por último, cabe destacar que el rango en el que se disponen los valores de error de frecuencia son notablemente pequeños, del orden de milésimas. Sin embargo, un pequeño error en la frecuencia haría que el error se vaya atrasando, si tiene una frecuencia mejor; o bien, adelantando, si tiene una frecuencia superior.

Una vez vista la tabla, vamos a representar estos datos en dos gráficos: el primero es la imagen (26), donde vemos todos los datos dispuestos en diagramas de frecuencia absoluta; el segundo es la imagen (27), donde estos se muestran como una serie temporal, sin hacer distinción entre etapas de disciplinado.

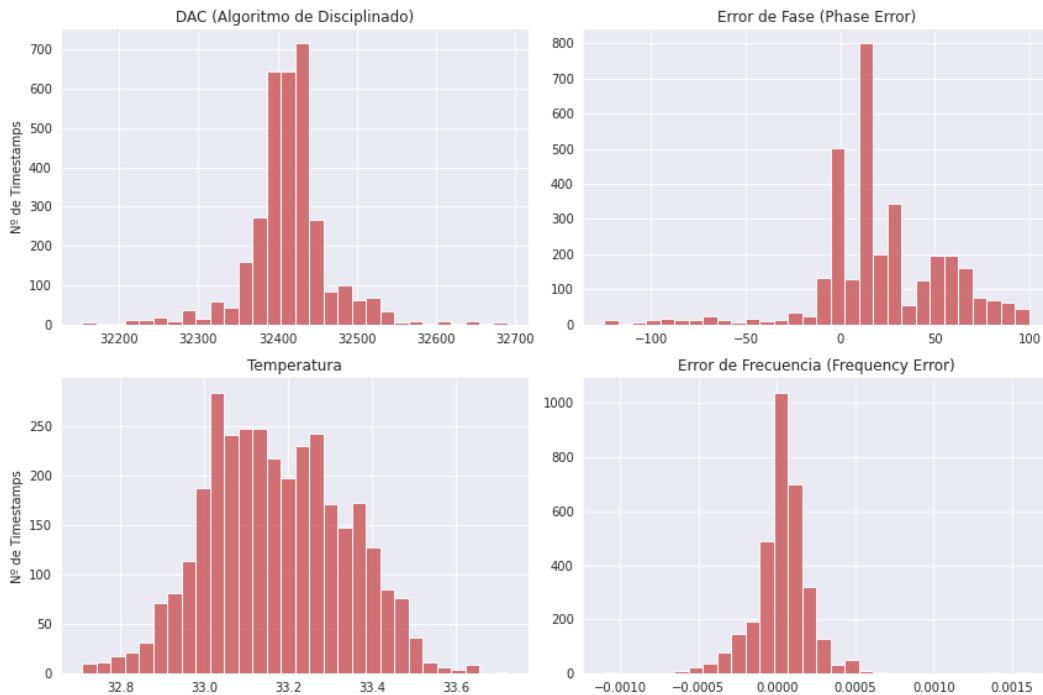


Figura 26: Distribución de los Datos por Variable

De la figura (26) podemos destacar que las 4 variables mantienen una distribución que se asemeja a una distribución normal, que será de gran ayuda de cara a mejorar las predicciones. Asimismo, se pueden observar algunos datos claramente alejados de la media tanto en el error de frecuencia como en el error de fase. Estas anomalías suelen deberse a las grandes rectificaciones que debe hacer el algoritmo de disciplinado al comienzo de una etapa. Por otro lado, como ya comentamos en el párrafo anterior, los datos de temperatura son muy homogéneos, todos ellos en el intervalo [32.6,33.8].

En el gráfico (27), además de ver la estacionalidad del valor del DAC, también podemos observar cierta estacionalidad con mismo periodo igual a 150 en el error de frecuencia y el error de fase. Se puede afirmar que de 150 en 150 observaciones, los valores de error "se reinician", puesto que cada etapa es un experimento diferente dentro del mismo oscilador, obteniendo más picos que en el resto de observaciones de cada etapa. Esto se debe a que tras el periodo de libre funcio-

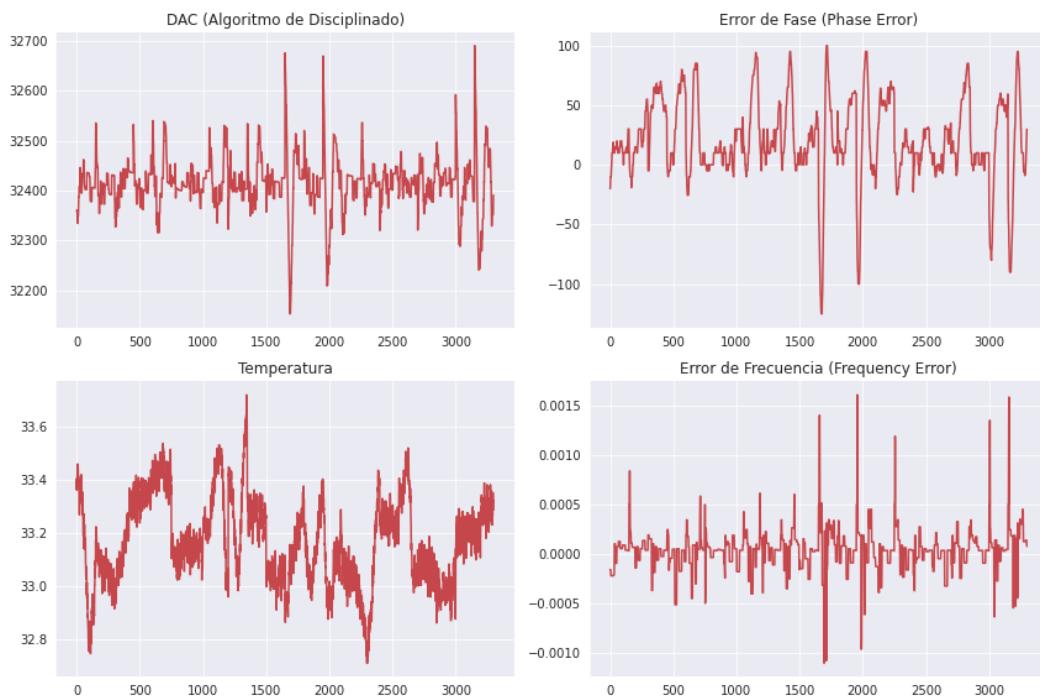


Figura 27: Distribución de los Datos por Variable medidos en el tiempo

namiento del reloj, este se ha desviado de su referencia externa, por lo que debe emplear el inicio de cada etapa en corregir adecuadamente ese desvío.

Por el contrario, la temperatura no muestra ningún patrón visible a simple vista; sino que mantiene signos de aleatoriedad en las observaciones.

Con respecto a figura (27), vale la pena destacar el índice que hemos usado para representar las observaciones. En los 4 casos, hemos empleado un índice ordinal, de manera que cada observación queda indexada por el orden en el que se encuentra; la n -ésima observación tiene índice n en el eje de abcisas. El por qué de esta decisión puede verse reflejado en la imagen (28), donde además de la anterior representación, la gráfica situada a la izquierda muestra los datos indexados por el índice original, con los *timesteps* originales. Al mostrar los datos de esta forma, existen discontinuidades (que en la gráfica se ven como una línea recta entre 2 observaciones) que se deben al tiempo que pasa el reloj en otros estados de disciplinado. Por tanto, de cara a los futuros modelos predictivos, usaremos gráficas como las de la derecha, donde empleamos un índice de manera que manejar la información sea mucho más sencillo.

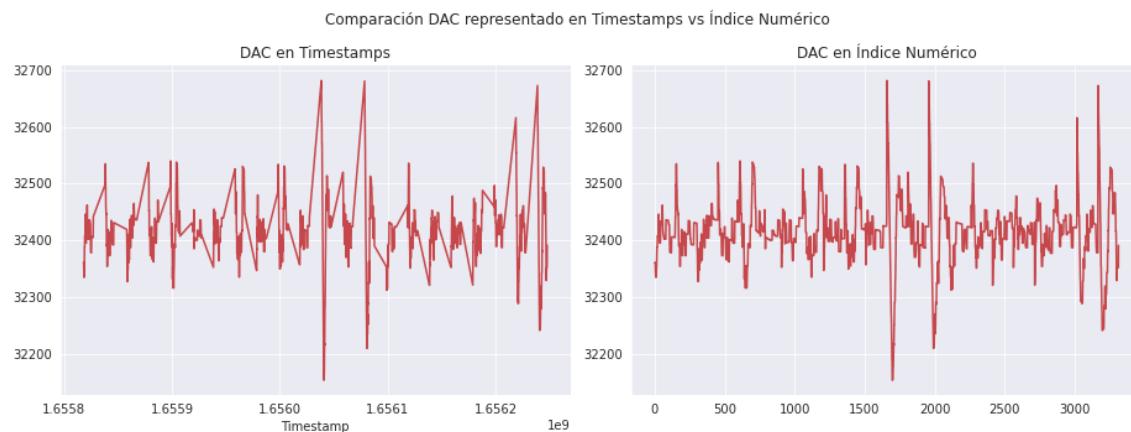


Figura 28: Comparación de los gráficos con distinto índice

Normalización de los Datos

El último paso que se debe llevar a cabo como parte del proceso de preprocesamiento de los datos es la normalización de estos. Consiste en escalar los valores linealmente mediante una transformación afín para obtener distribuciones normales centradas en 0 y con desviación estándar igual a 1, de forma que la distribución de tipo $N(\mu, \sigma^2)$ pasa a ser $N(0, 1)$ tomando $Z = \frac{X - \mu}{\sigma}$ como aplicación afín. Esto debe hacerse debido a los siguientes factores: [31]

- Evitar posibles problemas de convergencia: Los métodos de optimización de redes neuronales suelen funcionar de forma más eficiente si los valores de entrada mantienen una escala similar. Por el contrario, si introducimos valores con diferentes rangos o de diferente magnitud, el proceso de optimización puede llegar a no converger. En nuestro caso, por ejemplo, las diferencias entre el rango del valor del DAC y el rango del valor del error de frecuencia podrían ocasionar problemas.
- Mejorar la estabilidad de las predicciones: Reduce la sensibilidad de la red ante valores atípicos (*outliers*, en inglés), disminuyendo el impacto que estos pueden tener en el modelo.
- Acelerar el entrenamiento: Unos datos en una escala homogénea provoca que la función de coste converja con mayor rapidez, disminuyendo el tiempo de entrenamiento necesario.
- Facilitar la interpretación de los pesos: Al tener los datos en la misma escala, los pesos asociados a cada característica muestran directamente la importancia relativa de esta en el funcionamiento del modelo.

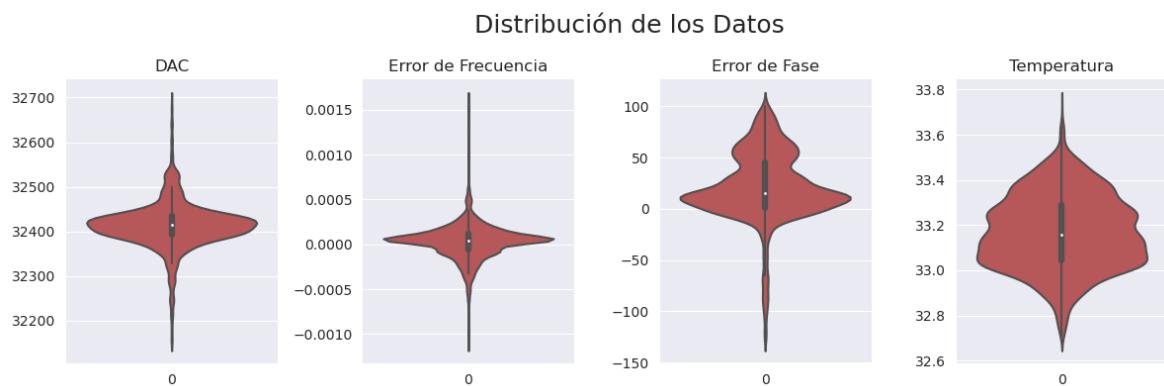


Figura 29: Distribución de los Datos antes de la Normalización

Nuestros datos originales presentan las siguientes distribuciones, mostradas en la figura (29). Como ya vimos en la tabla (2), ninguno de los atributos presenta una distribución normal $N(0, 1)$. Para normalizar los atributos, debemos restar a cada observación la media y dividir entre la desviación típica, obteniendo así el resultado que se ve en la imagen (30), donde todas son distribuciones normales centradas en 0 y con desviación típica 1.

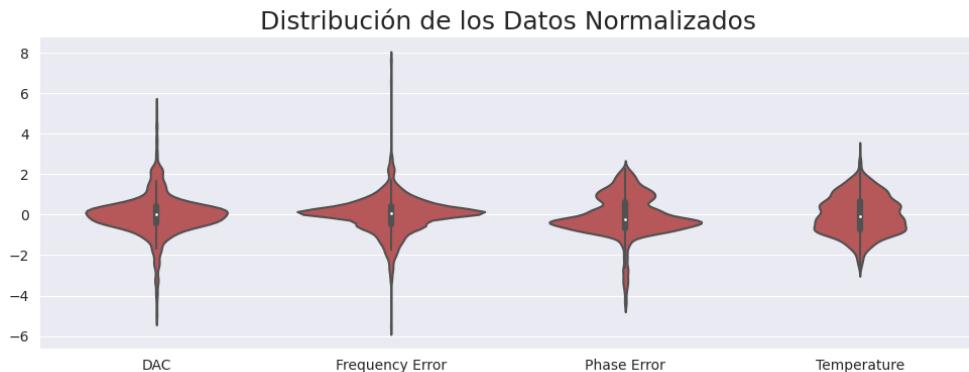


Figura 30: Distribución de los Datos tras la Normalización

ESTUDIO DE CORRELACIÓN

El trabajo con series temporales creando modelos predictivos mantiene ciertas similitudes con respecto al trabajo con otro tipo de datos de distinta naturaleza. No obstante, la mayor diferencia la provoca el factor temporal, que genera nuevas posibilidades de aprendizaje, entre ellas, la correlación (temporal y atemporal) entre los atributos que conforman una serie.

Estudiar la correlación de series temporales puede ahorrar tiempo de planteamiento de los problemas y de entrenamiento *a posteriori*, además de tener otras funciones como pueden ser:

- Identificar relaciones y patrones: A partir de la correlación, podemos identificar las variables que estén relacionadas y la forma en la que influyen entre sí. Así, se puede comprender la estructura y dinámica de los datos.
- Seleccionar características relevantes: Si identificamos las variables más correlacionadas con la variable de salida, podríamos reducir la dimensionalidad de los datos, mejorando la precisión y eficiencia del modelo.
- Interpretar los resultados del modelo: Al comprender la correlación entre las distintas variables, nos será más fácil justificar las predicciones; que puede ser crucial en campos como las finanzas o la medicina, donde el cómo juega un papel fundamental.

Al hablar de correlación, existen 2 tipos. El primero de ellos no tiene un carácter temporal, sino que es aplicable a datos de cualquier naturaleza. Sin embargo, el segundo sí que es exclusivo para series temporales. En cualquier caso, analizaremos ambos.

11.1 CORRELACIÓN ENTRE ATRIBUTOS

En esta parte, veremos cómo se relacionan los atributos si eliminamos el factor temporal; esto es, suponiendo que son simples observaciones de un fenómeno atemporal.

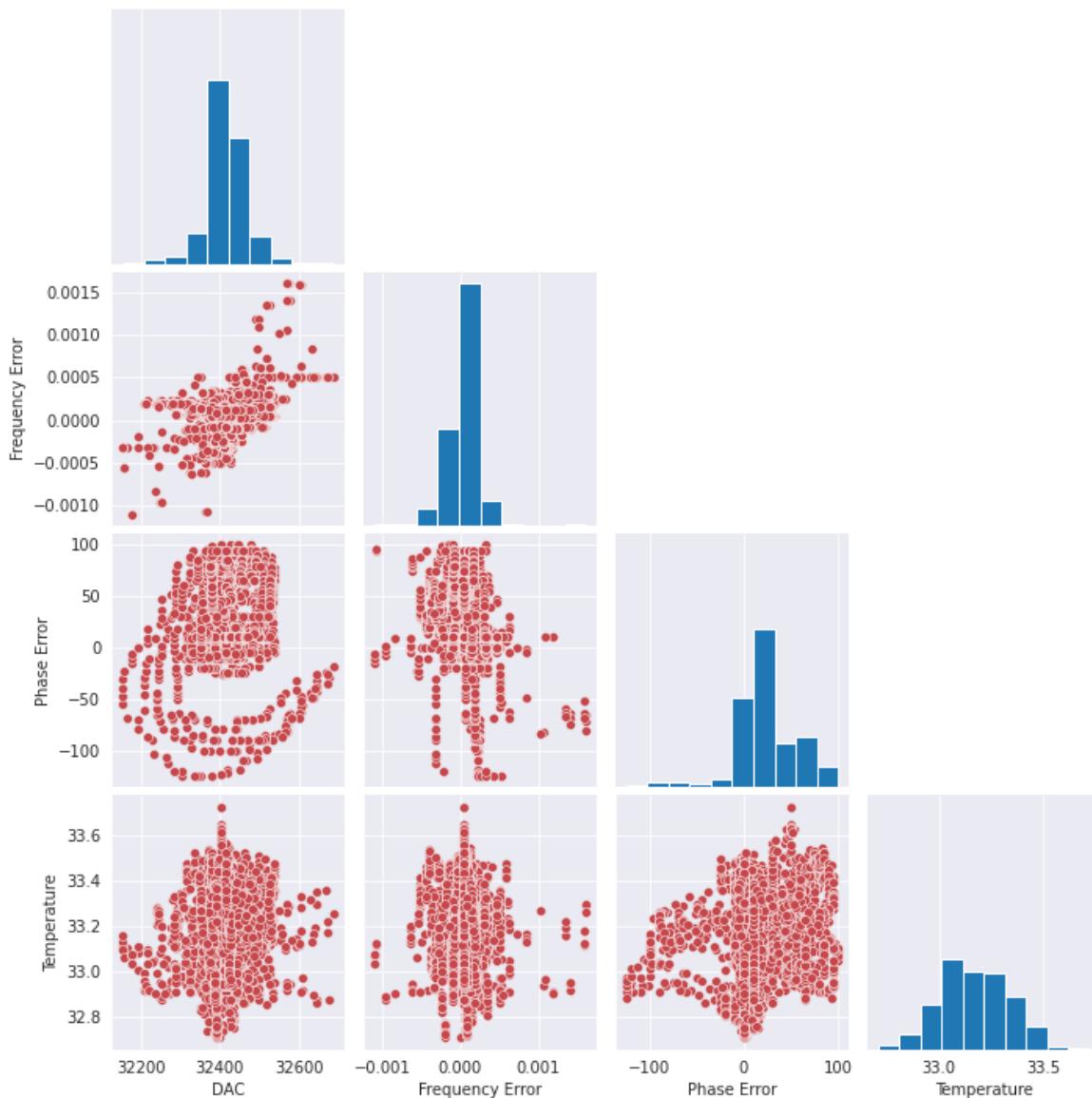


Figura 31: Distribución de los Atributos 2 a 2

En primer lugar, tenemos la figura (31), donde están dispuestas las distribuciones de las variables como datos en 2 dimensiones, de donde se desprenden algunos patrones observables a simple vista. Por ejemplo, podemos ver que existe

una correlación lineal notable entre el valor del DAC y el valor del error de frecuencia; esto es, mediante una recta del tipo $y = \alpha x + \beta$ (que se puede conseguir mediante el método de mínimos cuadrados o similar), podríamos estimar el valor de una variable, dada la otra. Esto se debe a que la función del DAC es corregir los errores de frecuencia, por lo que a más error de frecuencia, más valor de DAC. No obstante, este procedimiento no es automático. Primero se mide el error de frecuencia, para posteriormente corregirlo con un algoritmo de disciplinado.

Por otro lado, la imagen correspondiente a la nube de puntos entre el error de fase y el error de frecuencia, también refleja una ligera correlación entre ambas variables. A simple vista, se puede considerar que una recta prácticamente vertical puede ser una buena aproximación lineal. No obstante, es fácil observar que la correlación es menor que la que mantienen el DAC con el error de frecuencia.

Finalmente, también podemos observar que la correlación entre las variables DAC y temperatura es prácticamente nula. En este caso, puede parecer que la nube de puntos es totalmente aleatoria. Sin embargo, debemos seguir considerando esta variable porque los relojes físicamente sí que dependen de la temperatura. De la misma manera, en la mayor parte del resto de las imágenes sucede algo parecido, pues apenas existe ninguna correlación entre ninguna pareja de atributos.

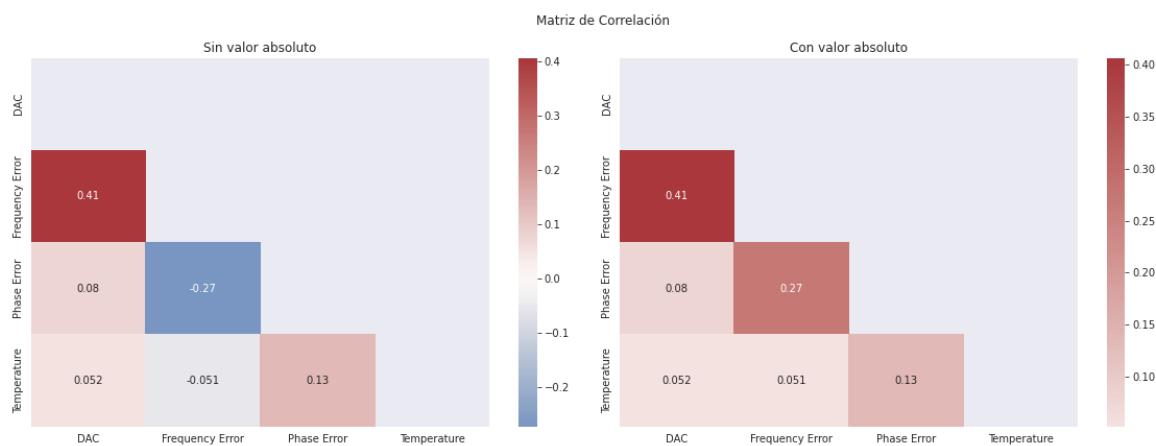


Figura 32: Matrices de Correlación de Atributos

Para cuantificar los valores de correlación anteriores se presenta el gráfico (32), donde para cada pareja de variables, se indica el valor nominal de la correlación entre ambos. En la imagen de la izquierda, también se incluyen los valores de correlación negativa; mientras que en la derecha se toma el valor absoluto con el objetivo de que el degradado de color represente de forma más clara las correlaciones más altas.

En particular, podemos destacar el valor de 0,41 como correlación entre DAC y error de frecuencia, o el valor de -0,27 entre el error de fase y de frecuencia, que indica una correlación negativa entre ambos. Normalmente, estos valores se consideran bajos, pues se dice que dos variables están altamente correladas cuando el valor absoluto supera el umbral de 0,7 – 0,8. Esto implica que necesitamos más información para poder sacar conclusiones, que es algo coherente, pues eliminar el factor temporal de nuestros datos está limitando la información de la que disponemos.

11.2 AUTOCORRELACIÓN, AUTOCORRELACIÓN PARCIAL Y CORRELACIÓN CRUZADA

Las funciones de autocorrelación (ACF, por sus siglas en inglés) y autocorrelación parcial (PACF, por sus siglas en inglés) de las variables de una serie temporal tratan de cuantificar la relación existente entre los valores de una serie a lo largo del tiempo. Al igual que las correlaciones anteriores, tiene un rango de [-1, 1] donde 1 indica correlación directa perfecta y -1 correlación inversa perfecta.

Autocorrelación

En particular, la autocorrelación [32] en series temporales se refiere a la relación entre los valores pasados y futuros de una serie temporal. Esta medida estadística detecta las dependencias y patrones sistemáticos, si los hubiera, entre las observaciones sucesivas a lo largo del tiempo de una serie.

La autocorrelación se obtiene a partir del coeficiente de autocorrelación; esto es, el usado en la figura (32), solo que comparando los valores de la serie con esos mismos valores en distintos momentos de tiempo. Así, si la función de autocorrelación en el instante t es alta, implica que existe una gran dependencia entre los valores actuales y los valores pasado un tiempo t .

Una vez introducida la función de correlación, se computa para las 4 variables de nuestro estudio en la gráfica (33), donde se muestra la función para los primeros 50 instantes de tiempo. Se ha considerado que es suficiente esta cantidad de tiempo para poder identificar los patrones existentes. Viendo las imágenes, podemos constatar que el patrón de autocorrelación del DAC y del error de fase son muy similares: existe una gran dependencia temporal en ambas variables; es decir, el valor de la variable en tiempo t depende en gran medida de las observaciones en instantes anteriores.

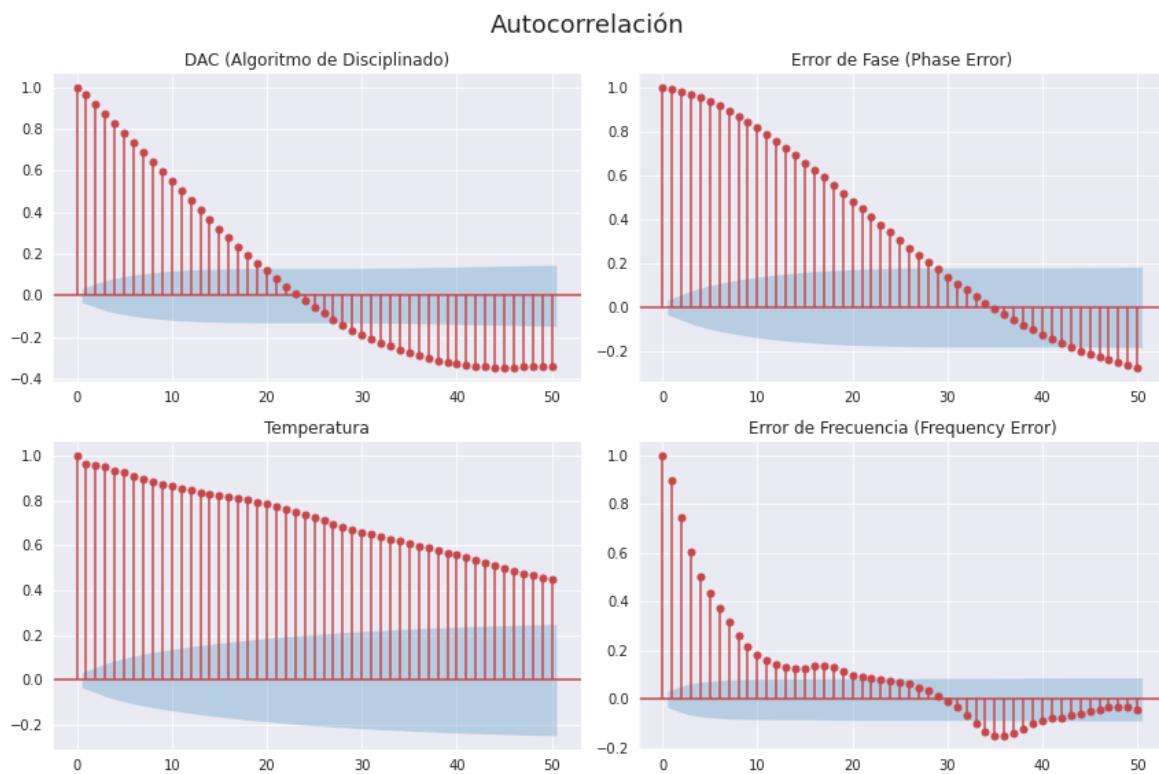


Figura 33: Función de Autocorrelación de las 4 Variables

La afirmación anterior proviene de unas gráficas con altos valores de autocorrelación en los primeros rezagos y un decaimiento gradual, hasta entrar dentro de los límites de confianza (sombreado azul).

Por otro lado, la temperatura en la figura (33) muestra un comportamiento donde la función de autocorrelación se demora en alcanzar el límite de confianza, bajo el que podemos considerar que no existe dependencia temporal. Por lo que diremos que esta variable presenta una alta dependencia temporal. Sin embargo, este comportamiento puede deberse a la baja variabilidad de los datos y a una desviación estándar muy pequeña.

Finalmente, la función de autocorrelación del error de frecuencia resulta engañosa, puesto que tiene una bajada muy temprana, hasta que se queda cerca del intervalo de confianza, donde se mantiene estable durante más de 10 instantes de tiempo. Por ello, diremos que esta variable mantiene una dependencia temporal media, sin poder obtener mayores conclusiones.

Con el objetivo de evitar el fenómeno de la autoregresión; es decir, que la mejor aproximación de un valor de una serie temporal no sea la observación inmediatamente anterior, se introducen las funciones de autocorrelación correspondientes

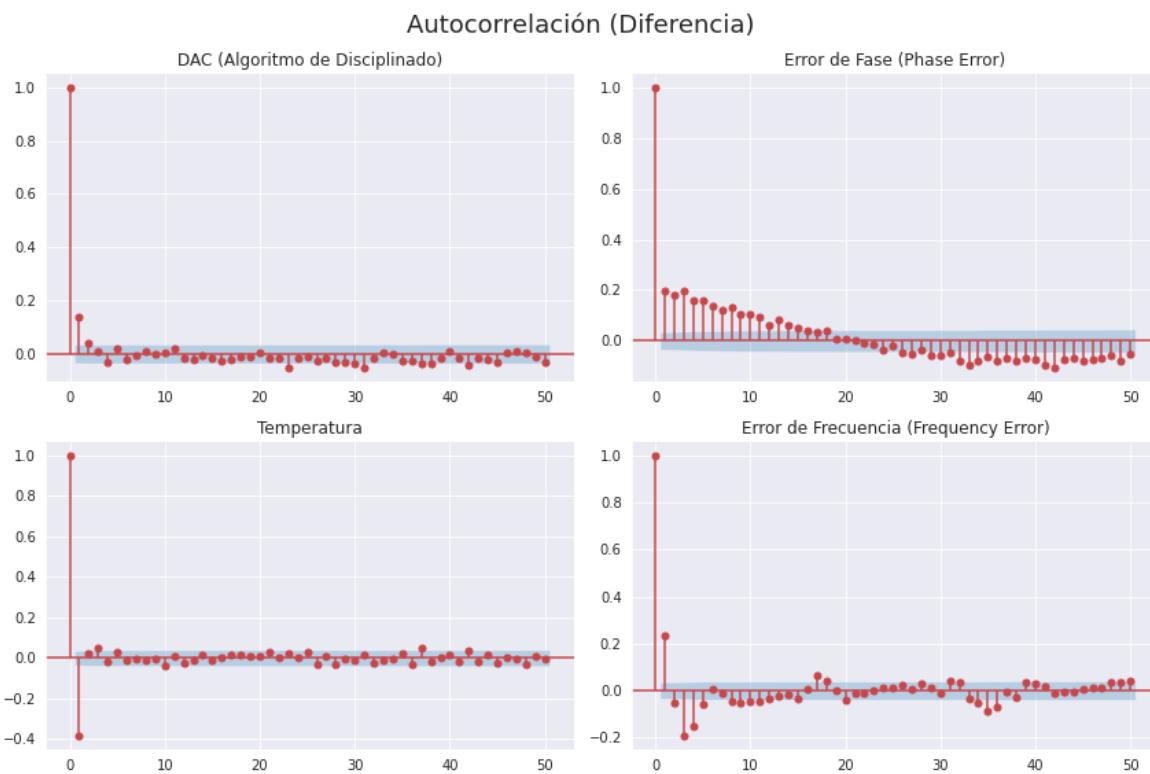


Figura 34: Función de Autocorrelación con respecto a la Diferencia de los Datos

a la diferencia entre observaciones consecutivas para las 4 variables. Se puede ver en la figura (34).

No nos vamos a detener demasiado, puesto que son las 4 gráficas muy similares. Al llevar a cabo la variación del valor nominal, se elimina la dependencia temporal, haciendo así que, por ejemplo, la función de autocorrelación correspondiente a la temperatura pierda el decaimiento habitual de una serie temporal. También podemos observar que la función del error de fase es la que mejor respeta este patrón.

Autocorrelación Parcial

Por su parte, la autocorrelación parcial [32] es una medida utilizada para examinar la relación lineal entre una observación y sus rezagos¹, teniendo en cuenta la correlación con los rezagos intermedios. Se diferencia de la autocorrelación simple en que no tiene en cuenta la influencia de las correlaciones indirectas. En otras

¹ Un rezago en una serie temporal se define como el período de tiempo que transcurre entre que una variable cambia y los efectos que provoca ese cambio sobre valores próximos de la serie.

palabras, la autocorrelación parcial muestra la correlación entre dos puntos en el tiempo después de eliminar la influencia de los rezagos intermedios.

Con la definición anterior, se puede deducir fácilmente que la autocorrelación y a la autocorrelación parcial coinciden en el primer y segundo instante de tiempo; debido a que no ha habido que eliminar el residuo provocado por observaciones anteriores.

Para computar la autocorrelación parcial, se emplea el método de la eliminación de mínimos cuadrados recursivos (RMSE, por sus siglas en inglés). En él, se realiza una regresión lineal entre la observación actual y sus rezagos correspondientes, para posteriormente eliminar gradualmente la influencia de los rezagos intermedios.

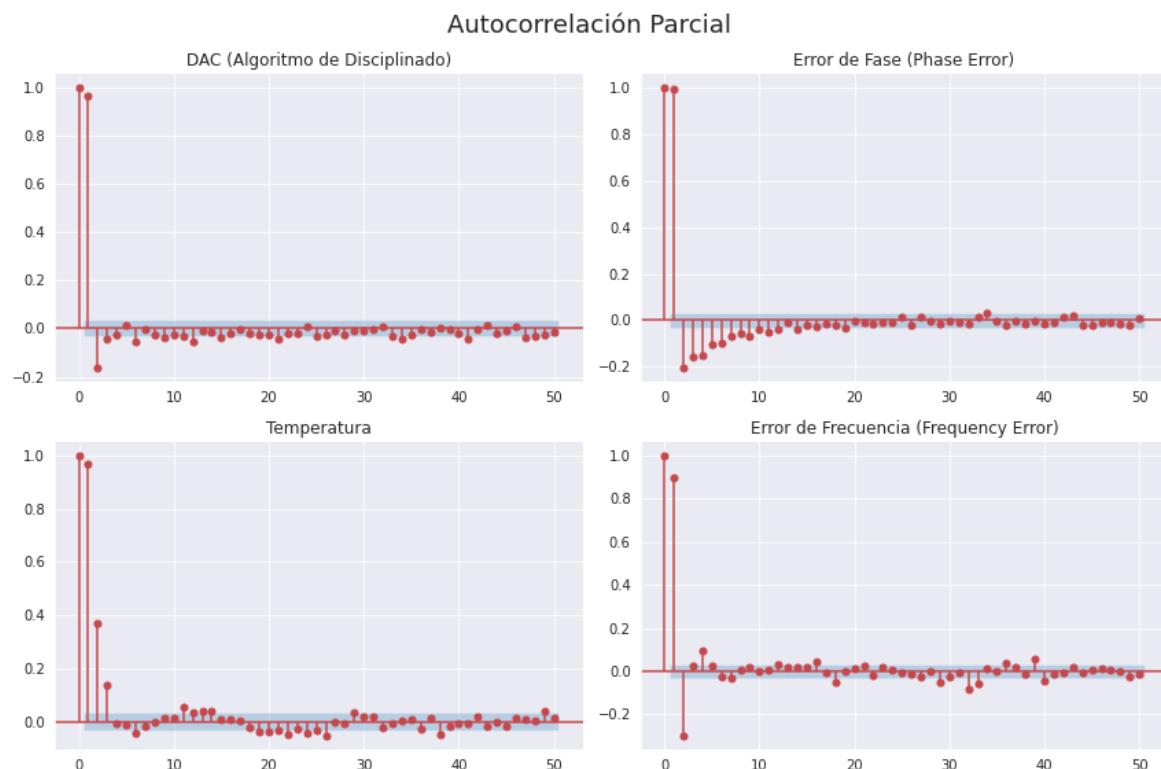


Figura 35: Función de Autocorrelación Parcial de las 4 Variables

En la figura (35), se muestran las funciones de autocorrelación parcial correspondientes a nuestras 4 variables. En este caso, como nuestros modelos predictivos estarán basados en redes neuronales recurrentes, el análisis de la correlación mediante las funciones de autocorrelación parcial es menos relevante. Esto se debe

a que la autocorrelación parcial aisla la dependencia más a largo plazo, pero este tipo de patrones son fácilmente reconocibles para una red de neuronas artificial.

En todo caso, podemos destacar la tendencia de la función de autocorrelación parcial del error de fase, donde se observa una dependencia inversa con el tiempo para ciertos valores. Por su parte, la temperatura es la variable que muestra una mayor dependencia temporal, como puede verse en (35).

Correlación Cruzada

Hasta ahora, tanto la ACF como la PACF eran funciones que operaban sobre una única variable. Para obtener mejores nociones sobre nuestro datos, se introduce el concepto de correlación cruzada [32]. Con ella, se busca comprender la relación entre dos series temporales, para identificar patrones y posibles dependencias temporales entre ellas.

En estas gráficas, el eje X representa el retraso (o *lag*) entre las series que estamos comparando y el eje Y, el valor de la correlación en cada *lag*. Debemos buscar los picos más altos (o bajos, para correlación inversa) de correlación para encontrar los retrasos que muestran mayor dependencia.

Para ejemplificar lo anterior, supongamos que tenemos una serie temporal que mide el agua que hay en el suelo y otra serie que mide la cantidad de lluvia que cae del cielo. Suponiendo que el agua de la lluvia tarde unos 30 segundos en caer al suelo, entonces estas dos series mantendrán una correlación cruzada igual a 1 en el lag 30, puesto que el valor de una está perfectamente correlacionado con el de la otra, 30 instantes de tiempo más tarde.

En nuestro caso, hemos optado por no incluir la variable temperatura entre estas gráficas, debido a su poca variabilidad y a su alta dependencia temporal ya vista en su función de autocorrelación. Por lo tanto, realizaremos la correlación cruzada entre las 3 variables restantes, 2 a 2.

En la imagen (36) presentamos 3 gráficas:

1. DAC - Error de Frecuencia: nos muestra que la mayor correlación entre estas variables se da sin retraso en el tiempo; es decir, que la mejor manera de predecir una sabiendo la otra es buscar un valor directamente proporcional al valor actual de la que conocemos.
2. DAC - Error de Fase: en este caso, la mayor correlación es de 0,68 y se da con un retraso de unos 20 instantes de tiempo. Esto nos hace ver que en nuestro



Figura 36: Correlación Cruzada entre las Variables 2 a 2

modelos, como poco, debemos incluir como valores de entrada 20 valores. De esta manera, la red podrá predecir con mayor facilidad el valor del DAC a partir del valor del error de fase este tiempo después. Todo ello, en caso de captar correctamente los patrones.

3. Error de Frecuencia - Error de Fase: Esta gráfica se comporta de forma similar que la anterior, salvo que los valores de correlación se reducen a la mitad. Así, sabemos que la correlación más alta entre ambas variables se da con un retraso de unos 26 instantes de tiempo. Sin embargo, por ser una correlación baja, de 0,32, esta es insuficiente como para poder considerarla interesante para nuestro estudio.



Figura 37: Correlación Cruzada entre la Diferencia de los Datos 2 a 2

Al igual que hicimos con la figura (34), en la (37) se presenta la correlación cruzada entre las series temporales que representan la diferencia entre dos valores consecutivos. En este caso, no podemos destacar ninguna información relevante, puesto que todos los valores de correlación a lo largo de los 200 *lags* que se representan, son bajos.

12

MODELOS PREDICTIVOS CON REDES LSTM

Una vez conocemos más información sobre los datos de los que disponemos, podemos comenzar a construir modelos predictivos con ellos. En este apartado, vamos a realizar diferentes experimentos basados en redes neuronales recurrentes con arquitectura LSTM, variando tanto el objetivo de la predicción como los parámetros asociados.

Todos estos experimentos se encuentran ubicados dentro del [repositorio del trabajo](#), en todos aquellos *notebooks* cuyo nombre comienza por *rnn*. Por ejemplo, el archivo *rnn_multistep.ipynb* comienza con unas redes LSTM para predicciones multipaso (más de un dato de salida por cada predicción); o el archivo *rnn_crossval_mov_win.ipynb*, que trata con ventanas temporales móviles optimizadas mediante diferentes técnicas de validación cruzada.

12.1 DIFERENCIACIÓN ENTRE ETAPAS

Antes de construir las RNN, tenemos que tratar la problemática de las diferentes etapas de disciplinado. Lidiar con ellas supone una dificultad añadida, en referencia a que se puede tomar cada etapa como un experimento distinto. Por ello, nuestras futuras predicciones deben ser parte de únicamente una etapa. En particular, esto se ve reflejado en que las ventanas temporales siempre se mantienen dentro de la misma etapa.

Ejemplifiquemos lo anterior, supongamos una [Ventana Temporal](#) móvil, con los siguientes atributos:

- Datos de Entrada: 30
- Offset: 1
- Datos de Salida: 1

entonces esta ventana móvil recorrerá todas las etapas una por una, y para cada etapa, tendríamos un total de ($longitud_etapa - 30$) instancias que posteriormente se dividirán entre entrenamiento, validación y testeo.

Para ello, tenemos en nuestro código una función encargada de obtener todas las ventanas posibles dentro de un conjunto de etapas, dado el atributo o *target* que queremos predecir (DAC), la longitud de las etapas (*stages_length*), el número de datos de entrada (*n_input*) y el número de datos de salida (*n_output*). La función se llama *generate_moving_windows()* y solo necesita un *dataframe* con todas las etapas concatenadas una a continuación de la otra.

```
def generate_moving_windows(df, target, stages_length, n_input, n_output):
    input_windows = []
    output_windows = []

    for stage in range(int(len(df)/stages_length)):
        for i in range(int((stages_length - n_input - n_output))):
            input_window = df[stage*stages_length+i:stage*stages_length+i+n_input]
            output_window = df[stage*stages_length+i+n_input:stage*stages_length+i+n_input+n_output][target]
            input_windows.append(input_window)
            output_windows.append(output_window)
    input_windows = np.array(input_windows)
    output_windows = np.array(output_windows)
    output_windows = np.expand_dims(output_windows, axis=1)

    return input_windows, output_windows
```

En este trabajo no se ha incluido nada de código, pues todo puede verse en el [repositorio del trabajo](#); sin embargo, esta función es altamente relevante por ser la encargada de resolver la mayor complejidad que encontramos al tratar con series temporales dispuestas por etapas de experimentos independientes.

12.2 TÉCNICAS DE ENTRENAMIENTO PARA RNN

Al tratar con redes neuronales, surgen intuitivamente cuestiones acerca de la optimización del entrenamiento de estas. En nuestro caso, vamos a hablar en este trabajo tres aspectos importantes sobre el entrenamiento: optimizadores del entrenamiento, ajuste de hiperparámetros de las redes y validación cruzada para potenciar las predicciones.

12.2.1 Técnicas de Ajuste Dinámico

Este tipo de métodos se emplean exclusivamente para potenciar y acelerar el proceso de entrenamiento de una red, mejorando los resultados y reduciendo los tiempos. Estas técnicas son relativamente recientes y por ello, su traducción al castellano no es del todo exacta, pudiendo variar incluso en inglés, dependiendo del *framework* utilizado. [1]

Entre las más utilizadas, se encuentran las siguientes:

- *Early Stopping*: Esta técnica, como su nombre indica, consiste en detener el proceso de entrenamiento antes de tiempo, a pesar de no haber finalizado todas las etapas (*epochs*), en función de ciertos parámetros o requisitos que ya se habrían cumplido. La idea de esto es evitar el sobreajuste que podría darse si la red sigue entrenándose con los mismos datos, a pesar de ya "haberlos aprendido". Se aplicará el *early stopping* cuando el rendimiento (función de coste) en el conjunto de validación, empeore con respecto a anteriores valores, a pesar de que el rendimiento en el conjunto de entrenamiento continúe mejorando. En la práctica, durante el entrenamiento se monitorea alguna de las métricas de evaluación sobre el conjunto de validación, y si esta métrica no mejora durante una cantidad predefinida de *epochs* consecutivas, entonces se detiene automáticamente el entrenamiento. En el *framework* que nosotros utilizamos, *keras*, esta función *EarlyStopping()* tiene como parámetros la métrica a medir y la "paciencia" (*patience*); es decir, el número de *epochs* sin mejora que se deben dar para detener el entrenamiento.
- *Reduce Learning Rate On Plateau*: Al igual que la técnica anterior, esta funciona a partir del monitoreo de la pérdida del conjunto de validación. En este caso, el procedimiento consiste en ir reduciendo progresivamente la tasa de aprendizaje (*learning rate*) a medida que la métrica monitoreada deja de disminuir. De nuevo, para poner esta técnica en funcionamiento, debemos predefinir un número de *epochs* tal que, si la métrica no ha mejorado pasado este número, reduciremos la tasa de aprendizaje multiplicándola por un factor (obviamente menor que 1) que también vendrá dado como parámetro. Reducir la tasa de aprendizaje permite que el algoritmo de descenso del gradiente pueda adaptarse progresivamente al aspecto del gradiente, superando mínimos locales y "cayendo" de manera adecuada en los mínimos globales buscados.
- *Data Augmentation* o Ampliación de Datos: Técnica utilizada para aumentar la cantidad de datos de entrenamiento. Para ello, se aplican modificaciones

pseudoaleatorias a las instancias de las que disponemos para obtener nuevos datos con los que entrenar nuestros modelos. Esto puede ayudar a que la red generalice de mejor manera datos que no ha visto antes. Sin embargo, dependiendo de la naturaleza de los datos, esta técnica puede carecer de sentido. En nuestro caso, como los datos provienen de relojes atómicos con una configuración muy específica, puede resultar muy complejo modificarlos para encontrar nuevas series temporales que pudiesen haber salido de estos relojes. Por tanto, hemos optado por no usarla.

- *Gradient Clipping* o Recorte del Gradiente: El recorte del gradiente es una técnica empleada con el objetivo de limitar el rango de los gradientes obtenidos durante el proceso de entrenamiento de la red. Esta técnica trata de evitar que los gradientes que se usan en el [Algoritmo de Descenso del Gradiente](#) se vuelvan demasiado grandes, pudiendo así desestabilizar el proceso de optimización, llegando incluso a no converger a la solución óptima. De esta forma, nos aseguramos que las variaciones en los pesos se realizan gradualmente, buscando dar una mayor estabilidad en el proceso. [22]

En nuestro trabajo en particular, como es difícil que se de sobreajuste (debido a la escasez de datos) y vamos a utilizar tanto *early stopping* como *reduce learning rate on plateau*, hemos optado por no llevar a cabo más técnicas de ajuste dinámico.

12.2.2 Ajuste de Hiperparámetros

En cualquier red neuronal, el ajuste de hiperparámetros (o *hyperparameter tuning*, en inglés) es una tarea que no debe pasarse por alto en el entrenamiento de la red, pues gran parte del éxito o fracaso de un modelo predictivo depende de tener una buena configuración de hiperparámetros.

Los hiperparámetros son configuraciones de atributos que una red no puede aprender durante el entrenamiento, sino que se indican a la hora de comenzar el aprendizaje. Estas configuraciones afectan tanto al rendimiento y como a la capacidad de generalización de la red.

Algunos de los hiperparámetros más conocidos que se pueden ajustar en una red recurrente LSTM son los siguientes. [1]

- i. **Número de unidades LSTM:** Indica la cantidad de bloques LSTM de los que dispone la red neuronal. Este hiperparámetro puede ajustarse puesto que un mayor número de unidades podría aumentar la capacidad de aprendizaje de la red en cuanto a patrones complejos en la serie temporal. Sin embargo, un

número alto de bloques aumentaría los requerimientos computacionales, así como el riesgo de sobreajuste. Es por esto que se debe buscar un equilibrio adecuado.

- II. **Número de capas LSTM:** Se refiere al número de capas ocultas de bloques LSTM de la red. Al igual que con el número de bloques LSTM, añadir más capas ocultas puede ayudar a modelar relaciones más complejas; no obstante, esto podría aumentar el tiempo de entrenamiento de la red, así como la complejidad del modelo.
- III. **Número de datos de entrada:** Especifica el número de pasos de tiempo (*timesteps*) que se van a emplear para la predicción de los siguientes valores de la serie. En esta caso, si el número de datos de entrada es alto, el modelo podrá capturar la dependencia temporal con mayor seguridad; sin embargo, también requiere más memoria al estar trabajando con más datos, además de aumentar la complejidad del cómputo.
- IV. **Tamaño del lote (batch size):** Se refiere a la cantidad de instancias del conjunto de datos de entrenamiento que se emplean cada vez que actualizamos los valores de los pesos de la red. Al igual que los anteriores, un número alto de batch puede acelerar el proceso de entrenamiento, pero también podría hacer que el algoritmo de descenso del gradiente no converja a la solución óptima de la función de coste. Además, cuantos más datos necesitemos para actualizar los pesos, más memoria se requerirá para llevar a cabo el cómputo.
- V. **Tasa de aprendizaje (learning rate):** Indica el valor de la tasa de aprendizaje empleada en el [Algoritmo de Descenso del Gradiente](#); es decir, controla el tamaño de las actualizaciones de los pesos en la etapa de entrenamiento de la red. Por un lado, un valor demasiado alto de *learning rate* puede acelerar el proceso; sin embargo podría hacer que la red converja de manera inestable, o incluso no converja. Por otro lado, una tasa de aprendizaje baja puede provocar que una convergencia más lenta, o en el peor de los casos, que el algoritmo se quede estancado en mínimos locales y converja.

Además de estos hiperparámetros considerados los más básicos. Existen otras formas de mejorar el rendimiento de una red, solo que están más orientados a evitar el sobreajuste¹ de una red a los valores de entrenamiento. En nuestro caso, disponemos de un conjunto de datos relativamente pequeño, donde tras obtener las

¹ Sobreajuste es un comportamiento de aprendizaje automático no deseado que se produce cuando el modelo de aprendizaje automático proporciona predicciones precisas para los datos de entrenamiento, pero no para los datos nuevos o hacer un modelo tan ajustado a los datos de entrenamiento que haga que no generalice bien a los datos de test.

ventanas temporales correspondientes, disponemos de entre 20 y 2000 instancias para el entrenamiento. Por ello, ninguno de nuestros modelos llega a sobreajustarse y no tenemos que emplear este tipo de recursos. En todo caso, los describimos a continuación. [1].

- **Regularización:** Existen varios tipos de regularización en redes neuronales y los más usados son la regularización L1 y la regularización L2. Es uno de los principales mecanismos a la hora de prevenir el sobreajuste de una red. La idea de la regularización es controlar la penalización que se aplica a los pesos de la red durante el proceso de aprendizaje. De esta forma, se trata de buscar pesos con valores pequeños, que no puedan ajustarse por demás a los datos de entrenamiento.
- **Dropout:** Esta técnica que consiste en eliminar de la red (desactivar o apagar) aleatoriamente algunas unidades LSTM durante el entrenamiento. Esto provoca que el aprendizaje mantenga una componente estocástica que pueda ayudar a evitar el sobreajuste. La idea es que aunque se entrene el modelo con los mismos datos, algunas veces se procesan con unas neuronas y otras veces con otras, haciendo que el resultado no sea el mismo. Para llevarlo a cabo, el hiperparámetro *dropout* representa la probabilidad (un número entre 0 y 1) de que dicha neurona se desactive durante el procesamiento de un lote. Habitualmente, el valor de *dropout* se mantiene dentro del intervalo [0.2,0.4].

Para poder llevar toda la información sobre los parámetros utilizados, existen diversas herramientas. La más conocida a día de hoy es [wandb.ai](#), un software diseñado para el testeo de diferentes configuraciones de hiperparámetros dentro de una red, de forma que es muy sencillo compararlos mediante las métricas que se quieran medir.

En nuestro caso, por falta de tiempo y por tener muy pocos hiperparámetros que se mueven en un pequeño rango, hemos decidido evaluar y almacenar los resultados de diversas configuraciones por nuestra cuenta. Para poder agilizar el proceso, hemos creado una función que dada una lista con los valores de los hiperparámetros usados (número de capas ocultas, número de unidades por capa, tamaño de lote y tasa de aprendizaje), recoge las métricas asociadas a cada configuración dentro de todas las combinaciones posibles.

Esta función se denomina *tuning()* y tiene la siguiente estructura descrita en (4), donde únicamente hemos incluido el pseudocódigo, puesto que el código en python ocupa bastante más espacio:

Por ejemplo, para las siguientes listas de hiperparámetros:

Algorithm 4 Función para el Ajuste de Hiperparámetros

Require: dataframe, n_input, n_output, deep_layers[], lstm_units[], batch_sizes[], learning_rates[]

```

1: for [dl,u,bs,lr] in [deep_layers,lstm_units, batch_sizes, learning_rates] do
2:   train, val, test ← generar_ventanas_temporales(dataframe)
3:   modelo ← train_model(train, val, n_output, dl, u, bs, lr)
4:   predicciones ← modelo.predict(test)
5:   metricas ← obtener_metricas(predicciones,test)
6:   csv ← [n_input, n_output, dl,u,bs,lr, metricas]
7: end for
8: return csv

```

- Capas Ocultas: [1,2]
- Unidades LSTM por capa: [40,60,80,100,120]
- Tamaño del Lote: [2,4,8]
- Tasa de Aprendizaje: [0.05, 0.01, 0.005, 0.001]

obtendremos un total de $2 \times 5 \times 3 \times 4 = 120$ configuraciones distintas; es decir, 120 modelos diferentes con sus respectivas métricas sobre el conjunto de testeo. Para almacenar esta información, hemos creado diversos archivos *csv* donde cada línea se corresponde a una configuración distinta, que almacena las siguientes variables: error absoluto medio, raíz cuadrado del error cuadrático medio, número de datos de entrada, número de datos de salida, número de capas ocultas, número de unidades LSTM por capa, tamaño de lote, tasa de aprendizaje, número de épocas de entrenamiento.

Hemos creado un *csv* distinto para cada combinación de número de datos de entrada y número de datos de salida. Por lo que dentro de un *csv*, estos 2 valores siempre serán los mismos. Una vez tenemos estos *csv*, ya podemos afirmar que hemos encontrado una configuración de hiperparámetros con altas probabilidades de ser óptima para nuestro problema. Con esta configuración, que debería ser la mejor en alguna métrica, o mantenerse entre las mejores en varias métricas, ya podemos entrenar un nuevo modelo con relativa seguridad de poder obtener un buen rendimiento.

12.2.3 Validación Cruzada *leave one/two out*

La validación cruzada (*cross validation*) en K -iteraciones [2] ya se presentó en la sección 8.2, ahora vamos a llevarla a la práctica como un método para optimizar las predicciones de nuestros modelos.

En primer lugar, mediante las [Técnicas de Ajuste Dinámico](#) y el [Ajuste de Hipérparámetros](#), debemos encontrar un modelo adecuado de entre todas las combinaciones que podamos llegar a probar. Una vez encontrado una configuración adecuada, aplicamos validación cruzada. En nuestro caso, debido a la escasez de datos (solo disponemos de 22 experimentos de nuestro reloj atómico), vamos a llevar a cabo el algoritmo de *leave two out*.

La idea entonces es ir dividiendo nuestro conjunto de datos, de forma que en cada iteración de nuestro algoritmo, tengamos 20 experimentos para entrenamiento, dejando fuera del entrenamiento 2 instancias (de ahí el nombre de *leave two out*): 1 para validar y 1 como testeo. Así, podemos obtener, en cada iteración, tanto las predicciones del modelo, como las métricas asociadas a la instancia de testeo.

Una vez acabado el proceso, habremos conseguido un total de 22 predicciones con sus métricas. Con ellas, podremos finalmente graficar la media de todas ellas sobre la media de las instancias de entrenamiento, consiguiendo así una mejora en los resultados, pues habremos evitado el **sesgo** que provocaría una separación arbitraria entre conjuntos de entrenamiento, validación y testeo.

El proceso anterior es el algoritmo (5), que se presenta en el código como la función `cross_validation_training()` y que está presente en el [repositorio del trabajo](#) en todos aquellos *notebooks* que incluyen el término `cross_val` en el nombre del archivo.

12.3 PREDICCIÓN DEL DISCIPLINADO DE UNA ETAPA COMPLETA

Una vez conocemos el proceso con el que hemos optimizado al máximo los resultados de las predicciones, vamos a presentar las predicciones propiamente dichas. Para ello, vamos a darle 2 enfoques distintos, de forma que cada uno de ellos sirve para dar solución a dos problemas totalmente independientes.

Algorithm 5 Entrenamiento con Validación Cruzada

Require: dataframe, n_input, n_output, layers, lstm_units, batch, lr

```

1: for iteration in [0,...,n_etapas-1] do
2:   train, val, test ← cross_val_sets(dataframe, iteracion)
3:   train, val, test ← normalizar(train, val, test)
4:   train_win, val_win, test_win ← generar_ventanas_temporales(train, val,
   test)
5:   modelo ← train_model(train_win, val_win, n_output, layers, lstm_units,
   batch, lr)
6:   predicciones[] ← modelo.predict(test_win)
7:   metricas[] ← obtener_metricas(predicciones, test)
8: end for
9: mean_predictions ← predictions.mean(axis=0)
10: mean_metrics ← metrics.mean(axis=0)
11: return mean_predictions, mean_metrics
  
```

12.3.1 Explicación del Problema

El primer problema consiste en, dadas las primeras n observaciones, predecir las $longitud_etapa - n$ observaciones restantes antes de finalizar la etapa de disciplinado. De esta manera, se busca encontrar el comportamiento que debería seguir un oscilador para finalizar el disciplinado correctamente.

Este problema tiene una aplicación práctica. Mediante el disciplinado, se debe estabilizar el error de fase y de frecuencia a lo largo de una etapa, por lo que el final de la misma, estos valores deben cumplir un patrón que consideremos adecuado en función de las características de la etapa. Por ejemplo, si al inicio de la etapa había grandes irregularidades en el oscilador, es normal que "le cueste estabilizarse" y seguirá un patrón más irregular; por otro lado, si el inicio muestra un comportamiento estacionario, el final debe seguir siéndolo. Esto se refleja en la imagen (24) que presentamos anteriormente, donde algunas etapas tienen un inicio con valores inestables de disciplinado, tratando de corregir el error de fase; mientras que otras tienen un comportamiento más regular durante toda la etapa.

En el caso de que en alguno de los experimentos futuros, el reloj no cumpla los patrones que indicará nuestro modelo, habremos detectado un problema en el disciplinado, que es el objetivo final de este tipo de predicción.

Para llevar a la práctica esto, se debe tomar una decisión antes de empezar, que es definir exactamente qué se entiende por el principio de una etapa de disciplinado. Todas las etapas disponen un total de 150 observaciones y tenemos que mantener cierto equilibrio. Por un lado, no tendría sentido práctico tener 149 datos

de entrada para predecir una salida; sin embargo, por otro lado, ningún modelo sería capaz de, dadas las primeras 50 observaciones, ofrecer un buen rendimiento prediciendo las 100 siguientes. Finalmente, se ha optado por tomar las primeras 100 para predecir las 50 restantes, manteniendo así un equilibrio de 2/3 de datos de entrada por 1/3 de salida.

12.3.2 Desarrollo de los Modelos para Etapas Completas

Con estas indicaciones, se ha llevado a cabo la búsqueda de una óptima configuración de hiperparámetros probando un total de 1412 combinaciones distintas, donde hemos obtenido que las 5 mejores configuraciones son las que se disponen en la tabla (3).

Deep Layers	LSTM Units	Learning Rate	Batch Size	MAE	RMSE
1	120	0.008	1	0.201229	0.240943
1	60	0.010	1	0.204994	0.269339
1	50	0.0600	2	0.205801	0.248949
1	80	0.040	4	0.206696	0.246657
1	80	0.006	1	0.211499	0.265740

Cuadro 3: Búsqueda de Hiperparámetros para la Predicción de una Etapa Completa

Cabe destacar que los 5 modelos de la tabla (3) están ordenados por error absoluto medio (MAE) y lo ideal sería encontrar una configuración que mantuviese el nivel de precisión tanto para MAE como para RMSE. Por ello, hemos escogido tanto la primera combinación de hiperparámetros, como la tercera, para graficar las predicciones sobre el conjunto de testeo². Los resultados del primer modelo se presentan en la imagen (38), mientras que los del tercero corresponden a la imagen (39).

Antes de pasar a la descripción de las gráficas de predicción, se debe explicar la leyenda de todas ellas. En cada imagen de las que se presentan a continuación, en el eje de abcisas se presentan los instantes de tiempo, donde cada observación

² Como estamos en el proceso de búsqueda de hiperparámetros, aun no se ha llevado a cabo la validación cruzada. Esto puede provocar que los resultados estén sesgados dependiendo de la división entre entrenamiento, validación y testeo.

se corresponde con un minuto. De esta forma, todas las gráficas deben disponer de 150 observaciones, puesto que esa es la longitud de una etapa (experimento) del algoritmo de disciplinado. Por otro lado, el eje de ordenadas representa el valor nominal normalizado del DAC. Para llevar a cabo esta normalización, se toma la media y desviación estándar del conjunto de entrenamiento correspondiente, de forma que la distribución de los datos de entrenamiento siga una distribución normal $N(0, 1)$, como ya se explicó anteriormente.



Figura 38: Predicciones de 1 etapa completa dados 100 datos de entrada (Modelo 1)

El primer modelo obtenido, el de la figura (39) ha sido capaz de aprender los patrones finales de las etapas, siguiendo de una manera relativamente adecuada el valor del DAC y sus variaciones. El final de la segunda instancia de testeо muestra que el modelo tiene alguna dificultad para obtener con precisión los grandes picos de diferencia entre valores consecutivos (entre la observación 147 y 148); sin embargo, en general tenemos un resultado decente, donde las métricas sobre estas 2 instancias de testeо son: 0.201229 de MAE y 0.240943 de RSME.

En cuanto al segundo modelo, del que presentamos sus predicciones en la figura (39), sacamos 2 conclusiones distantes de las 2 instancias de testeо. De la primera, vemos que es capaz de predecir la tendencia, quedando cerca de los valores exactos del DAC. De la segunda, vemos que ha aprendido los giros que se van a producir, al igual que los picos; pero tiene cierta dificultad en situarnos a lo largo del tiempo. De todas formas, sigue siendo un resultado válido para no



Figura 39: Predicciones de 1 etapa completa dados 100 datos de entrada (Modelo 2)

haberse realizado con validación cruzada. Las métricas son: 0.205801 de MAE y 0.248949 de RMSE.

Validación Cruzada Leave two out

Una vez que se han presentado estos 2 modelos para visualizar la bondad de las predicciones, es el momento de realizar validación cruzada (en particular, vamos a recurrir al método de *leave two out*, para obtener una gráfica que mostrar al final) para optimizar al máximo estos resultados.

Para poder representar gráficamente los resultados del *leave two out*, en cada iteración del algoritmo, una de las etapas que "dejamos fuera" se usará para testeо. Posteriormente, se computará la media de los resultados de estas predicciones y se comparará con la media de las 22 etapas. De esta forma, obtendremos un gráfico donde quedan representados: la media de las 22 etapas (como variable a predecir), la media de las predicciones de los 22 modelos (cada uno sobre 1 instancia de testeо diferente), la desviación estándar de las 22 predicciones. Además, podremos obtener la media de las métricas sobre las 22 predicciones, de cara a poder compararlas después.

Es importante saber que no disponemos de ninguna medida de referencia para conocer la bondad de las predicciones de los modelos con validación cruzada *leave two out*. Las predicciones que se reflejan en (38) y (39) pueden estar sesgadas por la elección de etapas para testeо, por lo que puede suceder tanto que estas

etapas sean "las más normales" o "las más extrañas", en comparación con el resto. En el primer caso, las métricas de estas predicciones resultarían inalcanzables con validación cruzada; mientras que en el segundo caso, serían fácilmente mejorables al eliminar el sesgo mencionado.

Para computar los diferentes modelos de validación cruzada, se emplea el algoritmo (5), donde las ventanas temporales son "fijas" y representan una etapa completa. Por ello, en cada una de las 22 iteraciones del algoritmo, tendremos 20 etapas de entrenamiento, 1 de validación y 1 de testeo. Además, todos los modelos mantienen la configuración de hiperparámetros del mejor modelo que se obtuvo previamente. Los resultados de la media de las predicciones con *leave two out* se presentan en (40).

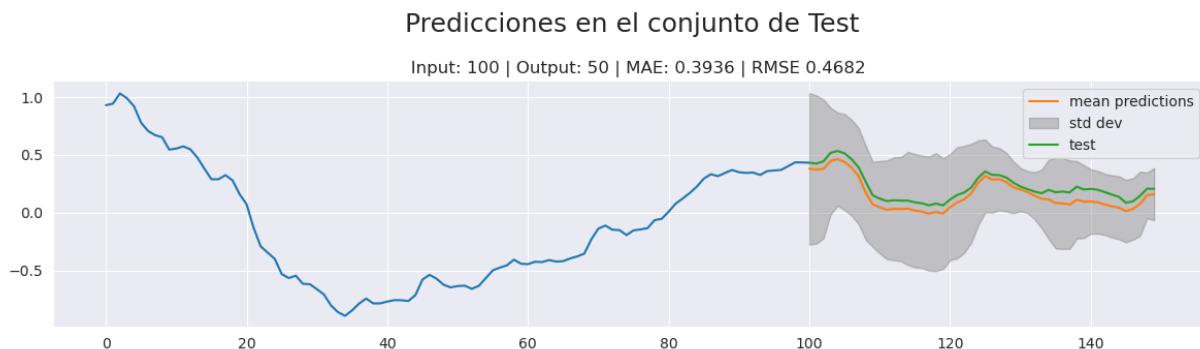


Figura 40: Validación Cruzada *leave two out* para Predicciones de 1 etapa completa con 100 datos de entrada

Las media de las métricas obtenidas con validación cruzada, como se puede ver en la imagen (40) son: 0.3936 de MAE y 0.4682 de RMSE. Por lo que podemos concluir que las etapas que se escogieron en las predicciones anteriores fueron de las mejores que se podían tomar. Además, en la gráfica también se ha incluido un sombreado que representa la desviación estándar de todas las predicciones realizadas; esto es, los límites del sombreado quedan definidos como la media más la desviación estándar en cada instante de tiempo.

En todo caso, teniendo en cuenta la escasez de datos (estos modelos están preparados para entrenarse con miles de datos y solamente disponemos de 20), podemos concluir que los resultados son mejores de lo que se podía esperar. Si se dispusiera de un conjunto más amplio de datos y de más tiempo de entrenamiento, probablemente se conseguirían unos resultados mejores, con unos modelos que pudieran generalizar perfectamente el comportamiento de los osciladores al final de cada etapa.

12.4 PREDICCIONES CON VENTANA MÓVIL POR ETAPAS

Una vez finalizadas las predicciones referentes al primer problema, podemos introducir el segundo de los problemas que pueden resolver nuestros modelos predictivos.

12.4.1 *Explicación del Problema de Predicción de Errores*

El segundo problema consiste en, dado un número n_i de datos de entrada del oscilador (incluyendo las 4 variables que empleamos en nuestro estudio), poder predecir un número n_o de valores de disciplinado posteriores; es decir, predecir futuros valores de DAC, siempre dentro de la misma etapa.

La naturaleza de este problema nos permite disponer, dependiendo de las dimensiones de las ventanas, de un mayor número de datos de entrenamiento que el problema anterior. Por ello, a priori, los modelos asociados a esta parte tendrán un mejor rendimiento.

Al igual que el anterior problema, este también tiene una utilidad práctica: la predicción de errores. Supongamos que tenemos un modelo que, dadas 20 observaciones, predice las 5 siguientes. En tal caso, si las predicciones difieren significativamente de los valores reales de disciplinado del oscilador, esto puede ser un indicio de que se está produciendo alguna irregularidad en el reloj. De alguna manera, estos modelos podrían usarse como alarma que puede saltar cuando algo no funcione como debería.

Para llevar a cabo la resolución de este problema, de igual forma que en el anterior optamos por mantener fijos tanto el número de datos de entrada como de salida, en este caso los variaremos. Construiremos diferentes modelos con un número distinto tanto de datos de entrada como de datos de salida, manteniendo algunas premisas:

- Como se estudió en el apartado de [Autocorrelación, Autocorrelación Parcial y Correlación Cruzada](#), debido a la correlación cruzada entre variables, siempre se deben tener en cuenta 20 datos de entrada como mínimo. Esto es necesario para que los modelos puedan aprender los patrones que presentan los datos.
- Se debe mantener un equilibrio entre el número de datos de entrada y de salida, de forma que estos nunca serán mayores que los primeros. Igualmente,

es preferible (y los modelos mostrarán un mejor rendimiento) que se disponga de más datos de entrada que de salida.

12.4.2 Desarrollo de los Modelos con Ventana Temporal Móvil

Teniendo en cuenta todo lo anterior, hemos decidido probar las siguientes configuraciones de datos de entrada y datos de salida: $(20 - 1)$, $(20 - 5)$, $(20 - 10)$, $(30 - 15)$, $(30 - 20)$, $(30 - 30)$. La idea es ir probando y analizando de qué manera se van degradando las predicciones a medida que aumentamos el tamaño de la ventana de salida. La lógica indica que un modelo con 30 datos de entrada y de salida tendrá un rendimiento considerablemente peor que el que predice un único dato.

Modelo con 20 Datos de Entrada y 1 de Salida

El primer modelo que vamos a realizar es el de $(20 - 1)$, para ello, debemos realizar la búsqueda de los hiperparámetros más adecuados. Como vamos a realizar un total de 6 modelos distintos, únicamente incluiremos la tabla con las mejores configuraciones de hiperparámetros referentes a los 2 primeros tipos; es decir, $(20 - 1)$ y $(20 - 5)$. En todo caso, el procedimiento será el mismo con todos: definir entre 50 y 200 configuraciones distintas³, tomar la que mejor se comporta en la mayoría de las métricas, y a partir de esa configuración, ir entrenando modelos hasta lograr un rendimiento aceptable.

Las combinaciones de hiperparámetros que mejor han funcionado para modelos con 20 datos de entrada y 1 de salida se muestran en la tabla (4). Tomando la mejor configuración y entrenando convenientemente una red recurrente LSTM, se consiguen los resultados que se muestran en (42), donde se observa que el modelo es capaz de entender las tendencias, aunque muestra algunos picos en las predicciones que no concuerdan con los valores reales del DAC. En todo caso, un valor de MAE de 0.058 indica un buen resultado.

Ahora, de cara a entender cómo se está llevando a cabo el proceso de entrenamiento de una red neuronal, se incluye en la imagen (41), las funciones de coste sobre el conjunto de entrenamiento y validación de uno de estos modelos.

³ En este problema probaremos un menor número de configuraciones de hiperparámetros para cada modelo, debido a que estas redes tardan mucho más en entrenarse. Esto se debe a que por ser ventanas móviles, en lugar de tener una instancia por etapa, obtenemos un total de $longitud_etapa - n_input - n_output$ instancias. Para el caso de $(20 - 1)$, serían 129 instancias por etapa.

Deep Layers	LSTM Units	Learning Rate	Batch Size	MAE	RMSE
1	160	0.010	1	0.058718	0.110545
1	80	0.010	1	0.060749	0.111386
1	100	0.005	1	0.061617	0.106597
1	60	0.005	4	0.062245	0.106922
1	40	0.010	4	0.062720	0.109896

Cuadro 4: Búsqueda de Hiperparámetros para Predicciones con Ventana Móvil (20 – 1)

La imagen deja ver que podría estar produciéndose *underfitting* debido a la escasez de datos de los que disponemos. Cuando en una gráfica de función de coste vemos picos tan pronunciados, pueden no significar nada y ser una irregularidad espontánea del entrenamiento; o bien puede ser que el algoritmo no ha sido capaz de generalizar conocimiento, simplemente porque no tenemos tanto conocimiento "que aportarle".

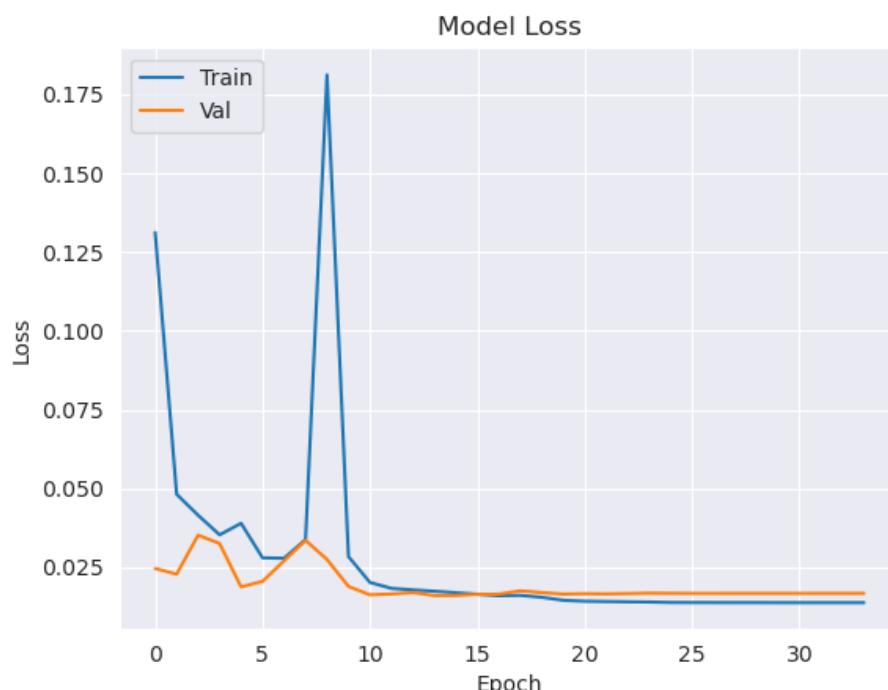


Figura 41: Función de Coste sobre el conjunto de entrenamiento y validación de un modelo (20 – 1)

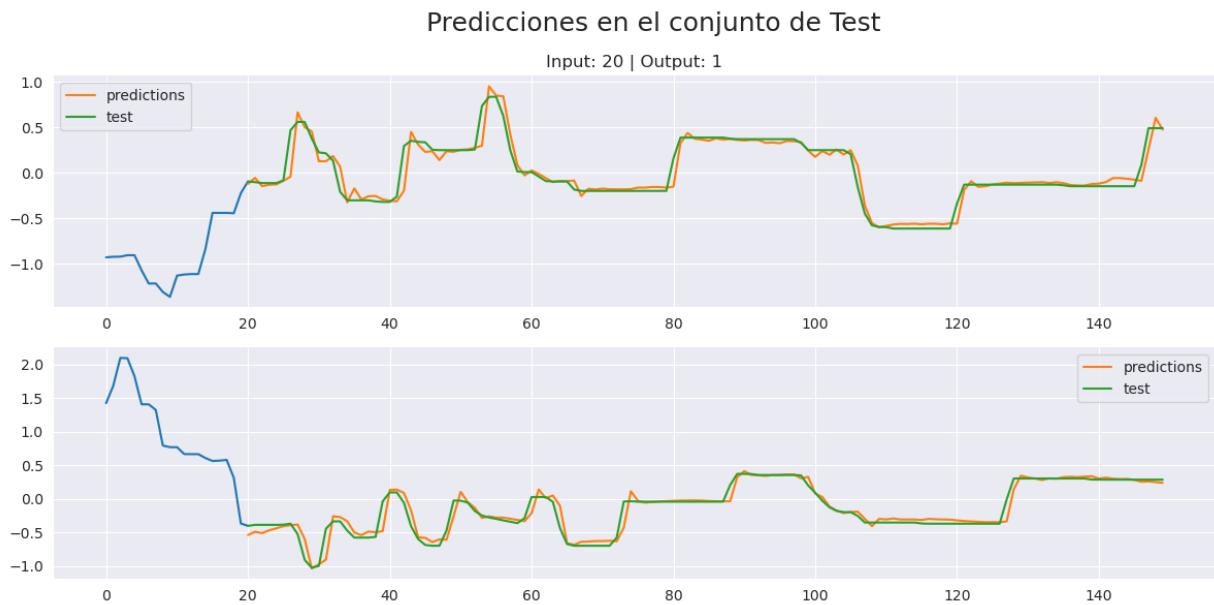


Figura 42: Predicciones con una Ventana Temporal Móvil de tamaño (20 – 1)

Aunque los resultados de las predicciones de la figura (4) son buenos en la mayoría de observaciones, esto era lo esperado. Al fin y al cabo, dados 20 datos de entrada, a priori no debería ser complicado predecir 1 dato; sobre todo para un modelo potente como son las redes recurrentes LSTM. A continuación, veremos modelos más complejos donde obtener buenos resultados será más complicado.

Modelo con 20 Datos de Entrada y 5 de Salida

Para llevar a cabo el ajuste de hiperparámetros relativo al modelo correspondiente a la configuración de datos de entrada y salida (20 – 5), se han probado un total de 190 combinaciones distintas; de donde, tras ordenarlas por los mejores valores de las métricas, se ha obtenido el *top 5* incluido en la tabla (5). Si comparamos estos resultados con los de la tabla (4), podemos observar que las configuraciones que mejor rendimiento han proporcionado son muy similares; sin embargo, si comparamos las métricas, podemos ver una clara degradación en las referentes a la configuración que tiene más datos de salida (20 – 5 en este caso). Esta degradación es totalmente normal, como preveíamos; de hecho, se hará más notable conforme aumentemos el ratio $\frac{n_output}{n_input}$.

En este caso, la mejor configuración de hiperparámetros es aquella que se presenta la primera (en rojo) en la tabla (5) y será la que utilicemos para entrenar el modelo cuyas predicciones se incluyen en la figura (43).

Deep Layers	LSTM Units	Learning Rate	Batch Size	MAE	RMSE
2	160	0.010	4	0.162086	0.237612
1	100	0.050	1	0.165304	0.245987
2	120	0.005	2	0.166026	0.239662
2	80	0.005	4	0.166096	0.241567
1	140	0.005	2	0.166556	0.251542

Cuadro 5: Búsqueda de Hiperparámetros para Predicciones con Ventana Móvil (20 – 5)

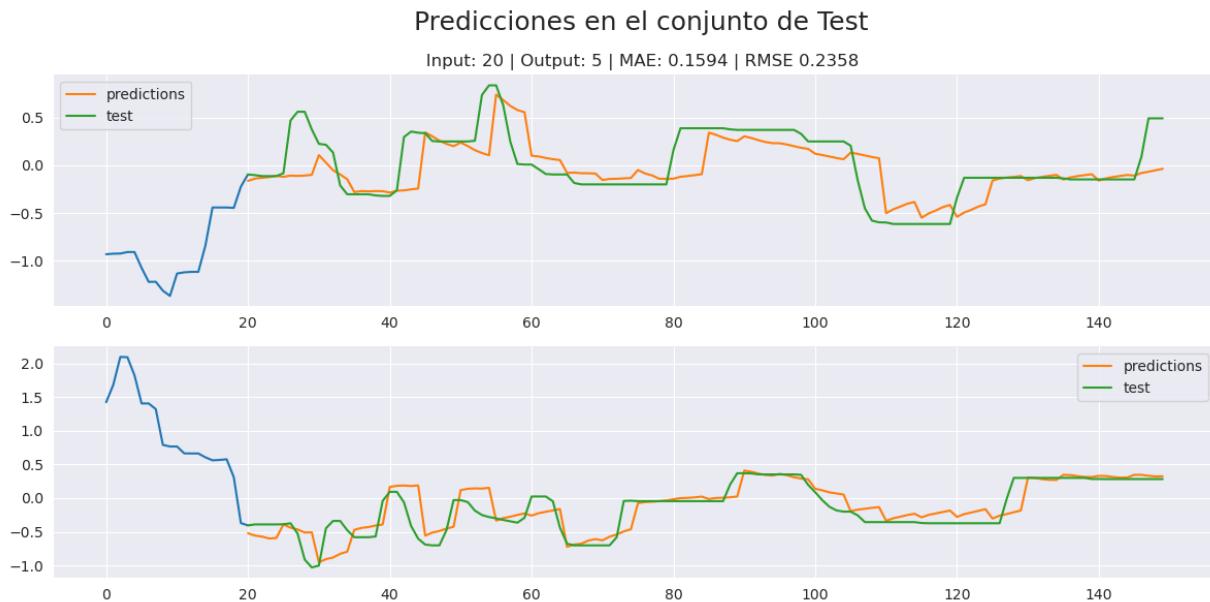


Figura 43: Predicciones con una Ventana Temporal Móvil de tamaño (20 – 5)

Vamos ahora a comentar los resultados de las predicciones del modelo (20 – 5) sobre el conjunto de testeо. Antes, debemos tener claro que existe cierto sesgo en el entrenamiento; puesto que, de nuevo, qué etapas estén en el conjunto de test resulta decisivo teniendo tan pocos datos. En todo caso, mirando la figura (43), observamos que el modelo es capaz de predecir cambios en el valor del DAC, pero que en ciertas ocasiones estos cambios los predice con cierto retraso. Esto puede deberse a que predecir únicamente 5 datos puede no ser suficiente para que el modelo "exprese" toda la información que ha deducido sobre los futuros valores de la serie.

Modelo con 20 Datos de Entrada y 10 de Salida

A partir de este modelo, vamos a dejar de incluir la tabla con las mejores configuraciones de hiperparámetros⁴. Tras analizar diferentes combinaciones, el mejor modelo se ha obtenido con los siguientes hiperparámetros:

$$\left\{ \begin{array}{l} \text{Capas Ocultas: 1} \\ \text{Unidades LSTM: 80} \\ \text{Tasa de Aprendizaje: 0.01} \\ \text{Tamaño del Lote: 4} \end{array} \right.$$

y los resultados se muestran en la figura (44).

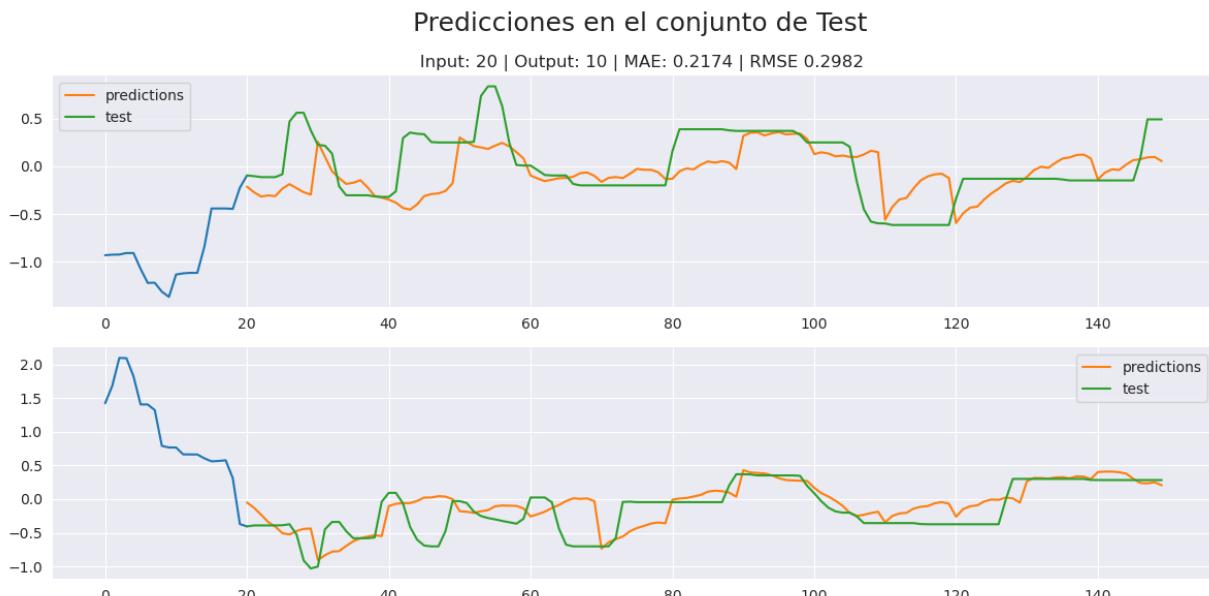


Figura 44: Predicciones con una Ventana Temporal Móvil de tamaño (20 – 10)

Si observamos las predicciones de la figura (44), podemos ver como claramente los resultados son peores que los modelos anteriores, pues tanto el MAE como el RMSE han aumentado al menos 0.06 con respecto a los anteriores. Sin embargo, el modelo es capaz de seguir al DAC, aunque no de manera excesivamente precisa. De alguna manera, la red neuronal conoce cuál debe ser el siguiente movimiento

⁴ Todas las tablas relativas al ajuste de hiperparámetros para cada uno de los modelos realizados se incluyen en el [repositorio del trabajo](#), si se quisieran visualizar de manera más distendida.

del DAC (aumentar o disminuir), pero no es capaz de acertar con cuánto crece o cuánto disminuye.

A continuación, se presentan 3 modelos más. Todos ellos admiten 30 datos de entrada, por lo que tienen más información para aprender los posibles patrones presentes en las etapas. De todas formas, a partir de ahora deberíamos ver unas predicciones peores que las anteriores.

Modelo con 30 Datos de Entrada y 15 de Salida

El primer modelo con 30 datos de entrada es este (30 – 15), que presenta una degradación con respecto a todos los anteriores. El valor del MAE y del RMSE han subido prácticamente un 50 % con respecto al modelo que realizaba predicciones del tipo (20 – 5). No obstante, estos valores pueden estar sesgados por la elección de las etapas usadas para testeo, por lo que debemos esperar a los posteriores resultados de la validación cruzada para obtener conclusiones definitivas.



Figura 45: Predicciones con una Ventana Temporal Móvil de tamaño (30 – 15)

Las predicciones del modelo (30 – 15) se ven en la imagen (45). En la primera etapa de testeo, se ve como aun con este gran número de datos de salida, el modelo es capaz de deducir los movimientos del DAC. Eso sí, en algunos casos lo hace con excesivo retraso de tiempo. En la segunda etapa, vemos como el modelo apenas predice cambios y se mantiene prácticamente estable, lo que al final de la

etapa será lo más óptimo, pues el valor del DAC se mantiene alrededor del valor 0.

Modelo con 30 Datos de Entrada y 20 de Salida

El segundo modelo de entre los que funcionan con 30 datos de entrada, predice 20 valores en el futuro, que son un número de predicciones bastante alto en comparación con los primeros que se han desarrollado.

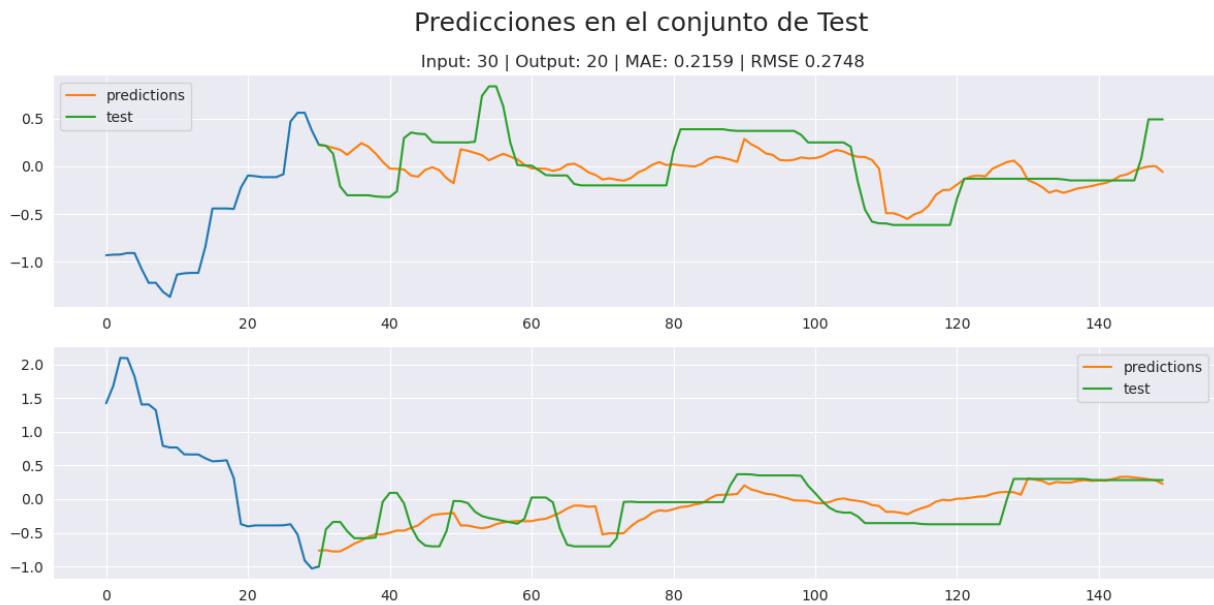


Figura 46: Predicciones con una Ventana Temporal Móvil de tamaño (30 – 20)

La gráficas presentes en la figura (46) representan las predicciones del mejor modelo obtenido de entre los que, dados 30 datos de entrada, predice 20. Los resultados que arroja son mejores de lo esperado: en la primera gráfica, al final de la etapa se consigue seguir la tendencia; mientras que en la segunda gráfica, por momentos se logra predecir con cierta precisión el valor del DAC. En todo caso, las métricas son 0.2159 de MAE y 0.2748 de RMSE, bastante peores que los anteriores modelos.

Modelo con 30 Datos de Entrada y 30 de Salida

El último modelo que presentamos para este problema es el (30 – 30). Con él, se pretende mostrar la degradación en la precisión de las predicciones. En la figura

(47) podemos observar que el modelo apenas es capaz de predecir los cambios de tendencia. Incluso, en algunos instantes de tiempo se "inventa" cambios de tendencia que en absoluto se asemejan a los valores objetivo del DAC.

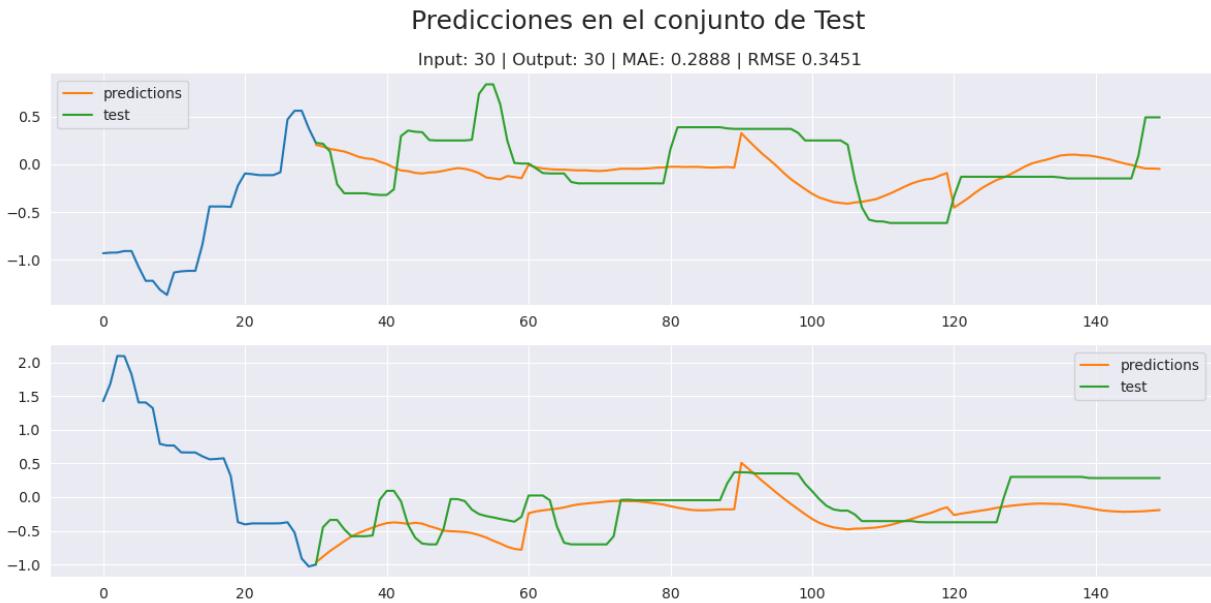


Figura 47: Predicciones con una Ventana Temporal Móvil de tamaño (30 – 30)

Validación Cruzada Leave two out

Una vez se han presentado todos los modelos que se han realizado con sus respectivas predicciones sobre conjuntos de testeо, vamos a tratar de mejorar al máximo el rendimiento de estos. Como ya sabemos, el gran problema de tener pocos datos es que el hecho de incluir unos o otros en depende qué conjunto (entrenamiento, validación y testeо) puede provocar un aprendizaje totalmente distinto.

Para evitar este sesgo, usamos un algoritmo de validación cruzada, dejando únicamente 2 etapas fuera del entrenamiento, una se usa para validación y otra para testeо. Este proceso se lleva a cabo mediante el algoritmo (5) que explicamos anteriormente. Asimismo, las gráficas que se van a exponer a continuación, siguen la misma metodología que aquellas referentes a la validación cruzada para el primer problema; esto es, se grafica la media de las predicciones, con unos márgenes correspondientes a la desviación estándar, sobre la media de todas las etapas de las que disponemos.

En el código, todo este proceso se encuentra en uno de los *notebook* de Python llamado *rnn_crossval_mov_win.ipynb*, donde debemos resaltar la importancia de algunas funciones clave como son: *generate_cross_validation_sets()* y *cross_validation_tuning()*. Estas dos funciones, junto con las que ya se emplearon para la creación de los modelos anteriores, son las que se encargan de preparar los datos convenientemente para cada iteración, además de llevar el entrenamiento de las diferentes redes.

Emplearemos una vez la técnica de validación cruzada *leave two out* para cada uno de los modelos anteriores y veremos como las gráficas mejoran con respecto a las anteriores. Cabe destacar que las métricas que se obtienen en estos casos deben tomarse como meras aproximaciones. Al fin y al cabo, un modelo que está entrenado con 20 instancias, es altamente improbable que pueda generalizar el suficiente conocimiento como para tener un buen rendimiento en específicamente 1 instancia de testeo que no ha visto antes. Para entender el potencial de la validación cruzada, tenemos que quedarnos con la precisión de las predicciones medias, que no es la que aparece reflejada en las imágenes posteriores. Las métricas de las figuras indican la precisión media de las predicciones sobre las etapas, no la precisión de la media de las predicciones con respecto a la media de las etapas. Es muy importante tener esto en cuenta para evaluar la bondad de los modelos.

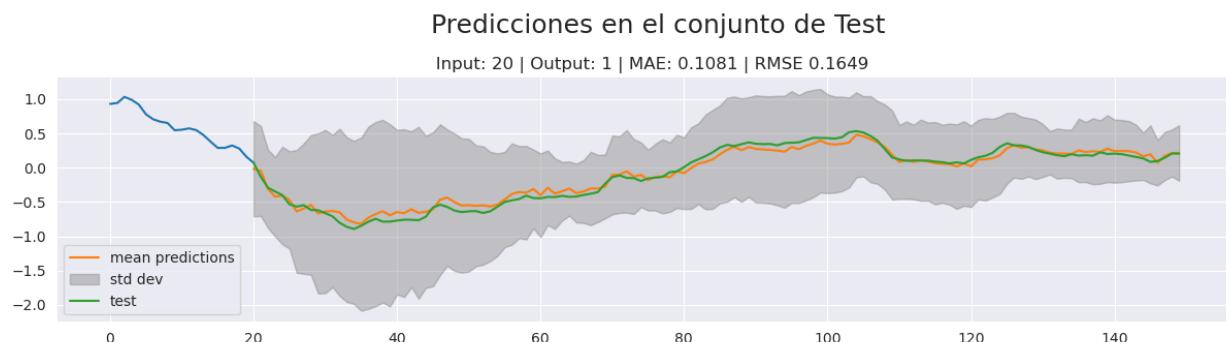


Figura 48: Validación Cruzada para una Ventana Temporal Móvil de (20 – 1)

La primera figura de predicciones medias con validación cruzada es (48), se corresponde con el modelo (20 – 1) y podemos ver como las predicciones tienden con una precisión excelente a la media de las etapas. Si la comparamos con la figura (42), observaremos un cambio notable.

Los siguientes 3 modelos, que son (20 – 5), (20 – 10) y (30 – 15), tienen sus predicciones en las figuras (49), (50) y (51). Estos cumplen con un mismo patrón, las predicciones se desvían muy ligeramente de la media de las etapas, pero al

final todas predicen con exactitud notable la mayoría de los instantes de tiempo que conforman una etapa. De nuevo, podemos comparar estas gráficas con los modelos anteriores sin validación cruzada y la diferencia es sustancial.

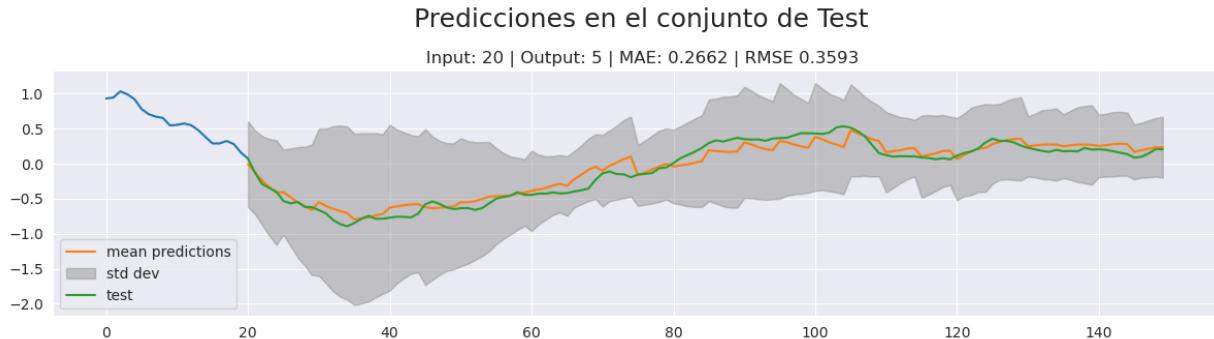


Figura 49: Validación Cruzada para una Ventana Temporal Móvil de (20 – 5)

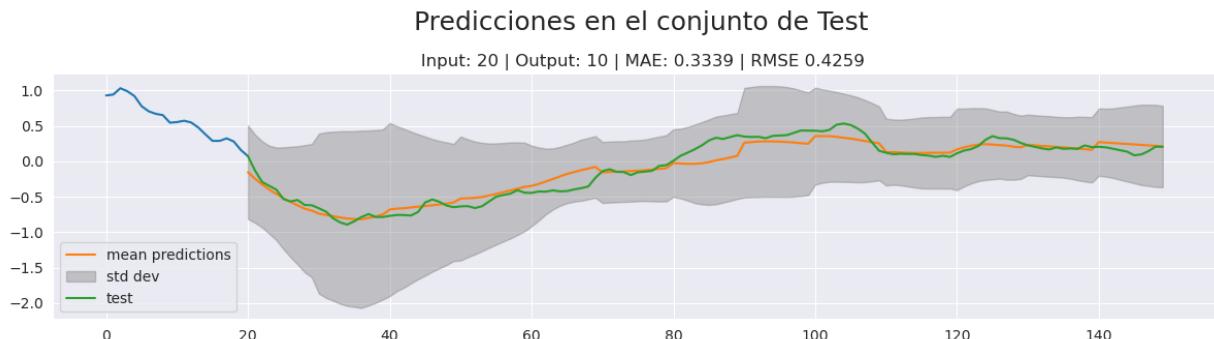


Figura 50: Validación Cruzada para una Ventana Temporal Móvil de (20 – 10)

Finalmente, con respecto a los modelos (30 – 20) y (30 – 30) representados en (52) y (53), se ve como las predicciones se van degradando al haber aumentado el tamaño de la ventana de salida. No obstante, en la comparación con las redes que no tenían validación cruzada, se aprecia un cambio, en especial al principio y al final de las etapas.

12.5 EVALUACIÓN DE LOS RESULTADOS

A la vista de los resultados anteriores, se pueden sacar algunas conclusiones que vamos a enumerar a continuación:

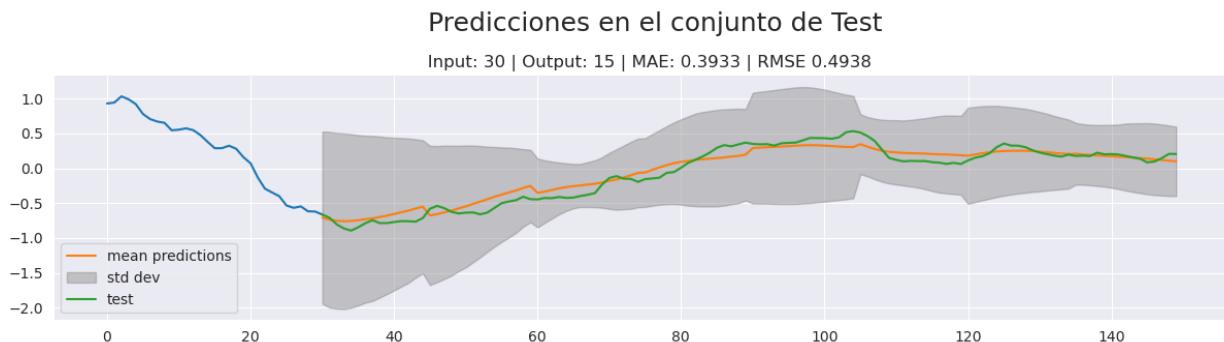


Figura 51: Validación Cruzada para una Ventana Temporal Móvil de (30 – 15)

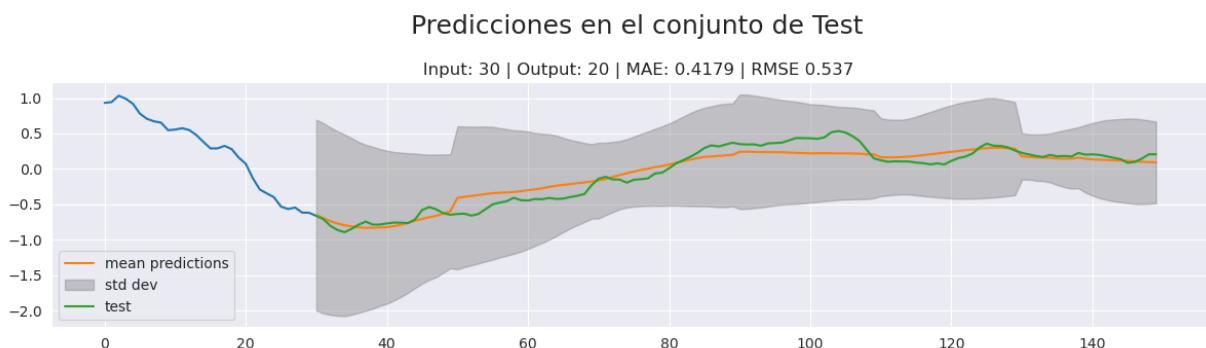


Figura 52: Validación Cruzada para una Ventana Temporal Móvil de (30 – 20)

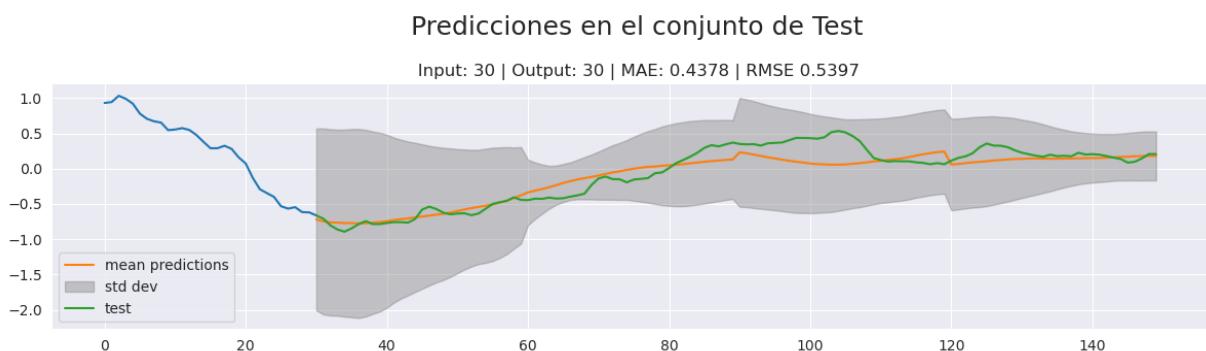


Figura 53: Validación Cruzada para una Ventana Temporal Móvil de (30 – 30)

- La escasez de datos ha propiciado el fenómeno de *underfitting*, de manera que por mucho que sigamos entrenando nuestros modelos, resultará difícil que sean capaces de reconocer patrones y generalizar el conocimiento.
- El código que se ha generado para el desarrollo de los modelos, gráficos y algoritmos es perfectamente reutilizable si se quisieran añadir más datos, dinamizando todo el proceso.
- Los resultados con respecto al primer problema arrojan la siguiente información:

- Los modelos son capaces de detectar los cambios de tendencia y controlar la cantidad de variación del DAC, con una precisión notable teniendo en cuenta la escasez de datos.
 - Al realizar validación cruzada, se observa que los modelos no son capaces de generalizar el conocimiento. Esto se debe a la separación 20-1-1 del *leave two out*.
- Los resultados con respecto al segundo problema arrojan la siguiente información:
- Los modelos presentan una clara degradación de las predicciones conforme el coeficiente n_output/n_input aumenta.
 - Los modelos con ventanas del tipo (20 – 1), (20 – 5) y (20 – 10) son capaces de generalizar el conocimiento de manera adecuada, consiguiendo predicciones bastante buenas.
 - Los modelos con ventanas del tipo (30 – 15), (30 – 20) y (30 – 30) apenas pueden reconocer tendencias de la serie y suelen realizar las predicciones con cierto retardo.
 - La validación cruzada aplicada a estos modelos aumenta notablemente la precisión, por lo que se deja entrever que ante una mayor cantidad de datos, posiblemente el rendimiento fuese muy bueno.

Teniendo en cuenta todo lo anterior podemos concluir que los resultados son relativamente buenos teniendo en cuenta lo disruptivo que puede resultar tener que dividir los datos en función de las etapas de disciplinado, creando ventanas móviles y sobre todo, disponiendo de muy pocos datos para llevar a cabo el entrenamiento.

13

CONCLUSIONES Y TRABAJO FUTURO

Esta sección se compone de 2 partes diferenciadas. En primer lugar, se tratan los objetivos iniciales del proyecto, analizando cuáles se han cumplido y con qué nivel de éxito. Posteriormente, se detallará el posible trabajo futuro que queda abierto tras el desarrollo de este proyecto.

13.1 REPASO DE LOS OBJETIVOS DEL PROYECTO

Tras haber llevado a cabo la evaluación de resultados vista en el anterior capítulo y haber realizado un análisis global de nuestro trabajo, podemos afirmar que se han cumplido los 5 objetivos que se plantearon en un comienzo.

En lo referente al (OBJ-1.), se ha logrado realizar un exhaustivo estudio teórico de las bases matemáticas relacionadas con procesos estocásticos y el algoritmo del descenso del gradiente, con un enfoque específico en el análisis de series temporales. Hemos introducido conceptos fundamentales de campo de la matemática muy diversos, como pueden ser la estadística y el análisis matemático; en particular, un estudio de la diferenciabilidad, de donde podemos destacar un teorema fundamental en la definición del gradiente, como es la regla de la cadena.

En cuanto al (OBJ-2.), se ha llevado a cabo una extensa búsqueda y estudio sobre los relojes atómicos y su relevancia en el contexto actual de un mundo completamente globalizado e interconectado, donde la sincronización cobra una vital importancia. Se han analizado las características y principios del funcionamiento de osciladores, así como su importancia en diversas aplicaciones tecnológicas. Por último, se ha realizado un análisis de los algoritmos de disciplinado utilizados para mantener la sincronización precisa de los relojes, comprendiendo las bases teóricas y los fundamentos matemáticos involucrados.

El ([OBJ-3.](#)), se ha cumplido mediante un repaso de las bases teóricas y directrices fundamentales para la construcción de redes neuronales recurrentes, centrandonos en las que están compuestas de bloques de neuronas LSTM (Long Short-Term Memory), de las que hemos estudiado minuciosamente su funcionamiento, adquiriendo un conocimiento sólido sobre ellas. Se destaca la capacidad que tienen este tipo de neuronas de capturar dependencias a largo plazo en series temporales. Finalmente, se ha explorado una variedad de técnicas de optimización utilizadas para el entrenamiento eficiente de estas redes, permitiendo obtener modelos de alta calidad y rendimiento. Entre ellas, destacamos las de ajuste dinámico, ajuste de hiperparámetros y la validación cruzada.

En relación al ([OBJ-4.](#)), se ha logrado construir modelos predictivos capaces de predecir las observaciones restantes en una etapa de disciplinado de un reloj atómico. Estos modelos se han llevado a cabo con redes recurrentes LSTM y su entrenamiento se ha basado en el comportamiento previo de un reloj atómico que tomamos como datos de entrada. Los modelos obtenidos han demostrado su capacidad para encontrar el comportamiento adecuado que debe seguir el algoritmo de disciplinado para finalizar correctamente la etapa. Asimismo, cabe destacar que estos modelos mantienen rango de mejora, que no ha podido efectuarse debido a la escasez de datos; en todo caso, igualmente podrían utilizarse como una herramienta de detección de problemas en el disciplinado, alertando cuando los patrones esperados no se cumplen, lo que permitiría identificar posibles errores o irregularidades en el oscilador.

Por último, referente al ([OBJ-5.](#)), se han obtenido modelos predictivos con la capacidad de deducir futuros valores (1, 5, 10, 15, 20 y 30) del algoritmo de disciplinado en el corto plazo. Estos modelos han sido entrenados utilizando técnicas similares a las anteriores, aunque con una metodología diferente debido a la inclusión de ventanas temporales móviles. Las predicciones aun tienen margen de mejora y necesitarían más tiempo de entrenamiento, aunque algunos de los modelos ya podrían ser capaces de identificar discrepancias entre los valores predichos y los valores reales de disciplinado, permitiendo corroborar el valor nominal correcto que debería haber sido alcanzado en caso de posibles errores externos o fallos humanos.

13.2 TRABAJO FUTURO

Habiendo logrado los objetivos establecidos, existen diversas líneas de trabajo futuro que podrían ser exploradas para ampliar y mejorar aún más el conocimiento y el desarrollo de los modelos predictivos conseguidos.

Por un lado, se puede buscar una mejora en los modelos predictivos. A pesar de haber obtenido modelos capaces de predecir el comportamiento de los relojes atómicos en distintas etapas de disciplinado, se pueden explorar enfoques adicionales para mejorar la precisión y confiabilidad de las predicciones. Esto podría implicar la exploración de arquitecturas de redes neuronales más complejas o la incorporación de nuevos datos provenientes de osciladores de la misma naturaleza. La escasez de datos ha provocado que se eleve la dificultad de entrenar redes neuronales preparadas para aprender a partir de miles de datos.

Por otro lado, podría desarrollarse alguna mejora para algoritmos de disciplinado. Podría investigarse y desarrollar algoritmos más sofisticados que utilicen técnicas de control avanzadas, como controladores predictivos o adaptativos, con el fin de lograr disciplinados más regulares sobre los que poder aplicar técnicas de predicción de series temporales con mayor probabilidad de éxito.

Finalmente, los conocimientos sobre algoritmos de disciplinado y predicción en relojes atómicos pueden tener aplicaciones en diversos campos y tecnologías. Puede explorarse la transferencia de este conocimiento para abordar problemas de sincronización y predicción en sistemas de comunicaciones, de navegación por satélite o cualquier otro ámbito donde se haga necesaria la sincronización temporal.

BIBLIOGRAFÍA

- [1] A. C. Ian Goodfellow, Yoshua Bengio, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [2] T. Mitchell, *Machine Learning*. McGraw Hill series in computer science, McGraw Hill, 2017.
- [3] B. Lim and S. Zohren, "Time-series forecasting with deep learning: a survey," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 379, p. 20200209, feb 2021.
- [4] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, "Deep learning with long short-term memory for time series prediction," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 114–119, 2019.
- [5] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, pp. 850–855 vol.2, 1999.
- [6] H. Yiğitler, B. Badihi, and R. Jäntti, "Overview of time synchronization for iot deployments: Clock discipline algorithms and protocols," *Sensors*, vol. 20, no. 20, 2020.
- [7] R. Nandita, S. Maharana, R. Bijoy, S. G. T, and S. Krishnamoorthy, "An artificial neural network model for timescale atomic clock ensemble algorithm," *MAPAN*, vol. 35, 12 2020.
- [8] M. Spiegel, J. Schiller, and R. Srinivasan, *Probability and Statistics*. McGraw Hill, New York, 2002.
- [9] R. Durrett, *Essentials of Stochastic Processes*. Springer Texts in Statistics, Springer Cham, 2016.
- [10] S. M. Ross, *Stochastic Processes*. John Wiley & Sons Inc., 2nd ed., 1996.
- [11] B. Abraham and J. Ledolter, "Statistical methods for forecasting," 1983.
- [12] W. Fleming, *Functions of Several Variables*. No. 2, Springer New York, NY, 1977.

- [13] C. Aparicio and R. Payá, *Análisis Matemático I*. Textos Universitarios Universidad de Granada, 7th ed., 1999.
- [14] T. Apostol, *Análisis Matemático*. No. 2, REVERTE, 1976.
- [15] Electrical 4 U, "Oscillators: What are they? (definition, types & applications)." <https://www.electrical4u.com/what-is-an-oscillator>, 2020. Accessed: 2023-03-23.
- [16] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle, *Global Navigation Satellite Systems - GPS, GLONASS, Galileo and more*. No. 1, Springer-Verlag Wien New York, 2008.
- [17] L. Galleani and P. Tavella, "The dynamic allan variance," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 56, no. 3, pp. 450–464, 2009.
- [18] The Secure Network Time Protocol (NTPsec) Distribution, "Clock discipline algorithm." <https://docs.ntpsec.org/latest/discipline.html>. Accessed: 2023-03-23.
- [19] D. Mills, "Adaptive hybrid clock discipline algorithm for the network time protocol," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 505–514, 1998.
- [20] J. Levine, M. Lombardi, and A. Novick, "Nist computer time services: Internet time service (its), automated computer time service (acts), and time.gov web sites," *NIST Special Publication 250-59*, p. 79 pages, 05 2002.
- [21] J. Martin, J. Burbank, W. Kasch, and D. L. Mills, "Network Time Protocol Version 4: Protocol and Algorithms Specification." RFC 5905, June 2010.
- [22] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *CoRR*, vol. abs/2104.05314, 2021.
- [23] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [24] Z. Shen, Y. Zhang, J. Lu, J. Xu, and G. Xiao, "A novel time series forecasting model with deep learning," *Neurocomputing*, vol. 396, pp. 302–313, 2020.
- [25] J. Connor, R. Martin, and L. Atlas, "Recurrent neural networks and robust time series prediction," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 240–254, 1994.

- [26] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [27] C. Tallec and Y. Ollivier, "Unbiasing truncated backpropagation through time," 2017.
- [28] Christopher Olah, "Understanding lstm networks." <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, 2015. Accessed: 2023-05-28.
- [29] F. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, pp. 189–194 vol.3, 2000.
- [30] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.
- [31] D. Peña, *Análisis de Series Temporales*. 2010.
- [32] G. C. R. G. M. L. George E. P. Box, Gwilym M. Jenkins, *Time Series Analysis: Forecasting and Control*. John Wiley & Sons Inc., 5th ed., 2016.

A

APÉNDICE

A.1 OTROS RELOJES ATÓMICOS

Se incluye la información referente a los osciladores descartados (54) y (55) por ser de una naturaleza distinta al usado.

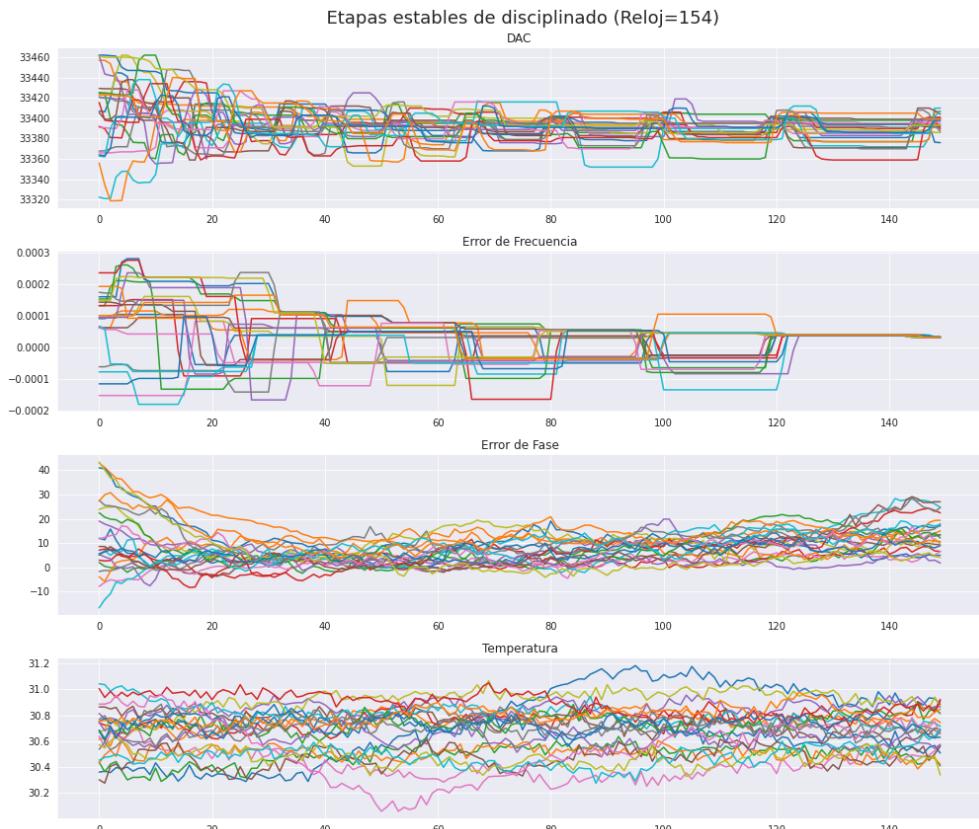


Figura 54: Oscilador 154

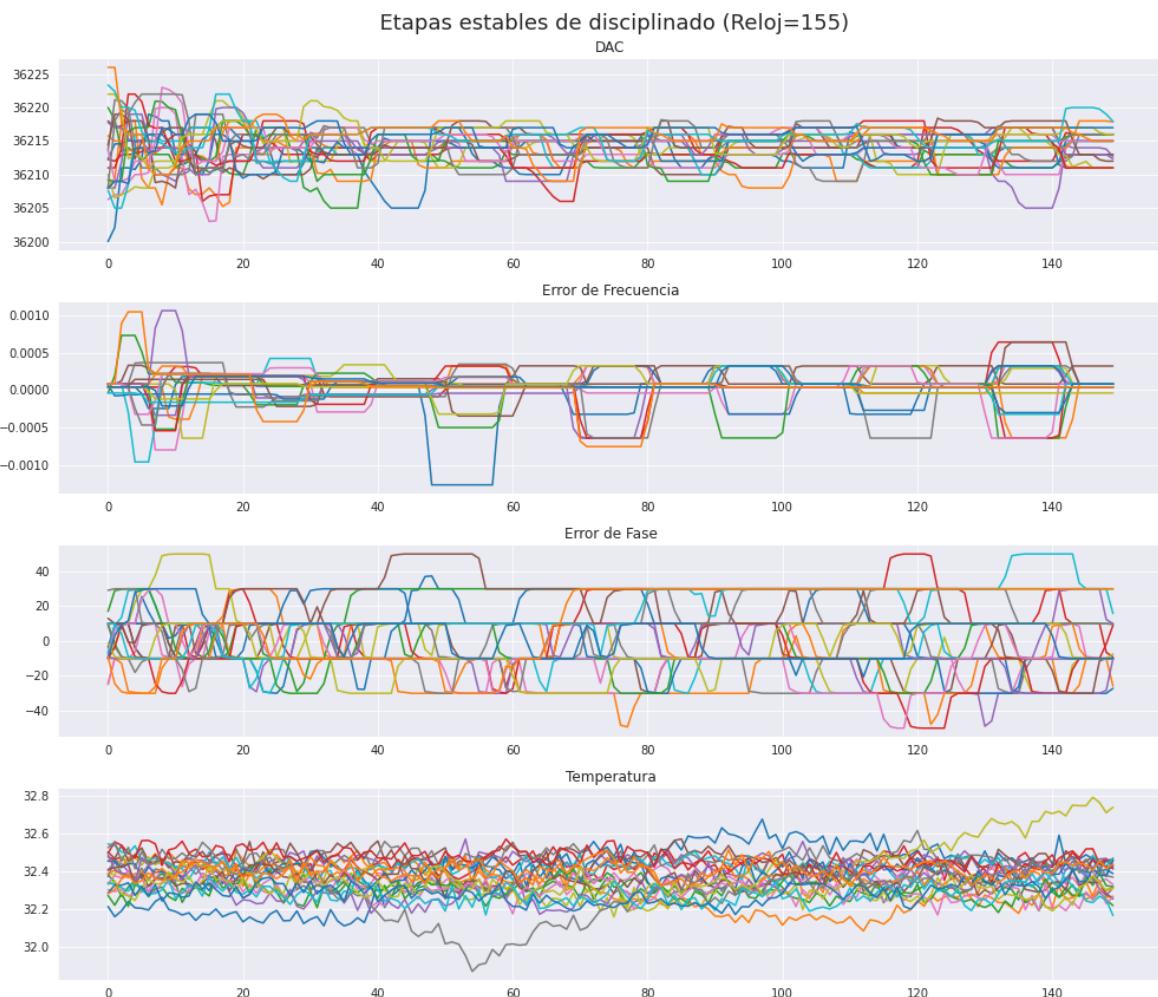


Figura 55: Oscilador 155