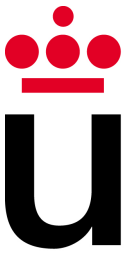


Universidad
Rey Juan Carlos

TESIS DOCTORAL

Estudio de métodos estadísticos y de machine learning avanzados para la estimación de momentos de segundo orden, en problemas de regresión con énfasis en el análisis de series temporales

Autor:

León Beleña

Directores:

Prof. Luca Martino

Prof. Valero Laparra

Programa de Doctorado en
Tecnologías de la Información y las Comunicaciones

Escuela Internacional de Doctorado

2024

Agradecimientos

Esta tesis tiene que agradecer mucho a mucha gente.

Quería comenzar agradeciendo a mis directores Valero Laparra y Luca Martino.

A Valero le agradezco aquella clase de redes neuronales en la que me ayudó a crear una primera idea de tesis. Por fin años después queda materializada.

Luca, tú has sido capaz de liderar este proyecto con verdadera maestría. Jamás olvidaré que la frase "confía en Luca", está llena de verdad. Gracias a los dos por todo lo que he aprendido (y reído!!), mucho más de lo que jamás pude imaginar. Hacéis un buen tándem investigador, y vuestra amistad ha permitido consolidar esta tesis en algo real. Habéis aportado y compartido mucho conocimiento y valor. Si me puedo sentir orgulloso de esta tesis, es en gran parte gracias a vuestra ayuda.

Agradecimiento especial a Ernesto C., espero que sepas todo lo que vales y que no te conformes con menos que la excelencia. Eres el óxido nitroso de las ideas. Nos queda mucho por hacer.

A mis profesores de universidad, en especial a Miguel Jerez, quien ha estado siempre disponible aún con el paso de los años para resolver dudas, e incluso para motivar a la hora de hacer la tesis. Y también a Gregorio R. Serrano, al que le tengo que agradecer que me contagiase su interés en aprender a programar en R. Eternamente agradecido.

A Carmen, Ana, Pablo, Víctor, Javier y Emilio: Las conversaciones que he tenido con cada uno de vosotros han aportado ese punto de vista tan distinto y tan necesario para avanzar. Esta tesis tiene mucho que agradeceros. Y yo también.

Mi agradecimiento y reconocimiento especial a Julio. Creo que todavía no sabes lo que has supuesto en todo este proceso. Marcaste un antes y un después.

A mis alumnos, los verdaderos maestros. Sin vosotros, esto no tendría prácticamente ningún sentido.

A mi familia, en especial a mis padres, hermanos y abuelas, por estar y teneros siempre presentes de alguna manera. A Fátima y a Ángel, por todas vuestras enseñanzas y buenos consejos.

A mis amigos universitarios Clara, Inés, Dani y Marcos, los "analistas inconformistas". Ningún modelo predictivo "stricto sensu", hubiera sido jamás capaz de predecir dónde y cómo estamos ahora. Y que siga siendo así.

A Paco y a Alberto, por vuestra eterna escucha y consejos. Sois un referente. Gracias por motivarme e inspirarme a llegar cada día más lejos.

A mis amigos, particularmente a Ernesto y a Victoria, que gracias a su amistad y conocimiento, me han dado claridad y guía cuando más falta hacía. A Bea, Pepa y Álvaro por vuestro constante interés y muestras de ánimo.

Gracias a María, por ser cada día mejor e ir avanzando sin que se note. Gracias por acompañarme en este proceso tan enriquecedor como agotador. Prometo "garchear" un poquito menos y devolverte todo este tiempo invertido. Sé que harás lo posible para que deje de buscar nuevos retos y aventuras. No prometo nada.

Resumen

Esta tesis doctoral explora la intersección entre los métodos clásicos de análisis de series temporales y técnicas avanzadas de machine learning. La investigación se centra en la integración de estos enfoques para mejorar la precisión, la estabilidad y la interpretabilidad de los modelos predictivos en diversos campos, especialmente en series temporales. A través de un estudio y de la implementación de metodologías avanzadas, este trabajo ofrece nuevas perspectivas y soluciones a problemas complejos, destacando la importancia de la adaptabilidad y la innovación en la ciencia de datos actual. Se ha incorporado una técnica innovadora, la normalización divisiva generalizada (GDN) adaptándola al contexto de series temporales, y se ha demostrado su efectividad en mejorar la estabilidad y reducir la variabilidad en modelos de predicción complejos. También se exploran métodos avanzados para la estimación de volatilidad y predicción de series temporales financieras, incorporando los métodos de suavizado de kernel y la normalización divisiva generalizada. Ambas se presentan como un enfoque novedoso que supera las limitaciones de los modelos GARCH tradicionales.

El principal objetivo de esta investigación es desarrollar y validar métodos estadísticos avanzados para la estimación de la volatilidad y la predicción de series temporales financieras que mejoren la capacidad predictiva y la interpretación en comparación con los modelos tradicionales. De forma más precisa, se trata de desarrollar un método de suavizado de kernel para la estimación de la varianza local en datos de series temporales, que se adapte de forma dinámica a las distintas estructuras de dichas series. También se integra la normalización divisiva generalizada (GDN) dentro de las diferentes arquitecturas de redes neuronales, enfocadas a la predicción de series temporales para mejorar el entrenamiento y la estabilidad de las predicciones, extendiendo su aplicabilidad más allá de su uso tradicional en el procesamiento de imágenes.
La investigación desarrolla dos métodos principales de suavizado de kernel adaptados a la estimación de varianza local, junto con la aplicación de la normalización divisiva generalizada para mejorar la interpretación y estabilidad de las predicciones en series temporales financieras. Esta integración de técnicas clásicas y modernas forma un nuevo marco para el análisis y modelado de series temporales que es tanto flexible como robusto. La investigación ha adoptado un enfoque metodológico compuesto de revisión exhaustiva de literatura, y desarrollo y optimización de algoritmos en Python y MATLAB. Entre las contribuciones principales, se incluye la publicación de un artículo en una revista de alto impacto y la participación en conferencias internacionales, reforzando el impacto y la relevancia de la investigación realizada.

Los resultados principales de esta tesis demuestran que la aplicación del suavizado de kernel en la estimación de la volatilidad proporciona un marco flexible y eficaz, capaz de captar dinámicas complejas que los modelos GARCH tradicionales pueden no detectar. La comparativa de rendimiento respecto los modelos GARCH revela que los métodos basados en kernel ofrecen una mejora en la estimación de la volatilidad con menor sesgo y mayor precisión. Adicionalmente, la incorporación de la normalización divisiva generalizada (GDN) en los modelos basados en redes neuronales representa un avance notable en el tratamiento de problemas financieros, al proporcionar una mejora significativa tanto en la precisión de las predicciones como en la estabilidad del modelo frente a variaciones abruptas en la volatilidad de los datos. Esta tesis demuestra que la combinación de técnicas clásicas con innovaciones en machine learning ofrece un enfoque muy completo para el análisis de series temporales, donde la adaptación de la GDN ha sido particularmente efectiva, lo que sugiere caminos prometedores para futuras investigaciones en esta disciplina.

Abstract

This doctoral thesis explores the intersection between classical time series analysis methods and advanced machine learning techniques. The research focuses on integrating these approaches to enhance the accuracy, stability, and interpretability of predictive models across various fields, particularly in time series. Through the study and implementation of advanced methodologies, this work offers new perspectives and solutions to complex problems, emphasizing the importance of adaptability and innovation in contemporary data science. An innovative technique, Generalized Divisive Normalization (GDN), has been adapted to the context of time series, demonstrating its effectiveness in enhancing stability and reducing variability in complex prediction models. Advanced methods for estimating volatility and predicting financial time series are also explored, incorporating kernel smoothing methods and Generalized Divisive Normalization. Both are presented as a novel approach that surpasses the limitations of traditional GARCH models.

The primary objective of this research is to develop and validate advanced statistical methods for estimating volatility and predicting financial time series that improve predictive ability and interpretation compared to traditional models. More precisely, it aims to develop a kernel smoothing method for estimating local variance in time series data, which dynamically adapts to the various structures of these series. Additionally, Generalized Divisive Normalization (GDN) is integrated within different neural network architectures, focused on time series prediction to improve training and the stability of predictions, extending its applicability beyond its traditional use in image processing.
The research develops two main kernel smoothing methods adapted for local variance estimation, along with the application of Generalized Divisive Normalization to enhance the interpretation and stability of predictions in financial time series. This integration of classic and modern techniques forms a new framework for time series analysis and modeling that is both flexible and robust. The research has adopted a methodological approach consisting of a comprehensive literature review, and the development and optimization of algorithms in Python and MATLAB. Among the main contributions are the publication of an article in a high-impact journal and participation in international conferences, reinforcing the impact and relevance of the conducted research.

The principal results of this thesis demonstrate that applying kernel smoothing in volatility estimation provides a flexible and effective framework capable of capturing complex dynamics that traditional GARCH models may not detect. Performance comparisons with GARCH models reveal that kernel-based methods offer improvements in volatility estimation with less bias and greater accuracy. Additionally, the incorporation of Generalized Divisive Normalization (GDN) into neural network-based models represents a significant advancement in addressing financial problems, providing notable improvements in prediction accuracy and model stability against abrupt changes in data volatility. This thesis shows that combining classical techniques with innovations in machine learning offers a very comprehensive approach to time series analysis, where the adaptation of GDN has been particularly effective, suggesting promising paths for future research in this discipline.

"No hay que hacer las cosas como los demás, como cabras; hay que hacer las cosas como uno piensa que son buenas, y no hacerlas únicamente porque los demás hagan las cosas todos iguales, como cabras."(Luca Martino)

Índice general

1. Introducción y objetivos **1**
- 1.1. Contexto científico . . . 1
- 1.2. Objetivos iniciales de la tesis doctoral . . . 2
- 1.3. Desarrollo: descripción de la modalidad de trabajo seguida . . . 2
- 1.4. Objetivos logrados . . . 3
- 1.5. Estructura de la memoria de la tesis . . . 5

2. Caracterización y modelización tradicional de series temporales **7**
- 2.1. Series temporales: notación y definiciones . . . 7
- 2.2. Modelos clásicos para la modelización de series temporales: modelos estacionarios . . . 15
  - 2.2.1. Modelo de medias móviles — MA(q) . . . 16
  - 2.2.2. Modelo autorregresivo de order p — AR(p) . . . 19
  - 2.2.3. Modelo autorregresivo de media móvil — ARMA(p, q) . . . 23
- 2.3. Modelos no estacionarios . . . 24
  - 2.3.1. Modelo autorregresivo integrado de media móvil — ARIMA(p, d, q) . . . 24
- 2.4. Modelos de análisis de volatilidad para series temporales . . . 28
  - 2.4.1. Introducción a la volatilidad . . . 28
  - 2.4.2. Propiedades de la volatilidad en series temporales . . . 28
  - 2.4.3. Modelos clásicos y limitaciones . . . 29
  - 2.4.4. Homocedasticidad y heterocedasticidad . . . 29
- 2.5. Extensiones de los modelos ARCH y GARCH . . . 31
- 2.6. Volatilidad estocástica . . . 32
  - 2.6.1. Modelo de Heston . . . 32
- 2.7. Métodos modernos más avanzados . . . 33
  - 2.7.1. Promedio móvil exponencialmente ponderado - exponentially weighted moving average (EWMA) . . . 33

3. Kernel smoothers **35**
- 3.1. Metodología de suavizadores de kernel para regresión con pesos normalizados . . . 36
- 3.2. Métodos para selección del bandwidth (anchura del kernel) . . . 36
  - 3.2.1. Regla de Silverman . . . 37
- 3.3. Principales tipos de funciones kernel . . . 38
  - 3.3.1. Kernel Gaussiano . . . 38
  - 3.3.2. Kernel t-Student . . . 39
  - 3.3.3. Kernel Rectangular . . . 40
  - 3.3.4. Kernel Triangular . . . 40
  - 3.3.5. Kernel Tricubo . . . 41
  - 3.3.6. Kernel Epanechnikov . . . 42
- 3.4. Ejemplo de métrica de evaluación . . . 43

4. Estudio de series temporales con algoritmos de machine learning 45
4.1. Arquitectura de las redes neuronales . . . 46
4.1.1. Breve historia e introducción . . . 46
4.1.2. Componentes clave y configuración en redes neuronales artificiales: neuronas, pesos, capa densa, operación flatten, inicializador de pesos (kernel) y épocas . . . 47
4.2. Funciones de activación . . . 49
4.2.1. Función de activación sigmoide . . . 50
4.2.2. Función de activación tangente hiperbólica (tanh) . . . 50
4.2.3. Función de activación Unidades Lineales Rectificadas (RELU) . . . 51
4.2.4. Función de activación Unidad lineal exponencial escalada (SELU) . . . 52
4.2.5. Función de activación Softmax . . . 53
4.3. Función de coste . . . 54
4.4. Regularización . . . 56
4.5. Entrenamiento de la red . . . 58
4.5.1. Función de propagación: forward propagation, backward propagation . . . 58
4.5.2. Regla de la cadena . . . 59
4.5.3. Métodos de optimización . . . 59
4.6. Capas específicas para datos temporales . . . 62
4.6.1. Redes neuronales recurrentes . . . 62
4.6.2. Redes neuronales convolucionales . . . 66
4.7. Normalización divisiva . . . 76
4.7.1. Breve historia e introducción . . . 76
4.7.2. Formulación matemática de la normalización divisiva: . . . 79
4.7.3. Normalización Divisiva Generaliazda (GDN): . . . 80
4.7.4. Ventajas y aplicaciones de la GDN . . . 81

5. Aproximaciones del momento segundo mediante kernel smoother con aplicación a la estimación de la volatilidad 83
5.1. Introducción . . . 83
5.2. Aproximando la tendencia . . . 85
5.3. Procedimientos de estimación de varianza . . . 86
5.4. Extensiones y variantes . . . 87
5.4.1. Uso de métodos de regresión genéricos . . . 87
5.4.2. Estimación de la covarianza entre dos entradas genéricas . . . 88
5.4.3. Escenario de múltiples salidas y otras extensiones . . . 88
5.4.4. Añadiendo más flexibilidad al modelo inicial . . . 89
5.4.5. Uso conjunto de un kernel rectangular y una solución de mínimos cuadrados para el análisis de series temporales . . . 89
5.5. Experimentos numéricos . . . 90
5.5.1. Aplicaciones de series temporales . . . 90
5.5.2. Aplicación en dimensiones de entrada superiores: datos reales sobre emociones de paisajes sonoros . . . 95
5.6. Conclusiones . . . 95

6. Redes neuronales para el análisis de series temporales financieras 97
6.1. Objetivo inicial . . . 97
6.2. Metodología aplicada . . . 98
6.2.1. Tabla de librerías, usos y versiones . . . 98
6.2.2. Obtención y correlación de los datos . . . 98
6.2.3. Descripción de los datos . . . 100

6.2.4. Normalización de los datos . . . 102
6.2.5. Partición de los datos y validación cruzada . . . 102
6.2.6. Entrenamiento . . . 103
6.2.7. Evaluación de modelos . . . 104
6.3. Resultados obtenidos de los experimentos realizados . . . 104
6.3.1. Modelo Lineal & Lineal GDN . . . 105
6.3.2. Modelo GRU & GRU GDN . . . 106
6.3.3. Modelo LSTM & LSTM GDN . . . 107
6.3.4. Modelo CNN & CNN GDN . . . 108
6.4. Conclusiones . . . 110

7. Conclusiones **113**
7.1. Conclusiones y contribuciones principales . . . 114
7.2. Otros hallazgos . . . 114
7.3. Posibles líneas de investigación futuras . . . 115

Bibliografía **117**

Capítulo 1

Introducción y objetivos

1.1 Contexto científico

En las últimas décadas se han producido avances muy significativos en el campo de la ciencia de datos y el análisis de series temporales que han redefinido distintas áreas, tanto industriales como del ámbito más científico. La ciencia de datos es un campo interdisciplinar que utiliza algoritmos, procesos, sistemas y métodos científicos para extraer conocimiento, tanto de datos estructurados como no estructurados. Esta ciencia se ha convertido en esencial para una mejor toma de decisiones estratégicas y operativas en empresas y gobiernos [127].

Las series temporales son secuencias de datos indexados con un orden temporal. Esta tipología de datos permite observar tendencias, ciclos y patrones estacionales, lo que facilita la predicción de eventos futuros con alta precisión. Esto es especialmente relevante en áreas como la economía, las finanzas, la meteorología y la salud, donde existen una gran parte de las fuentes de datos que tiene una temporalidad inherente a ellos [30]. El valor de la ciencia de datos y el análisis de series temporales ha crecido exponencialmente con el avance de la tecnología de almacenamiento y de computación. Esto ha permitido que la capacidad de procesar grandes cantidades de datos en tiempo real se haya transformado por completo, permitiendo a las empresas y organizaciones responder de forma ágil a los cambios de mercado actuales.

Este desarrollo ha sido impulsado por la creciente disponibilidad de grandes conjuntos de datos y el avance de técnicas de aprendizaje automático y algoritmos estadísticos. A día de hoy, las diferentes técnicas como pueden ser las redes neuronales, los modelos ARIMA (entre otros) para el análisis de series temporales y los modelos GARCH para la modelización y tratamiento de la volatilidad en series temporales, han abierto la puerta a poder abordar complejidades en los datos que antes resultaban inalcanzables [28]. Por todo ello, la ciencia de datos y el análisis de series temporales se consideran, hoy en día, esenciales e indispensables debido principalmente a su capacidad para transformar grandes volúmenes de datos, en información y conocimiento accionable. Estas disciplinas ofrecen una ventaja competitiva que permite a las organizaciones anticipar tendencias y adaptarse a distintos entornos dinámicos y en constante cambio.

1.2 Objetivos iniciales de la tesis doctoral

El objetivo inicial y general de esta investigación es estudiar la viabilidad y eficacia de integrar las técnicas clásicas de estudio de análisis y modelización de series temporales, con nuevas tecnologías más modernas, basadas en machine learning (como las redes neuronales). Esta integración tiene como objetivo fortalecer el arsenal metodológico para analizar datos de series temporales, así como ampliar el alcance y la precisión de los modelos predictivos en estos campos. Al mismo tiempo, los métodos clásicos de predicción y análisis pueden mejorar la *interpretability* de los métodos basados en machine learning.
Entre los objetivos específicos de esta investigación, destaca la adaptación de la normalización divisiva generalizada propuesta en [13] al marco de series temporales. Cabe destacar que originalmente la normalización divisiva generalizada nació con el propósito de mejorar el procesamiento de imágenes, obteniendo además resultados muy prometedores [13]. Esto supuso una notable contribución científica, que fue desarrollada inicialmente por el Prof. Valero Laparra y los co-autores.

1.3 Desarrollo: descripción de la modalidad de trabajo seguida

La modalidad de trabajo llevada a cabo para la realización de esta investigación, ha estado compuesta de varias fases con el objetivo de abarcar todos los aspectos principales de este estudio. Antes de describir estas fases en detalle, se considera importante remarcar dos aspectos que han impactado en el desarrollo de la tesis, así como en el posible alcance de los objetivos iniciales planteados:

- Cabe resaltar que el doctorando León Beleña, en todo el periodo de desarrollo de la tesis, ha trabajado siempre en el ámbito académico como Profesor Asistente en la Universidad Francisco de Vitoria. Claramente, todas las tareas de trabajo realizadas por León Beleña han incrementado su experiencia académica y conocimientos (así como el CV) del doctorando, pero han ralentizado el desarrollo de la tesis (respecto a un estudiante con el $100\,\%$ de dedicación a la tesis doctoral).
- Además, al comienzo de la tesis, el background del doctorando León Beleña era esencialmente en ciencias económicas. Por esta razón, cabe resaltar el gran esfuerzo hecho por el estudiante para aprender nuevos enfoques, técnicas, y lenguajes de programación, así como la necesidad de cubrir las lagunas de conocimiento.

A continuación, se detalla la metodología de trabajo empleada, divididas en fases/tareas:

[T1] Estudio bibliográfico. Se ha realizado un estudio bibliográfico exhaustivo[1] en el análisis de series temporales y en técnicas modernas de machine learning. Este enfoque multidisciplinar, ha sido necesario para comprender las teorías existentes y explorar cómo la integración de estas áreas puede conducir a importantes innovaciones en el ámbito de la ciencia de datos.
Esta tarea ha resultado ser más compleja de lo previsto (sobretodo en lo que se refiere al tiempo de desarrollo) y sin duda la más subestimada (incluso por parte de los directores). Esto es debido a la inmensa cantidad de literatura de las dos áreas abarcadas en esta tesis: (a) por un lado en lo que respecta a series temporales y volatilidad con modelos clásicos, y por el otro lado (b) regresión/smoothing vía machine learning.

[T2] Programación en Python y Matlab: el doctorando ha incrementado sus habilidades de programación con Python y Matlab, ambos necesarios para testear los diferentes métodos.

[1] Mejor dicho: todo lo exhaustivo que se ha podido, visto la cantidad de literatura relacionada.

[T3] Análisis y diseños de nuevas metodologías. Comparando y valorando a nivel teórico los métodos clásicos ("pro y contra") así como las oportunidades de machine learning, se han intentado diseñar nuevos esquemas mejorando los actualmente existentes en la literatura.

[T4] Desarrollo en Python de algunos métodos. Bajo la ayuda principal del Prof. Valero Laparra, se ha llevado a cabo la programación del código de Python, específicamente diseñado para la implementación y optimización de los algoritmos de redes neuronales, y de las tareas específicas y necesarias en la investigación.

[T5] Desarrollo en MATLAB de algunos métodos. Con la colaboración principal del Prof. Luca Martino y con la ayuda del estudiante Ernesto Curbelo de la Universidad Carlos III de Madrid (UC3M), se ha utilizado MATLAB para implementar y optimizar las técnicas de suavizado mediante kernel smoother, aplicado posteriormente a series temporales.

[T6] Escritura del primer paper, los experimentos para el segundo paper y memoria: La escritura de esta memoria se fue realizando con la suficiente antelación debido a la extensión de dicha tarea. La escritura del primer paper se realizó con la ayuda de ambos directores, aportando así una gran enseñanza y valor, que fue puesto en práctica a la hora de escribir el capítulo referente a predicción de series temporales. Capítulo que servirá de base para un próximo paper de investigación.

El diagrama temporal de Gantt que se muestra en la Figura 1.1, se puede observar cómo la tarea 1 se ha prolongado a lo largo de todo el segundo año (cuando la previsión era la de no exceder los primeros 12 meses).

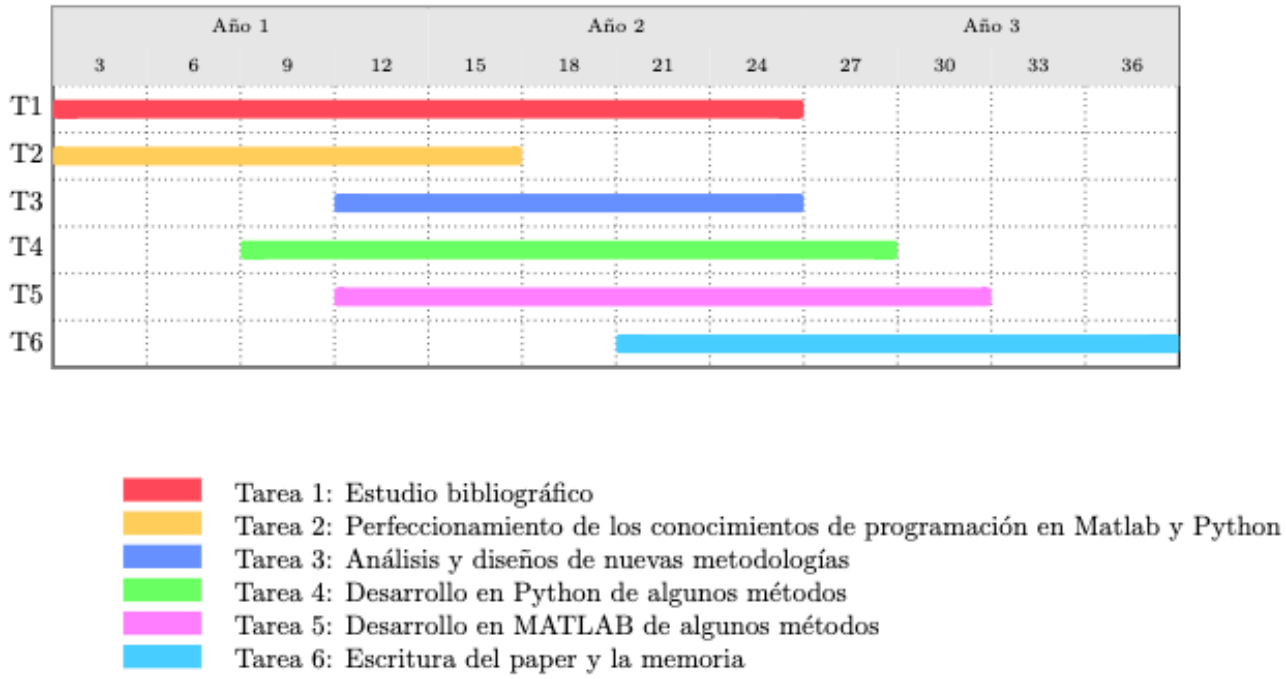


Figura 1.1: Esquema temporal de las tareas de la tesis.

1.4 Objetivos logrados

Se ha logrado el objetivo principal de la tesis: se ha conseguido llevar el método llamado "normalización divisiva generalizada" propuesto en [13], dentro de un contexto de series temporales a través de la metodología

clásica de los kernel smoothers. Varias variantes y generalizaciones han sido diseñadas y probadas (considerando incluso varias extensiones, como los escenarios multi-outputs y otros enfoques).
Dada la complejidad de la tarea T1 (debido a la gran extensión de la literatura relacionada), otras contribuciones, relacionadas con llevar y combinar resultados e ideas de los modelos clásicos dentro de métodos avanzados de machine learning, no se han terminado, requiriendo más tiempo y estudio (están "working in progress").

Publicaciones y participaciones a congresos

A lo largo de esta investigación, se ha conseguido publicar un artículo, así como la asistencia como ponente a un congreso internacional, que ilustran un primer esfuerzo en esta dirección. Este primer estudio, realizado en colaboración con el Dr. Valero Laparra, el Dr. Luca Martino, y Ernesto Curbelo, marca el comienzo de una trayectoria investigadora que se origina bajo una sólida formación en economía y se expande hacia la ingeniería y la ciencia de datos. El autor, compaginando su rol de investigador con responsabilidades laborales como docentes universitario a tiempo completo, ha logrado una significativa contribución en estos tres años, a pesar de las limitaciones temporales.

Además, aunque no directamente relacionado con la línea principal de esta tesis, el autor ha publicado un artículo en coautoría con el Dr. Julio Emilio Sandubete y el Dr. Juan Carlos García-Villalobos, titulado "Testing the efficient market hypothesis and the model-data paradox of chaos on top currencies from the foreign exchange market (FOREX)", publicado en enero de 2023. Este trabajo fue también presentado en el VI Congreso Iberoamericano de Jóvenes Investigadores en Ciencias Económicas y Dirección de Empresas en noviembre de 2023. Asimismo, el autor participó como ponente en el IV Congreso Iberoamericano AJICEDE en Madrid, presentando la ponencia Çonsecuencias en el Índice de Confianza Económico del análisis del sentimiento de tweets asociados a la pandemia.^{en} diciembre de 2021.

Este trabajo no solo aborda la síntesis de la economía con la ciencia de datos sino también establece un diálogo entre la teoría econométrica clásica, que incluye modelos como ARMA y GARCH, y las recientes innovaciones en redes neuronales y machine learning.

Aportes relacionados con la tesis

- Publicación del artículo en una revista internacional JCR Q1:

  Beleña, L., Curbelo, E., Martino, L., and Laparra, V., "Second-Moment/Order Approximations by Kernel Smoothers with Application to Volatility Estimation". Mathematics, 12(9), 1406, 2024.

- Participación como ponente en un congreso internacional, de los resultados presentes en la publicación anterior. Machine Learning, Information Theory, Signal Processing and Communications Workshop, 2024, Universidad Rey Juan Carlos, Madrid.

Claramente, el autor ha adquirido conocimientos profundos en las dos áreas de investigación (modelos clásicos para series temporales y métodos de machine learning), que se verán reflejados también en futuras publicaciones. Además, el autor de la tesis ha aprendido a manejar profesionalmente Python, Matlab y LaTeX.

Otros hitos durante el periodo de la tesis

- Publicación del artículo en una revista internacional JCR Q1:

  Sandubete, J. E., Beleña, L., and García-Villalobos, J. C., "Testing the efficient market hypothesis and

the model-data paradox of chaos on top currencies from the foreign exchange market (FOREX)". Mathematics, 11(2), 286, 2023.

- Presentación de dicho artículo en congreso: León Beleña, P., Hidalgo, P., Lazcano, A., & Sandubete, J. E. (24, Noviembre 2023 - Universidad de la Rioja, Logroño). Análisis de la Hipótesis de Mercados Eficientes, la Triple Hora Bruja y el Efecto del Día de Vencimiento, en las Principales Monedas del Mercado de Divisas (FOREX). VI Congreso Iberoamericano de Jóvenes Investigadores en Ciencias Económicas y Dirección de Empresas.
- Ponencia en congreso: "Hidalgo, P., Lazcano, A., and Beleña, L. (16 Diciembre 2021 - Universidad Pontificia Comillas ICADE, Madrid). Consecuencias en el Índice de Confianza Económico del análisis del sentimiento de tweets asociados a la pandemia. IV Congreso Iberoamericano de Jóvenes Investigadores en Economía y Empresa".

La expectativa futura de este trabajo inicial, es el fundamento para futuras investigaciones que seguirán ampliando los horizontes de las aplicaciones de redes neuronales (y más en general, el machine learning) en el análisis de series temporales. Esta tesis supone un punto y seguido al desarrollo de la carrera como investigador del autor, no un punto y final.

1.5 Estructura de la memoria de la tesis

En general, la primera parte de la tesis hace referencia al marco teórico, que está desglosado en un primer capítulo donde se explican los distintos métodos de modelización clásicos de series temporales. En el segundo capítulo se pone en relieve la técnica de suavizado de kernel, y para finalizar el marco teórico, se explicará en detalle las redes neuronales, sus elementos, las funciones de coste y de activación. También se incluye una explicación detallada de las redes convolucionales, recurrentes y la capa de normalización divisiva generalizada. La segunda parte de la memoria está dedicada a los experimentos realizados. Un primer capítulo sobre el artículo de referencia de esta tesis doctoral: .ªproximaciones del segundo momento/orden mediante kernel smoother con aplicación a la estimación de la volatilidad", publicado en la revista Mathematics MDPI, el 4 de mayo de 2024. A continuación se dedica otro capítulo que contiene otros experimentos y simulaciones. Para finalizar, el último capítulo está dedicado a las conclusiones y los próximos trabajos que darán lugar, a raíz de las aportaciones de esta tesis doctoral. Mas específicamente, El resto de la memoria de la tesis se estructura de la siguiente forma:

- **Capitulo 2:** En este capítulo se introducen primero los fundamentos teóricos esenciales. Se explica la modelización de series univariantes estacionarias mediante modelos autorregresivos (AR), modelos de media móvil (MA) y la combinación de ambos en modelos autorregresivos de media móvil (ARMA). También se introduce el concepto de series no estacionarias con modelos como ARIMA y ARIMAX, destacando la inclusión de métodos avanzados recientes en la literatura. Finalmente, se detallan modelos específicos para la volatilidad de las series temporales, proporcionando una comprensión más profunda de los enfoques clásicos y modernos en la modelización de series temporales financieras.
- **Capitulo 3:** En este capítulo se aborda de forma detallada los suavizadores de kernel como herramientas no paramétricas para la estimación de funciones de densidad de probabilidad y regresión. Se describe la formulación básica del suavizador de kernel, originada en los trabajos de Rosenblatt y Parzen, y su extensión a problemas de regresión por Nadaraya y Watson. También se detalla la metodología de los suavizadores de kernel, destacando la importancia de la selección del ancho de banda, para lo cual se discuten varios métodos como la validación cruzada, el método plug-in y la regla de Silverman. Además, se analizan diferentes tipos de funciones kernel, como el lineal, Gaussiano y Epanechnikov, y se concluye con una discusión sobre las métricas de evaluación utilizadas para comparar la eficacia de los estimadores.

- **Capitulo 4:** Este capítulo detalla la modelización de series temporales mediante redes neuronales, enfocándose en su arquitectura, funciones de activación, y métodos de entrenamiento. Se exploran técnicas de regularización como lasso y ridge, junto con algoritmos de optimización como Adam, para mejorar la eficacia y estabilidad del aprendizaje. Para finalizar el capítulo, se desarrolla la explicación de redes neuronales recurrentes, convolucionales, con un enfoque particular sobre la normalización divisiva y su aplicaciones.
- **Capitulo 5:** Este capítulo hace referencia al paper que tiene el mismo nombre, donde se presentan métodos para estimar la volatilidad mediante técnicas de suavizado con kernel. También se introducen dos procedimientos para la estimación de varianzas locales en datos multidimensionales y múltiples salidas.
- **Capitulo 6:** Este capítulo explora el uso de distintas arquitecturas de redes neuronales para la predicción de series temporales financieras. Se introduce de forma adicional la aplicación de la capa de la normalización divisiva generalizada (GDN) a las distintas arquitecturas como una técnica avanzada para mejorar la precisión y estabilidad de los modelos, en comparación con esas mismas arquitecturas, sin la inclusión de esta capa.
- **Capitulo 7:** Se proporciona una discusión final y conclusiones respecto al trabajo hecho.

Capítulo 2

Caracterización y modelización tradicional de series temporales

En este capítulo se va a realizar primeramente una descripción de los conceptos básicos y fundamentos teóricos esenciales de series temporales financieras. Posteriormente, se van a explicar los modelos univariantes estacionarios, donde se encuentran los modelos autorregresivos (AR), los modelos de media móvil (MA) y la combinación de ambos en los modelos autorregresivos de media móvil (ARMA). También se hará una introducción a los modelos no estacionarios, incluyendo a los modelos autorregresivos integrados de media móvil (ARIMA) y los modelos autorregresivos integrados de media móvil y variable exógena (ARIMAX). Se tratarán también métodos más avanzados que se han encontrado en la literatura. Una vez expuesto el marco teórico a los modelos clásicos de series temporales, este capítulo finalizará con una explicación más detallada de los modelos que se utilizan para modelizar la volatilidad de las series temporales.

2.1 Series temporales: notación y definiciones

Una serie temporal o serie de tiempo, es una serie de valores en la que cada uno de ellos depende del tiempo. La diferencia entre un valor temporal y el siguiente (o el anterior) se llama periodo de tiempo, mientras que la periodicidad será la frecuencia temporal con la que se registran las observaciones en el conjunto de datos. Dicha frecuencia debe de ser constante, es decir, el periodo de tiempo empleado debe de ser el mismo para todo el conjunto de datos [124]. Un ejemplo de serie temporal es la cotización del índice bursátil 'National Association of Securities Dealers Automated Quotation', Nasdaq Composite tal y como se aprecia en la Figura 2.1.

Figura 2.1: Precio de cierre de la cotización diaria del índice Americano NASDAQ

La serie temporal puede ser univariante o multivariante. Una serie temporal univariante es una serie de datos que consta de una sola variable que está medida a lo largo del tiempo. Es decir, se trata de una serie temporal que registra valores de una única variable en diferentes momentos del tiempo, sin incluir ninguna otra variable adicional.

La ecuación general de una serie temporal univariante se puede expresar como:

$$y_t = f(y_{t-1}, y_{t-2}, ..., y_{t-k}, \epsilon_t), \tag{2.1}$$

donde y_t es el valor de la serie temporal en el tiempo t, f es una función que relaciona el valor actual de la serie temporal con los valores pasados y el término de error, $y_{t-1}, y_{t-2}, ..., y_{t-k}$ son los valores pasados de la serie temporal y ϵ_t es el término de error aleatorio en el tiempo t [124]. Una serie temporal multivariante es una serie de datos que consta de dos o más variables medidas a lo largo del tiempo. Esta serie registrará valores de varias variables en diferentes momentos del tiempo, permitiendo así analizar las relaciones entre dichas variables.
La ecuación general de una serie temporal multivariante se puede expresar como:

$$y_t = f(y_{t-1}, y_{t-2}, ..., y_{t-k}, \epsilon_t), \tag{2.2}$$

donde: Y_t es el vector de valores de la serie temporal en el tiempo t, f es una función que relaciona el vector de valores actual de la serie temporal con los vectores de valores pasados y el término de error, $y_{t-1}, y_{t-2}, ..., y_{t-k}$ son los vectores de valores pasados de la serie temporal, y ϵ_t es el vector de términos de error aleatorio en el tiempo t.
Es importante tener en cuenta que las funciones f pueden variar dependiendo del contexto específico de la serie temporal, así como la longitud del retraso k y la dimensión de los vectores y_t y $y_{t-1}, y_{t-2}, ..., y_{t-k}$ [124].

- **Proceso estocástico:** Un proceso estocástico se considera que es un proceso aleatorio, ya que se mueve al azar, y por tanto, no se puede predecir.

  Es una colección de variables aleatorias $x_1, x_2, x_3, ..., x_n$ que dependen del tiempo. Cada variable aleatoria x_i representa una observación en un momento específico del tiempo, y su distribución de probabilidad puede depender tanto del tiempo como de los valores anteriores en la secuencia [28].

  Existen varios tipos de procesos estocásticos de series temporales que son comúnmente utilizados en el análisis y modelado de datos temporales. Algunos de ellos son:

  Procesos autorregresivos (AR), procesos de media móvil (MA), procesos ARMA, procesos de diferencia integrada (I), procesos de ARIMA. En los siguientes puntos de este capítulo se explicará cada uno de ellos con detalle.

  La Figura 2.2 muestra un modelo autorregresivo de orden 1, AR(1), como un caso particular de proceso estocástico donde $\phi = 0{,}8$ y $c = 5$, del siguiente modo:

$$x_t = 5 + 0{,}8x_{t-1} + \epsilon_t. \tag{2.3}$$

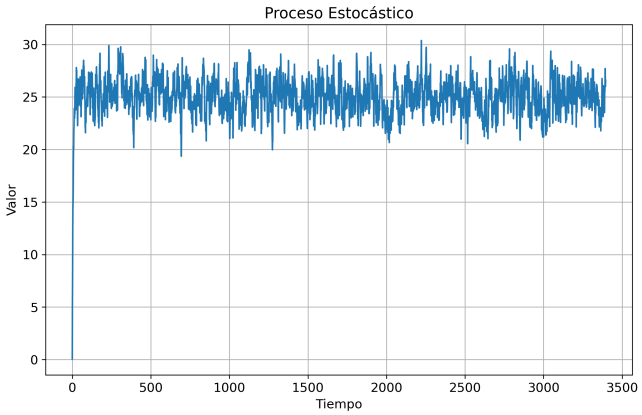


Figura 2.2: Proceso estocástico AR(1)

- **Función de autocorrelación (FAC) y función de autocorrelación parcial (FACP):** El análisis de series temporales a menudo requiere examinar la relación entre los valores de una serie en diferentes momentos del tiempo. Para este propósito, se utiliza la función de autocorrelación (FAC), la función de autocorrelación parcial (FACP) y el correlograma como herramientas para evaluar y visualizar dichas relaciones.

  La autocorrelación mide el nivel de semejanza que hay en una serie de datos consigo misma y con varios periodos atrás, también llamados "retardos" o "retrasos".

  Esto es:

  $$corr(x_t, x_{t-k}), \tag{2.4}$$

  siendo t el tiempo y k los retardos de la serie que se quiere analizar.

  La función de autocorrelación (FAC) es una medida que cuantifica el grado en que los valores de una serie temporal están correlacionados consigo mismos en diferentes retardos. Se puede representar matemáticamente como:

  $$\rho_k = \frac{\sum_{t=k+1}^{n}(y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^{n}(y_t - \bar{y})^2}, \tag{2.5}$$

  donde ρ_k es la autocorrelación en el retraso k, y_t es el valor en el tiempo t de la serie temporal, $\bar{y}$ es el promedio de la serie temporal y n es la longitud de la serie temporal.

  La función de autocorrelación se representa gráficamente mediante una función que muestra el valor de la correlación en función del número de retrasos. Los valores de autocorrelación están comprendidos entre -1 y 1 (al igual que en el coeficiente de correlación). Posteriormente se deberá de analizar si el valor de la correlación obtenido es significativo o no. Si el coeficiente de autocorrelación no es significativo, su valor será de cero.

  Una serie temporal con alta autocorrelación positiva en la FAC indica que los valores de la serie están fuertemente correlacionados con sus valores anteriores, lo que sugiere la presencia de una tendencia en la serie. Por otro lado, una serie temporal con baja autocorrelación en la FAC indica que los valores de la serie no están correlacionados con sus valores anteriores, lo que sugiere la presencia de ruido blanco en los datos.

  La función de autocorrelación parcial (FACP), recoge sólo la autocorrelación directa existente entre dos valores de la serie, mientras que la función de autocorrelación recoge los efectos tanto directos como indirectos de la relación que existe entre dos valores de la serie temporal [28].

Ambas funciones, FAC y FACP, son esenciales para entender las estructuras de dependencia en series temporales y seleccionar modelos adecuados para realizar predicciones.

El correlograma es una representación gráfica que combina las funciones FAC y FACP, facilitando la identificación de patrones y tendencias en la serie temporal. En un correlograma se representan dos gráficos, uno para la FAC y otro para la FACP, donde cada gráfico muestra barras verticales que representan el valor de la correlación para cada retardo. Las barras que superan un umbral de significancia indica que la correlación en ese retraso es estadísticamente significativa y, por lo tanto, relevante para el análisis.

En las siguientes Figuras, se puede ver la representación de la función de autocorrelación (ACF) para el precio de cierre del Nasdaq (ver Figura 2.3) y la función de autocorrelación parcial (PACF) para la misma serie temporal (ver Figura 2.4):

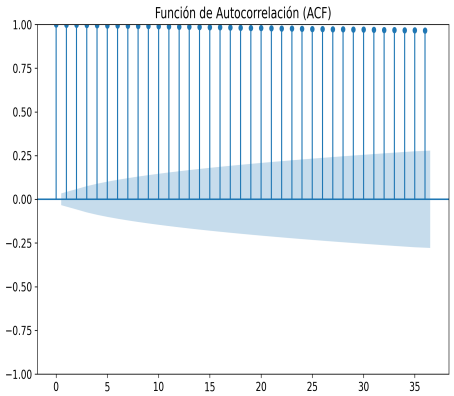


Figura 2.3: Función de autocorrelación de los precios de cierre del Nasdaq.

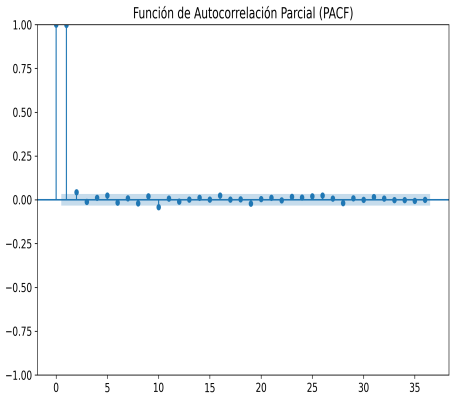


Figura 2.4: Función de autocorrelación parcial de los precios de cierre del Nasdaq.

- **Ruido blanco y paseo aleatorio:** Si los datos de la serie temporal no siguen ningún patrón, serán un proceso de ruido blanco. Debido a que no hay un patrón pasado en los datos, tampoco se podrá realizar ninguna predicción a futuro, ya que será una serie temporal compuesta por observaciones aleatorias [124]. Se suele asumir que el ruido blanco es Gaussiano y por lo tanto, sigue una distribución Gaussiana. La notación matemática para expresar el ruido blanco Gaussiano es la siguiente:

$$e_t \sim \mathcal{N}(0, \sigma^2), \tag{2.6}$$

donde e_t es el término de error aleatorio en el tiempo t, $\mathcal{B}$ indica 'Normal', 0 es la media, que se asume igual a cero, y por último, σ^2 es la varianza del ruido.
Dicha ecuación significa que el ruido blanco es un proceso estocástico en el que cada valor e_t de la serie temporal es una variable aleatoria que sigue una distribución normal con media cero (y por lo tanto, también constante) y varianza constante σ^2. Los valores del ruido blanco son independientes entre sí y no están correlacionados (ningún periodo tiene autocorrelaciones) tal y como se puede ver en la Figura 2.5.

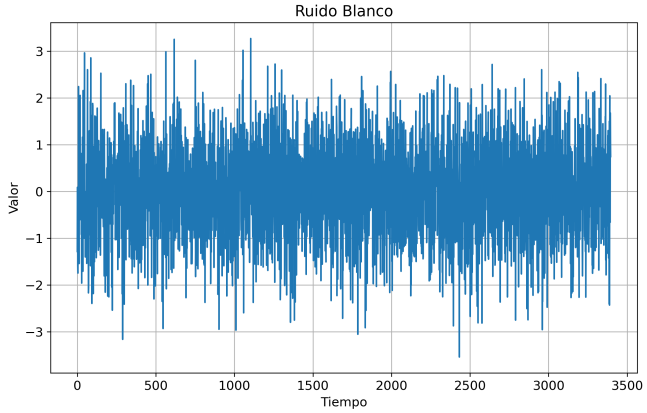


Figura 2.5: Ruido blanco

Por otro lado, el paseo aleatorio (en inglés, 'random walk') es un modelo de series temporales que describe un proceso estocástico en el que el valor de una variable cambia de manera aleatoria y sin una dirección clara. Dicho de otro modo, el valor de una variable en cualquier momento es igual al valor anterior más un término de error aleatorio o ruido blanco [124].
Formalmente, un paseo aleatorio se puede escribir como:

$$y_t = y_{t-1} + e_t, \tag{2.7}$$

donde y_t es el valor de la variable en el tiempo t, y_{t-1} es el valor de la variable en el tiempo t-1, e_t es un término de error aleatorio que sigue una distribución normal con media cero y varianza constante.

La principal característica de un paseo aleatorio, es que los cambios en el valor de la variable a lo largo del tiempo parecen ser aleatorios y no hay una tendencia clara, tal y como se puede apreciar en la Figura 2.6.

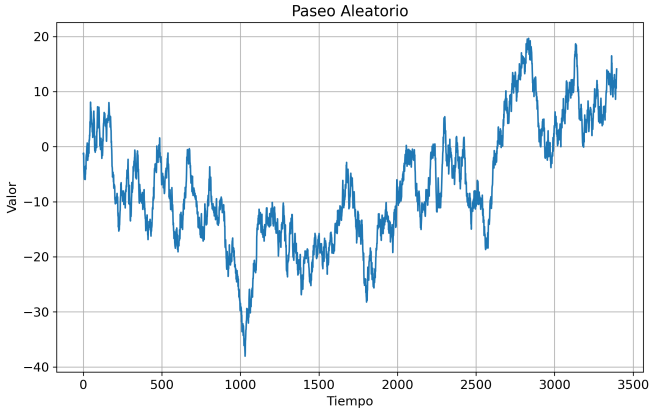


Figura 2.6: Paseo aleatorio

- **Serie temporal estacionaria:** Una serie temporal estacionaria es aquella que es estable en el tiempo, es decir, que no tiene tendencia. Se puede definir matemáticamente como aquella serie de datos que cumple con las siguientes condiciones:

  La media de la serie temporal es constante e igual para todos los momentos temporales, $E(y_t) = \mu$, y la varianza también es constante e igual para toda la serie, $\text{Var}(y_t) = \sigma^2$. Además, la covarianza entre dos valores de la serie temporal depende únicamente de la distancia temporal de los parámetros, y no del momento temporal específico: $\text{Cov}(y_t, y_{t+h}) = \text{Cov}(y_{t+k}, y_{t+k+h})$, para todo t, todos los intervalos de tiempo h y todas las distancias k.

  En términos matemáticos, una serie temporal estacionaria se puede expresar como [53]:

  $$y_t = \mu + e_t, \tag{2.8}$$

  donde y_t es el valor de la serie temporal en el tiempo t, μ es la media constante de la serie temporal, y e_t es el término de error aleatorio en el tiempo t, que sigue una distribución con media cero y varianza constante.

  Por otro lado, una serie no estacionaria será aquella que no es estable en el tiempo, ya que tendrá media no constante y por lo tanto, también tendrá tendencia.

  El test de raíces unitarias propuesto por Wayne Arthur Fuller, fue de los primeros métodos empleados para comprobar la existencia de estacionariedad en series temporales. Posteriormente Wayne Arthur Fuller desarrolló junto a David Dickey el test de Dickey-Fuller, en 1979 [53]:

  Esta prueba estadística se utiliza para determinar la presencia de raíces unitarias en una serie temporal. La existencia de dichas raíces sugiere que la serie tiene tendencia estocástica, implicando una alta correlación entre sus valores a lo largo del tiempo. Esto dificultará la predicción y el modelado de la serie.

  El test de Dickey-Fuller se estructura alrededor de dos hipótesis fundamentales:

  - **Hipótesis nula** (H_0): La serie temporal posee una raíz unitaria, lo que implica que es no estacionaria y tiene una tendencia determinista.
  - **Hipótesis alternativa** (H_1): La serie temporal no posee una raíz unitaria, es decir, la serie es estacionaria. que es estacionaria.

  La evaluación se realiza analizando la autocorrelación entre los valores de la serie en un instante dado t y su primer retraso $(t-1)$.

  Si el coeficiente de autocorrelación es significativamente distinto a cero, se rechaza la hipótesis nula y se concluye que la serie temporal es estacionaria. Si el coeficiente no es significativamente diferente de cero, no se rechaza la hipótesis nula y se concluye que la serie temporal tiene una raíz unitaria y, por lo tanto, es no estacionaria.

- **Componente estacional o estacionalidad de la serie:** Una serie temporal tendrá una componente estacional si la evolución de los datos sigue un ciclo a lo largo del tiempo. La estacionalidad ayuda a entender las fluctuaciones de la serie temporal. Para analizar de manera rigurosa la estacionalidad en una serie temporal, se va a realizar una descomposición de la serie en sus diferentes componentes [154], siendo estos:

  1. *Efecto de tendencia*: Este componente refleja la trayectoria subyacente que la serie sigue a lo largo del tiempo. La tendencia puede ser ascendente, descendente o estacionaria. En el procesamiento de señales, se le denomina *componente de baja frecuencia*.

2. *Efecto estacional*: Una serie temporal exhibirá una componente estacional si presenta fluctuaciones cíclicas o patrones que se repiten a intervalos regulares de tiempo. En el procesamiento de señales, se denomina como *componente que oscila a una frecuencia específica*, siempre mayor que cero.
3. *Efecto residual o error de predicción*: Este componente representa la diferencia entre los valores observados y los estimados por el modelo. Es un indicador de la precisión del modelo ajustado y captura los elementos aleatorios o anomalías no explicadas por la tendencia o la estacionalidad. En otras palabras, incluye todo aquello que no forma parte de la señal y es completamente aleatorio (sin ninguna dependencia entre las muestras).

Las Figuras 2.7, 2.8 y 2.9 ilustran respectivamente, los componentes de tendencia, estacionalidad y residual para un conjunto de datos de la serie temporal del Índice NASDAQ.

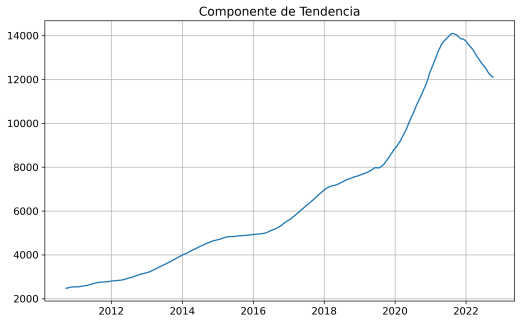


Figura 2.7: Tendencia de los precios de cierre del Nasdaq

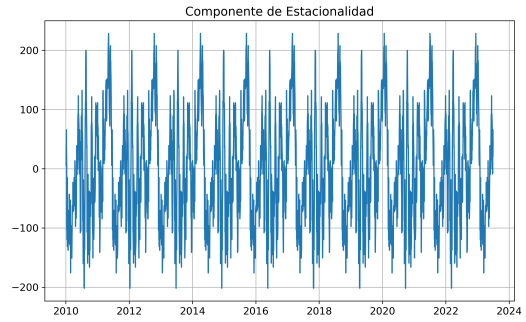


Figura 2.8: Estacionalidad de los precios de cierre del Nasdaq

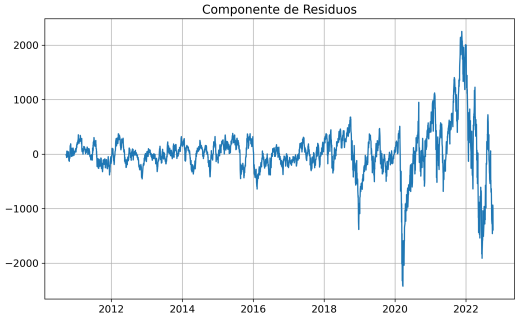


Figura 2.9: Residuos de los precios de cierre del Nasdaq

- **Modelos de suavizado: Definición y tipos:** La técnica de suavizado se utiliza para realizar predicciones de la serie temporal, eliminando fluctuaciones temporales y ruido de la serie, facilitando de este modo la identificación de patrones y tendencias.

  Los métodos de suavizado exponencial tienen el objetivo de dar pesos a los valores de la serie. Estos pesos van decreciendo de forma exponencial, de forma que dará más importancia a los valores de la serie que son más recientes.

  Dos de los enfoques más comunes para el suavizado de series temporales, son el suavizado exponencial simple y el suavizado exponencial doble (también conocido como modelo de Holt):

  El suavizado exponencial simple es adecuado para series temporales sin tendencia ni componentes estacionales. La fórmula para el suavizado exponencial simple es [78]:

  $$\hat{y}_{t+1} = \alpha y_t + (1-\alpha)\hat{y}_t,$$

  donde $\hat{y}_{t+1}$ es la previsión para el siguiente punto de tiempo $(t+1)$, y_t es el valor observado en el tiempo t, $\hat{y}_t$ es la previsión para el tiempo t, y α es el parámetro de suavizado que tendrá un valor entre 0 y 1.

  El suavizado exponencial doble se utiliza cuando hay una tendencia en los datos pero no hay estacionalidad. El modelo de Holt introduce un segundo componente de suavizado para modelar la tendencia. Las fórmulas para el suavizado exponencial doble son [78]:

  $$\begin{array}{lll} \text{Nivel:} & \ell_t & = \alpha y_t + (1-\alpha)(\ell_{t-1} + b_{t-1}), \\ \text{Tendencia:} & b_t & = \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1}, \\ \text{Previsión:} & \hat{y}_{t+1|t} & = \ell_t + b_t, \end{array}$$

  donde ℓ_t es el componente de nivel en el tiempo t, b_t es el componente de tendencia en el tiempo t, α es el parámetro de suavizado para el nivel, situando su valor entre 0 y 1, y β es el parámetro de suavizado para la tendencia, situado entre 0 y 1.
  En la Figura 2.10 se puede observar la serie temporal de la cotización de los precios de cierre del Índice bursátil Nasdaq, y la predicción para los siguientes 1000 días realizada por medio del modelo de suavizado doble de Holt-Winters.

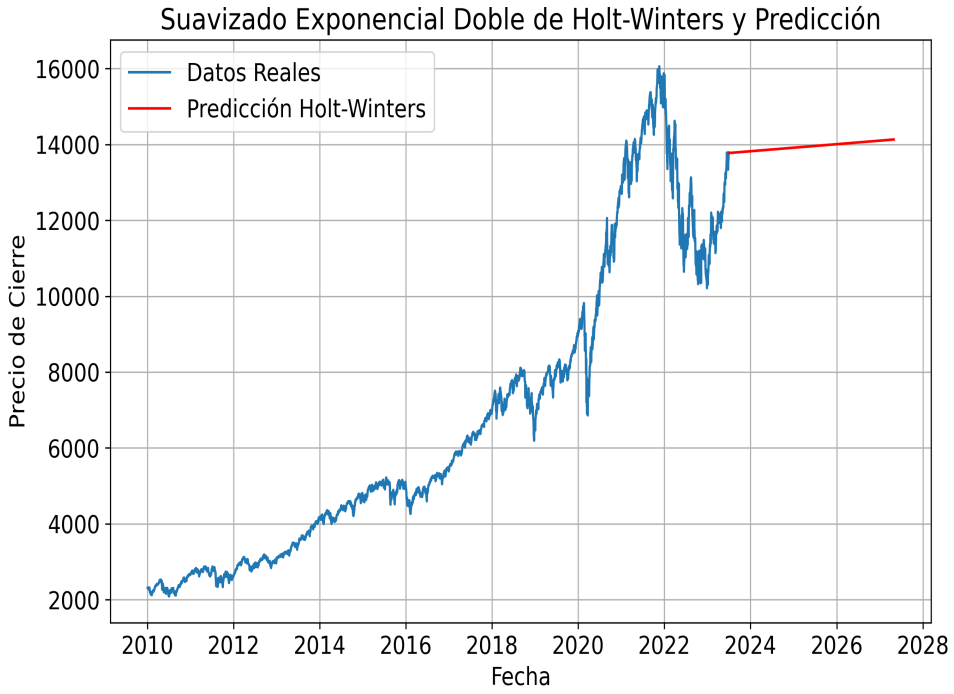


Figura 2.10: Suavizado exponencial doble de Holt-Winters aplicado al Índice Bursátil NASDAQ y predicción para los siguientes mil períodos

2.2 Modelos clásicos para la modelización de series temporales: modelos estacionarios

Debido a la gran cantidad de algoritmos que pueden dar una solución adecuada a la optimización de la serie temporal, surge la necesidad de establecer un criterio para el momento de seleccionar un algoritmo que modelice de forma correcta dicha serie, siendo habitual que se utilice el principio de parsimonia. Ello sugiere la adopción de modelos menos parametrizados en caso de que no exista una necesidad que justifique un algoritmo con alguna complejidad adicional [7].

Selección de modelo. Para determinar cuál es el modelo óptimo, se suelen emplear varios indicadores que evalúan la calidad del ajuste y la complejidad del modelo. Entre los más utilizados se encuentran el criterio de información de Akaike (AIC) [2] y el criterio de información Bayesiano (BIC) [136]. Ambos criterios buscan equilibrar el ajuste del modelo a los datos con el número de parámetros estimados, penalizando los modelos excesivamente complejos. El modelo óptimo será aquel que minimice el valor de dicho criterio [31]. Recientemente, se han propuesto métodos más avanzados dentro de la misma familia, como el criterio de información espectral (SIC) [107] y el detector automático universal de codos (UAED) [112].
Otro método para seleccionar el modelo óptimo de todos los candidatos que se han obtenido, es mediante el algoritmo de máxima verosimilitud (log-likelihood) [155]. En este caso, se prefiere el modelo que maximice este criterio.

Dentro del conjunto de modelos estacionarios, los modelos de media movil (en inglés, moving averaging - MA) y autorregresivos (AR) han mostrado ser especialmente útiles para describir la autocorrelación lineal existente en muchas series temporales.

Operador de retardos: Antes de comenzar, es necesario definir el Operador "retardo". El operador de retardos se utiliza para representar las observaciones pasadas en los componentes autorregresivos (AR) y de media móvil (MA) del modelo. Se denota por L y se aplica sobre una serie temporal para desplazarla un periodo hacia atrás en el tiempo. Dada una serie temporal y_t, el operador de retardos se define matemáticamente como:

$$L(y_t) = y_{t-1}. \tag{2.9}$$

La aplicación p-ésima del operador se denota como L^p, y tiene el efecto de retrasar la serie p periodos. Matemáticamente se define como:

$$L^p(y_t) = y_{t-p}. \tag{2.10}$$

2.2.1 Modelo de medias móviles — MA(q)

Los modelos de medias móviles surgen como una solución a las limitaciones de los modelos autorregresivos cuando se trata de modelizar y predecir procesos que son no estacionarios.
En lugar de explicar la variable de la serie en función de sus valores pasados (como harán los modelos autoregresivo), tal y como se hace con los modelos autorregresivos, los modelos de medias móviles utilizan los errores aleatorios (o residuos) de la estimación de la serie en periodos anteriores. Estos residuos son la diferencia entre el valor real y el valor predicho [124].

Por ejemplo, un modelo de medias móviles de primer orden, MA(1), consideraría el valor del residuo en el período anterior. Estos modelos son útiles para modelar la dependencia lineal entre los residuos en diferentes momentos del tiempo.

La ecuación de un modelo MA general de orden q es:

$$y_t = \mu + \epsilon_t - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2} - \cdots - \theta_q\epsilon_{t-q}, \tag{2.11}$$

donde y_t es la observación actual en el tiempo t, μ es una constante que representa el valor medio de la serie temporal, ϵ_t es el error aleatorio en el tiempo t. Por otro lado, $\theta_1, \theta_2, ..., \theta_q$ son los coeficientes de media móvil que miden la relación entre los errores aleatorios pasados y el actual, por ejemplo, θ_1 mide la relación entre el error aleatorio del periodo anterior (ϵ_{t-1}) y el error aleatorio actual (ϵ_t), θ_2 mide la relación entre el error aleatorio dos periodos atrás (ϵ_{t-2}) y el error aleatorio actual (ϵ_t), y así sucesivamente hasta θ_q.

El modelo de medias móviles de orden uno, MA(1), se define como:

$$y_t = \mu + \epsilon_t - \theta_1\epsilon_{t-1}, \tag{2.12}$$

donde y_t es la observación actual en el tiempo t, μ es una constante que representa el valor medio de la serie temporal, ϵ_t es el error aleatorio en el tiempo t, y θ_1 es el coeficiente de media móvil que mide la relación entre el error aleatorio inmediatamente anterior (ϵ_{t-1}) y el error aleatorio actual (ϵ_t).

Cálculo de la Función de Autocorrelación Simple (FAS) para un Modelo MA(1)

A continuación, se va a proceder a calcular la función de autocorrelación simple (FAS) para el modelo de media móvil de orden 1 (MA(1)). Este cálculo está formado por los siguientes pasos: primero, se ha calculado la esperanza matemática del proceso de media móvil con un retardo, MA(1). A continuación, se ha derivado la expresión de la autocovarianza en función del retraso k, se ha hallado la varianza del proceso, y finalmente, se ha calculado la función de autocorrelación simple (FAS) y particularizado para $k = 1$.

Paso 1: Esperanza matemática del proceso MA(1)

El modelo de media móvil de orden 1, MA(1), se define como:

$$y_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1}. \tag{2.13}$$

La esperanza matemática $E[y_t]$ es:

$$E[y_t] = E[\mu + \epsilon_t - \theta_1 \epsilon_{t-1}], \tag{2.14}$$

$$E[y_t] = \mu + E[\epsilon_t] - \theta_1 E[\epsilon_{t-1}]. \tag{2.15}$$

Dado que $E[\epsilon_t] = 0$ y $E[\epsilon_{t-1}] = 0$, la media $E[y_t]$ es constante en el tiempo:

$$E[y_t] = \mu. \tag{2.16}$$

Paso 2: Autocovarianza en función del Retraso k

La autocovarianza $\gamma(k)$ se define como:

$$\gamma(k) = \mathsf{Cov}(y_t, y_{t-k}) = E[(y_t - E[y_t])(y_{t-k} - E[y_{t-k}])]. \tag{2.17}$$

Para $k = 0$:

$$\gamma(0) = E[(y_t - \mu)^2], \tag{2.18}$$

sustituyendo $y_t - \mu$ usando la ecuación $2{,}13$:

$$\gamma(0) = E[(\epsilon_t - \theta_1 \epsilon_{t-1})^2], \tag{2.19}$$

expandiendo el cuadrado se obtiene:

$$\gamma(0) = E[\epsilon_t^2 - 2\theta_1 \epsilon_t \epsilon_{t-1} + \theta_1^2 \epsilon_{t-1}^2], \tag{2.20}$$

y dado que $E[\epsilon_t \epsilon_{t-1}] = 0$ debido a que el ruido blanco es no correlacionado, y $E[\epsilon_t^2] = E[\epsilon_{t-1}^2] = \sigma^2$, entonces:

$$\gamma(0) = \sigma^2 + \theta_1^2 \sigma^2 = \sigma^2(1 + \theta_1^2). \tag{2.21}$$

Para $k = 1$:

$$\gamma(1) = E[(y_t - \mu)(y_{t-1} - \mu)], \tag{2.22}$$

sustituyendo y_t y y_{t-1} se obtiene:

$$\gamma(1) = E[(\epsilon_t - \theta_1 \epsilon_{t-1})(\epsilon_{t-1} - \theta_1 \epsilon_{t-2})], \tag{2.23}$$

expandiendo el producto se llega a:

$$\gamma(1) = E[\epsilon_t \epsilon_{t-1} - \theta_1 \epsilon_{t-1}^2 - \theta_1 \epsilon_t \epsilon_{t-2} + \theta_1^2 \epsilon_{t-1} \epsilon_{t-2}], \tag{2.24}$$

dado que ϵ_t, ϵ_{t-1}, y ϵ_{t-2} son independientes y $E[\epsilon_t^2] = \sigma^2$:

$$\gamma(1) = -\theta_1 E[\epsilon_{t-1}^2] = -\theta_1 \sigma^2. \tag{2.25}$$

Para $k > 1$:

$$\gamma(k) = E[(y_t - \mu)(y_{t-k} - \mu)], \tag{2.26}$$

sustituyendo y_t y y_{t-k} se tiene:

$$\gamma(k) = E[(\epsilon_t - \theta_1 \epsilon_{t-1})(\epsilon_{t-k} - \theta_1 \epsilon_{t-k-1})], \tag{2.27}$$

pero debido a que el ruido blanco es no correlacionado, todos los términos que implican productos de errores en diferentes tiempos son cero. Entonces:

$$\gamma(k) = 0, \quad \text{para } k > 1. \tag{2.28}$$

Paso 3: Función de Autocorrelación Simple (FAS)

La función de autocorrelación simple (FAS) $\rho(k)$ se define como:

$$\rho(k) = \frac{\gamma(k)}{\gamma(0)}. \tag{2.29}$$

Para $k = 0$:

$$\rho(0) = \frac{\gamma(0)}{\gamma(0)} = 1. \tag{2.30}$$

Para $k = 1$:

$$\rho(1) = \frac{\gamma(1)}{\gamma(0)} = -\frac{\theta_1 \sigma^2}{\sigma^2(1 + \theta_1^2)} = -\frac{\theta_1}{1 + \theta_1^2}. \tag{2.31}$$

Generalizando para cualquier k:

$$\rho(k) = 0, \quad \text{para } k > 1. \tag{2.32}$$

Resumen y Conclusión

Por lo tanto, la función de autocorrelación simple (FAS) para un modelo MA(1) se expresa como:

$$\rho(k) = \begin{cases} 1 & \text{si } k = 0, \\ -\frac{\theta_1}{1+\theta_1^2} & \text{si } k = 1, \\ 0 & \text{si } k > 1. \end{cases} \tag{2.33}$$

Modelos de medias móviles de orden 2 (MA(2))

El modelo de medias móviles de orden dos, MA(2), se define como:

$$y_t = \mu + \epsilon_t - \theta_1 \epsilon_{t-1} - \theta_2 \epsilon_{t-2}, \tag{2.34}$$

donde: y_t es la observación actual en el tiempo t, μ es una constante que representa el intercepto o el valor medio de la serie temporal, ϵ_t es el error aleatorio en el tiempo t, θ_1 es el coeficiente de media móvil que mide la relación entre el error aleatorio inmediatamente anterior (ϵ_{t-1}) y el error aleatorio actual (ϵ_t), y θ_2 es el coeficiente de media móvil que mide la relación entre el error aleatorio dos veces antes (ϵ_{t-2}) y el error aleatorio actual (ϵ_t).

2.2.2 Modelo autorregresivo de order p — AR(p)

Los modelos autorregresivos fueron creados por George Udny Yule en su obra *'VII. On a method of investigating periodicities disturbed series, with special reference to Wolfer's sunspot numbers* en 1927 [162], aunque fueron popularizados por George Box y Gwilym Jenkins en *'Time Series Analysis: Forecasting and Control'* en 1970 [28]. Son procesos en los que cada valor en la serie temporal se explica como una combinación lineal de los valores anteriores de dicha serie más un término de error aleatorio. Estos modelos son útiles para modelar la dependencia lineal entre los valores en diferentes momentos del tiempo [124].

El número de observaciones pasadas utilizadas para predecir la observación actual, se conoce como 'orden' y se denota con p. La ecuación del modelo generalizado queda del siguiente modo:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t, \tag{2.35}$$

donde y_t es la observación actual en el tiempo t, c es una constante que representa el intercepto o el valor medio de la serie temporal, $\phi_1, \phi_2, ..., \phi_p$ son los coeficientes autorregresivos que miden la relación entre las observaciones pasadas y actuales. El coeficiente ϕ_1 mide la relación entre la observación inmediatamente anterior (y_{t-1}) y la observación actual (y_t), el coeficiente ϕ_2 mide la relación entre la observación dos veces antes (y_{t-2}) y la observación actual (y_t), y así sucesivamente hasta el coeficiente ϕ_p, que mide la relación entre la observación p veces antes (y_{t-p}) y la observación actual (y_t). Por último, ϵ_t, es el error aleatorio en el tiempo t que no se puede explicar mediante las observaciones pasadas y los coeficientes autorregresivos.
En esta sección, se presentan los modelos AR(1) y AR(2) como casos específicos del modelo autorregresivo genérico AR(p).

AR(1). El modelo AR(1) es un caso particular del modelo autorregresivo genérico AR(p) con orden $p = 1$, siendo su ecuación la siguiente:

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t, \tag{2.36}$$

donde y_t es la observación actual en el tiempo t, c es una constante que representa el intercepto o el valor medio de la serie temporal, ϕ_1 es el coeficiente autorregresivo que mide la relación entre la observación inmediatamente anterior (y_{t-1}) y la observación actual (y_t). Los valores entre los que se encuentra ϕ_1 para que el proceso sea estacionario son -1 y 1. El error aleatorio (o innovación) en el tiempo t es ϵ_t: no se puede explicar mediante las observaciones pasadas y los coeficientes autorregresivos.

Cálculo de la Función de Autocorrelación Simple (FAS) para un Modelo AR(1)

A continuación se va a proceder a calcular la función de autocorrelación simple (FAS) para el modelo autorregresivo de orden 1 [124], AR(1). Este cálculo está formado por los siguientes pasos: primero se ha calculado la esperanza matemática del proceso autorregresivo con un retardo, AR(1). A continuación se ha derivado la expresión de la autocovarianza en función del retraso k, se ha hayado la varianza del proceso, y finalmente, se ha calculado la función de autocorrelación simple (FAS) y particularizado para retardo $k = 1$.

Paso 1: Esperanza matemática del proceso AR(1)

El modelo autorregresivo de orden 1, AR(1), se define como:

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t. \tag{2.37}$$

La esperanza matemática $E[y_t]$ es:

$$E[y_t] = E[c + \phi_1 y_{t-1} + \epsilon_t], \tag{2.38}$$

y usando las propiedades de la esperanza y considerando que $E[\epsilon_t] = 0$, se obtiene

$$E[y_t] = c + \phi_1 E[y_{t-1}]. \tag{2.39}$$

Dado que $E[\epsilon_t] = 0$ y que el proceso es estacionario, la media $E[y_t]$ es constante en el tiempo:

$$\mu = c + \phi_1 \mu, \tag{2.40}$$

y resolviendo para μ:

$$\mu(1 - \phi_1) = c \implies \mu = \frac{c}{1 - \phi_1}. \tag{2.41}$$

Paso 2: Autocovarianza en función del retardo k

La autocovarianza $\gamma(k)$ se define como:

$$\gamma(k) = \text{Cov}(y_t, y_{t-k}) = E[(y_t - \mu)(y_{t-k} - \mu)]. \tag{2.42}$$

Para un proceso AR(1), se parte de la ecuación $2{,}37$:

$$y_t - \mu = \phi_1(y_{t-1} - \mu) + \epsilon_t, \tag{2.43}$$

y para $k = 1$, se sustituye $2{,}43$ en $2{,}42$:

$$\gamma(1) = E[(\phi_1(y_{t-1} - \mu) + \epsilon_t)(y_{t-1} - \mu)]. \tag{2.44}$$

Expandiendo el producto se obtiene:

$$\gamma(1) = \phi_1 E[(y_{t-1} - \mu)^2] + E[\epsilon_t(y_{t-1} - \mu)], \tag{2.45}$$

y dado que ϵ_t es independiente de y_{t-1} y $E[\epsilon_t] = 0$, se tiene:

$$E[\epsilon_t(y_{t-1} - \mu)] = 0. \tag{2.46}$$

Por lo tanto:

$$\gamma(1) = \phi_1 E[(y_{t-1} - \mu)^2] = \phi_1 \gamma(0), \tag{2.47}$$

para $k > 1$, se aplica el mismo razonamiento iterativamente:

$$\gamma(k) = \phi_1 \gamma(k - 1). \tag{2.48}$$

De este modo, se puede escribir la relación general:

$$\gamma(k) = \phi_1^k \gamma(0). \tag{2.49}$$

Paso 3: Varianza del proceso: el valor $\gamma(0)$

El valor $\gamma(0) = \text{Var}(y_t)$ representa la varianza del proceso AR, es decir:

$$\gamma(0) = E[(y_t - \mu)^2], \tag{2.50}$$

y sustituyendo $y_t - \mu$ usando la ecuación $2{,}43$:

$$\gamma(0) = E[(\phi_1(y_{t-1} - \mu) + \epsilon_t)^2]. \tag{2.51}$$

Si se expande el cuadrado:

$$\gamma(0) = \phi_1^2 E[(y_{t-1} - \mu)^2] + 2\phi_1 E[(y_{t-1} - \mu)\epsilon_t] + E[\epsilon_t^2]. \tag{2.52}$$

Dado que ϵ_t es independiente de y_{t-1}:

$$E[(y_{t-1} - \mu)\epsilon_t] = 0, \tag{2.53}$$

y $E[\epsilon_t^2] = \sigma^2$:

$$\gamma(0) = \phi_1^2 \gamma(0) + \sigma^2, \tag{2.54}$$

se puede resolver para $\gamma(0)$:

$$\gamma(0)(1 - \phi_1^2) = \sigma^2 \implies \gamma(0) = \frac{\sigma^2}{1 - \phi_1^2}. \tag{2.55}$$

En los pasos anteriores, se ha utilizado la propiedad de estacionariedad del proceso AR(1). Esto implica que la varianza es constante en el tiempo, es decir, $\text{Var}(y_t) = \text{Var}(y_{t-1}) = \gamma(0)$. Por tanto, se cumple que:

$$E[(y_t - \mu)^2] = E[(y_{t-1} - \mu)^2] = \gamma(0).$$

El subíndice t en el desarrollo se utiliza para denotar la variable en el tiempo actual, pero debido a la estacionariedad, las propiedades estadísticas (como son la media y la varianza) no cambian con el paso del tiempo. Por ello, t, es importante para indicar la relación temporal, especialmente en los cálculos de covarianza.

Paso 4: Función de autocorrelación simple (FAS)

La función de autocorrelación simple (FAS) $\rho(k)$ se define como:

$$\rho(k) = \frac{\gamma(k)}{\gamma(0)}. \tag{2.56}$$

Para $k = 1$:

$$\rho(1) = \frac{\gamma(1)}{\gamma(0)} = \frac{\phi_1 \gamma(0)}{\gamma(0)} = \phi_1. \tag{2.57}$$

Generalizando para cualquier k:

$$\rho(k) = \frac{\gamma(k)}{\gamma(0)} = \frac{\phi_1^k \gamma(0)}{\gamma(0)} = \phi_1^k. \tag{2.58}$$

Resumen y Conclusión

Por lo tanto, la función de autocorrelación simple (FAS) para un modelo AR(1) se expresa como:

$$\rho(k) = \phi_1^k, \tag{2.59}$$

y para $k = 1$, se obtiene:

$$\rho(1) = \phi_1. \tag{2.60}$$

Modelos autorregresivos de orden 2 — AR(2)

El modelo AR(2) es otro caso particular del modelo autorregresivo genérico AR(p) con orden $p = 2$. La ecuación del modelo AR(2) es la siguiente:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \epsilon_t. \tag{2.61}$$

En este caso, además del coeficiente ϕ_1 que mide la relación entre la observación inmediatamente anterior (y_{t-1}) y la observación actual (y_t), se introduce un nuevo coeficiente autorregresivo, ϕ_2, que mide la relación entre la observación dos veces anterior (y_{t-2}) y la observación actual (y_t). El resto de los términos se mantienen iguales respecto al modelo AR(1).

Transformación de modelos autorregresivos en modelos de media móvil

La inversión de un modelo autorregresivos, consiste en transformarlo en su correspondiente modelo de media móvil. Para un modelo AR(1), se parte de:

$$y_t = \phi_1 y_{t-1} + \epsilon_t, \tag{2.62}$$

o se reescribe como:

$$y_t - \phi_1 y_{t-1} = \epsilon_t. \tag{2.63}$$

Utilizando el operador retardo, se obtiene $y_t - \phi_1 L y_t = \epsilon_t$, y sacando factor común y_t, se obtiene:

$$(1 - \phi_1 L) y_t = \epsilon_t, \tag{2.64}$$

despejando y_t, se obtiene:

$$\begin{aligned} y_t &= (1 - \phi_1 L)^{-1} \epsilon_t, \\ &= (1 + \phi_1 L + \phi_1^2 L^2 + \phi_1^3 L^3 + \phi_1^4 L^4 + \dots) \epsilon_t, \\ &= \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_1^2 \epsilon_{t-2} + \phi_1^3 \epsilon_{t-3} + \phi_1^4 \epsilon_{t-4} + \dots . \end{aligned} \tag{2.65}$$

Por lo tanto, el modelo AR(1) estacionario se convierte en un modelo de medias móviles de orden infinito MA(∞). La condición de invertibilidad en modelos autorregresivos con un número finito de términos se satisface automáticamente [124].

Transformación de modelos de media móvil a modelos autorregresivos

Antes de nada, se necesita que el modelo MA sea invertible. Para que un modelo MA sea invertible, es necesario que las raíces del polinomio característico en términos de su módulo, sean inferiores a uno. A fin de determinar si el modelo puede invertirse, se procede a calcular las raíces del polinomio característico. Esto se logra igualando a cero la parte correspondiente a las medias móviles del modelo. Por simplicidad, se va a considerar un modelo MA(1). En este caso, se parte de:

$$\epsilon_t - \theta_1 \epsilon_{t-1} = 0, \tag{2.66}$$

donde se sustituye ϵ_t por λ^t, y se obtiene la siguiente ecuación:

$$\lambda^t - \theta_1 \lambda^{t-1} = 0, \tag{2.67}$$

y dividiendo por λ^{t-1} se obtiene:

$$\lambda - \theta_1 = 0. \tag{2.68}$$

La solución de la ecuación o raíz del monomio es:

$$\lambda_1 = \theta_1. \tag{2.69}$$

Dado que la raíz del monomio es real, es necesario que el valor absoluto de dicha raíz sea inferior a uno:

$$|\lambda_1| < 1 \quad \text{o bien} \quad |\theta_1| < 1. \tag{2.70}$$

Entonces, si el valor absoluto de θ_1 es menor que uno, el modelo $y_t = \epsilon_t - \theta_1\epsilon_{t-1}$ puede invertirse. Al invertir un modelo MA(1), es decir, al convertirlo en su correspondiente modelo AR, se obtiene el siguiente resultado:

$$y_t = \epsilon_t - \theta_1\epsilon_{t-1}. \tag{2.71}$$

Al emplear el operador de retardo, $y_t = \epsilon_t - \theta_1 L\epsilon_t$ y extrayendo el factor común ϵ_t, se obtiene:

$$y_t = (1 - \theta_1 L)\epsilon_t, \tag{2.72}$$

y despejando ϵ_t, se puede obtener:

$$\begin{aligned} \epsilon_t &= (1 - \theta_1 L)^{-1} y_t, \\ &= (1 + \theta_1 L + \theta_1^2 L^2 + \theta_1^3 L^3 + \theta_1^4 L^4 + \dots) y_t, \\ &= y_t + \theta_1 y_{t-1} + \theta_1^2 y_{t-2} + \theta_1^3 y_{t-3} + \theta_1^4 y_{t-4} + \dots \end{aligned} \tag{2.73}$$

Por lo tanto, el modelo MA(1) invertible se puede expresar como un modelo autorregresivo de orden infinito AR(∞) (pero con un solo polo multiple - y estable - a θ_1). La condición para que un modelo de medias móviles sea invertible exige que las raíces del polinomio característico, en términos de su módulo, sean inferiores a uno. A continuación sa van a explicar los modelos ARMA, donde se combinan los modelos autorregresivos y de media móvil, detallados anteriormente (de hecho, ARMA = AR + MA).

2.2.3 Modelo autorregresivo de media móvil — ARMA(p, q)

El modelo ARMA, es una combinación de los modelos autorregresivos AR(p) y los modelos de media móvil MA(q), donde dichos coeficientes coeficientes autorregresivos y de media móvil se combinan para explicar las relaciones entre las observaciones pasadas y los errores aleatorios pasados [28]. Este modelo es especialmente adecuado para capturar diversas dinámicas presentes en datos de series temporales, como tendencias, ciclos y estacionalidades. Además, se utiliza para predecir valores futuros en la serie temporal y para identificar la importancia relativa de las observaciones y los errores aleatorios pasados en la serie temporal.

Un modelo ARMA(p, q) se puede definir de la siguiente forma [124]:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}, \tag{2.74}$$

donde y_t es la observación actual en el tiempo t, c es una constante que representa el intercepto o el valor medio de la serie temporal, $\phi_1, \phi_2, ..., \phi_p$ son los coeficientes autorregresivos que miden la relación entre las observaciones pasadas y actuales, ϵ_t es el error aleatorio en el tiempo t, y $\theta_1, \theta_2, ..., \theta_q$ son los coeficientes de media móvil que miden la relación entre los errores aleatorios pasados y actuales.
Si se particulariza a un modelo ARMA(1, 1) se obtiene un modelo que incluye una única regresión autorregresiva

(AR) y un único término de media móvil (MA). La fórmula general de un modelo ARMA(p, q) es la siguiente [124]:

$$y_t = c + \phi_1 y_{t-1} + \epsilon_t + \theta_1 \epsilon_{t-1}, \tag{2.75}$$

donde c representa el término constante del modelo. Puede interpretarse como la media del proceso si el modelo es estacionario. El término autorregresivo ($\phi_1 y_{t-1}$) captura la dependencia lineal entre el valor actual de la serie y su valor en el periodo anterior. Está compuesto a su vez por ϕ_1, que es el coeficiente autorregresivo de orden 1, y y_{t-1}, que es el valor de la serie temporal en el periodo anterior. El error actual o ruido blanco en el tiempo t, sigue siendo ϵ_t. Por último, el término de media móvil ($\theta_1 \epsilon_{t-1}$), que captura la dependencia lineal entre el valor actual de la serie y el error en el periodo anterior, está formado por el coeficiente de media móvil de orden 1 θ_1, y el término de error en el periodo anterior ϵ_{t-1}.
La estimación de los parámetros en modelos ARMA, se realiza mediante el método de máxima verosimilitud. Una vez estimado el modelo, se aplican diversas métricas y pruebas estadísticas, como el criterio de Akaike (AIC) o el criterio Bayesiano de Schwarz (BIC) o el spectral information criterion (SIC) [107], para evaluar la calidad del ajuste y comparar diferentes configuraciones del modelo.

2.3 Modelos no estacionarios

Operador de diferencias: El operador de diferencia se utiliza para transformar una serie temporal no estacionaria en una serie estacionaria. La estacionariedad es una propiedad importante en series temporales, ya que facilita el modelado y la predicción de la misma. El operador de diferencia se denota por ∇ y se define como la diferencia entre observaciones consecutivas en una serie temporal. Para una serie temporal y_t, la primera diferencia se define como:

$$\nabla y_t = y_t - y_{t-1}. \tag{2.76}$$

Si se necesita aplicar el operador de diferencia d veces, se denota como

$$\nabla^d y_t = y_t - y_{t-d}, \tag{2.77}$$

y se define de manera recursiva como:

$$\nabla^d y_t = \nabla^{d-1}(y_t - y_{t-1}). \tag{2.78}$$

El orden de diferenciación d en un modelo ARIMA(p, d, q) indica cuántas veces se aplica el operador de diferencia a la serie original.

2.3.1 Modelo autorregresivo integrado de media móvil — ARIMA(p, d, q)

Este modelo se utiliza para modelar series temporales que presentan una tendencia temporal y/o estacionalidad, es decir, cuando el modelo no es estacionario. Añade como novedad respecto el modelo autorregresivo de media móvil (ARMA) la parte integrada I. Esta parte integrada indica cuántas diferencias no estacionales se deben de tomar para convertir la serie en estacionaria. En estos procesos se toman las diferencias entre los valores consecutivos de la serie temporal para eliminar la tendencia temporal [29]. La ecuación genérica del modelo ARIMA(p, d, q), es la siguiente:

$$\nabla^d y_t = c + \phi_1 \nabla^d y_{t-1} + \phi_2 \nabla^d y_{t-2} + \cdots + \phi_p \nabla^d y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}, \tag{2.79}$$

donde p representa los componentes de la parte autorregresiva del modelo, d representa el orden de integración. Es el número de veces que se debe integrar la serie de tiempo para que esta sea estacionaria y q representa

los componentes de la parte de medias móviles. $\nabla^d y_t$ es la serie temporal y_t después de aplicar el operador de diferenciación de orden 'd'. La diferenciación se utiliza para hacer que la serie sea estacionaria, lo cual facilita el modelado y la predicción de la serie temporal. El término independiente c, es una constante que representa la intercepción en el modelo, $\phi_1 \nabla^d Y_{t-1} + \phi_2 \nabla^d Y_{t-2} + \cdots + \phi_p \nabla^d Y_{t-p}$ hacen referencia a la parte autorregresiva (AR) del modelo ARIMA, que captura la dependencia entre una observación y un número específico de observaciones anteriores (retrasos). Los coeficientes $\phi_1, \phi_2, ..., \phi_p$ representan los pesos de las observaciones anteriores en el modelo (el término 'p' del subíndice, indica el orden del componente autorregresivo), mientras que ϵ_t es el término de error o ruido en el tiempo t que se asume como ruido blanco. Por otro lado, $\theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q}$ es la parte de media móvil (MA) del modelo ARIMA, que captura la dependencia entre una observación y un residuo de un promedio móvil aplicado a observaciones anteriores. Los coeficientes $\theta_1, \theta_2, \cdots, \theta_q$ representan los pesos de los errores anteriores en el modelo.

Se considere ahora el operador de retardos L y el operador de diferencias. La transición entre estos dos operadores se materializa cuando se busca construir un modelo temporal que sea lo más representativo posible de una serie dada. La combinación de ambos operadores a menudo se emplea en modelos ARIMA, donde L captura la autocorrelación en la serie y ∇ la convierte en una serie estacionaria. De este modo, el modelo ARIMA puede representar de forma más precisa a series que exhiben tanto autocorrelación como no estacionariedad. La combinación de los operadores de retardos y de diferencias en el modelo ARIMA ayuda a capturar tanto la autocorrelación como la no estacionariedad de las series temporales.

Se puede particularizar a un modelo ARIMA(1, 1, 1) donde $p = 1$, $d = 1$ y $q = 1$. Este modelo incluye una única regresión autorregresiva (AR), una diferenciación para hacer la serie estacionaria y un único término de media móvil (MA). Su fórmula queda del siguiente modo:

$$\nabla y_t = c + \phi_1 \nabla y_{t-1} + \epsilon_t + \theta_1 \epsilon_{t-1}, \tag{2.80}$$

donde $\nabla y_t = y_t - y_{t-1}$ representa la serie temporal diferenciada una vez. El término constante del modelo, se representa con (c) , que puede interpretarse como la media del proceso si el modelo es estacionario. El término autorregresivo $(\phi_1 \nabla y_{t-1})$ captura la dependencia lineal entre el valor actual de la serie diferenciada y su valor en el periodo anterior. Está compuesto a su vez por ϕ_1, que es el coeficiente autorregresivo de orden 1, y ∇y_{t-1}, que es el valor de la serie temporal diferenciada en el periodo anterior. El error actual o ruido blanco en el tiempo t, sigue siendo ϵ_t. Por último, el término de media móvil $(\theta_1 \epsilon_{t-1})$, que captura la dependencia lineal entre el valor actual de la serie y el error en el periodo anterior, está formado por el coeficiente de media móvil de orden 1 θ_1, y el término de error en el periodo anterior ϵ_{t-1}.

Sin embargo, la correcta identificación y estimación de estos modelos, requiere de un enfoque sistemático y riguroso. Es aquí donde la metodología Box-Jenkins [28]establece un procedimiento estructurado para la identificación, estimación, diagnóstico y predicción para los modelos ARIMA, asegurando de este modo que el modelo seleccionado sea el más adecuado para la serie temporal en cuestión. A continuación, se describe detalladamente cada una de las fases de la metodología Box-Jenkins, que permiten optimizar el ajuste y la capacidad predictiva del modelo ARIMA.

Metodología Box-Jenkins: La metodología de Box-Jenkins, trata de alcanzar el mejor ajuste del modelo autorregresivo integrado de media móvil (ARIMA) con el fin de tener unas predicciones más precisas respecto al resto de modelos candidatos. Fue nombrada así por sus creadores George Edward Pelham Box y Gwilym Jenkins en la década de 1970 y se ha constituido como un referente en la disciplina de las series temporales [144]. Este enfoque se descompone en cuatro fases, descritas a continuación [28]:

- **Primera fase: Identificación del modelo ARIMA.** La identificación del modelo ARIMA implica dos subprocesos fundamentales. Primero, la serie temporal debe ser transformada para alcanzar estacionariedad, utilizando métodos como la diferenciación o las transformaciones logarítmicas.

Una vez que la serie es estacionaria, se determinan los órdenes p y q para la serie del modelo ARMA correspondiente, utilizando las funciones de autocorrelación (ACF) y autocorrelación parcial (PACF).

- **Segunda fase: Estimación de parámetros.** Con los órdenes p y q establecidos, se procede a la estimación de los coeficientes del modelo mediante el método de máxima verosimilitud, que es el más recurrente para este tipo de modelos.
- **Tercera fase: Diagnóstico de los residuos.** En esta etapa, se evalúa si los residuos del modelo siguen una distribución de ruido blanco mediante pruebas estadísticas como el test de Ljung-Box. Si los residuos no cumplen con esta condición, se revisa y ajusta el modelo, lo cual podría incluir una revisión de los órdenes p, d y q o la aplicación de transformaciones adicionales. Si los residuos tienen estructura de ruido blanco, serán independientes entre si.
- **Cuarta fase: Predicción.** Una vez que se ha obtenido el modelo ARIMA adecuado, se puede utilizar para realizar predicciones de dicha serie temporal. Estas predicciones pueden ser puntuales o en intervalos.

Modelos Autorregresivos Integrados de Media Móvil con variable exógena — ARIMAX(p, d, q)

Los modelos ARIMAX amplían los modelos ARIMA al incorporar una o más variables exógenas, que podrían influir en la serie temporal en cuestión. Estas variables exógenas, se representan generalmente por $X_{t,i}$ [156].

La ecuación general del modelo ARIMAX(p, d, q) con k variables exógenas se puede representar de la siguiente forma:

$$\nabla^d y_t = c + \sum_{i=1}^{p} \phi_i \nabla^d y_{t-i} + \sum_{j=1}^{k} \beta_j x_{t,j} + \epsilon_t + \sum_{l=1}^{q} \theta_l \epsilon_{t-l} \tag{2.81}$$

donde y_t es el valor de la serie temporal en el tiempo t, p es el orden del componente autorregresivo (AR), d es el grado de diferenciación necesario para hacer la serie temporal estacionaria, q es el orden del componente de media móvil (MA), y k es el número de variables exógenas. Los coeficientes ϕ_i corresponden al componente autorregresivo y ajustan la influencia de los p valores previos de la serie temporal diferenciada, $\nabla^d y_{t-i}$. Los coeficientes β_j hacen referencia a las variables exógenas $x_{t,j}$, que son factores externos que afectan a la serie temporal en el tiempo t. El componente ϵ_t representa el error aleatorio en el tiempo t, θ_l son los coeficientes del componente de media móvil que modelan la relación entre el error actual y los q errores de observaciones pasadas. Por último, c es una constante que representa el término de intercepto en el modelo. La inclusión de variables exógenas en un modelo ARIMA es especialmente útil cuando se busca capturar el efecto de factores externos que no están reflejados en la serie temporal misma, lo cual puede dar lugar a predicciones más robustas y precisas [8].

Si se particulariza al modelo ARIMAX(1,1,1), se obtiene lo siguiente:

$$\nabla y_t = c + \phi_1 \nabla y_{t-1} + \beta_1 x_{t,1} + \epsilon_t + \theta_1 \epsilon_{t-1}, \tag{2.82}$$

donde $\nabla y_t = y_t - y_{t-1}$ representa la serie temporal diferenciada una vez. El término constante del modelo, representado por (c). Nuevamente, puede interpretarse como la media del proceso si el modelo es estacionario. El término autorregresivo $(\phi_1 \nabla y_{t-1})$ captura la dependencia lineal entre el valor actual de la serie diferenciada y su valor en el periodo anterior. Está compuesto a su vez por ϕ_1, que es el coeficiente autorregresivo de orden 1, y ∇y_{t-1}, que es el valor de la serie temporal diferenciada en el periodo anterior. El término de variable exógena $(\beta_1 x_{t,1})$ representa el impacto de la variable exógena $x_{t,1}$ en el valor actual de la serie temporal. Aquí, β_1 es el coeficiente que mide el efecto de la variable exógena en la serie temporal. El error actual o ruido blanco en el tiempo t, sigue siendo ϵ_t. Por último, el término de media móvil $(\theta_1 \epsilon_{t-1})$, que captura la dependencia lineal entre el valor actual de la serie y el error en el periodo anterior, está formado por el coeficiente de media móvil de orden 1 θ_1, y el término de error en el periodo anterior ϵ_{t-1}.

Modelos estacionales autorregresivos integrados de media móvil — SARIMA

Los modelos SARIMA son una extensión de los modelos ARIMA, solo que estos incorporan componentes estacionales para modelizar series temporales con patrones estacionales recurrentes. La estacionalidad se caracteriza por patrones regulares que se repiten en intervalos regulares de tiempo.

La notación SARIMA se expresa como SARIMA(p, d, q)(P, D, Q)$_s$ [149], donde:

- p: Orden del componente autorregresivo no estacional (AR).
- d: Orden de diferenciación no estacional.
- q: Orden del componente de media móvil no estacional (MA).
- P: Orden del componente autorregresivo estacional (SAR).
- D: Orden de diferenciación estacional.
- Q: Orden del componente de media móvil estacional (SMA).
- s: Periodo de estacionalidad.

La ecuación general del modelo SARIMA es:

$$\nabla^d \nabla_s^D y_t = c + \sum_{i=1}^{p} \phi_i \nabla^d \nabla_s^D y_{t-i} + \sum_{i=1}^{P} \Phi_i \nabla^d \nabla_s^D y_{t-is} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \sum_{j=1}^{Q} \Theta_j \epsilon_{t-js} + \epsilon_t, \tag{2.83}$$

donde y_t es el valor de la serie temporal en el tiempo t, ∇^d representa la diferenciación no estacional de orden d, ∇_s^D representa la diferenciación estacional de orden D y período s, c es una constante que representa el término de intercepto en el modelo, p es el orden del componente autorregresivo no estacional (AR), P es el orden del componente autorregresivo estacional (SAR), q es el orden del componente de media móvil no estacional (MA), Q es el orden del componente de media móvil estacional (SMA), ϕ_i son los coeficientes del componente autorregresivo no estacional, Φ_i son los coeficientes del componente autorregresivo estacional, θ_j son los coeficientes del componente de media móvil no estacional, Θ_j son los coeficientes del componente de media móvil estacional, y por último, ϵ_t es el error aleatorio en el tiempo t.

Si se particulariza a un modelo SARIMA(1,1,1)(1,1,1)$_s$, se obtiene la siguiente ecuación:

$$\nabla\nabla_s y_t = c + \phi_1 \nabla\nabla_s y_{t-1} + \Phi_1 \nabla\nabla_s y_{t-s} + \theta_1 \epsilon_{t-1} + \Theta_1 \epsilon_{t-s} + \epsilon_t, \tag{2.84}$$

donde $\nabla y_t = y_t - y_{t-1}$ representa la serie temporal diferenciada una vez y $\nabla_s y_t = y_t - y_{t-s}$ representa la diferenciación estacional. El término constante del modelo, representado por c, puede interpretarse como la media del proceso si el modelo es estacionario. El término autorregresivo no estacional ($\phi_1 \nabla\nabla_s y_{t-1}$) captura la dependencia lineal entre el valor actual de la serie diferenciada y su valor en el periodo anterior. Está compuesto a su vez por ϕ_1, que es el coeficiente autorregresivo no estacional de orden 1, y $\nabla\nabla_s y_{t-1}$, que es el valor de la serie temporal diferenciada una vez en el periodo anterior. El término autorregresivo estacional ($\Phi_1 \nabla\nabla_s y_{t-s}$) captura la dependencia lineal entre el valor actual de la serie y su valor en el mismo periodo de la estación anterior. Está compuesto por Φ_1, que es el coeficiente autorregresivo estacional de orden 1, y $\nabla\nabla_s y_{t-s}$, que es el valor de la serie temporal diferenciada estacionalmente en el periodo anterior. El término de media móvil no estacional ($\theta_1 \epsilon_{t-1}$) captura la dependencia lineal entre el valor actual de la serie y el error en el periodo anterior. Está formado por el coeficiente de media móvil no estacional de orden 1 θ_1 y el término de error en

el periodo anterior ϵ_{t-1}. El término de media móvil estacional ($\Theta_1\epsilon_{t-s}$) captura la dependencia lineal entre el valor actual de la serie y el error en el mismo periodo de la estación anterior. Está formado por el coeficiente de media móvil estacional de orden 1 Θ_1 y el término de error en el periodo anterior ϵ_{t-s}. El error actual o ruido blanco en el tiempo t, sigue siendo ϵ_t.

Como se puede observar, este modelo incluye una única regresión autorregresiva (AR), una diferenciación no estacional y una diferenciación estacional, un único término de media móvil (MA) y sus componentes estacionales.

El ajuste de un modelo SARIMA sigue un proceso iterativo que consta de las siguientes etapas:

1. **Identificación:** El objetivo en esta fase es identificar tanto los componentes no estacionales (p, d, q) como los estacionales (P, D, Q, s).

2. **Estimación:** Los parámetros del modelo se estiman generalmente mediante el método de máxima verosimilitud. Sin embargo, debido a la complejidad añadida de los componentes estacionales, la estimación puede requerir un mayor número de iteraciones y un enfoque más refinado en la selección inicial de parámetros.

3. **Diagnóstico:** La evaluación de los residuos en el caso de un modelo SARIMA es especialmente crítica debido a la presencia de componentes estacionales. Además de los residuos no estacionales, también se debe de verificar si los residuos estacionales siguen un proceso de ruido blanco.

4. **Predicción:** Similar a ARIMA, una vez que los componentes del modelo SARIMA están bien ajustados, el modelo se utiliza para hacer predicciones futuras. La peculiaridad en estos modelos radica en que las predicciones son especialmente sensibles a las fluctuaciones estacionales y por ello deben de ser ajustadas de acuerdo con los cambios estacionales.

2.4 Modelos de análisis de volatilidad para series temporales

2.4.1 Introducción a la volatilidad

La volatilidad es la medida de variabilidad o de dispersión de los datos a lo largo del tiempo, es decir, cómo de amplia es la distancia que hay entre el valor máximo y mínimo de los valores de la serie. Para estimar la volatilidad, se utilizan medidas de dispersión como la varianza o la desviación típica. Una volatilidad alta indicará cambios más bruscos en los valores de la serie (mayor rango de variación), mientras que una volatilidad baja indicará que la variación de los valores de la serie se darán con menor intensidad (dicha serie tendrá menor rango de variación).

2.4.2 Propiedades de la volatilidad en series temporales

A continuación, se detallan algunas de las propiedades clave a considerar sobre la volatilidad en series temporales. Estas son:

- La volatilidad de la serie temporal no es constante, sino que cambia a lo largo del tiempo: Habrá periodos más turbulentos y otros más tranquilos.

- Clustering de volatilidad: En muchas series temporales, se observa que períodos de alta volatilidad tienden a agruparse, seguidos por períodos de baja volatilidad. Este fenómeno se conoce como 'clustering de volatilidad' [102].

- Distribución leptocúrtica: la distribución de los datos tiene más curtosis que una distribución normal, siendo la curtosis en distribuciones normales igual a 3. Generalmente las variables financieras tienen distribuciones más elevadas que la de una distribución normal. Esto provoca que la distribución de los rendimientos (o cambios de precios) de esa serie sea más puntiaguda y que tenga colas más pesadas en comparación con una distribución normal. Como consecuencia de ello, la probabilidad de que ocurran eventos extremos (en comparación con la distribución normal) aumenta [47].

- Movimiento conjunto entre las volatilidades de distintas series temporales: Este comovimiento entre la volatilidad de ambas series temporales, se observa especialmente entre los cuadrados de los rendimientos de los precios de dichas series, ya que son datos que están centrados y estandarizados [57].

2.4.3 Modelos clásicos y limitaciones

Los modelos clásicos de series temporales, como los modelos ARIMA, asumen que la varianza es constante (es decir, homocedástica). Esta es una de las propiedades que deben cumplirse, junto con la ausencia de tendencia y una media constante. Sin embargo, estos modelos no son adecuados para describir procesos donde la serie presenta una varianza condicional no constante (heterocedástica). Por esta razón, se necesitan modelos alternativos para modelizar la varianza de la serie temporal. Entre estos, destacan los modelos ARCH (Autoregressive Conditional Heteroskedasticity) y GARCH (Generalized Autoregressive Conditional Heteroskedasticity), que permiten capturar y modelar la heterocedasticidad condicional en los errores.

2.4.4 Homocedasticidad y heterocedasticidad

Cabe destacar la necesidad de profundizar un poco más en los conceptos homocedasticidad y heterocedasticidad:

Homocedasticidad: Si la varianza de los errores de una serie temporal es constante, se dice que la serie es homocedástica. Es importante destacar que en este tipo de series, los errores son constantes, es decir, tienen la misma dispersión en los diferentes periodos de tiempo. Esto implica que la volatilidad en los datos financieros es constante y no presenta cambios significativos, permitiendo que el valor de una variable pueda predecir a otra, ya que el modelo es insesgado.

Heterocedasticidad: La heterocedasticidad, por otro lado, se refiere a la situación en la que la varianza de los errores cambia a lo largo del tiempo de la serie. En este caso, la dispersión de los errores no es constante y puede variar en función del tiempo, lo que refleja la presencia de volatilidad en los datos financieros. Como consecuencia, la volatilidad va a tender a agruparse durante la serie en ciertos periodos. Cuando la varianza condicional no es constante, las series suelen presentar ciertos rasgos:

1. Tienen poca estructura en la media.

2. Pueden modelizarse como procesos AR de orden bajo y coeficiente pequeño.

3. Presentan una distribución no normal, con colas pesadas y alta curtosis.

4. Aunque la autocorrelación en la serie es baja, para los cuadrados de dicha serie existe una gran estructura de dependencia.
5. En los datos de la serie existen errores en los cálculos de las matrices correspondientes a los estimadores.
6. Se pierde eficiencia y fiabilidad del modelo.

Dado que la heterocedasticidad es una característica común en los datos financieros, se han desarrollado modelos específicos para abordar este fenómeno. Los dos modelos principales son ARCH y GARCH.

Modelo Autoregressive Conditional Heteroskedasticity (ARCH)

El modelo ARCH, desarrollado por Robert Engle en 1982, fue una innovación significativa para modelar la volatilidad en series temporales, al incorporar la heterocedasticidad condicional. En lugar de asumir que la varianza de los errores es constante a lo largo del tiempo, el modelo ARCH permite que esta varíe en función de los errores pasados al cuadrado, capturando así la agrupación de la volatilidad y las variaciones en la varianza de los errores a lo largo del tiempo [59].
La formulación básica de un modelo ARCH(p) se expresa de la siguiente manera:

$$\epsilon_t = \sigma_t z_t, \tag{2.85}$$

donde

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2, \tag{2.86}$$

y ϵ_t es el error en el tiempo t, σ_t^2 es la varianza condicional, z_t es un ruido blanco con media cero y varianza uno, y $\alpha_0, \alpha_1, \ldots, \alpha_p$ son los parámetros del modelo.
La inclusión de estos parámetros permite que la varianza de los errores en un momento dado dependa de los errores pasados al cuadrado, proporcionando una estructura que puede capturar la dependencia temporal en la volatilidad. Esta característica es especialmente útil para modelar la variabilidad observada en las series temporales donde las fluctuaciones en la volatilidad son comunes.

Modelo Generalized Autoregressive Conditional Heteroskedasticity (GARCH)

El modelo GARCH, desarrollado por Tim Bollerslev en 1986, es una extensión del modelo ARCH que incluye términos adicionales para modelar la volatilidad. En el modelo GARCH, la varianza de los errores se explica en función de los errores pasados al cuadrado y de las varianzas pasadas. Esto permite que el modelo capture tanto la persistencia de la volatilidad como las variaciones a corto y largo plazo en la varianza de los errores [23]. La formulación matemática de un modelo GARCH(p, q) se expresa de la siguiente manera:

$$\epsilon_t = \sigma_t z_t, \tag{2.87}$$

donde ϵ_t representa el término de error en el tiempo t, σ_t es la desviación típica condicional en el tiempo t, y z_t es un término de ruido blanco con distribución normal estándar.
La varianza condicional σ_t^2 se define como:

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \tag{2.88}$$

donde α_0 es una constante, $\alpha_1, \ldots, \alpha_p$ son los coeficientes asociados a los errores pasados al cuadrado (ϵ_{t-i}^2), y $\beta_1, \ldots, \beta_q$ son los coeficientes asociados a las varianzas condicionales pasadas (σ_{t-j}^2).
La capacidad del modelo GARCH para incorporar estructuras dinámicas de la volatilidad a lo largo del tiempo, tanto a corto plazo (a través de los errores pasados) como a largo plazo (a través de las varianzas pasadas) lo hace especialmente útil en la modelización de series temporales, donde la volatilidad puede ser alta y variar significativamente con el tiempo.

2.5 Extensiones de los modelos ARCH y GARCH

También se han desarrollado varias extensiones de los modelos ARCH y GARCH para abordar diversas características y limitaciones de los modelos originales. Algunas de las extensiones más notables son:

- **EGARCH** (Exponential GARCH): Propuesto por Nelson en 1991, el modelo EGARCH incorpora el efecto de apalancamiento sobre la volatilidad y utiliza una especificación logarítmica para la varianza condicional [117]. La formulación básica de un modelo EGARCH(p, q) es:

$$\log(\sigma_t^2) = \omega + \sum_{i=1}^{q} \beta_i \log(\sigma_{t-i}^2) + \sum_{j=1}^{p} \alpha_j \frac{\epsilon_{t-j}}{\sigma_{t-j}} + \sum_{j=1}^{p} \gamma_j \left(\left| \frac{\epsilon_{t-j}}{\sigma_{t-j}} \right| - \mathbb{E} \left| \frac{\epsilon_{t-j}}{\sigma_{t-j}} \right| \right), \tag{2.89}$$

  donde σ_t^2 es la varianza condicional, ϵ_t es el término de error en el tiempo t, ω es una constante, β_i son los coeficientes de la varianza condicional pasada, α_j son los coeficientes de los términos de error estandarizados pasados, y γ_j son los coeficientes que capturan el efecto de apalancamiento. El modelo EGARCH permite capturar la tendencia de la volatilidad a aumentar más cuando los valores de la serie financiera bajan que cuando suben.

- **TGARCH** (Threshold GARCH): Desarrollado por Zakoian en 1994, el modelo TGARCH es una extensión del GARCH que permite capturar el efecto de apalancamiento mediante la inclusión de un término umbral [165]. La formulación básica de un modelo TGARCH(p, q) es:

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2 + \sum_{k=1}^{p} \gamma_k \epsilon_{t-k}^2 I(\epsilon_{t-k} < 0), \tag{2.90}$$

  donde σ_t^2 es la varianza condicional, ϵ_t es el término de error en el tiempo t, ω es una constante, α_i son los coeficientes de los términos de error pasados, β_j son los coeficientes de la varianza condicional pasada, γ_k son los coeficientes del término umbral, e $I(\epsilon_{t-k} < 0)$ es una variable indicadora que toma el valor 1 si ϵ_{t-k} es negativo y 0 en caso contrario. Este modelo permite que los choques negativos tengan un impacto diferente en la volatilidad que los choques positivos, capturando así el efecto de apalancamiento.

- **GARCH-M** (GARCH in Mean): Propuesto por Engle, Lilien y Robins en 1987, el modelo GARCH-M incorpora la volatilidad condicional en la ecuación de la media del modelo [60]. La formulación básica de un modelo GARCH-M(p, q) es:

$$y_t = \mu + \lambda \sigma_t^2 + \epsilon_t, \tag{2.91}$$

  donde

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2, \tag{2.92}$$

  y y_t es la variable dependiente en el tiempo t, μ es una constante, λ es el coeficiente que mide el efecto de la volatilidad condicional sobre la media de la serie, σ_t^2 es la varianza condicional, ϵ_t es el término de

error en el tiempo t, ω es una constante, α_i son los coeficientes de los términos de error pasados, y β_j son los coeficientes de la varianza condicional pasada. Este modelo es útil para situaciones donde se espera que el riesgo (volatilidad) afecte el nivel esperado de la serie temporal.

Estas extensiones y otras variantes de los modelos ARCH y GARCH buscan mejorar la capacidad de los modelos originales para capturar diversas características y comportamientos en la volatilidad de las series temporales.

2.6 Volatilidad estocástica

La volatilidad estocástica proporciona una alternativa a los modelos ARCH y GARCH para modelar la volatilidad condicional en series temporales. A diferencia de los modelos ARCH y GARCH, donde la volatilidad es una función determinística de los rendimientos pasados, en los modelos de volatilidad estocástica, la volatilidad misma sigue un proceso estocástico [81]. Entre sus propiedades destaca una mayor flexibilidad para capturar la complejidad de la volatilidad implícita, la capacidad de modelar el efecto de apalancamiento cuando una caída en los precios de los activos da lugar a un aumento de la volatilidad, y la capacidad de capturar cambios en la volatilidad de [22].

2.6.1 Modelo de Heston

Uno de los modelos más conocidos en esta categoría es el modelo de Heston [74]. En este modelo, tanto el precio del activo S_t como su volatilidad v_t son modelados como procesos estocásticos. Las ecuaciones diferenciales estocásticas (EDS) asociadas con el modelo de Heston son:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^{(1)} \tag{2.93}$$

$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^{(2)}, \tag{2.94}$$

donde $W_t^{(1)}$ y $W_t^{(2)}$ son movimientos brownianos con correlación ρ, y μ, κ, θ, y σ son parámetros que necesitan ser estimados. S_t describe la dinámica del precio del activo, mientras que v_t captura la evolución estocástica de la volatilidad. La correlación ρ entre los dos movimientos brownianos permite modelar el efecto de apalancamiento, reflejando el hecho de que la volatilidad tiende a aumentar cuando los precios de los activos caen [43]. Para la estimación de los parámetros del modelo de Heston, se pueden utilizar técnicas como el método de máxima verosimilitud o métodos de calibración basados en precios de opciones del mercado. Estas técnicas permiten ajustar los parámetros del modelo para que los precios teóricos de las opciones coincidan con los precios observados en el mercado, asegurando así la precisión y relevancia del modelo en aplicaciones prácticas. En el ámbito Bayesiano y/ para la maximación de la verosimilitud (en el caso de densidades a-priori impropias), el método en [108] es considerado estado del arte actualmente.

Para la estimación de los parámetros del modelo de Heston, se pueden utilizar técnicas como el Método de Máxima Verosimilitud Estocástica o métodos de calibración basados en precios de opciones del mercado [17]. Estas técnicas permiten ajustar los parámetros del modelo para que los precios teóricos de las opciones coincidan con los precios observados en el mercado, asegurando así la precisión y relevancia del modelo en aplicaciones prácticas [6].

2.7 Métodos modernos más avanzados

Se considera el problema de ajustar un modelo de tiempo-variable a una serie temporal de vectores, actualizándolo en cada período de tiempo a medida que se observan nuevos datos. Suponiendo que los datos recientes son más relevantes que los datos de muchos períodos en el pasado, el modelo se ajusta dando más peso a los valores recientes del pasado y menos peso a los valores lejanos en el pasado.

Modelo de ventana móvil. Un método simple para hacer esto es ajustar el modelo en el período de tiempo t usando una ventana móvil de R valores anteriores de la serie temporal. La elección de R implica un compensación. Cuando es pequeña, se tienen menos datos para ajustar nuestro modelo; cuando es grande, el modelo tarda más en adaptarse a los cambios en los datos subyacentes. Se puede pensar en un modelo de ventana móvil (RWM) con una ventana de longitud R como aquella que pone peso uno en los últimos R valores de datos, y peso cero en cualquier valor anterior a R períodos en el pasado. Una ventaja del modelo RWM, es que el problema de optimización que se resuelve para llevar a cabo el ajuste tiene el mismo tamaño en cada período de tiempo.

Modelo de promedio móvil exponencialmente ponderado. Otro método para ajustar un modelo de tiempo variable utiliza todos los datos pasados para crear el modelo, pero pone un peso variable en el tiempo sobre los valores pasados que decae suavemente a medida que se aleja más en el tiempo. Una elección natural para los pesos es un decaimiento exponencial. Dicho modelo actúa como un modelo de promedio móvil exponencialmente ponderado (EWMA). El parámetro en EWMA análogo a R en un RWM es la vida media, el número de períodos en el pasado donde el peso decae a la mitad.

2.7.1 Promedio móvil exponencialmente ponderado - exponentially weighted moving average (EWMA)

En esta sección, se introducire el concepto de *exponentially weighted moving average* (EWMA) general, enfocándonos en modelos que pueden ajustarse mediante optimización convexa. Se aborda la cuestión de cómo calcular un EWMA. Para funciones de pérdida cuadrática, existe un método recursivo exacto. Para pérdidas no cuadráticas, se describen métodos para ajustar aproximadamente un EWMA, utilizando solo una ventana de datos pasados de tamaño fijo y resolviendo un problema de tamaño fijo que no crece con el tiempo. Se demuestra el método recursivo aproximado con ejemplos, mostrando que encuentra modelos que están cerca de los modelos EWMA exactos, ajustándose usando todos los datos pasados. El método es práctico y extiende los muchos modelos de datos que se ajustan utilizando optimización convexa al entorno de tiempo variable exponencialmente.

En este documento no se aborda la cuestión de si un EWMA debe usarse en una aplicación, por ejemplo, en lugar de un RWM o cualquier otro método para ajustar un modelo de tiempo variable. Simplemente se asume que hay un uso para un EWMA, y se da un método práctico para llevar a cabo la evaluación con memoria y computación que no crecen con el tiempo.

Promedio móvil exponencialmente ponderado

Suponiendo que $\mathbf{x}_1, \mathbf{x}_2, \ldots \in \mathbb{R}^n$ es una serie temporal de vectores. Su promedio móvil exponencialmente ponderado (EWMA) es la serie temporal de vectores

$$\tilde{\mathbf{x}}_t = \alpha_t \sum_{\tau=1}^{t} \beta^{t-\tau} \mathbf{x}_\tau, \quad t = 1, 2, 3, \ldots, \tag{2.95}$$

donde $\beta \in (0,1)$ es llamado *factor de olvido*, y

$$\alpha_t = \left(\sum_{\tau=1}^{t} \beta^{t-\tau} \right)^{-1} = \frac{1-\beta}{1-\beta^t}, \tag{2.96}$$

es la constante de normalización. Esto significa que los pesos

$$\bar{w}_\tau = \alpha_t \beta^{t-\tau},$$

están normalizados (es decir, suma 1). La expresión arriba queda como:

$$\tilde{\mathbf{x}}_t = \sum_{\tau=1}^{t} \bar{w}_\tau \mathbf{x}_\tau. \tag{2.97}$$

El factor de olvido β se expresa usualmente en términos de la vida media $H = -\log 2 / \log \beta$, para la cual $\beta^H = 1/2$. Es importante remarcar que la $\mathbf{x}$ aquí puede ser multidimensional (es un vector en general).

Implementación recursiva. La secuencia EWMA (2.95) puede computarse recursivamente como:

$$\tilde{\mathbf{x}}_{t+1} = \frac{\alpha_{t+1}}{\alpha_t} \beta \tilde{\mathbf{x}}_t + \alpha_{t+1} \mathbf{x}_{t+1}, \quad t = 1, 2, \ldots. \tag{2.98}$$

Por lo tanto, se puede calcular $\tilde{\mathbf{x}}_t$ sin almacenar los valores pasados $\mathbf{x}_1, \ldots, \mathbf{x}_t$; solo se necesita llevar un registro del estado $\tilde{\mathbf{x}}_t$.

Interpretaciones. Hay varias formas de interpretar la serie temporal EWMA $\tilde{\mathbf{x}}$. Se puede pensar en ella como una versión de la serie temporal original $\mathbf{x}$ que ha sido suavizada en una escala de tiempo del orden de H. También como si fuera la transformación de la secuencia $\mathbf{x}$ a la secuencia EWMA $\tilde{\mathbf{x}}$ como una operación de filtrado de paso bajo, que elimina variaciones de alta frecuencia.

La interpretación más utilizada es que $\tilde{\mathbf{x}}_t$ es una estimación variable en el tiempo de la media de $\mathbf{x}_t$, formada a partir de $\mathbf{x}_1, \ldots, \mathbf{x}_t$, donde se supone que $\mathbf{x}_t$ proviene de una distribución variable en el tiempo con una media que varía lentamente. Se puede expresar esta interpretación usando una función de pérdida cuadrática:

$$\tilde{\mathbf{x}}_t = \arg\min_x \alpha_t \sum_{\tau=1}^{t} \beta^{t-\tau} \|\mathbf{x} - \mathbf{x}_\tau\|_2^2. \tag{2.99}$$

Así, la estimación EWMA $\tilde{\mathbf{x}}_t$ minimiza la suma exponencialmente ponderada de pérdidas cuadráticas pasadas $\|\mathbf{x} - \mathbf{x}_\tau\|_2^2$, para $\tau = 1, \ldots, t$.

Capítulo 3

Kernel smoothers

El suavizado de kernel, o 'kernel smoother', es una técnica no paramétrica que se centra en la estimación de una función de densidad de probabilidad desconocida a partir de un conjunto de datos. Este método le permite funcionar como una media móvil ponderada sin asumir ninguna estructura paramétrica particular.

Estimación de densidad (Kernel Density Estimation). La idea del suavizado de kernel nace probablemente de la publicación de Rosenblatt de 1960 llamada 'Remarks on Some Nonparametric Estimates of a Density Function', aunque fue Parzen en 1962 quien introdujo la función de estimación tipo núcleo (o kernel) [123]. con la siguiente ecuación:

$$\widehat{f_n}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right), \tag{3.1}$$

considerando

$$\int_{\mathcal{X}} k\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right) d\mathbf{x} = 1, \quad \text{(mas en general se asume } < \infty\text{)}, \tag{3.2}$$

donde $\widehat{f_n}(\mathbf{x})$ representa la estimación de la función de densidad en el punto $\mathbf{x}$, N es el número total de observaciones en la muestra, h es el bandwidth del kernel que también controla el grado de suavidad de la estimación, $k(\cdot)$ es la función kernel (o función núcleo). Esta función determina la forma y la importancia de los pesos que se asignan a las observaciones cercanas al punto $\mathbf{x}$, de esta forma pondera y asigna un peso mayor a los puntos de datos cercanos a $\mathbf{x}$ y menor peso a los puntos de datos más alejados, $\mathbf{x}_i$ representa el i-ésimo punto de datos de la muestra, $\sum_{i=1}^{N}$ es el sumatorio que recorre todas las observaciones de la muestra, y el término $\left(\frac{\mathbf{x}-\mathbf{x}_i}{h}\right)$ dentro de la función kernel determina cómo de lejos está el punto $\mathbf{x}$ del punto de datos $\mathbf{x}_i$, y luego escala ese valor por el bandwidth h.

Más tarde, esta idea de la función de suavizado, fue extendida al problema de la regresión por ejemplo por Nadaraya (1964) y Watson (1964), ampliando de este modo la variedad de problemas donde aplicar esta técnicas [114, 153]. También puede ser usada para clasificación.

3.1 Metodología de suavizadores de kernel para regresión con pesos normalizados

Dada una muestra de datos $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, el suavizador de kernel estima la función Nadaraya-Watson $\widehat{f}$ en un punto $\mathbf{x}$ como:

$$\widehat{f}(\mathbf{x}) = \widehat{f}(\mathbf{x}|h) = \sum_{n=1}^{N} \frac{k_\lambda(\mathbf{x}, \mathbf{x}_n)}{\sum_{j=1}^{N} k_\lambda(\mathbf{x}, \mathbf{x}_j)} y_n = \sum_{n=1}^{N} \phi_n(\mathbf{x}, \mathbf{x}_n) y_n, \tag{3.3}$$

donde $\phi_n(\mathbf{x}, \mathbf{x}_n) = \frac{h_\lambda(\mathbf{x}, \mathbf{x}_n)}{\sum_{j=1}^{N} h_\lambda(\mathbf{x}, \mathbf{x}_j)}$, y $\widehat{f}(\mathbf{x})$ es la estimación de la función de regresión en el punto $\mathbf{x}$, h es una función de kernel, λ indica el ancho de la banda, $\mathbf{x}_n$ y $\mathbf{x}_j$ son los puntos de datos de la muestra. Es importante saber que $h_\lambda(\mathbf{x}, \mathbf{x}_n) \geq 0$, es decir, los pesos que asigna la fución en base a la distancia que hay entre $\mathbf{x}_n$ y $\mathbf{x}$ siempre van a ser positivos y mayores que 0. Al estar ante una función de normalización de pesos, la suma de todos ellos va a ser 1. Es decir,

$$\sum_{n=1}^{N} \phi_n(\mathbf{x}, \mathbf{x}_n) = 1. \tag{3.4}$$

3.2 Métodos para selección del bandwidth (anchura del kernel)

Una parte importante del suavizado de kernel es el modo de elección del bandwidth o parámetro de suavizado, h. Este parámetro es un valor fijo para todo el conjunto de datos y actúa como una ventana que determina cuántos datos cercanos se tienen en cuenta al estimar la densidad en un punto concreto de los datos.
La selección de este valor es muy importante, ya que determina el nivel de suavidad de la estimación final. En la Figura 3.1 se puede apreciar cómo un valor de h demasiado pequeño (h=0.001), puede dar lugar a un suavizado excesivo. Los modelos capturan características detalladas en los datos en lugar de la tendencia subyacente. Por otro lado, un valor de h demasiado grande (h=10) puede llevar a un suavizado insuficiente, donde el modelo generalice mejor pero no capta suficientes detalles relevantes de los datos.

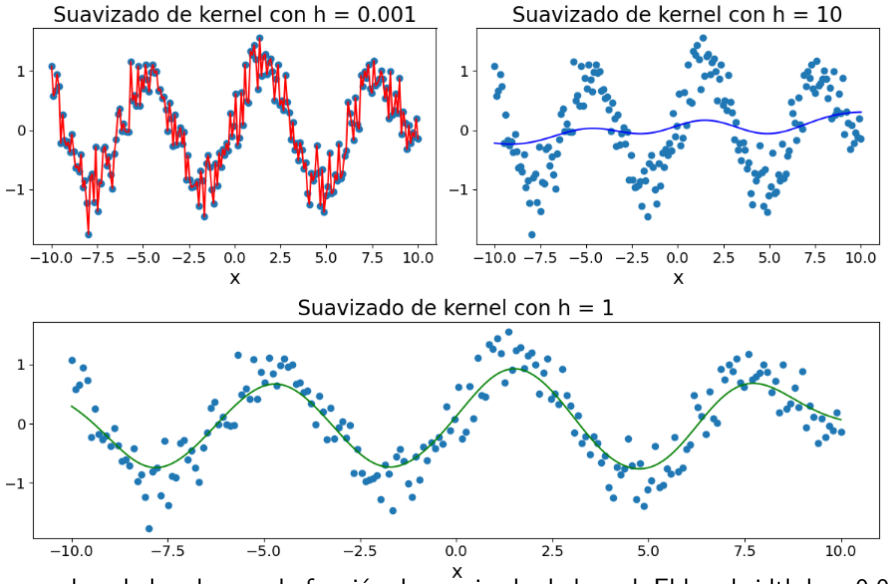


Figura 3.1: Distintos anchos de banda para la función de suavizado de kernel. El bandwidth h = 0.001 muestra un exceso de suavizado, el gráfico dónde h tiene un valor de 10 muestra un suavizado demasiado suave, mientras que el tercer gráfico con un h = 1 se asimila más a un suavizado de los datos óptimo.

Es por ello que uno de los objetivos más relevantes es seleccionar el valor óptimo de h, que será aquel donde el bandwidth tenga un buen equilibrio entre el sesgo de los datos y la varianza del estimador, tal y cómo se ve en el caso de la Figura en cuestión, con $h = 1$.

Problema. Se pretende calcular el h que minimiza el mean integrated squared error (MISE) [138], es decir:

$$MISE(\widehat{f}) = E\left[\int \left(f(\mathbf{x}) - \widehat{f}(\mathbf{x}|h)\right)^2 d\mathbf{x}\right]. \tag{3.5}$$

Pero no se puede calcular el MISE: no es posible ya que no se conoce la verdadera función $f(\mathbf{x})$. Por lo tanto, habrá que estimar la función desconocida $f(\mathbf{x})$.

Existen diversos métodos destinados a la elección óptima del bandwidth del estimador. Los métodos que se explican a contiunación son cross-validation, y diferentes "rules of thumb"(por ejemplo la regla de Silverman).

Método cross-validation (CV) para la selección del bandwidth: En este enfoque, se busca minimizar el error de predicción, y para ello, se dividen los datos en un conjunto de entrenamiento y otro conjunto de validación o test. Se entrena el modelo en el conjunto de entrenamiento para varios valores de h, y se selecciona el valor que minimiza el criterio de error utilizado para la predicción del conjunto de validación [27]. La validación cruzada se realiza siguiendo los siguientes pasos:

1. Dividir el conjunto de datos en k subconjuntos o 'folds' (por ejemplo, $k = 10$).
2. Para cada valor de h considerado:
   a) Entrenar el modelo en $k - 1$ subconjuntos y validar en el subconjunto restante.
   b) Calcular el error de predicción para ese subconjunto y ese valor de h.
3. Promediar el error de predicción sobre todos los subconjuntos para cada valor de h.
4. Seleccionar el valor de h que minimiza el error promedio.

La ventaja de la validación cruzada, es que esta utiliza una porción del conjunto de datos para validar el modelo, y puede proporcionar una estimación robusta del rendimiento del suavizado en datos no vistos. Aunque es más costosa computacionalmente que métodos como la regla de Silverman, su flexibilidad y precisión la convierten en una opción muy útil para seleccionar el bandwidth óptimo.
La formulación típica del error que se busca minimizar en la validación cruzada es el error cuadrático medio (MSE), aunque se pueden utilizar otros criterios de error en función de las características y necesidades del análisis.

3.2.1 Regla de Silverman

Se pueden encontrar en la literatura, estudios teóricos de bandwidth óptimos bajo ciertas hipótesis y circunstancias. Estas expresiones no son muy útiles desde el punto de vista practico. El bandwidth h óptimo para minimizar el MISE bajo ciertas condiciones regulares puede expresarse en términos de una constante C, la desviación estándar σ de los datos, y una derivada de la función de densidad real f. La relación aproximada es:

$$h_{opt} \approx C \times \sigma \times N^{-\frac{1}{d+4}}, \tag{3.6}$$

donde N es el tamaño de la muestra y d es el orden de la derivada de la función de densidad que minimiza el MISE. Dado que estas expresiones no son muy útiles desde el punto de vista practico, otras reglas heuristicas

se han diseñado en la literatura.

Regla de Silverman: Silverman propuso una regla basada en la minimización del error cuadrático medio asintótico de la función de densidad [140] para una clase particular. Esta regla se puede usar para determinar una primera aproximación de h. Esta fórmula se basa en la suposición de que los datos tienen una distribución aproximadamente a un Gaussiana. La fórmula del método 'rule of thumb' de Silverman es la siguiente:

$$h = \left(\frac{4\widehat{\sigma}^5}{3N} \right)^{\frac{1}{5}}, \tag{3.7}$$

donde $\widehat{\sigma}$ es la desviación típica de la muestra y n es el tamaño de la muestra. Como se dijo anteriormente, la regla de Silverman se deriva suponiendo que la distribución subyacente de los datos es normal. Esta suposición da lugar a una expresión para el bandwidth que minimiza el error asintótico cuadrático medio (AMISE) para un estimador de densidad kernel Gaussiano. Asumir normalidad puede limitar la aplicabilidad de las reglas a datos que se desvían significativamente de una distribución normal.

3.3 Principales tipos de funciones kernel

En esta sección en particular, se analizan algunos kernels específicos: Cada uno de estos kernels se caracteriza por sus funciones matemáticas únicas y sus diferentes aplicaciones prácticas. Esto refleja la importancia de elegir de forma correcta el kernel para optimizar el rendimiento del modelo en diferentes contextos. A continuación, se presenta una descripción de cada uno, destacando su definición matemática y características clave.

3.3.1 Kernel Gaussiano

El kernel Gaussiano (ver la Figura 3.2) es uno de los más utilizados debido a sus propiedades matemáticas inherentes, y a su efectividad en múltiples de sus aplicaciones.

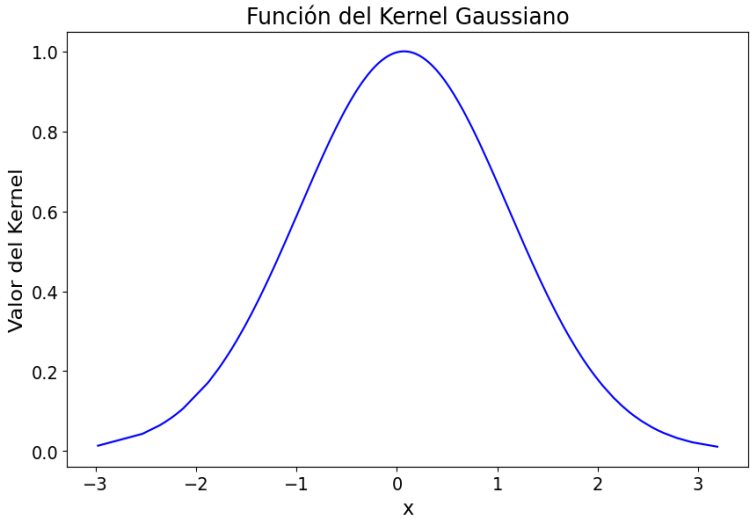


Figura 3.2: Función de kernel Gaussiano con un bandwidth igual a 1

La función de kernel Gaussiano se define como:

$$k(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}, \tag{3.8}$$

donde el término u se define como $u = \frac{\mathbf{x}-\mathbf{x}_i}{h}$, $\mathbf{x}$ es el punto de consulta, $\mathbf{x}_i$ es el punto de datos del conjunto y h es el bandwidth del kernel. La naturaleza 'en forma de campana' del kernel Gaussiano implica que las observaciones cercanas a $\mathbf{x}$ son las más relevantes, y su influencia disminuye rápidamente a medida que se aleja de $\mathbf{x}$. Esta característica es muy importante a la hora de obtener estimaciones robustas, especialmente donde los datos pueden contener ruido o valores atípicos.
Una de las propiedades más notables de este kernel, es que es 'infinitamente diferenciable'. Esto significa que todas las derivada de la función kernel son continuas en todos los puntos. Esta propiedad es fundamental en cuando es necesario analizar la derivada de una función estimada, como puede ser por ejemplo la identificación de puntos de inflexión o máximos y mínimos locales. Además, garantiza que las estimaciones producidas tengan transiciones suaves, evitando a su vez discontinuidades.
Vale la pena mencionar que el kernel Gaussiano ha demostrado ser especialmente efectivo en una gran variedad de aplicaciones, desde la estimación de densidad hasta la regresión y la clasificación, entre otras [105].

3.3.2 Kernel t-Student

El kernel t-Student (ver la Figura 3.3 puede ser especialmente útil cuando los datos presentan colas pesadas o cuando se pueden observar valores atípicos. Este kernel se basa en la distribución t-Student y es conocido por su flexibilidad para modelar datos con diferentes grados de kurtosis en comparación con la distribución normal [151].

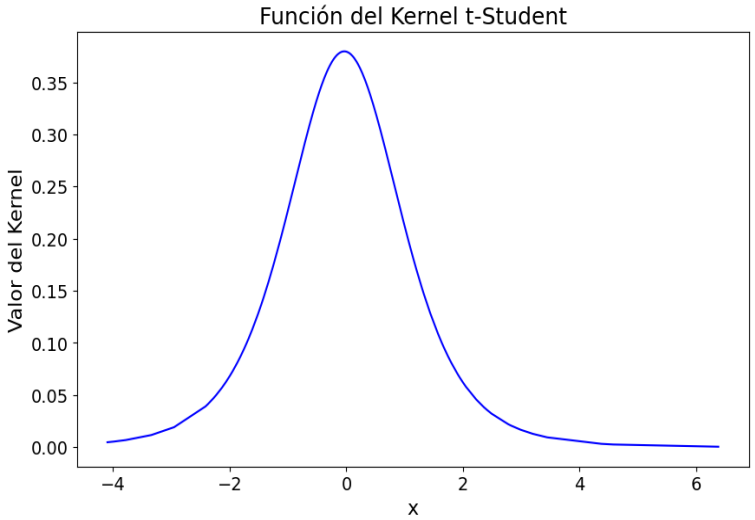


Figura 3.3: Función de kernel t-Student con 5 grados de libertad y un bandwidth igual a 1

La función de kernel t-Student se define de la siguiente manera:

$$k(u) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\,\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{u^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{3.9}$$

donde ν son los grados de libertad de la distribución t-Student, Γ es la función Gamma, y u es una medida de la distancia entre el punto de datos y el punto de consulta, normalizada por el bandwidth h.

Una de las características más destacadas de la distribución t-Student es que, en función del número de grados de libertad ν, puede tener colas más pesadas que la distribución normal. Esto permite que el kernel t-Student dé más peso a los datos que están más alejados del punto central, proporcionando robustez frente a valores atípicos. Además, el parámetro ν se puede ajustar para cambiar la forma del kernel, permitiendo que se adapte mejor a la estructura de los datos. Un valor más bajo de ν produce un kernel con colas más pesadas, mientras que un valor más alto se aproximará más al kernel Gaussiano.

3.3.3 Kernel Rectangular

El kernel rectangular, también conocido como kernel uniforme (ver la Figura 3.4), se caracteriza por su asignación de pesos uniforme dentro de su rango definido. Esto simplifica los cálculos y puede ser útil en situaciones donde la igual ponderación de observaciones es deseable [64]. Su formulación matemática es la siguiente:

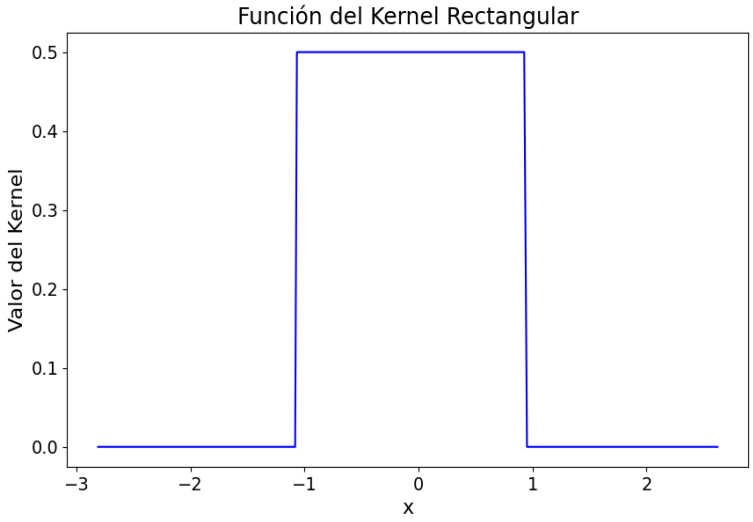


Figura 3.4: Función de kernel Rectangular con un bandwidth igual a 1

La función de kernel rectangular se define de la siguiente manera:

$$k(u) = \begin{cases} \frac{1}{2} & \text{si } |u| \leq 1 \\ 0 & \text{si } |u| > 1, \end{cases} \tag{3.10}$$

donde $u = \frac{x - x_i}{h}$, y h es el ancho de banda del kernel.

A diferencia de otros kernels como el Gaussiano o el t-Student, el kernel rectangular tiene un límite claramente definido. Esto puede resultar en estimaciones menos suaves, ya que los puntos fuera del rango no influyen en absoluto, mientras que los puntos dentro del rango tienen la misma influencia.

3.3.4 Kernel Triangular

El kernel triangular (ver la Figura 3.5) aporta una ponderación linealmente decreciente a los datos según se alejan del punto de consulta, lo que hace que los puntos más cercanos tienen mayor influencia que aquellos más alejados dentro de la ventana del bandwidth [125].

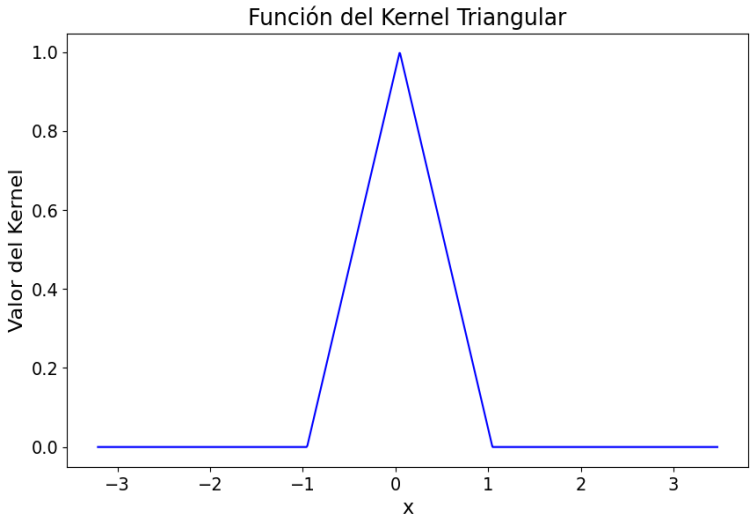


Figura 3.5: Función de kernel Triangular con un bandwidth igual a 1

La función de kernel triangular se define de la siguiente manera:

$$k(u) = \begin{cases} 1 - |u| & \text{si } |u| \leq 1 \\ 0 & \text{si } |u| > 1 \end{cases} \tag{3.11}$$

donde $u = \frac{x - x_i}{h}$, y h es el bandwidth del kernel. Este kernel integra a 1 en su dominio, asegurando que la estimación de densidad sea un estimador válido. Este kernel se recomienda en situaciones donde es deseable una ponderación menos agresiva hacia puntos muy distantes, manteniendo una influencia significativa sobre los puntos cercanos.

3.3.5 Kernel Tricubo

El kernel tricubo (ver la Figura 3.6) proporciona una disminución muy suave del peso asignado a las observaciones a medida que se alejan del punto de consulta. Se recomienda en situaciones donde se requiere un alto grado de suavidad sin que la precisión de la estimación se vea afectada [128].

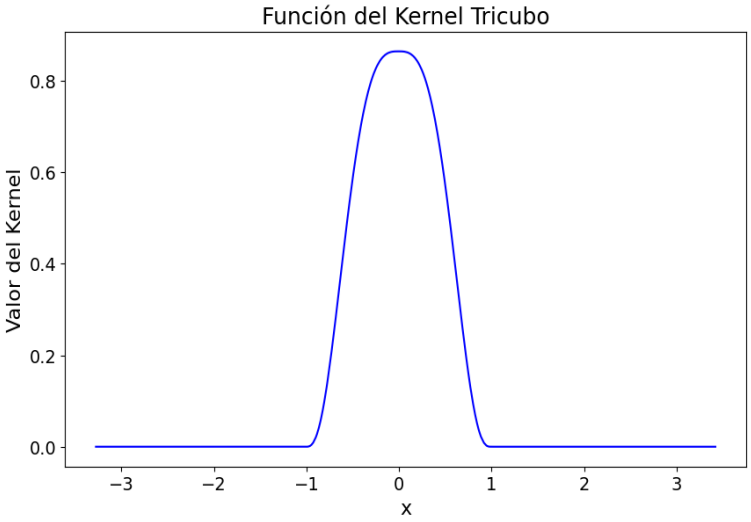


Figura 3.6: Función de kernel Tricubo con un bandwidth igual a 1

La función de kernel tricubo se define de la siguiente forma:

$$k(u) = \begin{cases} (70/81)\left(1 - |u|^3\right)^3 & \text{si } |u| \leq 1 \\ 0 & \text{si } |u| > 1, \end{cases} \tag{3.12}$$

donde $u = \frac{x - x_i}{h}$, y h es el bandwidth del kernel. Esta función es conocida por su suave transición a cero en los bordes del rango definido, minimizando el impacto de los valores atípicos y las discontinuidades. Cabe destacar que la forma cúbica de la caída del peso asegura una transición muy suave.

3.3.6 Kernel Epanechnikov

El kernel Epanechnikov lleva el nombre de V. A. Epanechnikov, quien propuso este método en 1969. Este método es conocido por tener propiedades óptimas en términos de minimizar el error cuadrático medio asociado con la estimación de la densidad. La Figura 3.7 muestra el kernel Epanechnikov con bandwidth de 1.

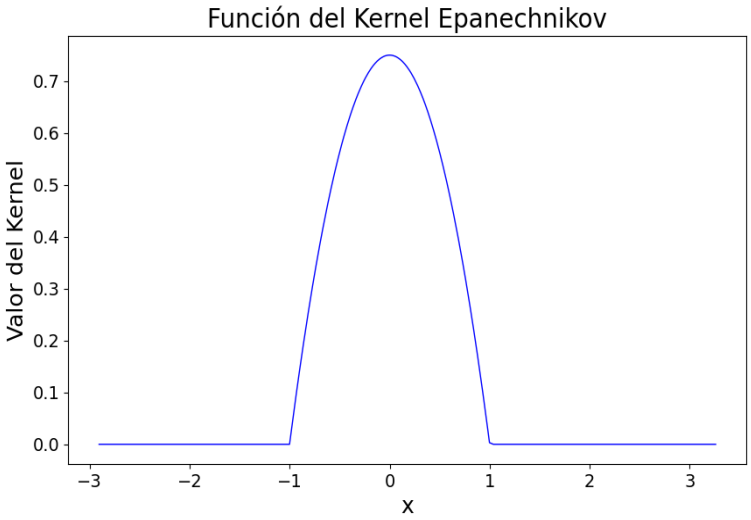


Figura 3.7: Función de kernel Epanechnikov con un bandwidth igual a 1

Una de sus ventajas es que es computacionalmente menos costoso que otros kernels debido a su soporte acotado, ya que los datos que se encuentra a más de h del punto de consulta pueden ser ignorados [140]. El kernel Epanechnikov se define como:

$$k(u) = \begin{cases} \frac{3}{4}(1-u^2) & \text{si } |u| < 1 \\ 0 & \text{si } |u| \geq 1. \end{cases} \tag{3.13}$$

El valor u es una medida normalizada de la distancia entre un punto de datos dado $\mathbf{x}_i$ y el punto de consulta $\mathbf{x}$, ajustado por el bandwidth h. Específicamente, se define de la siguiente forma: $u = \frac{\mathbf{x}-\mathbf{x}_i}{h}$, donde $\mathbf{x}$ es el punto de consulta, $\mathbf{x}_i$ es el punto de datos específico, y h es el bandwidth. La estructura de esta función es la siguiente: Vale $\frac{3}{4}(1-u^2)$ entre -1 y 1 y cero fuerta de este intervalo. Esta expresión define el comportamiento del kernel dentro del intervalo $[-1, 1]$. La función tiene forma parabólica en este intervalo, alcanzando su valor máximo en $u = 0$ (el punto medio). A medida que u se aleja del centro, el valor del kernel disminuye. El kernel tiene Valor 0 para valores de u fuera del intervalo $[-1, 1]$. Esto significa que cualquier punto de datos que esté a más de h unidades de distancia del punto de consulta no contribuirá a la estimación.

3.4 Ejemplo de métrica de evaluación

Las métricas de evaluación o de error permiten comparar y evaluar el rendimiento y la eficacia de los distintos estimadores con la finalidad de encontrar la configuración de parámetros óptima. Esta sección se presenta algunas métricas importantes utilizadas para evaluar la calidad de diferentes estimadores de densidad. Incluye error absoluto integrado (IAE). El error absoluto integrado es una métrica utilizada para evaluar la precisión de los estimadores de densidad basados en kernel que mide la diferencia entre la función real y la función estimada [140]. El error absoluto integrado entre la función de densidad real f y su función estimada $\widehat{f}$ se define como:

$$IAE(f, \widehat{f}) = \int |f(\mathbf{x}) - \widehat{f}(\mathbf{x})| \, d\mathbf{x}. \tag{3.14}$$

La expresión matemática 3.14 evalúa la suma de las desviaciones, siendo estas el resultado de la diferencia entre la función estimada y la función real en todo el dominio. El IAE penaliza linealmente las desviaciones utilizando valores absolutos, independientemente de si la diferencia es positiva o negativa. También proporciona una medida robusta del error de estimación, lo cual es especialmente útil cuando hay valores atípicos o irregularidades en los

datos. Cuanto más pequeño sea el IAE, más cerca estará la estimación $\widehat{f}$ de la verdadera función de densidad f.

Capítulo 4

Estudio de series temporales con algoritmos de machine learning

A través de este capítulo, el objetivo es proporcionar una base sólida y detallada para comprender las redes neuronales y su aplicación en la modelización de series temporales. Para ello, se comenzará con la presentación de la arquitectura básica de las redes neuronales, incluyendo su historia y evolución. A partir de ahí, se realizará una exploración de los conceptos fundamentales de las redes neuronales, como los perceptrones, las neuronas, los pesos, las capas densas, y los inicializadores de pesos, así como la importancia de las épocas en el entrenamiento de la red. A continuación, se estudiarán las funciones de activación más utilizadas en la actualidad, como la función sigmoide, la tangente hiperbólica, las Unidades Lineales Rectificadas (RELU) y la Unidad lineal exponencial escalada (SELU).
En la siguiente sección, se examinarán las diversas funciones de coste empleadas en el entrenamiento de las redes neuronales. Específicamente, la entropía cruzada, el error cuadrático medio, el error absoluto medio y la raíz del error cuadrático medio. Una vez analizadas las funciones de coste, se analizarán los métodos de regularización. Estos métodos se utilizan para prevenir el sobreajuste y mejorar la capacidad de generalización de las redes neuronales. Destacan algunos métodos como Lasso (L1), Ridge (L2), Dropout, batchnormalization (normalización por lotes) y early-stopping (parada temprana).
El siguiente paso será examinar el algoritmo de entrenamiento de la red neuronal. En esta sección, se abordarán las funciones de propagación, comenzando con la función de propagación hacia adelante (forward propagation) y continuando con la función de retropropagación o propagación hacia atrás (backpropagation) y la regla de la cadena. También se presentarán varios métodos de optimización avanzados, incluyendo el descenso de gradiente estocástico, momentum, el descenso de gradiente acelerado de Nesterov (NAG), el algoritmo de gradiente adaptativo (Adagrad), la propagación de raíz cuadrática media (RMSprop), la estimación del momento adaptativo (Adam) y Adadelta. Posteriormente, el análisis continuará hacia las capas especializadas diseñadas para manejar datos temporales. Se hará especial hincapié en las redes neuronales recurrentes y convolucionales, así como en las ventajas y desventajas de cada una de ellas.
Por último, este análisis concluirá con el concepto de 'Contrast Gain Control', donde se hará enfoque en la normalización divisiva local. Estos métodos de normalización son particularmente relevantes para el procesamiento de imágenes y la detección de patrones en las redes neuronales, es por ello por lo que en esta tesis se plantea su uso en datos temporales, para evaluar su eficacia más allá de las imágenes.

4.1 Arquitectura de las redes neuronales

Esta parte del trabajo explora la evolución, componentes y aplicaciones de la Red Neuronal Artificial (ANN por sus siglas en inglés, Artificial Neural Network). Se hace especial énfasis en la arquitectura del perceptrón y el perceptrón multicapa. Se profundiza en los aspectos fundamentales de la comunicación neuronal, incluyendo las sinapsis, neurotransmisores, y la transmisión de información. Además, se examinan los componentes clave de las ANN como las neuronas, pesos, capa densa, inicializador de pesos y épocas. Esta sección proporciona una retrospectiva sobre la historia y el progreso de las ANN, delineando su relación con los procesos neuronales y su evolución desde los primeros modelos hasta las sofisticadas redes de hoy en día. A continuación se examina la evolución de los modelos de perceptrón, desde el perceptrón simple hasta el perceptrón multicapa, ilustrando la transición de modelos lineales simples a estructuras complejas capaces de capturar relaciones no lineales.
Finalmente, se detallan los componentes y configuraciones esenciales en las ANN, proporcionando una comprensión integral de los elementos que componen estos modelos.

4.1.1 Breve historia e introducción

El cerebro humano es un órgano muy complejo, formado por neuronas, que están interconectadas a su vez por redes más complejas. Todo esto le otorga la capacidad de procesar la información que recibe, y a aprender de sus experiencias. Además, tiene una gran capacidad para aprender y adaptarse. Lleva años siendo objeto de estudio por la comunidad científica. Las redes neuronales artificiales, son modelos informáticos que intentan emular el funcionamiento del cerebro humano. Las ANN están compuestas por nodos, que son similares a las neuronas del cerebro. Los nodos están interconectados y pueden comunicarse entre sí. También se utilizan estas redes para aprender de los datos y realizar tareas complejas. Desde una perspectiva biológica, el fenómeno de la sinapsis es el proceso que permite la 'transmisión de información' entre neuronas. Esta transmisión ocurre desde las terminaciones de las ramificaciones del axón de una neurona hacia las dendritas de otra.
Cuando un estímulo (o impulso eléctrico) alcanza un terminal nervioso, provoca la liberación de neurotransmisores por parte del nervio. Dependiendo del tipo de neurotransmisor liberado, las neuronas receptoras pueden activarse si reciben el estímulo de las neuronas a las que están conectadas, o inhibirse si dicha información no llega a recibirse. Esto genera una respuesta específica en cada caso, ya sea de excitación o inhibición.
Las redes neuronales artificiales buscan emular este mecanismo de transmisión de información y adaptación a través de un sistema de nodos interconectados, comúnmente denominados 'neuronas artificiales'. Al igual que en su contraparte biológica, estas neuronas artificiales pueden 'activarse' o 'inhibirse' basándose en las señales que reciben.
Algunas redes neuronales artificiales se estructuran en capas, debiendo existir al menos:

- Una capa de entrada: asociada a las variables explicativas.
- Una capa de salida: asociada a la/s variables target u objetivo.

Opcionalmente, una red neuronal puede llevar asociada una o varias capas intermedias u ocultas, que ofrecen a la red gran flexibilidad en los tipos de relaciones input – output que puede manejar, permitiendo reflejar relaciones más complejas (no lineales) entre las variables. El estudio de las redes neuronales artificiales comenzó en 1943 con la propuesta de un neurona artificial, por Warren McCulloch y Walter Pitts [109].
El perceptrón [132], comúnmente reconocido como la primigenia red neuronal fue desarrollado por Frank Rosenblatt en la década de 1950. Supuso un gran avance en el área de la inteligencia artificial. A lo largo del tiempo, su diseño elemental evolucionó, adquiriendo una mayor complejidad que dió lugar al perceptrón multicapa (MLP, por sus siglas en inglés). Se considera el fundamento de las redes neuronales actuales. Es un tipo de clasificador lineal binario y se considera el primer modelo y el más básico de red neuronal. La idea

subyacente del perceptrón es imitar la función básica de una neurona en el cerebro. Para ello, toma un conjunto de información como datos de entrada o input (por ejemplo, las características de un conjunto de datos), posteriormente pondera dichos datos según unos pesos asignados previamente, los suma y luego aplica una función de activación para producir una salida o output.
La salida del perceptrón puede ser en el caso más simple binaria (por ejemplo, tomando valores 0 o 1), aunque también puede tomar más valores en versiones más complejas del perceptrón. Esta salida se puede interpretar como una clase en un problema de multiclasificación. A pesar de la importancia que tiene el perceptrón al establecer las bases de las redes neuronales tal y como se conocen, cabe recordar que también tiene una serie de limitaciones significativas. La más relevante es que solo puede tratar con problemas linealmente separables.
El perceptrón multicapa (Figura 4.2) surge como una evolución del perceptrón simple (Figura 4.1), incorporando una mayor complejidad y flexibilidad para modelar relaciones no lineales. Cada neurona de la capa de entrada está conectada a todas las neuronas de la capa siguiente, permitiendo la capacidad de modelar relaciones no lineales entre las entradas y la salida.

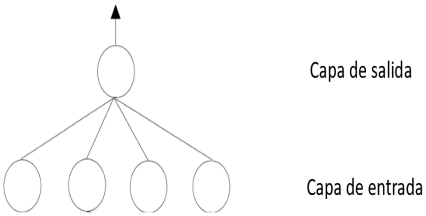


Figura 4.1: Perceptrón simple

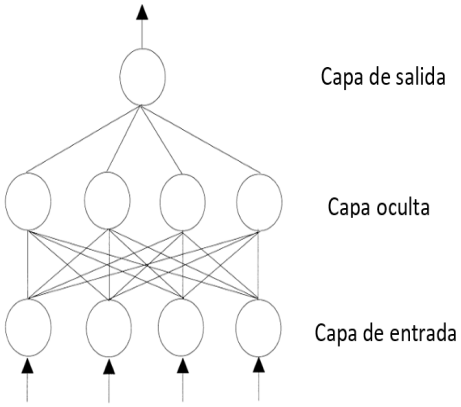


Figura 4.2: Perceptrón multicapa

En el contexto del análisis de la volatilidad de las series temporales financieras, el perceptrón multicapa adquiere un papel fundamental. Gracias a su diseño con capas de neuronas interconectadas que incluyen capas de entrada, ocultas y de salida, es capaz de modelar relaciones no lineales que a menudo se presentan en datos financieros. Esta capacidad de representar y aprender de patrones complejos y no lineales lo hace especialmente adecuado para capturar la volatilidad y la complejidad inherentes en las series temporales financieras.

4.1.2 Componentes clave y configuración en redes neuronales artificiales: neuronas, pesos, capa densa, operación flatten, inicializador de pesos (kernel) y épocas

Para comprender cómo funcionan las redes, es esencial analizar sus componentes básicos, así como su configuración. Los elementos descritos a continuación, no solo definen la arquitectura subyacente de las redes, también determinan su eficacia durante el aprendizaje y cómo dicho aprendizaje impacta en la resolución de problemas específicos.
Esta sección está enfocada en explicar cada uno de los principales componentes de la red:

- **Neuronas:** La neurona es una componente fundamental dentro de la arquitectura de la red neuronal. El concepto se inspira en las neuronas biológicas del cerebro humano. Cada neurona en una red neuronal artificial procesa la información recibida y produce una salida. Matemáticamente, esto se representa

mediante la fórmula:

$$a = f\left(\sum_{i=1}^{N} w_i x_i + b\right),$$

donde x_i representa las entradas a la neurona, w_i son los pesos asociados a estas entradas, b es el término de sesgo, y f es la función de activación. Esta fórmula refleja cómo cada neurona realiza una suma ponderada de sus entradas, que luego se transforma a través de la función de activación para producir la salida de la neurona.
Cada neurona está conectada a varias neuronas de la capa anterior y/o de la capa siguiente. Cada conexión tiene un 'peso' asociado que determina la importancia relativa de la entrada correspondiente al cálculo que realiza la neurona. La cantidad de neuronas indica la complejidad de la red en sí. Cuantas más neuronas tenga la red más fácil será que aprenda, pero si se tienen pocos datos, será muy probable que sobreajuste y que memorice en lugar de aprender [71]. En función de su pertenencia a cada capa, existen varios tipos de neuronas:

- **Neuronas de entrada:** a través de ellas se introducen los valores de las variables explicativas. Es habitual que dichos valores sean previamente estandarizados.
- **Neuronas de salida:** representan las variables objetivo o 'target' en la red neuronal. Su función es producir los valores finales que la red genera como salida:
  - Si la variable objetivo es categórica o nominal, el número de neuronas de salida generalmente será igual al número de categorías o clases posibles, menos uno. Esto se debe a que cada neurona de salida puede representar la probabilidad de que la entrada pertenezca a una clase particular.
  - Si la variable objetivo es de naturaleza continua, se empleará una única neurona de salida que proporcionará un valor numérico correspondiente a la predicción de la red.
  - Si existen múltiples variables objetivo, habrá varias neuronas de salida, cada una correspondiente a una variable objetivo distinta. En este caso, cada neurona de salida proporcionará una predicción independiente para su variable objetivo correspondiente.
- **Neuronas intermedias u ocultas:** reciben la información de las neuronas en las capas de entrada, y a través de una serie de cálculos y transformaciones, propagan la información procesada a las neuronas en las capas de salida. Son esenciales para el aprendizaje de representaciones internas y la captura de patrones y relaciones complejas en los datos de entrada.

- **Pesos:** Los pesos, son los parámetros que la red aprende durante el proceso de entrenamiento para minimizar la función de pérdida. Estos pesos se aplican a los datos de entrada para determinar la importancia relativa que va a tener cada entrada en la salida de la neurona [134]. Los pesos sinápticos (w_{ij}) representan la fuerza de una conexión sináptica entre dos neuronas, la presináptica "i" y la postsinápica "j". Si los pesos son próximos a cero, se considera que no existe comunicación entre el par de neuronas.

- **Capa densa:** Es una capa en la que todas las neuronas están conectadas a todas las neuronas en la capa anterior y a todas las neuronas en la capa siguiente. Cada una de las neuronas de esta capa, se compone de un peso asignado a cada una de sus conexiones a las neuronas de la capa anterior, así como un término de sesgo. Cada neurona toma la suma ponderada de sus entradas, aplica el término de sesgo, y posteriormente pasa el resultado a través de una función de activación para producir su salida [66].

- **Operación "flatten" en redes neuronales:** La operación flatten o de aplanamiento, tiene mucha relevancia en el procesamiento de datos en arquitecturas de redes neuronales, especialmente en aquellas que involucran capas convolucionales y de pooling. Esta operación implica convertir estructuras de datos multidimensionales (habitualmente creadas por capas anteriores de la red neuronal), en vectores unidimensionales. Esta transformación es importante para integrar de manera eficiente las capas convolucionales y recurrentes con capas densas, ya que permite aumentar significativamente la cantidad de parámetros de las capas densas posteriores en la red.

- **Inicializador de pesos (kernel):** El 'Inicializador de pesos' o 'Inicializador de kernel' es un método que establece los valores iniciales de los pesos en una red neuronal. El objetivo es proporcionar una buena inicialización que permita a la red aprender eficazmente a partir de los datos y converger a un buen modelo que minimice la función de pérdida. Esta inicialización de pesos tiene un gran impacto en el entrenamiento de redes neuronales, ya que puede influir en la convergencia y el rendimiento del modelo. Si los pesos se inicializan con valores muy grandes o muy pequeños, puede llevar a problemas como la desaparición o explosión del gradiente, dificultando de este modo el aprendizaje de la red. Por otro lado, si todos los pesos se inicializan con el mismo valor, todas las neuronas en una capa aprenderán las mismas características, lo que reduce la capacidad de la red para aprender patrones complejos en los datos [65].

- **Épocas, (epoch en inglés):** Hace referencia a un pase completo por todo el conjunto de datos de entrenamiento, durante el entrenamiento de la red neuronal. En cada época, el modelo intenta aprender patrones y características de los datos, y ajusta sus pesos y sesgos en función del error que cometió en sus predicciones y entrenamientos anteriores. El número de épocas es un parámetro que determina la cantidad de veces que se va a llevar a cabo el proceso de aprendizaje.

  Un número muy pequeño de épocas puede resultar en un aprendizaje insuficiente, mientras que un número excesivamente grande puede llevar a un sobreajuste del modelo, en donde la red neuronal aprende demasiado bien los detalles y el ruido de los datos de entrenamiento. Esto puede perjudicar la capacidad del modelo para generalizar y funcionar bien con datos nunca vistos [66].

4.2 Funciones de activación

Una función de activación es una función que se agrega a las neuronas de una red neuronal, con el objetivo de que la red pueda aprender relaciones complejas en los datos. Esta función de activación seleccionada actúa como un umbral de decisión para determinar qué información se transmite hacia las siguientes neuronas, añadiendo de esta forma complejidad a la red. Cabe señalar que la función de activación es no lineal, esto es esencial en el entendimiento de dichas funciones, ya que sin esta no linealidad, cada neurona de la red solo realizaría transformaciones lineales de las entradas usando los pesos y sesgos, realizando únicamente sumas ponderadas. Es decir, la función de activación actúa sobre la señal de salida proveniente de una neurona en una capa específica, transformándola en información de entrada para la siguiente neurona.
La elección de la función de activación ejerce una influencia determinante en la capacidad y rendimiento de la red neuronal. Se pueden utilizar distintas funciones de activación en diferentes segmentos del modelo, dependiendo de las características del problema y de los objetivos del aprendizaje. Una red neuronal típica consta de tres tipos de capas: entrada, oculta y salida. Por lo general, todas las capas ocultas comparten la misma una función de activación, mientras que la capa de salida adopta una función de activación distinta. Dicha selección depende del problema de aprendizaje supervisado específico que se va a resolver. Las funciones de activación deben de ser diferenciables y preferentemente continuas. Esta característica es esencial ya que el entrenamiento de las redes neuronales se lleva a cabo habitualmente mediante el algoritmo de retropropagación, el cual requiere el cálculo de la derivada de primer orden del error de predicción para actualizar los pesos del modelo en ser preferiblemente diferenciables y continuas.
Entre las funciones de activación más importantes se encuentran la sigmoide, la Tangente Hiperbólica, la Unidad Lineal Rectificada (ReLU), la Unidad Lineal Exponencialmente Escalada (SELU) y la función softmax. En este contexto, se va a introducir y analizar dichas funciones con el fin de comprender sus propiedades y aplicaciones en el diseño y entrenamiento de modelos de redes neuronales.

4.2.1 Función de activación sigmoide

La función sigmoide está definida como [164]:

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{4.1}$$

Esta función se distingue por una serie de características y limitaciones. En términos de sus características, tal y como se muestra en la Figura 4.3, la función tiene una forma de S y produce una salida que centrada en 0.5, con un rango que se extiende de 0 a 1. Esta función se utiliza frecuentemente en la capa final de los problemas de clasificación binaria debido a su capacidad para ser una función diferenciable.

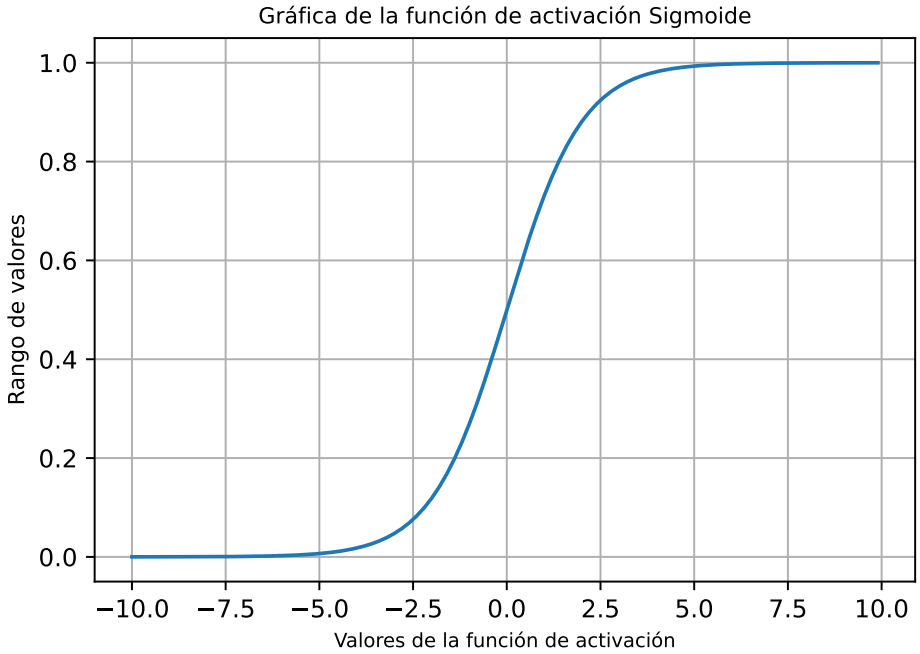


Figura 4.3: Función de activación sigmoide

Sin embargo, la función de activación sigmoide también tiene ciertas desventajas. Uno de los problemas más notables es la fuga del gradiente. Este problema ocurre cuando las entradas a la función son demasiado pequeñas o demasiado grandes, lo que provoca que la función se sature en 0 o 1. Como resultado, la derivada de la función se acerca demasiado a 0. Esto puede provocar que el gradiente que se propaga a través de la red sea casi nulo, dejando muy poco gradiente para las capas inferiores de la red. Además, la función de activación sigmoide puede ser costosa en términos computacionales. Esto se debe a que la función implica operaciones exponenciales, que son más costosas computacionalmente que otras operaciones matemáticas más simples.

4.2.2 Función de activación tangente hiperbólica (tanh)

La función de activación de la tangente hiperbólica queda representada del siguiente modo [86]:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{4.2}$$

La función de activación tangente hiperbólica presenta una serie de propiedades y limitaciones. En cuanto a sus propiedades, se puede observar en la Figura 4.4, que la función muestra una curva en forma de S, similar a la función de activación logística. A diferencia de la función de activación sigmoide, la tangente hiperbólica está centrada en 0 y su rango se extiende de -1 a 1. Al igual que la función sigmoide, la tangente hiperbólica es diferenciable. Aunque la función es monótona, su derivada no lo es.

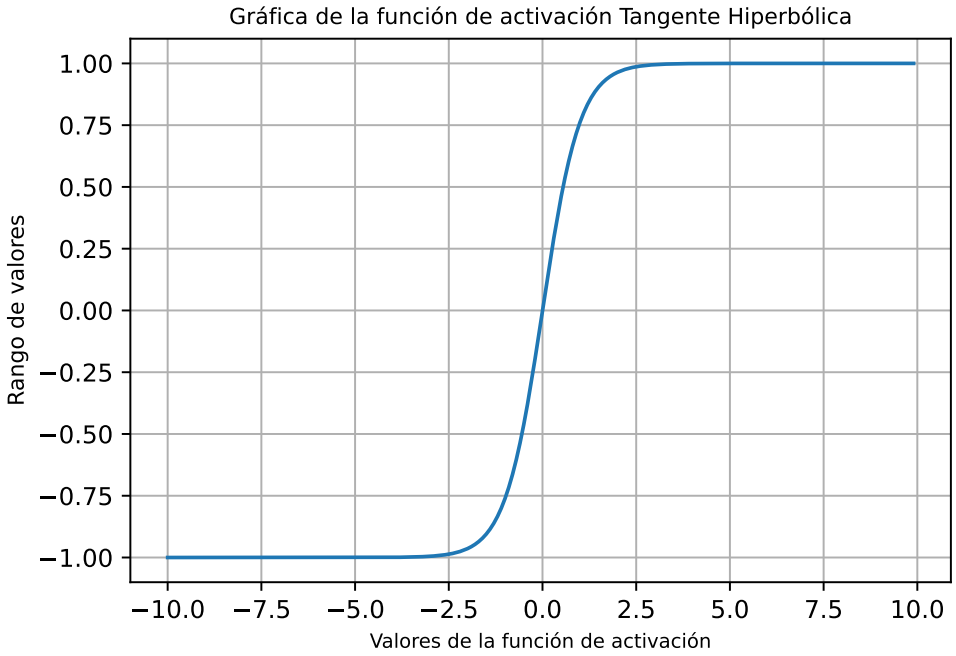


Figura 4.4: Función de activación Tangente hiperbólica

No obstante, la función de activación tangente hiperbólica también tiene ciertas limitaciones. Una de las más destacadas es el problema de la fuga de gradiente, que es similar al problema de la función sigmoide. Cuando las entradas a la función son muy pequeñas o muy grandes, la función se satura en 0 o 1, lo que lleva a una derivada muy cercana a 0. Esto puede dar como resultado un gradiente casi nulo que se propaga a través de la red, dejando muy poco gradiente para las capas inferiores de la red. Esta función puede resultar costosa en términos computacionales debido a que la función implica operaciones exponenciales, siendo este tipo de operaciones más costosas que otras más simples.

4.2.3 Función de activación Unidades Lineales Rectificadas (RELU)

La función ReLU es una función de activación muy popular en las redes neuronales debido a su simplicidad y eficiencia. Esta función es lineal para valores positivos y cero para valores negativos, lo que reduce el problema del desvanecimiento del gradiente en redes neuronales profundas. La función de activación RELU es muy utilizada en redes neuronales convolucionales, y su ecuación es [1]:

$$ReLU(x) = \max(0, x). \tag{4.3}$$

Una ventaja de la función de activación ReLU es que la activación es escasa, con aproximadamente el '50 %' de las unidades ocultas activadas en una red inicializada de manera aleatoria. Además, proporciona una mejor propagación del gradiente con menos problemas de fuga de gradiente en comparación con las funciones de

activación sigmoide y tangente hiperbólica. La ReLU también es eficiente en términos de cálculo, ya que solo implica operaciones de comparación, suma y multiplicación. Esto permite que sea más rápida de calcular, respecto las funciones sigmoide y tangente hiperbólica.
Por otro lado, la función de activación ReLU también tiene ciertas limitaciones. Una de ellas es que no está centrada en cero, tal y como se puede observar en la Figura 4.5. Además, aunque es diferenciable en cualquier valor, no lo es en 0, y el valor de la derivada en este punto puede elegirse arbitrariamente para ser 0 o 1. Otro problema es el de la 'ReLU moribunda', donde algunas neuronas dejan de producir algo distinto de 0 durante el entrenamiento, especialmente si se utiliza una tasa de aprendizaje alta. Cuando esto sucede, estas neuronas continuan generando ceros y ya no se ven afectadas por el descenso del gradiente, ya que el gradiente de la función ReLU es 0 cuando su entrada es negativa. Por último, debido a que la ReLU no tiene un límite superior, esto puede provocar activaciones de forma explosiva, lo que a veces resulta en nodos inutilizables.

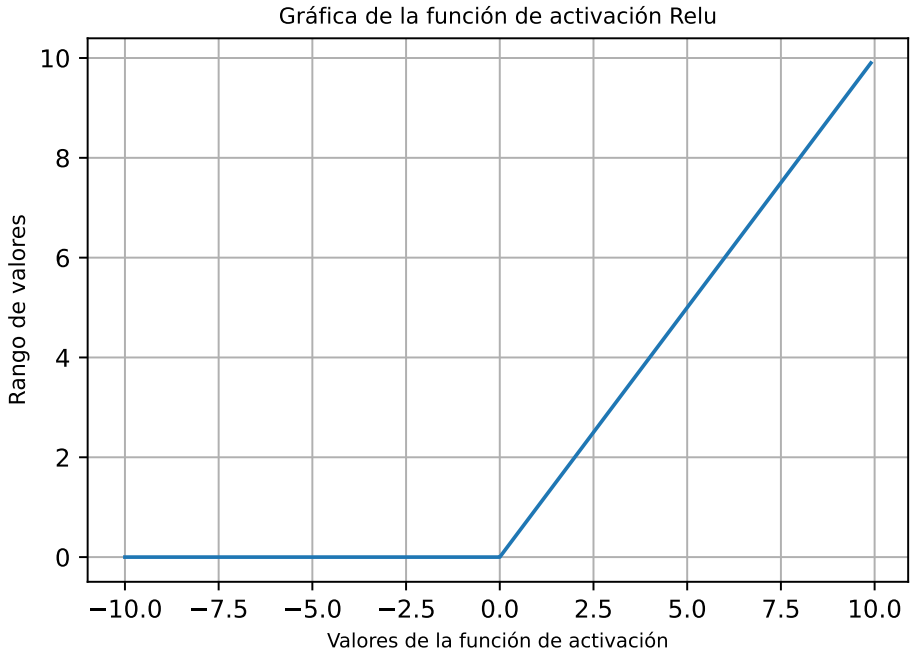


Figura 4.5: Función de activación relu

4.2.4 Función de activación Unidad lineal exponencial escalada (SELU)

Es una variante avanzada de la función Rectified Linear Unit (ReLU). Cuando se implementa una red neuronal exclusivamente compuesta por capas con la función de activación SELU, dicha red tiene la propiedad de auto-normalización [90]. Esta característica significa que durante el entrenamiento, la salida de cada capa se mantendrá con una media cercana a cero y una desviación estándar cercana a uno. Esta propiedad ayuda a resolver problemas críticos en el entrenamiento de redes neuronales como la desaparición y explosión de gradientes, a menudo resultan en un rendimiento superior en comparación con otras funciones de activación. La función SELU se define como:

$$SELU(x) = \lambda \begin{cases} x, & \text{si } x > 0 \\ \alpha(e^x - 1), & \text{si } x \leq 0. \end{cases} \tag{4.4}$$

Al igual que otras funciones de activación populares, como ReLU, SELU introduce una no linealidad en el modelo, lo que permite a la red aprender relaciones más complejas entre las características de entrada. Además,

SELU es diferenciable en todos los puntos, lo que facilita el cálculo de los gradientes durante el entrenamiento. A diferencia de la función ReLU, que puede causar el problema de las neuronas 'muertas' (neuronas que siempre se activan a 0), SELU tiene una salida negativa para las entradas negativas, tal y como se puede observar en la Figura 4.6. Esta característica puede ayudar a mantener las neuronas activas y a aliviar este problema.

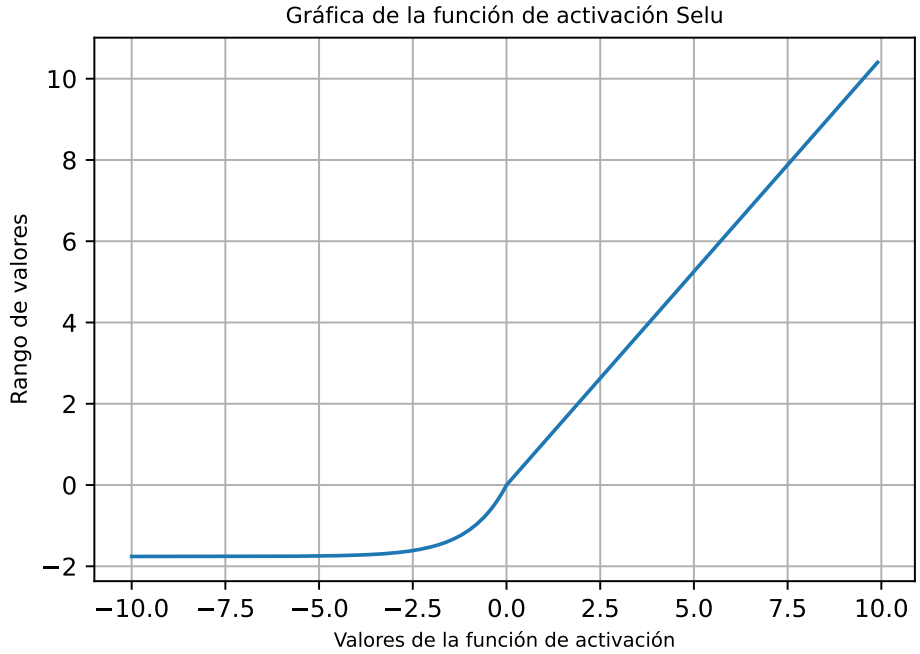


Figura 4.6: Función de activación Scaled Exponential Linear Unit (SELU)

La función de activación SELU también tiene algunas limitaciones que deben tenerse en cuenta. En primer lugar, SELU está diseñada principalmente para redes neuronales compuestas por capas densas, lo que significa que puede no ser la opción más adecuada para redes neuronales convolucionales. En segundo lugar, para que la función SELU funcione correctamente, es necesario que los pesos de cada capa oculta se inicialicen utilizando el método de inicialización normal de LeCun. Este requisito puede añadir complejidad al proceso de configuración de una red neuronal.
Finalmente, para que la función SELU realice su trabajo de manera efectiva, es necesario que las variables de entrada estén normalizadas con una media de 0 y una desviación estándar de 1. Este requisito de normalización puede requerir un preprocesamiento adicional de los datos, lo que puede aumentar la complejidad y el tiempo de procesamiento.

4.2.5 Función de activación Softmax

La función de activación Softmax se define matemáticamente como:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \tag{4.5}$$

donde $\mathbf{z}$ representa el vector de entrada de la capa de activación y k es el número de clases en el problema de clasificación. Esta función se utiliza ampliamente en las capas de salida de las redes neuronales para resolver problemas de clasificación multiclase. Su principal ventaja radica en poder convertir un vector de valores reales

en una distribución de probabilidad, donde las salidas tienen un rango de $[0, 1]$ y la suma de todas las salidas es igual a 1, tal y como se puede apreciar en la Figura 4.7.

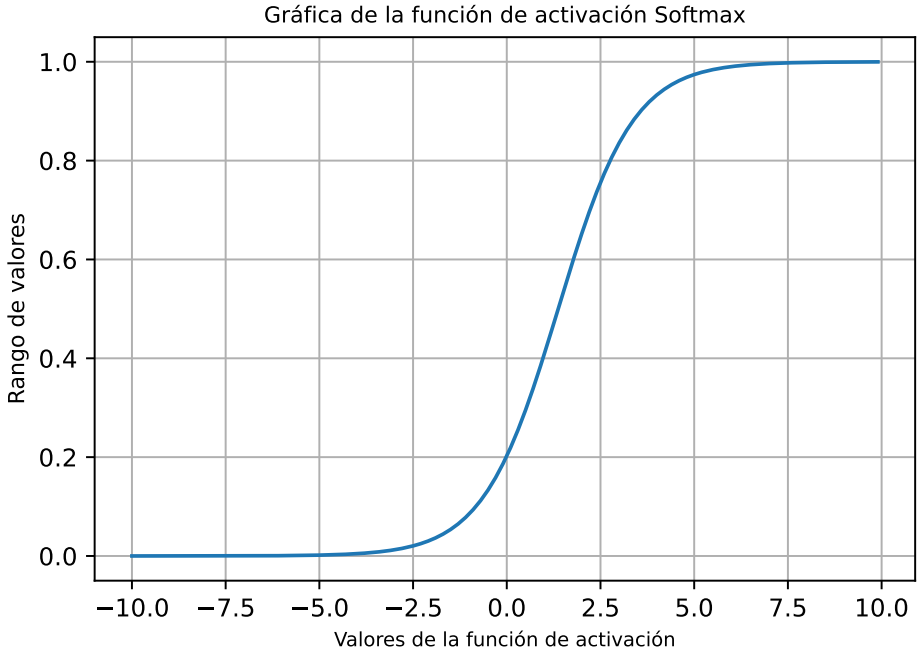


Figura 4.7: Función de activación Softmax

Una de las características distintivas de la función softmax es que es capaz de manejar varias clases. Esto no ocurre con la función sigmoidea, que solo puede identificar dos clases. Además, en problemas donde las clases no son mutuamente excluyentes, la función softmax sigue siendo una elección apropiada debido a su naturaleza probabilística. Sin embargo, la función softmax tiene limitaciones. Una de ellas es que puede ser computacionalmente intensiva debido a la operación exponencial y la normalización, especialmente cuando el número de clases k es grande.
En el contexto de las redes neuronales profundas, la función softmax juega un papel crucial importante en el proceso de entrenamiento. Es común combinarla con la función de pérdida de entropía cruzada, que mide la distancia entre la distribución de probabilidad predicha y la distribución de probabilidad real de las etiquetas de entrenamiento. La combinación de la función softmax y la entropía cruzada permite que el cálculo de gradiente informe sobre cómo ajustar los pesos de la red para minimizar el error de clasificación. La aplicación de la función softmax en la última capa de la red neuronal, permite que los resultados puedan interpretarse directamente como una probabilidad, facilitando de este modo la toma de decisiones en tareas de clasificación.

4.3 Función de coste

La función de coste (o función de pérdida o de error) es una medida escalar que tiene el propósito principal de cuantificar la diferencia que hay entre el valor de la predicción realizada por la red neuronal y el valor real. Esta diferencia de valores se utiliza como indicador cuantitativo respecto a la precisión de las predicciones de la red neuronal, siendo el objetivo de la red minimizar dicho error durante el proceso de entrenamiento, alcanzando de este modo un mínimo local o global. Este procedimiento de minimización se lleva a cabo mediante la optimización de los pesos y sesgos de la red [66].

Existen distintas funciones de coste en función del tipo de problema que se quiera resolver: por ejemplo, **regresión** o **clasificación**. A continuación, se va a detallar funciones de coste comúnmente utilizadas, junto con sus definiciones matemáticas (independientemente que sean para regresión o clasificación).

1. **Entropía cruzada (Cross-Entropy en inglés)**: La entropía cruzada se define como:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}), \tag{4.6}$$

   donde N es el número total de muestras, C es el número de clases, y_{ij} son los valores verdaderos y $\hat{y}_{ij}$ son las probabilidades predichas por la red neuronal. La entropía cruzada es una de las funciones de coste más comunes en problemas de clasificación. En este contexto, y_{ij} es 1 si la muestra i pertenece a la clase j y 0 en caso contrario, y $\hat{y}_{ij}$ es la probabilidad predicha de que la muestra i pertenezca a la clase j. Esta métrica es diferenciable, lo cual facilita la optimización. Además, al estar basada en el logaritmo, proporciona una penalización más severa para las predicciones incorrectas que están lejos de los valores verdaderos, favoreciendo de este modo la convergencia rápida [66]. A pesar de su eficacia, la interpretación de la entropía cruzada puede ser poco intuitiva, ya que no se expresa directamente en términos de error de clasificación.

2. **Error cuadrático medio (MSE, por sus siglas en inglés Mean Squared Error)**: Mide la calidad de un estimador al cuantificar cuánto se 'equivoca' en promedio con respecto a los valores reales observados. El error cuadrático medio se define como:

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{4.7}$$

   donde N es el número total de muestras, y_i son los valores reales y $\hat{y}_i$ son los valores predichos por la red neuronal. Su formulación consiste en calcular la media de los cuadrados de las diferencias entre los valores predichos o estimados y los valores reales [152, 70]. Al elevar al cuadrado dichas diferencias, se vuelve sensible respecto a valores atípicos, haciendo que los errores grandes sean penalizados especialmente.

3. **Error absoluto medio (MAE, por sus siglas en inglés Mean Absolute Error)**: El error absoluto medio se define como:

$$MAE(y, \hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|. \tag{4.8}$$

   Se obtiene calculando la media de las diferencias absolutas entre los valores verdaderos y los predichos. El uso del MAE puede conducir a una tasa de convergencia más lenta en comparación con otras funciones de pérdida, como el Error Cuadrático Medio (MSE). Sin embargo, el MAE penaliza menos los errores grandes, lo que puede hacerlo más robusto a valores atípicos. A pesar de estas complejidades, el MAE es altamente interpretable debido a su escala directamente comparable con los datos de entrada, lo cual facilita la evaluación del rendimiento del modelo [77]. Al igual que el MSE, el MAE puede ser útil para problemas de regresión.

4. **Raíz del error cuadrático medio (RMSE, por sus siglas en inglés Root Mean Squared Error)**: La raíz del error cuadrático medio raíz se define como:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}. \tag{4.9}$$

   Al estar los errores elevados al cuadrado, esta métrica va a penalizar de forma severa los errores de gran magnitud. También hace que esta métrica sea poco interpretable. A pesar de estos retos interpretativos, el RMSE se ha consolidado como una métrica muy útil en la optimización de modelos de regresión, gracias a ser diferenciable [77].

Es muy importante que al diseñar una red neuronal, se seleccione una función de coste acorde al problema a resolver, ya que esta decisión tiene un impacto significativo en la capacidad de la red para aprender de los datos. Una elección inadecuada de la función de coste puede llevar a un aprendizaje deficiente, sobreajuste o subajuste, y a resultados inferiores a los óptimos.

4.4 Regularización

Los modelos demasiado complejos tienden a sobreajustar. El sobreajuste, es un fenómeno común donde el modelo se ajusta muy bien a los datos de entrenamiento pero falla al generalizar a nuevos datos no vistos durante el entrenamiento [37]. El objetivo es encontrar un modelo que no sólo aprenda bien respecto los datos de entrenamiento, si no que además tenga un buen rendimiento con los datos no observados anteriormente. La regularización es una técnica para prevenir el sobreajuste del modelo, ya que cuando se aplica la regularización, se minimiza la complejidad del modelo y la función de coste de forma simultánea. Esto se debe a que los regularizadores permiten aplicar penalizaciones a los parámetros de la capa durante su optimización, y estas penalizaciones se incorporan en la función de coste que optimiza la red. De este modo el modelo se convierte en un modelo más simples tendiendo a generalizar mejor [120].

- **Lasso (L1, Least Absolute Shrinkage and Selection Operator en inglés):** La regularización L1 busca minimizar la suma de los valores absolutos de los pesos de la red. Esta técnica tiene una propiedad llamada 'selección de características' (feature selection), que consiste en hacer que algunos de los pesos de la red sean cero. Mediante esta propiedad, L1 ayuda a la creación de modelos más simples y comprensibles que a su vez generalizan mejor, gracias a la eliminación de características irrelevantes. También ayuda a prevenir que estos modelos tengan sobreajuste [146].

  La aplicación consiste en añadir un término adicional a la función de coste que es proporcional a la suma de los valores absolutos de los pesos. Este término de regularización penaliza a los modelos con pesos grandes, forzándolos a ser pequeños a menos que sean necesarios para reducir significativamente el error de entrenamiento. La fórmula matemática para la regularización L1 dada una función de coste, se puede expresar como:

  $$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \lambda \sum_{l=1}^{L} |w_l|, \tag{4.10}$$

  donde $J(W,b)$ es la función de coste regularizada, $L(\hat{y}^{(i)}, y^{(i)})$ es la función de coste para la i-ésima instancia de entrenamiento, λ es el parámetro de regularización, $|w_i|$ denota la norma L1 de los pesos en la capa l, que es simplemente la suma de los valores absolutos de estos pesos, y la suma sobre l se extiende a todas las capas de la red.

- **Ridge (L2):** Esta regularización, ayuda a evitar el sobreajuste al penalizar los pesos grandes en el modelo. Es decir, reduce la magnitud de los pesos pero no llegan a valer 0. Se aplica al igual que L1 en la función de coste de la red neuronal durante el entrenamiento, y tiene el efecto de suavizar el error y simplificar el modelo [115]. Esta minimización de los pesos, permite que el algoritmo generalice mejor y ayude a evitar el sobreajuste. Matemáticamente, se define como:

  $$J(W,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \lambda \sum_{l=1}^{L} w_l^2, \tag{4.11}$$

  donde $J(W,b)$ es la función de coste regularizada, $L(\hat{y}^{(i)}, y^{(i)})$ es la función de coste para la i-ésima instancia de entrenamiento, λ es el parámetro de regularización. Un valor de λ más alto resulta en más regularización y un modelo más simple, mientras que un valor de λ más bajo resulta en menos regularización y un modelo más complejo. w_l^2 denota el cuadrado de los pesos en el modelo.

- **Dropout:** El dropout ayuda a prevenir el sobreajuste en el proceso de entrenamiento de datos. Según un porcentaje especificado, selecciona una parte aleatoria de las neuronas y les asigna un valor de 0 durante las etapas de entrenamiento. Mientras se entrena, el dropout 'desactiva' de manera aleatoria ciertas neuronas en una capa, lo que obliga a la red a aprender representaciones redundantes y más resistentes. De modo que esas neuronas no se utilizan durante el entrenamiento. Este proceso podemos observarlo en la Figura 4.8. Durante el entrenamiento, el dropout 'apaga' aleatoriamente algunas neuronas de la red durante el entrenamiento de la red, lo que ayuda a prevenir el sobreajuste [142]. Esto se hace normalmente con una probabilidad fija (por ejemplo, 0.5) para cada neurona en una capa. Por medio de esta operación de apagado, se crean varias redes más pequeñas (subredes), siendo entrenadas en paralelo todas ellas. Cada subred adquiere una visión ligeramente diferente de los datos durante el entrenamiento, y esto promueve la robustez y la diversidad en la representación de los mismos.

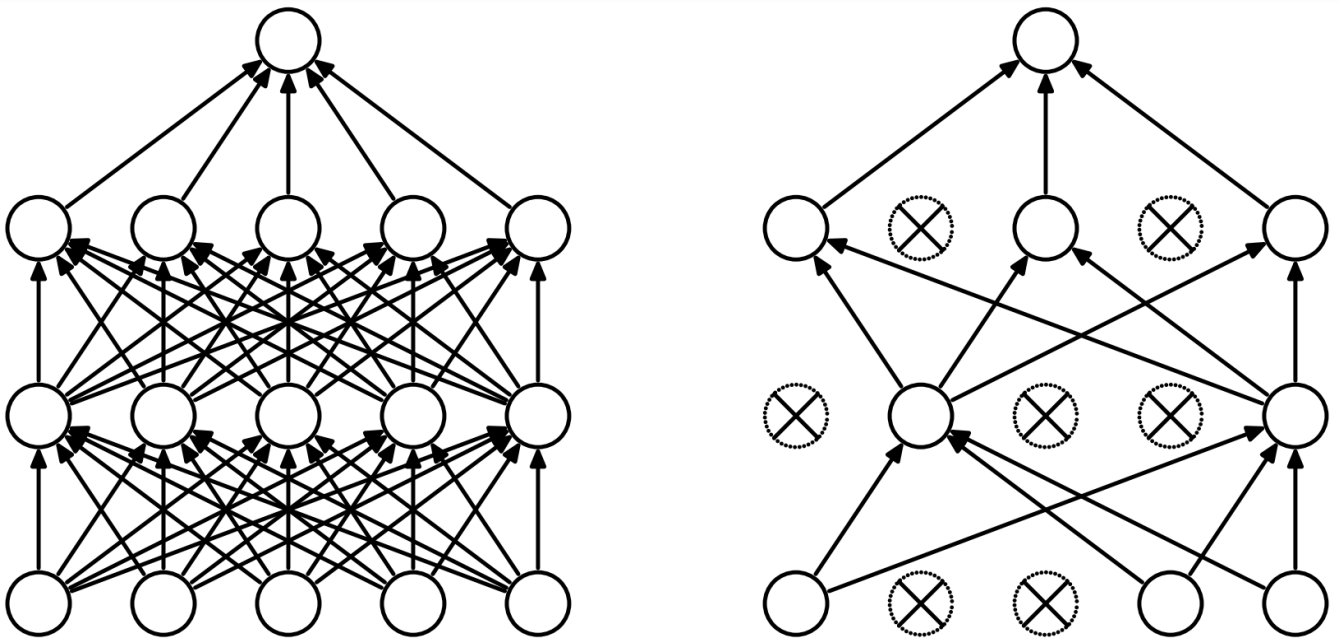


Figura 4.8: La Figura de la izquierda es una red neuronal con 2 capas ocultas, mientras que la Figura de la derecha es una red neuronal donde se ha aplicado dropout, desactivando ciertas neuronas para el entrenamiento de la red (Figura extraída de [142]). Figura extraída de 'Deep learning with gated recurrent unit networks for financial sequence predictions', Procedia Computer Science, Elsevier, 2018.

En la fase de evaluación o prueba, todas las unidades neuronales de la red se mantienen activas, pero sus salidas se multiplican por la probabilidad de dropout utilizada durante la fase de entrenamiento. Este ajuste es necesario para mantener la coherencia del valor esperado de las salidas entre las fases de entrenamiento y prueba. Esto conduce a que las decisiones finales del modelo sean producto de un tipo de consenso entre todas las subredes que fueron entrenadas a lo largo del proceso, donde cada subred puede considerarse como una interpretación única de los datos basada en un conjunto específico de neuronas activas.

La introducción de dropout tiene una consecuencia interesante y útil en términos de robustez del modelo: dado que cualquier neurona puede ser 'apagada' durante el entrenamiento, el modelo no puede depender excesivamente de la presencia de ninguna neurona o característica en particular. Esto obliga al modelo a aprender representaciones más robustas y redundantes de los datos, capturando los patrones subyacentes en los datos de una manera que no esté fuertemente ligada a la presencia de características específicas. Este aprendizaje de representaciones más generalizadas y robustas proporciona a menudo mejoras notables en la capacidad de generalización del modelo, en términos de su rendimiento con datos no vistos durante

el entrenamiento, reduciendo de este modo la probabilidad de overfitting.

- **Normalización por lotes, (Batch normalization por sus siglas en inglés):** Esta técnica fue propuesta por Sergey Ioffe y Christian Szegedy en 2015. Se desarrolló con la idea de mejorar el rendimiento y la estabilidad de las redes neuronales. La normalización por lotes consiste en centrar y normalizar cada mini-lote calculando la media y la desviación típica en cada uno de los mini-lotes que forman el conjunto de datos. De esta manera, se pueda obtener mini-lotes que tengan una media cercana a 0 y una desviación típica cercana a 1.
  El objetivo inicial es reducir el problema conocido como 'cambio de la covarianza interna' (internal covariate shift). Este problema hace referencia a la variación de las distribuciones de los inputs (datos de entrada) en cada capa de la red durante su entrenamiento, lo cual puede provocar un efecto negativo en la convergencia de la red [83]. Al reducir el problema de 'internal covariate shift', las redes pueden aplicar tasas de aprendizaje más altas, lo que facilita el uso de funciones de activación saturantes (es decir, aquellas que tienen a aplanarse y producir una salida constante a medida que la entrada se aleja de 0 en cualquier dirección). También hace que las redes sean menos sensibles a la inicialización de los pesos. También reduce el tiempo de optimización de la red, reduciendo la cantidad de épocas a utilizar.

- **Parada temprana, (Early stopping por sus siglas en inglés):** Esta técnica de regularización, interrumpirá el entrenamiento de la red si pasados un número 'n' de épocas el error del conjunto de validación deja de disminuir. Es una forma sencilla y eficaz de prevenir el sobreajuste. Asume que durante el entrenamiento de la red, comenzará a sobreajustar la modelización de dicho entrenamiento, dando lugar a un incremento en los errores de validación. Por lo tanto, si se consigue detener el entrenamiento en el momento en el que el error de validación comienza a aumentar, se evitará que el modelo llegue a sobreajustar [126].
  La elección de 'n' es muy importante para la modelización de la red, ya que un valor muy pequeño puede hacer que se detenga el entrenamiento de forma prematura, mientras que un valor de 'n' muy grande, puede dar lugar a un claro sobreajuste del mismo. Este algoritmo permite ahorrar tiempo de ejecución, ya que no es necesario que dicho modelo se entrene hasta el número total de épocas indicadas, si no que se va a detener tan pronto como deje de mejorar.

4.5 Entrenamiento de la red

Para poder entrenar eficientemente redes más complejas, se requería de una nueva técnica de aprendizaje. Esta necesidad condujo al desarrollo de un nuevo algoritmo llamado función de propagación. Para entender correctamente el funcionamiento de este algoritmo, se explicará posteriormente la regla de la cadena.

4.5.1 Función de propagación: forward propagation, backward propagation

Esta técnica ha proporcionado un método muy utilizado para el entrenamiento de redes neuronales multicapa, permitiendo el ajuste de los pesos y sesgos de cada neurona en función del error de salida. El proceso de entrenamiento de una red neuronal se divide en dos fases principales: la propagación hacia adelante (forward propagation) y la propagación hacia atrás (backward propagation).

- **Forward propagation:** Esta es la primera fase en el proceso de entrenamiento de una red neuronal. Durante la propagación hacia adelante, la red toma una entrada, realiza operaciones sucesivas en cada capa de la red, y produce una salida. Es esencial entender este proceso antes de poder entender la backpropagation o la regla de la cadena [94]. La función de propagación asociada a la neurona "j",

$(h_j(n))$, determina el potencial postsináptico resultante de la interacción de una neurona "j" con las neuronas precedentes "i" en el análisis del patrón (vector) "n". Esta regla resume, para una neurona "j", la información proveniente de una relación de neuronas "i" que "le entran", tal que:

$$h_j(n) = \sigma_j(w_{ij}, x_i(n)). \tag{4.12}$$

La regla de propagación más simple y utilizada consiste en sumar las entradas ponderadas por sus pesos sinápticos correspondientes del siguiente modo:

$$h_j(n) = \sigma_j(w_{ij}, x_i(n)) = \sum_i w_{ij} x_i(n). \tag{4.13}$$

- **Backward propagation:** Una vez que se ha calculado la función de pérdida, el siguiente paso es ajustar los pesos y sesgos de la red neuronal para tratar de minimizar dicha función y que se aproxime cada vez más a los valores verdaderos. Este proceso se realiza evaluando el error del output, siendo este error la diferencia que hay entre la salida de la red y la salida esperada o real (el output verdadero) [134]. Posteriormente, este ajuste se realiza a través de un proceso llamado 'propagación hacia atrás' donde el error se transmite a través de la red, modificando los pesos y sesgos de cada neurona de manera correspondiente.

4.5.2 Regla de la cadena

Para actualizar los parámetros de la red con el fin de minimizar la función de pérdida, es necesario calcular el gradiente de dicha función respecto a los parámetros de la red. Aquí es donde se implementa la regla de la cadena, descomponiendo el cálculo del gradiente en una serie de cálculos de derivadas más simples. La regla de la cadena es fundamental para el proceso de retropropagación, ya que permite calcular las derivadas parciales de la función de pérdida con respecto a cada parámetro en cada capa de la red, desde la salida de la red, es decir, comenzando por la última capa y retrocediendo hasta la primera capa, con cada uno de sus pesos [66].

4.5.3 Métodos de optimización

Los algoritmos de optimización juegan un papel vital en el proceso de entrenamiento de la red neuronal. El algoritmo de optimización más utilizado es 'Descenso por gradiente', y su objetivo es minimizar la función de pérdida o de coste en los datos de entrenamiento. Este algoritmo garantiza que durante la optimización se va a llegar a un mínimo local, pero no necesariamente se llegará a un mínimo local. La actualización de los parámetros del modelo se realiza utilizando la siguiente fórmula [66]:

$$x' = x - \epsilon \cdot \nabla_x f(x), \tag{4.14}$$

donde x representa los parámetros del modelo, $f(x)$ es la función de costo o pérdida, $\nabla_x f(x)$ denota el gradiente de la función de costo con respecto a los parámetros, ϵ es la tasa de aprendizaje, y donde x' y x son los valores de los parámetros después y antes de la actualización, respectivamente. El proceso del descenso de gradiente, se lleva a cabo a través de los siguientes pasos:
Se establece un punto de partida con los valores de los parámetros del modelo seleccionados de forma aleatoria. A continuación se calcula el gradiente de la función de pérdida para cada uno de los parámetros. Es importante remarcar que el gradiente indica la dirección en la que la función tiene su mayor incremento, pero como el objetivo será minimizar dicha función, el algoritmo se moverá en dirección opuesta al gradiente buscando el mínimo de la función de pérdida.
Este movimiento se logra actualizando cada parámetro, restando el gradiente multiplicado por un hiperparámetro

conocido como tasa de aprendizaje (learning rate en inglés). El valor que toma la tasa de aprendizaje es muy relevante en la optimización del algoritmo, ya que determina la magnitud del paso que se da en cada iteración, tal y como se puede observar en la Figura 4.9

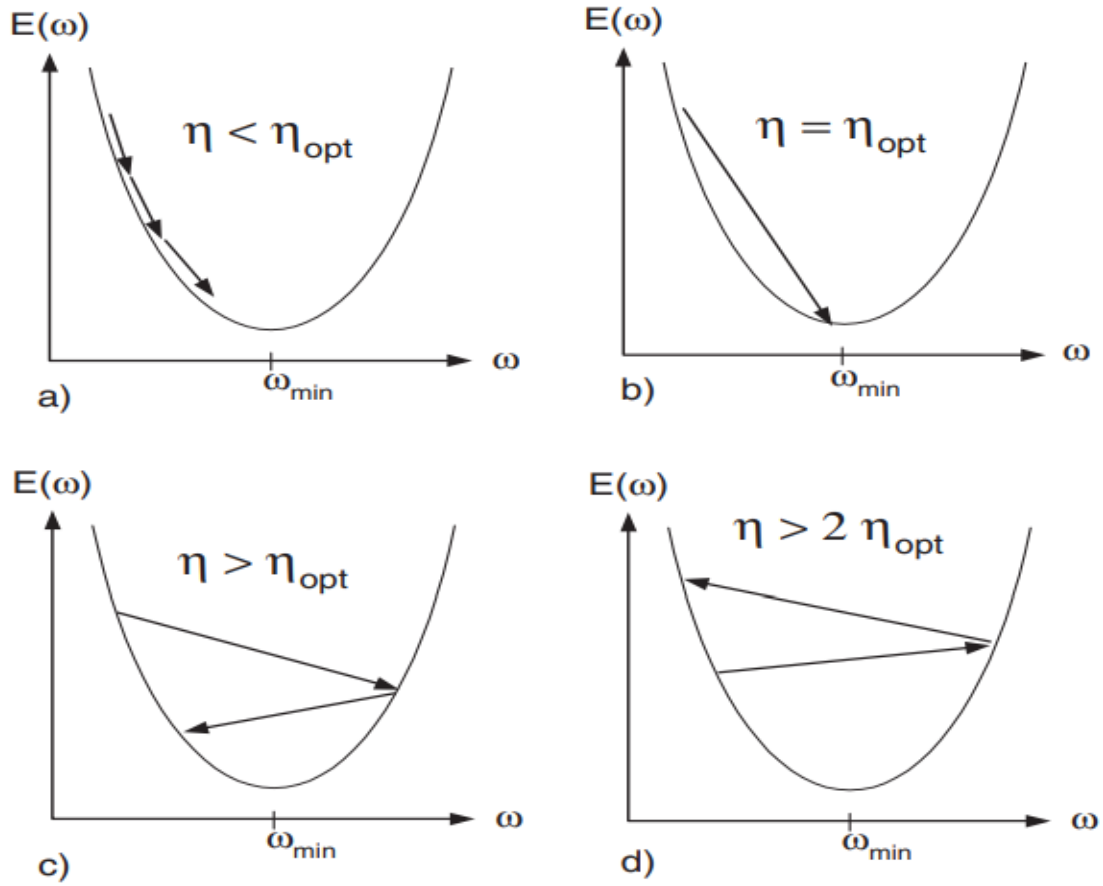


Figura 4.9: Comportamiento del algoritmo descenso del gradiente usando distintas tasas de aprendizaje [95]. Figura extraída de 'Efficient backprop', en Neural networks: Tricks of the trade, páginas 9–50, Springer, 2002.

Este proceso se repetirá hasta que se alcance un mínimo local de la función de pérdida, o se haya alcanzado el número máximo de iteraciones, también llamado, criterio de parada[95]. Existen una serie de versiones de dicho algoritmo que pueden mejorar la convergencia y evitar quedar atrapado en mínimos locales durante el proceso de minimización de la función de pérdida. A continuación se detallan algunas de estas variaciones.

- **Descenso de gradiente estocástico, Stochastic Gradient Descent por sus siglas en inglés (SGD):** El SGD introduce un elemento de aleatoriedad en cada iteración dentro del proceso de optimización, de modo que el gradiente se va a calcular y actualizar los parámetros en base a una única muestra seleccionada de forma aleatoria [26].
- **Momentum:** El método de momentum acelera el aprendizaje con el fin de evitar los mínimos locales. Introduce una variable de velocidad que acumula los gradientes anteriores acelerando de este modo la convergencia al mínimo de la función de pérdida. En cada iteración, la variable de velocidad se actualiza con el gradiente actual y una fracción de su valor anterior. De este modo, los parámetros del modelo se actualizan con esta variable de velocidad en lugar del gradiente directamente [143].
- **Descenso de gradiente acelerado de Nesterov, Nesterov accelerated gradient en inglés (NAG):**

Este método primero realiza un salto en la dirección del momentum acumulado anterior. Posteriormente calcula el gradiente y realiza una corrección. Esta anticipación permite a NAG frenar antes de que la pendiente de la función de pérdida se vuelva demasiado pronunciada, mejorando la eficiencia de la optimización [118].

- **Algoritmo de gradiente adaptativo, adaptive gradient algorithm por sus siglas en inglés (Adagrad):** Este método de optimización adapta la tasa de aprendizaje de cada parámetro de forma individual, basándose en la suma de los cuadrados de los gradientes pasados de ese parámetro. A los parámetros que tienen gradientes grandes se les asigna una tasa de aprendizaje más pequeña, mientras que a los parámetros con gradientes más pequeños se les asigna una tasa de aprendizaje más grande [54]. Este método funciona muy bien con parámetros que tienen gradientes muy pequeños. Como inconveniente, cabe destacar que la tasa de aprendizaje para cada parámetro disminuye monótonamente con el tiempo. Esto puede llevar a que la tasa de aprendizaje se vuelva muy pequeña y el entrenamiento del modelo se estanque.

- **Propagación de raíz cuadrática media, Root Mean Square Propagation en inglés (RMSprop):** Este método ha sido diseñado para abordar algunos de los problemas que surgen al usar Adagrad, principalmente, el problema de la disminución rápida y monótona de la tasa de aprendizaje. Al igual que Adagrad, RMSProp ajusta individualmente la tasa de aprendizaje de cada uno de los parámetros. RMSProp se distingue por el uso de una media móvil del cuadrado del gradiente, permitiéndole de este modo que el impacto de los gradientes antiguos en la tasa de aprendizaje disminuya con el tiempo, evitando así una disminución excesivamente rápida de la tasa de aprendizaje [75].

- **Estimación del momento adaptativo, Adaptive Moment Estimation en inglés (Adam):** Adam es un método de optimización que combina los aspectos de Momentum y RMSprop, ha demostrado ser eficaz en varias tareas de aprendizaje profundo. Al igual que RMSProp, utiliza medias móviles exponenciales de los gradientes para escalar la tasa de aprendizaje y, además, utiliza la media móvil exponencial del cuadrado de los gradientes. En relación con Momentum, añade una fracción de la dirección de la iteración anterior al parámetro actual [89].
  La combinación de estas características hace que Adam sea un algoritmo de optimización eficiente desde un punto de vista computacional y con un rendimiento robusto.

- **Adadelta:** Adadelta es una extensión más robusta del algoritmo Adagrad, desarrollada para abordar los problemas relacionados con el rápido y monótono descenso de la tasa de aprendizaje en Adagrad. Este problema puede dar lugar a una convergencia anticipada, antes de que el algoritmo alcance el mínimo óptimo. Para afrontar dicho problema, este método limita la acumulación de gradientes pasados a un tamaño de ventana fijo, utilizando un promedio móvil del cuadrados de los gradientes recientes.
  De modo que la actualización del parámetro en Adadelta depende tanto de la media móvil de los gradientes pasados como de la media móvil de las actualizaciones de los parámetros pasados [166].

Los algoritmos de optimización son fundamentales para el entrenamiento de redes neuronales. La elección del algoritmo puede depender de la tarea específica que se desea resolver, así como de la propia arquitectura de la red. La correcta aplicación puede ayudar a mejorar significativamente la precisión y eficiencia de los modelos de redes neuronales. En la Figura 4.10, se puede observar cómo se comportan los distintos métodos de optimización.

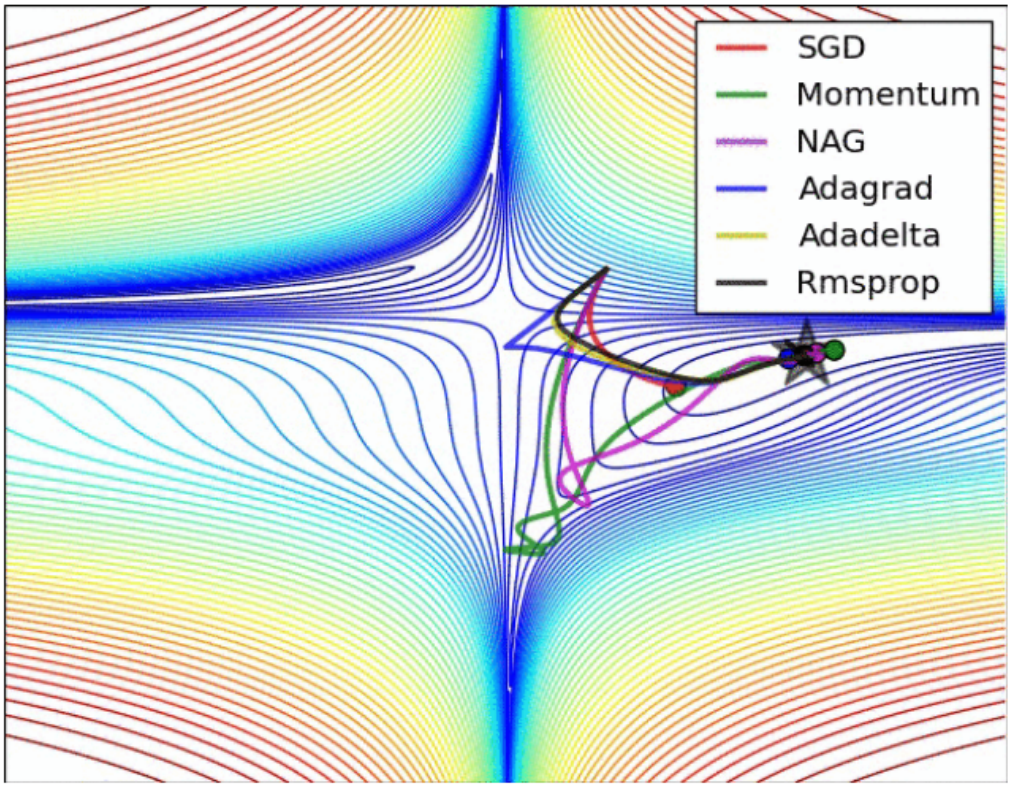


Figura 4.10: Camino que han tomado cada uno de los métodos de optimización, en los contornos de la superficie de la función de Beale (función de pérdida) [133]. Figura extraída de 'An overview of gradient descent optimization algorithms', arXiv preprint arXiv:1609.04747, 2016.

Como se puede apreciar, los distintos métodos han comenzado en el mismo punto, y que cada uno ha tomado un camino distinto hasta llegar al mínimo. Los métodos Adagrad, Adadelta y RMSProp convergieron rápidamente hacia el mínimo, mientras que Momentum y NAG se desviaron de la ruta del resto y tardaron más en llegar al mínimo. SGD fue el método que más tardó en converger, aunque comenzó en la dirección correcta. A continuación se pasa a detallar el marco teórico de las redes neuronales utilizadas con series temporales.

4.6 Capas específicas para datos temporales

Las redes neuronales, con su capacidad para modelar relaciones no lineales y complejas, son particularmente adecuadas para el análisis y la predicción en campos que tratan con grandes volúmenes de datos como las finanzas. En particular, las redes neuronales recurrentes (RNN, por sus siglas en inglés) y las redes neuronales convolucionales (CNNs, por sus siglas en inglés) han demostrado ser útiles para predecir movimientos de precios en los mercados financieros [137].

4.6.1 Redes neuronales recurrentes

Las redes neuronales recurrentes representan un avance significativo en relación con las redes neuronales tradicionales, como el perceptrón multicapa, debido a su capacidad para procesar secuencias de datos de longitud variable. Esta capacidad es especialmente relevante en contextos como la modelización de series temporales

financieras.

A diferencia de las redes con estructuras más simples y menos dinámicas, que están limitadas en su capacidad para manejar dependencias complejas en los datos, las RNN introducen una funcionalidad avanzada para el manejo de secuencias. En las RNN, la información de las salidas anteriores influye en las entradas actuales. Esto se logra mediante el uso de 'neuronas recurrentes', que permiten que la información persista y sea considerada en la toma de decisiones secuenciales, facilitando así el procesamiento de datos que presentan dependencias temporales. Las neuronas recurrentes son una pieza esencial en las RNN y se diferencian de las neuronas en una red neuronal regular en que también tienen bucles de retroalimentación (figura 4.11). Esto significa que no sólo procesan la entrada actual del paso de tiempo, sino que también mantienen y usan información de los pasos de tiempo anteriores en su cálculo. Su estado interno, o 'memoria', les permite retener información a lo largo del tiempo, facilitando el procesamiento de secuencias de datos y la captura de dependencias temporales [157].

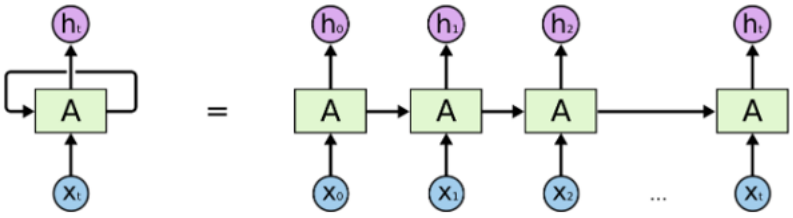


Figura 4.11: Desenrollando un bucle de retroalimentación de una RNN [121]. Figura extraída de 'Understanding LSTM Networks', disponible en https://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. Accedido: 2023-07-28.

Sin embargo, a pesar de las ventajas que las neuronas recurrentes proporcionan a las RNN, también introducen un desafío conocido como el problema del desvanecimiento del gradiente, donde los gradientes tienden a disminuir exponencialmente durante la retropropagación a través del tiempo, lo que dificulta el aprendizaje de la red a partir de errores en pasos de tiempo lejanos [19]. Para abordar este problema, se han desarrollado distintas variantes de las RNN. Destaca la Red de Memoria a Largo y Corto Plazo (LSTM), diseñada específicamente para evitar el desvanecimiento del gradiente. Las LSTM incorporan una estructura de 'puerta' que regula el flujo de información a través de la red, lo que les permite aprender dependencias a largo plazo [76]. De este modo, deciden explícitamente qué información debe guardarse y qué información debe descartarse en cada paso de tiempo, lo que les permite manejar de manera efectiva las dependencias a largo plazo.

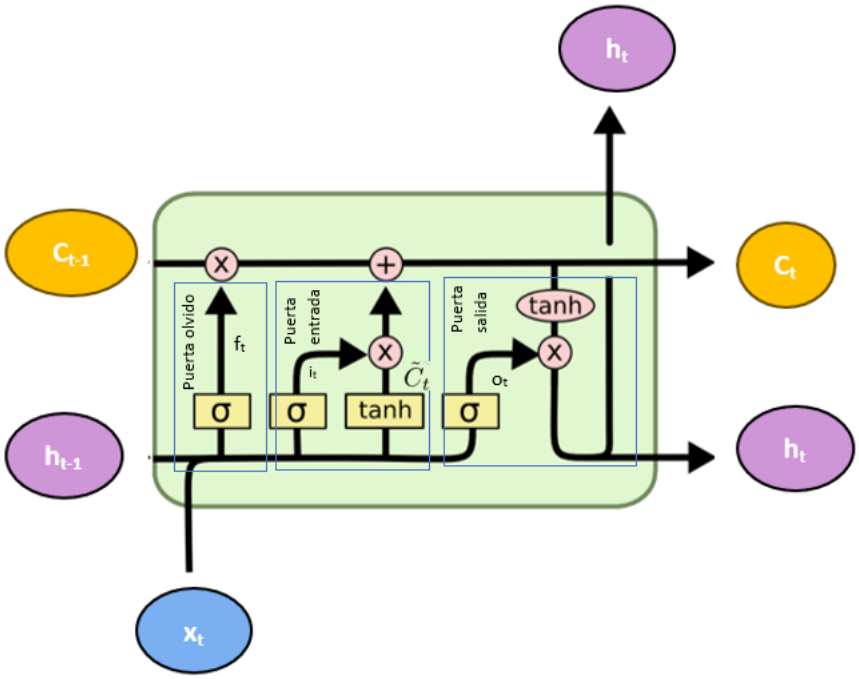


Figura 4.12: Arquitectura de una red neuronal de Memoria a Largo y Corto Plazo (LSTM)

Cada neurona de una red LSTM procesa una secuencia de datos paso a paso, tal y como vemos en la Figura 4.12, en cada paso de tiempo t, la red recibe una entrada x_t y la salida de la red LSTM del paso de tiempo anterior h_{t-1}. La LSTM tiene un estado interno C_t que se actualiza en cada paso de tiempo. El proceso de actualización utiliza tres puertas. La puerta de olvido f_t, actúa como un interruptor que controla cuánto de la memoria anterior C_{t-1} se retendrá o se 'olvidará'. Su fórmula es: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ donde W_f es la matriz de pesos para la puerta de olvido y b_f es el sesgo.
La puerta de entrada i_t decide cuánta de la nueva información se agregará al estado de la celda. Se calcula como: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ donde W_i es la matriz de pesos para la puerta de entrada, b_i es el sesgo y σ es la función sigmoidea que escala los valores entre 0 y 1. Al mismo tiempo que se genera la puerta de entrada, se genera una versión candidata $\tilde{C}_t$ de la actualización del estado de la celda utilizando la entrada actual y la salida anterior. Esta versión candidata puede considerarse como una propuesta para la actualización del estado de la celda, siendo su fórmula: $\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$. En esta ecuación, W_C y b_C son la matriz de pesos y el sesgo para el estado de la celda candidato, respectivamente. La función tangente hiperbólica, se utiliza para escalar los valores entre -1 y 1.
Hasta aquí se ha obtenido la información sobre cuánto de la memoria antigua se va a conservar por medio de la puerta del olvido, y cuánto de la nueva información se decide permitir por medio de la puerta de entrada. El siguiente paso es calcular la actualización del estado de la celda C_t como una combinación de las puertas de olvido y de entrada: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$. Una vez actualizado C_t, se calcula la salida h_t de la unidad en este paso de tiempo. Para ello se calcula la puerta de salida o_t que decide cuánto de la memoria actual se emitirá como salida. La fórmula es: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ donde W_o es la matriz de pesos para la puerta de salida y b_o es el sesgo. Por último, se genera la salida h_t aplicando la función tangente hiperbólica a la memoria actual y multiplicándola por la puerta de salida del siguiente modo: $h_t = o_t \odot tanh(C_t)$ [121].
Este proceso se repite para cada paso de tiempo en la secuencia de datos, lo que permite a la LSTM gestionar y propagar información a lo largo de toda la secuencia. A través de este mecanismo, las LSTM son capaces de aprender y recordar información relevante a largo plazo. En la predicción de series temporales, el contexto temporal es esencial. Si se está intentando predecir el precio de una acción, es importante tener en cuenta no sólo el precio actual de la acción sino también la evolución del precio en el pasado. Este tipo de red neuronal

es muy eficaz a la hora de tener en cuenta este contexto temporal [158], [111].
Además del desarrollo de la red LSTM, existen otras variantes de RNN como es el caso de la Red Neuronal Recurrente de Unidad recurrente cerrada (GRU, Gated Recurrent Unit), que incorpora mecanismos de puertas para controlar el flujo de información. Esta red fue introducida por Chung et al. (2014), [45], es una adaptación de la Red Neuronal de Memoria a Largo y Corto Plazo (LSTM) que simplifica su estructura, manteniendo su capacidad de capturar dependencias a largo plazo. Las GRU ajustan el fujo de información que va hacia la memoria a través de dos puertas: la puerta de actualización, que controla cuánta información del pasado se deja pasar al futuro, ayudando de este modo a evitar el problema de la desaparición del gradiente. Por otro lado, la puerta de reinicio determina cuánta información pasada va a ser olvidada. Este sistema ayuda a reducir el problema de la desaparición del gradiente y permitir que el modelo aprenda dependencias temporales de largo alcance. La operación de las puertas y el estado oculto en la GRU se pueden definir matemáticamente de la siguiente manera:

- **Puerta de actualización (z_t)**: Se define matemáticamente como:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]),$$

  donde z_t decide cuánta información del pasado (h_{t-1}) se debe llevar al estado actual h_t. La función sigmoide (σ) asegura que los valores estén entre 0 y 1.

- **Puerta de reinicio (r_t)**: Su formula se define como:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]),$$

  donde la puerta de reinicio r_t permite ajustar cuánta información anterior considerar al calcular el nuevo contenido de $\tilde{h}_t$. Un valor cercano a cero, cuando debe ignorar mucha información pasada.

- **Estado candidato ($\tilde{h}_t$)**: Este es un estado temporal basado en el almacenamiento que tiene la red en un momento dado. Se calcula combinando la nueva entrada con la información pasada relevante, filtrada por la puerta de reinicio.

$$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t]).$$

  Si la puerta de reinicio determina que la información pasada es importante, gran parte de esa información se utiliza para crear el estado candidato pertinente $\tilde{h}_t$.

- **Estado oculto (h_t)**: Es la memoria actualizada que la red lleva al siguiente paso en el tiempo. Se calcula en función del estado candidato y la información del estado oculto anterior, ajustado por la puerta de actualización del siguiente modo:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t,$$

  donde la puerta de actualización actúa como un filtro que determina la cantidad a conservar del estado oculto antiguo, así como cuánto estado candidato nuevo se debe incluir.

La GRU tiene menos parámetros que LSTM, ya que combina las puertas de entrada y olvido en la puerta de actualización, lo cual reduce la complejidad del modelo sin perder capacidad de captura de dependencias a largo plazo. Además, al tener menos parámetros, su entrenamiento es más eficiente, sin que esto afecte a su capacidad para capturar dependencias a largo plazo.

4.6.2 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN, por sus siglas en inglés) son un tipo de red neuronal artificial con aprendizaje supervisado que procesa sus capas imitando al córtex visual, ubicado en el polo posterior de la corteza occipital, permitiendo la identificación de distintas características en las entradas y, en última instancia, la identificación de objetos y su capacidad para 'ver' [129]. Las CNN se caracterizan por una estructura simple con pocos parámetros de entrenamiento y alta adaptabilidad, especialmente en procesamiento de imágenes [167]. Estas capas pueden conectarse entre sí para formar arquitecturas más profundas y complejas, de modo que la salida de una capa convolucional puede utilizarse como entrada para la siguiente capa. Esto permite a la red aprender representaciones jerárquicas y composicionales de los datos. También se caracterizan por tener múltiples capas ocultas, cada una con distintas especializaciones y jerarquías. Las primeras capas trantan de detectar elementos básicos como líneas y curvas [96], y a medida que se avanza hacia capas más profundas, estas se especializan en el reconocimiento de formas más complejas [49]. Estas redes neuronales también se utilizan para detectar patrones en series temporales [116], entre otros usos destacables.
Una operación importante dentro de las CNNs es la convolución. Matemáticamente, esta implica aplicar una función llamada kernel, sobre la función, normalmente una señal o imagen, para producir una tercera función que describe cómo el kernel se aplica a la señal o imagen en cada punto.

Redes neuronales convolucionales unidimensionales (1D-CNN)

Un kernel en una red convolucional unidimensional (1D-CNN), se define como una matriz unidimensional de pesos, que se aplica a una secuencia de datos siendo estos típicamente series temporales, mediante una operación de convolución. Esta operación involucra desplazar el kernel a lo largo del input o secuencia de entrada, y realizar el producto escalar entre los valores del kernel y la ventana correspondientes para los datos de entrada en cada posición. A medida que el kernel se va desplazando a lo largo de la serie, se calcula el producto escalar y se obtiene una nueva secuencia de valores, llamada mapa de características. Este mapa de características representa las características locales que el kernel ha detectado en los datos originales.
El filtro en una 1D-CNN se refiere a un conjunto de kernels, pero dado que en las 1D-CNN el filtro está constituido en esencia por un único kernel, los términos filtro y kernel se pueden utilizar como sinónimos. Cada filtro (o kernel) en una 1D-CNN está diseñado para identificar un tipo específico de característica de los datos de entrada. En el contexto de series temporales financieras, las capas convolucionales 1D tienen un especial protagonismo, ya que permiten extraer características relevantes a lo largo del tiempo [25]. Esto se puede apreciar en la Figura 4.13 muestra una representación esquemática de una serie temporal unidimensional. Esta estructura de datos es muy común en el ámbito financiero, donde cada uno de los elementos t, representa, por ejemplo, el precio de cierre de un activo bursátil. De modo que $t1$ sería el precio de cierre del primer día, $t2$ el precio de cierre del segundo día, y así sucesivamente. De este modo se crea una serie temporal únicamente con los precios de cierre de un activo bursátil concreto.

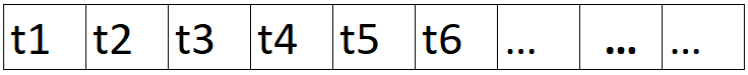


Figura 4.13: Estructura de datos de una dimensión

Estas capas aplican filtros convolucionales a lo largo de la dimensión temporal, lo que permite identificar patrones y estructuras temporales en los datos financieros. La elección del tamaño del filtro, la cantidad de filtros y el tipo de activación son hiperparámetros clave para el diseño de estas capas. Los filtros convolucionales en una capa convolucional 1D son vectores de pesos que se desplazan a lo largo de la dimensión temporal de los datos. Este cambio o desplazamiento, hace referencia a la aplicación secuencial de filtros a diferentes partes

de la serie temporal, avanzando a través de los datos paso a paso. Cada posición del filtro corresponde a una ventana temporal específica sobre la cual se realiza la convolución, aplicando los pesos del filtro a los datos de esa ventana. La cantidad de filtros determina la cantidad de patrones distintos que la capa convolucional puede aprender a detectar en los datos. El tamaño del filtro, determina la ventana temporal durante la cual se analiza el comportamiento y se evalúan los patrones de las series temporales. Un tamaño de filtro más pequeño permite a la capa convolucional detectar cambios y rápidos y variaciones locales en los datos, tales como la volatilidad y el ruido en series temporales. Por otro lado, un filtro más grande permite capturar patrones a lo largo de un rango temporal más amplio, incluyendo tendencias y patrones a largo plazo [87].
Las capas convolucionales 1D pueden aprender a detectar patrones y características locales a lo largo de la dimensión temporal de los datos. Esto es especialmente útil en el análisis de series temporales financieras, donde la detección de patrones temporales puede ser clave para predecir cambios futuros en los precios y otros indicadores financieros [25]. La capacidad de elegir la cantidad de filtros y su tamaño proporciona un control granular sobre la capacidad del modelo y la complejidad de las características aprendidas.
Debido a que las capas convolucionales 1D están diseñadas para trabajar secuencialmente a lo largo de una sola dimensión, hay ciertos tipos de series temporales financieras multidimensionales como datos de alta frecuencia, donde los datos no solo se obtienen a través de los precios, sino también de los volúmenes y otras variables de mercado. Cuando se trabaja con este tipo de datos, las capas convolucionales 1D presentan limitaciones para capatar la totalidad de la estructura de los datos. Es por ello por lo que las capas convolucionales 2D son ampliamente utilizadas para este tipo de situaciones, ya que son capaces de capturar patrones complejos a través de las múltiples dimensiones de los datos [148]. No obstante, las capas convolucionales de una dimensión tambien pueden ser multicanales. Esto quiere decir que pueden tomar como entrada múltiples series temporales, o 'canales', a la vez. Cada canal puede representar una característica diferente de los datos financieros. Por ejemplo, un canal podría ser los precios de cierre diarios de una acción, mientras que otro canal podría ser el volumen de negociación diario de esa acción. Al procesar múltiples canales a la vez, la capa 1D puede aprender a detectar patrones que involucran múltiples características a la vez.

Redes neuronales convolucionales bidimensionales (2D-CNN)

En lo que respecta a las redes neuronales convolucionales bidimensionales (2D-CNN), un kernel es una matriz bidimensional de pesos que se aplica a los datos de entrada (por ejemplo, una imagen) para realizar una operación de convolución. Esta operación implica desplazar el kernel sobre la imagen completa, computando en cada posición el producto escalar entre los valores del kernel y los valores de la imagen bajo este. El resultado es un mapa de características que refleja ciertos aspectos de la imagen, como bordes, texturas o patrones específicos. Estas capas requieren de más parámetros durante su entrenamiento, y como consecuencia de ello, las capas convolucionales bidimensionales requieren de más tiempo de entrenamiento, en comparación con las capas convolucionales unidimensionales.
En cuanto al filtro en una 2D-CNN, se trata de un conjunto de varios kernels donde cada uno de los cuales se aplica a la misma imagen de entrada. Cada kernel dentro de un filtro se encarga de detectar diferentes tipos de características en la imagen. Cada kernel genera un mapa de características, y el conjunto de todos los mapas de características generados por cada kernel dentro de un filtro proporciona una representación comprensiva de las diversas características presentes en la imagen. Cada filtro en una red neuronal convolucional bidimensional (2D-CNN) consta de un único kernel multidimensional, con el fin de explorar y detectar diferentes tipos de características en los datos de entrada. Este kernel se aplica a los datos de entrada, que pueden ser representados en múltiples canales. Por ejemplo, en el caso del procesamiento de imágenes, estos canales podrían ser los distintos colores de una imagen RGB.
Una vez aplicado cada filtro, se genera un mapa específico de características, y cada uno de estos mapas reflejará patrones o aspectos específicos detectados por ese filtro. El conjunto de todos estos mapas de características (uno por cada filtro), compone el volumen de salida de la capa. La ecuación de convolución continua se define

como [66]:

$$s(t) = \int x(a)w(t-a)da, \tag{4.15}$$

donde $s(t)$ es la señal de salida obtenida al aplicar la función de peso $w(t)$ a la señal de entrada $x(a)$. Esta operación se aplica en el contexto de señales continuas. Al procesar señales discretas, como píxeles en una imagen digital, la convolución se aplica a un dominio discreto y se expresa como:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{4.16}$$

donde el sumatorio se extiende sobre el rango finito de x y w, que son finitos.
La operación de convolución en bidimensional para el procesamiento de imágenes o matrices de datos en las CNNs se define por

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n), \tag{4.17}$$

donde $S[i,j]$ es el valor del elemento en la posición $[i,j]$ del mapa de características resultante. Aquí, I representa la imagen de entrada y K el kernel. La suma se realiza sobre todas las posiciones m y n, superponiendo el kernel K y la imagen I.
A continuación, en la Figura 4.14 se muestra como la red toma de entrada los píxeles de una imagen. Por ejemplo, una imagen de 28x28 píxeles en escala de grises tendría 784 neuronas de entrada.

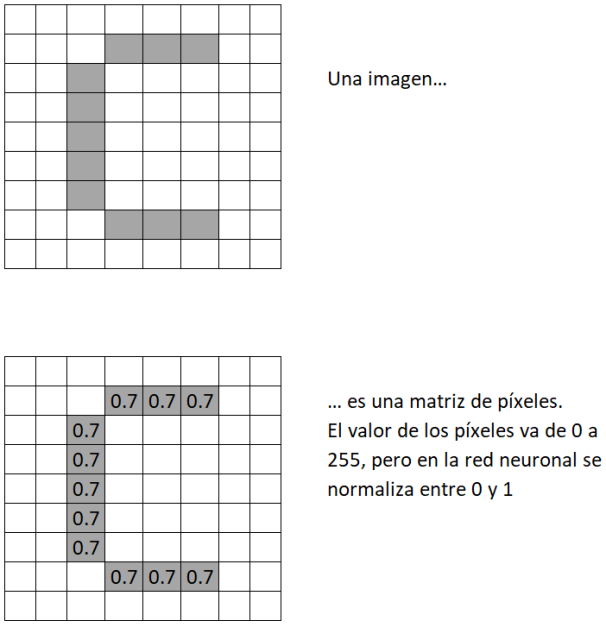


Figura 4.14: Estructura 2D blanco y negro

Si se tuviera una imagen a color (tal y como se puede observar en 4.15), se necesitaría 3 canales (rojo, verde y azul) y se usaría 28x28x3 = 2352 neuronas de entrada. Es importante normalizar los valores de entrada, transformando cada píxel mediante la operación 'valor/255' para obtener valores entre 0 y 1. Esta normalización garantiza que todas las características de entrada tengan el mismo rango de valores (ya que en las imágenes sin procesar, los píxeles varían de 0 a 255). Cuando dichos valores están normalizados, la red se entrena con más velocidad y con mayor estabilidad, ya que todas las características de entrada están en escalas similares. En cambio, si los valores de entrada no están normalizados, las neuronas de las primeras capas de la red pueden recibir entradas con grandes variaciones en sus magnitudes, pudiendo dar lugar a que algunas características

dominen sobre otras durante el entrenamiento de la red neuronal. Esto da lugar a un ajuste desproporcionado de los pesos asociados con ciertas neuronas durante el entrenamiento, lo que puede producir sesgos en el modelo y afectar al rendimiento general.
Si los valores de entrada son muy altos, existe el riesgo de que las neuronas se saturen, es decir, que se activen en regiones planas de la función de activación donde la pendiente es casi nula. Esto puede detener el proceso de aprendizaje ya que los ajustes necesarios en los pesos y sesgos de la red se vuelven muy pequeños.

Por otro lado, no será necesario normalizar si el rango de valores de datos originales contiene información importante que puede perderse durante la normalización. Por ejemplo, en algunas aplicaciones de imágenes médicas, las intensidades de píxeles precisas pueden ser fundamentales para la detección de enfermedades. Tampoco será necesario en algunas arquitecturas de redes neuronales, especialmente aquellas que utilizan normalización por lotes (batch normalization), pueden manejar eficientemente entradas no normalizadas, ya que estas capas ajustan la escala de los datos durante el entrenamiento.

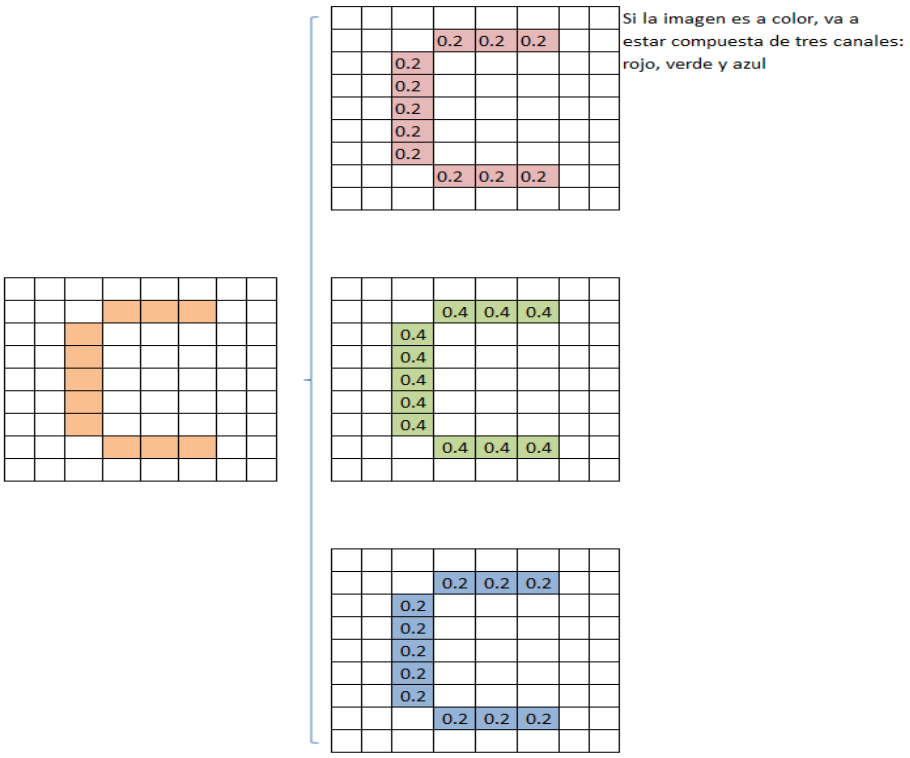


Figura 4.15: Estructura 2D a color

Estas capas también presentan robustez a pequeñas transformaciones en los datos, como desplazamientos y rotaciones [42].

Arquitectura CNN

En esta tesis, se pone foco en el uso de las CNN para el análisis de series temporales financieras, explorando las capas convolucionales unidimensionales (1D) y bidimensionales (2D), las diferentes operaciones y tipos de capas utilizadas, y cómo estas técnicas pueden aplicarse al estudio de datos financieros. En la Figura 4.16 se puede observar cómo se estructura una CNN con 4 capas convolucionales.
El input inicialmente tiene unas dimensiones de 150x150 y profundidad de 3 canales. Estos datos de entrada

pasan a través de una primera capa convolucional, donde 32 filtros distintos generan 32 mapas de características cada uno de un tamaño de 148x148. Esto se debe a que los filtros son de 3xx3 píxeles, modificando de ese modo el tamaño de los datos (150-3+1 debido al padding de 1 píxel). Cada uno de los mapas de características captura particularidades específicas de dichos datos.
A continuación los datos fluyen hacia la capa de max-pooling donde sigue reduciendo las dimensiones de cada mapa de características a la mitad, de 128x148 a 74x74. Esto se debe a que la ventana móvil de 2x2 que se aplica, se desplaza con un stride=2.
Posteriormente, una segunda capa convolucional incrementa la profundidad de los mapas de características a 64 filtros a cada uno de los 32 mapas anteriores de 74x74. Con un stride =1 y sin padding, se recalcula el tamaño de la salida: 74-3+1=72. Obteniendo así mapas de 72x72. Estos mapas proporcionan una representación más compleja en un espacio más reducido (72x72). Unido a ello, una segunda capa de max-pooling con una ventana de 2x2 y un stride=2, reduce aún más esta dimensión espacial, de 72 a 36, produciendo mapas de 36x36.
El proceso se repite en una tercera y cuarta capa convolucional y max-pooling respectivamente, finalizando el proceso con 32 canales y una dimensión del mapa de características de 3x3. Es entonces cuando la capa densa toma el vector aplanado de las características (32 canales de 3x3, totalizando 288 características) y lo transforma a través de una o más capas completamente conectadas. Esta transformación pondera las características para producir un vector de salida (output), que finalmente llevan a una salida de 6 valores, a menudo mediante la capa de activación softmax que distribuye las probabilidades de pertenencia entre las distitnas clases posibles.

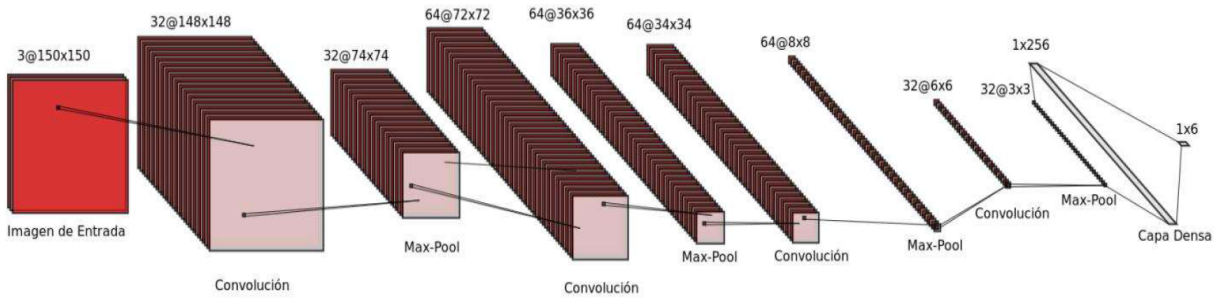


Figura 4.16: Arquitectura básica de una red neuronal convolucional con cuatro capas convolucionales [150]. Figura extraída de 'Diseño de una arquitectura de red neuronal convolucional para la clasificación de objetos', publicado en Ciencia Nicolaita, número 81, páginas 46-61, 2020.

Proceso de Convolución

El proceso de convolución se lleva a cabo desplazando los filtros a lo largo de la imagen de entrada y calculando el producto escalar en cada posición. Esta operación produce una serie de mapas de características que reflejan la presencia de diversos patrones en la imagen original. Este proceso se puede ver en la Figura 4.17

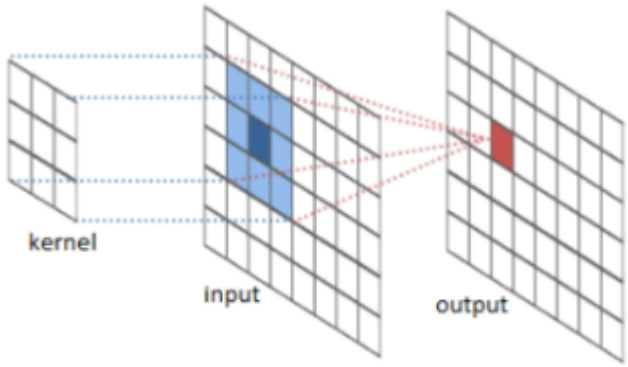


Figura 4.17: Proceso de convolución usando kernel 3x3 [50]. Imagen extraída de 'Visión artificial: Redes convolucionales (CNN)', El Laberinto de Falken, 2019.

En el caso de imágenes a color, el kernel tendría una dimensión de 3x3x3, es decir, un filtro con tres kernels de 3x3. Posteriormente, se suma la salida del filtro, junto con una unidad bias, dando como resultado una única salida (como si fuera un solo canal). Este proceso está ilustrado en la Figura 4.18, donde podemos ver que la aplicación independiente de cada kernel facilita la extracción de características esenciales y además permite integrar distintas señales en un tensor más complejo. Se puede observar como la suma de las contribuciones de cada canal, más el ajuste del bias, permite construir representaciones abstractas que posteriormente pasarán a ser fundamentales para las futuras capas de la CNN.

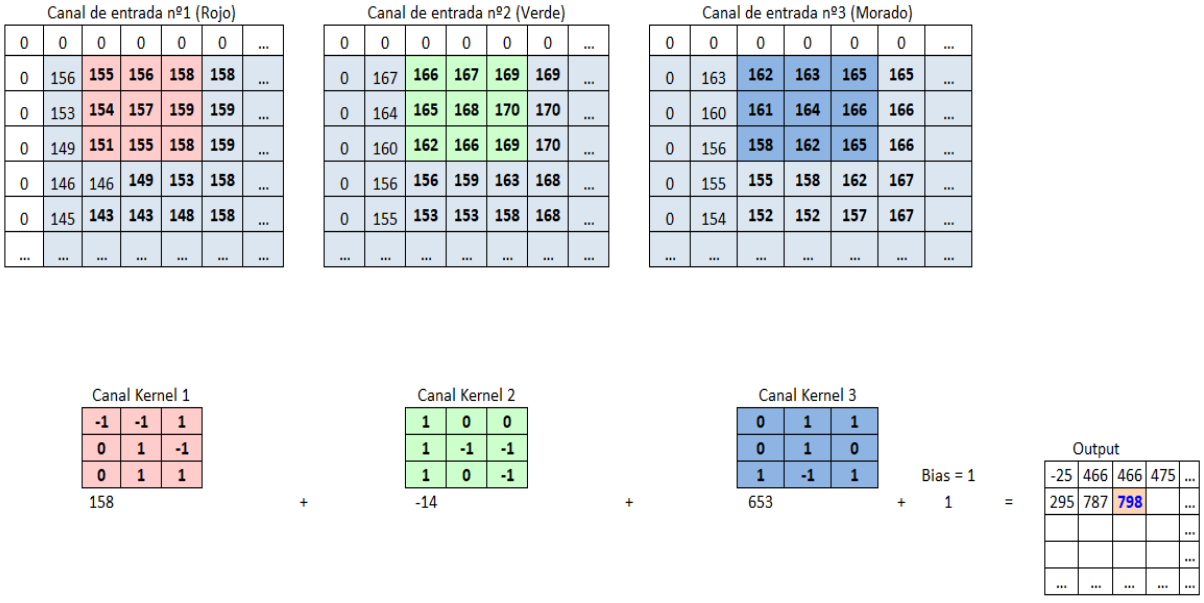


Figura 4.18: Proceso de convolución en redes neuronales convolucionales. La Figura muestra la convolución de una entrada multicanal con tres kernels específicos para cada canal de color (rojo, verde y azul), seguido de la suma de los resultados y la aplicación de un bias. El resultado es un mapa de características de salida que sintetiza la información relevante detectada por cada filtro, demostrando cómo la operación de convolución fusiona distintos aspectos de la entrada en una output, unificando el resultado.

Cabe destacar que, en lugar de aplicar un único kernel, se emplean múltiples kernels (cuyo conjunto se denomina filtros). La cantidad de neuronas en una CNNs, permite calcular la cantidad de parámetros a aprender y cómo de compleja será la red, computacionalmente hablando. Por ejemplo, suponiendo una imagen de entrada

de tamaño 28x28x3, (siendo ancho x alto x número de canales respectivamente), un kernel de tamaño 3x3, un stride de 1, y un padding "valid"(sin padding), se procede a calcular el número de neuronas una vez aplicada la operación de convolución.

$$\text{Tamaño del mapa de características} = \left(\frac{\text{Dimensión de la imagen} - \text{Tamaño del kernel}}{\text{Stride}}\right) + 1 \tag{4.18}$$

Aplicando la fórmula a cada dimensión espacial tenemos:

$$\text{Tamaño del mapa de características} = \left(\frac{28-3}{1}\right) + 1 = 26. \tag{4.19}$$

Por lo tanto, cada mapa de características tendrá un tamaño de 26x26. Para una operación de convolución con un único filtro que genera un mapa de características, el número total de neuronas es:

$$\text{Número total de neuronas} = 26 \times 26 = 676. \tag{4.20}$$

Por lo tanto, dadas las especificaciones anteriores, el número total de neuronas es de 676 en la capa convolucional. Esta cantidad representa las unidades activas en el mapa de características cuando solo tenemos un filtro. Si hubiera más de un filtro, este número se multiplicaría por la cantidad de filtros utilizados. Cada filtro produce un mapa de características independiente. Por ejemplo: si tuviéramos 32 filtros, el número total de neuronas sería 676 x 32.

Padding en redes neuronales convolucionales

Al abordar el proceso de aplicar un filtro convolucional a los datos de entrada, surge un problema relacionado con el encaje del filtro en los bordes de los datos. Dicha problemática ha sido objeto de estudio [98] y se han propuesto diferentes enfoques para mitigarla, conocidos como 'padding' (relleno). En la literatura, se identifican principalmente tres tipos de padding: 'valid', 'same' y 'full'. En la Figura 4.19 se muestra cómo actúa cada uno de estos filtros en el conjunto de datos.

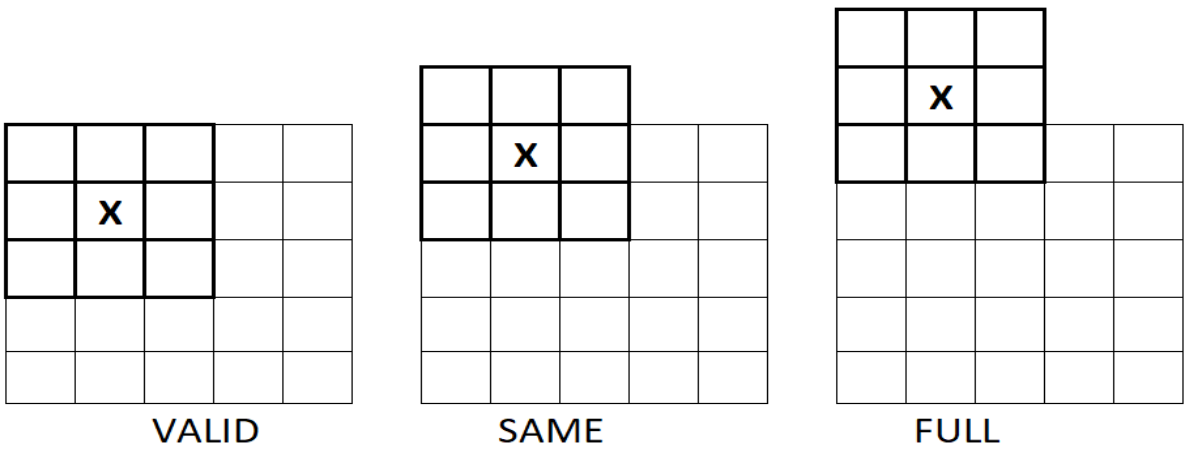


Figura 4.19: Tipos de padding. "Valid"no añade bordes y reduce la dimensionalidad del output. "Sameçonserva las dimensiones originales del input. "Full"permite maximizar la inclusión de las características de los bordes, y ello aumenta a su vez las dimensiones de salida.

A continuación, se describen en detalle cada uno de estos tipos [55].

- **Valid:** El padding 'valid' es un enfoque que no introduce ningún relleno adicional en los datos de entrada. En este caso, el filtro convolucional se aplica únicamente a las posiciones donde encaja por completo dentro de los datos. Como consecuencia, la dimensión de los datos de salida se verá reducida, respecto a la dimensión de los datos de entrada. La fórmula para calcular el tamaño del output O es:

$$O = \left\lfloor \frac{I - K}{S} \right\rfloor + 1, \tag{4.21}$$

donde I es el tamaño de la entrada, K es el tamaño del kernel, y S es el stride. El relleno P es cero, ya que no se introduce relleno adicional junto a los datos de entrada.

Ejemplo: Datos de entrada con tamaño $I = 7$, un kernel con tamaño $K = 3$, y un stride $S = 1$. Aplicando el padding 'valid', no se añade relleno adicional, así que $P = 0$. La fórmula para calcular el tamaño del output O es:

$$O = \left\lfloor \frac{I - K}{S} \right\rfloor + 1 = \left\lfloor \frac{7 - 3}{1} \right\rfloor + 1 = 5. \tag{4.22}$$

Por lo tanto, el tamaño del output es 5.

- **Same:** El padding 'same' consiste en agregar relleno suficiente a los datos de entrada de tal manera que la dimensión de los datos de salida sea igual a la de los datos de entrada, siempre que se asuma un stride de uno. Este enfoque permite que el filtro convolucional se aplique incluso en los bordes de los datos, lo cual puede resultar en una mejor detección de características en estas regiones. La fórmula para el tamaño del output O y el relleno P es:

$$O = \left\lceil \frac{I}{S} \right\rceil, \quad P = \left\lfloor \frac{K - 1}{2} \right\rfloor. \tag{4.23}$$

Ejemplo: Datos de entrada con tamaño $I = 7$, un kernel con tamaño $K = 3$, y un stride $S = 1$. Para el padding 'same', se añade relleno para mantener el tamaño de la entrada. La fórmula para el relleno P y el tamaño del output O es:

$$P = \left\lfloor \frac{K - 1}{2} \right\rfloor = \left\lfloor \frac{3 - 1}{2} \right\rfloor = 1, \quad O = \left\lceil \frac{I}{S} \right\rceil = \left\lceil \frac{7}{1} \right\rceil = 7. \tag{4.24}$$

Por lo tanto, se añade un relleno de 1 y el tamaño del output sigue siendo 7.

- **Full:** El 'full padding' en redes neuronales convolucionales se refiere a una técnica de relleno que añade ceros a los datos de entrada para permitir que los filtros convolucionales se apliquen incluso en los bordes de los datos. Esta estrategia aumenta las dimensiones de la salida, proporcionando una representación más completa de las características detectadas, mejorando así la detección de patrones en regiones cercanas a los límites de la imagen. La fórmula para el tamaño del output O es:

$$O = I + K - 1. \tag{4.25}$$

Una peculiaridad distintiva del padding full es la ausencia de una fórmula para determinar la cantidad exacta de relleno a aplicar. Esto se debe a que el relleno debe ajustarse en función del tamaño del filtro y del modo específico en que se pretende que este interactúe con los bordes del input.

Ejemplo: Datos de entrada con tamaño $I = 7$, y un kernel con tamaño $K = 3$. En el caso del padding 'full', la fórmula para el tamaño del output O es:

$$O = I + K - 1 = 7 + 3 - 1 = 9. \tag{4.26}$$

Así, el tamaño del output es 9. El relleno específico depende del tamaño del filtro y de cómo queremos que interactúe con los bordes del input, pero no está definido por una fórmula estándar.

Importancia del stride en las operaciones de convolución

Un aspecto importante de la operación de convolución, es la elección del stride. Este indica el desplazamiento mediante el cual el filtro se mueve a lo largo de la imagen de entrada, es decir, va a determinar el número de píxeles que el filtro mueve en dicho input durante cada operación de convolución. El stride determina cómo se superponen las regiones de la imagen cuando se aplica el filtro, influyendo directamente en el tamaño del mapa de características resultante y en la capacidad de la red para capturar información espacial. En la Figura 4.20 (a) se puede observar que un stride más pequeño produce mapas de características más detallado, permitiendo a la red identificar detalles más finos en la imagen. Por otra parte, tal y como vemos en 4.20 (b) un stride más grande generalmente da como resultado un mapa de características de menor tamaño. Esto reduce la resolución espacial pero aumenta la eficiencia computacional [159].

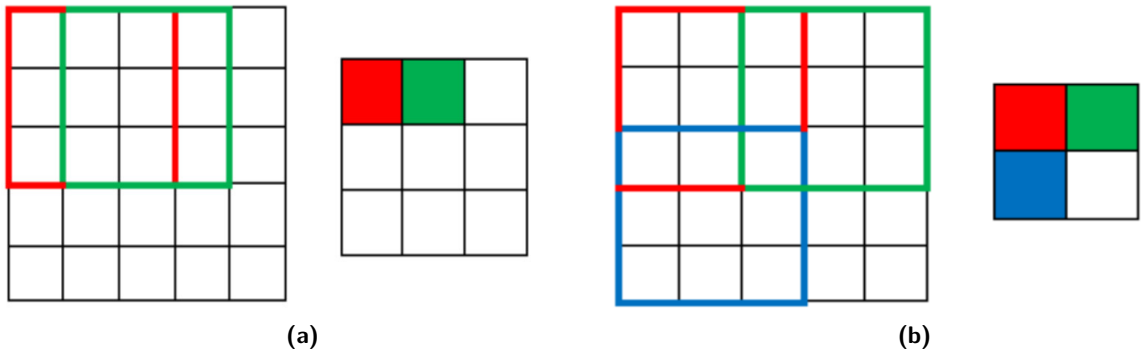


Figura 4.20: Efectos de distintos valores de stride en la convolución: (a) muestra la detallada captura de características con un stride = 1; (b) ilustra la reducción en la resolución espacial y el incremento de eficiencia con un stride = 2 [159]. Figura extraída de 'Stride 2 1-d, 2-d, and 3-d winograd for convolutional neural networks', IEEE Transactions on Very Large Scale Integration (VLSi) Systems, IEEE, 2020.

Un stride grande, reduce el tamaño del mapa de características resultante capturando características más generales, lo que es útil para disminuir la cantidad de parámetros y el coste computacional en la red, y un stride pequeño permite una mayor sensibilidad a las características finas.
Por lo tanto, la elección de stride debe de ser un equilibrio entre la resolución de las características que se desean detectar, y la eficiencia computacional. Esto es muy relevante cuando se procesan grandes volúmenes de datos.

Subsampling y Pooling en CNN

El downsampling hace referencia a aquellas técnicas que tienen el objetivo de disminuir la cantidad de parámetros y la complejidad computacional, permitiendo tambien conservar las caracteristiscas relevantes de los datos de entrada. Las técnicas de downsampling que se van a explicar a continuación son: subsampling, y distintas operaciones de pooling. Las técnicas de downsampling que se van a explicar a continuación son: subsampling, y distintas operaciones de pooling.
El subsampling es una técnica empleada para reducir la resolución espacial o temporal de los datos, disminuyendo la cantidad de parámetros y la complejidad computacional del modelo. Esto se logra mediante el parámetro 'stride', ya que hace referencia al número de píxeles por los que se desplaza la ventan del subsampling a través del mapa de características en cada paso. Este efecto del stride dentro del subsamplig se puede apreciar en la Figura 4.21. Cuando stride es igual a 1, cada pixel contiguoo es considerado en el proceso de subsampling. Este enfoque mantiene la mayor cantidad de información espacial, y a su vez no reduce significativamente la dimensión de la salida. Cuando stride toma un valor de 2, reduce el tamaño del mapa de características a

la mitad en cada dimensión, dado que solo se considera cada segundo píxel tanto en la dirección horizontal como en la vertical. Por último, cuando stride vale 3 solo seleccionará un píxel por cada tres. Esto lleva a una reducción aún mayor en la dimensión espacial del mapa de características.

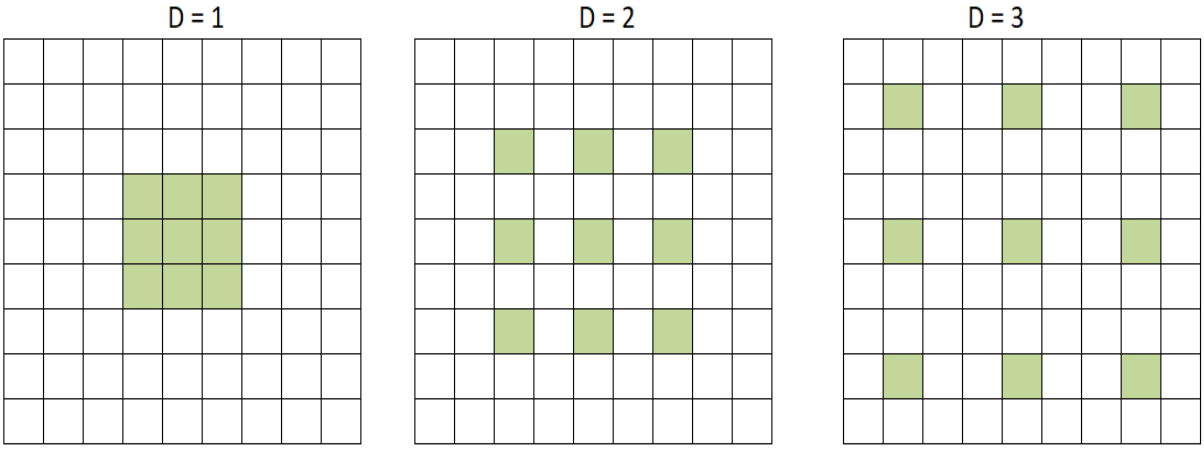


Figura 4.21: Tipos de Subsampling. D=1 significa que el filtro no salta ningún pixel, mientras que un stride de D=2 o D=3 significa que el filtro se salta uno o dos píxeles respectivamente.

Se puede observar que el efecto del incremento del valor del stride es una disminución progresiva en la resolución de la salida después del subsampling. Con un stride mayor, se pierde más información espacial, pero el modelo se vuelve más robusto a las variaciones de la entrada y se reduce el riesgo de overfitting. También se reduce la cantidad de cálculos necesarios para procesar los datos, dando lugar a un entrenamiento más rápido de la red.

El pooling es otra técnica que reduce la resolución de los datos y mejora la invariancia a pequeñas variaciones en los datos de entrada. Entre las operaciones de pooling se encuentran el "max poolingz el .ªverage pooling". La operación "max pooling", va a ayudar a resaltar las características más llamativas en la región de la ventana móvil deslizante, mejorando la robustez del modelo frente a pequeñas variaciones en la entrada. Por otro lado, la técnica .ªverage poolingçalcula el promedio de los valores dentro de dicha ventana, sin destacar las características más prominentes como lo hace el max pooling. La técnicas average pooling puede proporcionar una representación más suave y con un ruido menor de las características en los datos [163].

Tal y como se aprecia en la Figura 4.22, durante la realización de la técnica de "max pooling"se hace respetando el tamaño de la ventana móvil de 2x2. De estos cuatro pixeles iniciales en la ventana, se mantendrá únicamente aquel de valor más alto. Por otra parte, en lo que respecta al criterio "average pooling" se realizará un valor promedio de esos 4 píxeles.

Ambos métodos permiten reducir la dimensionalidad de la imagen y de la cantidad total de información mostrada, el diseño del pooling está orientado a retener las características más sobresalientes dentro de cada ventana movil de datos. Aunque por otro lado, es importante remarcar que este intento de retener características sobresalientes, puede suponer la pérdida de cierta información relevante, especialmente aquella que no es capturada como máximos (en el caso del max pooling) o como un promedio (en el caso del average pooling).

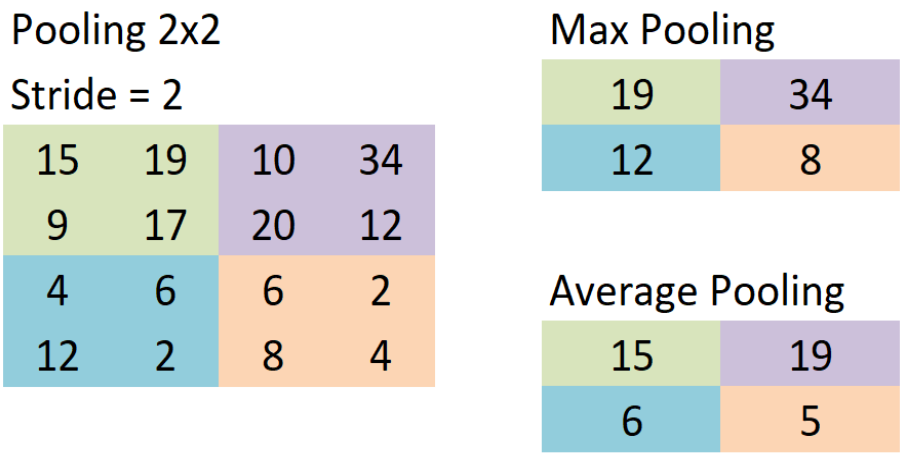


Figura 4.22: Tipos de pooling

4.7 Normalización divisiva

4.7.1 Breve historia e introducción

La normalización divisiva es una transformación introducida en 1992, basada en las propiedades no lineales observadas en las neuronas corticales, que están situadas en las primeras etapas de la corteza visual primaria (área V1) [72]. Hoy en día se valora la posibilidad de que tenga presencia en todo el sistema visual, así como en otras regiones cerebrales [130]. Se ha convertido en una transformación estándar para describir las neuronas del sistema visual [34]. Más concretamente, es una forma de control de ganancia local en la que las respuestas de las neuronas se dividen por un factor que es la actividad conjunta de las vecinas rectificadas con algunos pesos [72]. Este mecanismo nos permite reconocer (clasificar) imágenes bajo condiciones variadas, como puede ser en un orientación, la distancia a la que se encuentre, su iluminación, o con cambios en la textura o el color. La aplicación de la normalización divisiva se extendió desde un punto de vista psicofísico, aplicándose a texturas de imágenes [56], e incluse en problemas de color [38, 39], entre otros.
La idea principal era simular este proceso que nuestro cerebro realiza casi a la perfección. Esta transformación se llamó normalización divisiva y se ha demostrado que mejora el rendimiento del reconocimiento cuando se aplica a redes neuronales profundas [84]. Posteriormente se abordó desde una perspectiva estadística y se aplicó en problemas de clasificación [46], y de segmentación semántica [73]. En clasificación, supone que sólo hay un objeto por imagen, asignándole una etiqueta discreta como salida del modelo. Esta etiqueta se corresponde con el objeto de la imagen. La detección de objetos es el siguiente paso que da lugar a la generalización de la clasificación y localización, cuando hay múltiples objetos en una misma imagen [32].
En el caso de la segmentación semántica, cada píxel de la imagen se etiqueta, distinguiendo entre los individuos de una misma categoría, por lo que el modelo clasifica cada instancia por separado. Sin embargo, en el caso de la segmentación semántica (probablemente la más utilizada en conducción autónoma), se asocia cada píxel de una imagen con una etiqueta de clase, como puede ser 'persona', 'carretera' o 'coche', pero sin diferenciar entre los objetos de la misma clase. Este proceso facilita la identificación del entorno a la hora de tomar decisiones [32].
La Figura 4.23, está compuesta por un paisaje, siendo este la vista de un valle con zonas boscosas. Se puede observar que la textura y el color del bosque cambian debido a factores como la iluminación que reciben, la distancia o profundidad donde se encuentra dicho fragmento del objeto, la niebla, o las sombras que se generan. Esto está marcado en las zonas resaltadas en rojo, ya que corresponden al mismo objeto: el bosque. Estas variaciones no informativas implican que el mismo objeto tiene distintas visualizaciones, generando un problema

para la identificación y clasificación de un mismo objeto dentro de la misma imagen.

En la Figura que está siendo analizada, arriba a la derecha, se aprecia un diagrama de texturas 3D. Representa las características de textura del paisaje en un espacio tridimensional donde cada eje (X1, X2, X3) corresponde a propiedades distintas a nivel de textura como son el contraste, la orientación y la frecuencia de las propiedades visuales. La nube de puntos del interior del diagrama refleja la distribución de estas características. Una mayor dispersión indica una mayor variación en la textura de la imagen.

Por último, esta Figura está compuesta por un diagrama de colores situado abajo a la derecha, donde dicho triángulo representa el espacio de color visible. Cada vértice corresponde a un color primario (rojo, verde, azul). El diagrama interno, con vértices en 526 nm, 555 nm, 487 nm, 484 nm y 483 nm, indica el gamut de color (el subconjunto del espacio de color) que captura las características de color presentes en la imagen. Las marcas de longitud de onda (que se miden en nanómetros, nm) indican las longitudes de onda dominantes para el análisis de color en la imagen. Por ejemplo, 555 nm corresponde a un color verde amarillento, un tono que destaca en escenas naturales.
El diagrama en Figura 4.23 proporciona información sobre la saturación, el brillo y la gama de colores capturados en la imagen del paisaje.

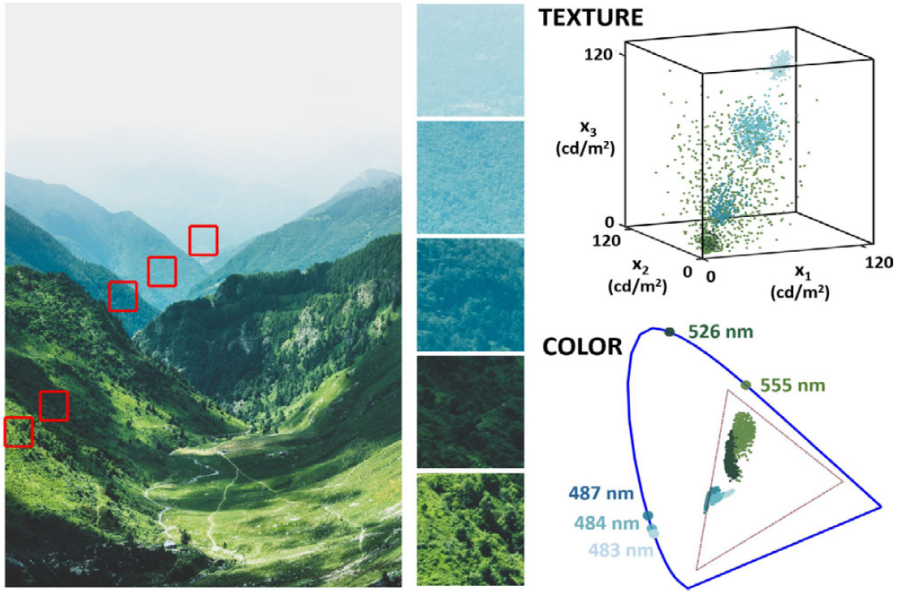


Figura 4.23: Los modelos de aprendizaje automático enfrentan el desafío de las variaciones no informativas de las imágenes, como cambios de textura y el color debido a la iluminación o sombras que recibe dicha parte de la imagen. Estos cambios no informativos crean múltiples fragmentos visuales distintos para un mismo objeto de la imagen, lo que dificulta el análisis y la identificación de los mismos [73]. Figura extraída de 'Neural networks with divisive normalization for image segmentation', Pattern Recognition Letters, Elsevier, 2023.

La Figura 4.24 desglosa el procesamiento de una escena natural. Concretamente, es una de las partes de la imagen anterior 4.23, donde se encontraban problemas en determinar la clase a la que corresponde cada una de las distintas regiones marcadas en rojo. Estos casos están agrupados en cada uno de los recuadros rojos de la imagen anterior. La Figura consta de varias partes:

- Imagen original: Se muestra la imagen sin procesar, donde las variaciones de luz y textura pueden afectar a la consistencia visual del objeto (en este caso, el bosque).
- Características de wavelet, z: donde la imagen X se ha transformado utilizando un banco de filtros wavelet,

que descompone la imagen en componentes que representan distintas escalas y orientaciones de la textura.

- Características normalizadas divisivas, y: Las características wavelet se someten a un proceso de normalización divisiva para reducir las variaciones causadas por factores no informativos como la iluminación y las sombras. Estas variaciones están resaltadas en los círculos amarillos y blancos respectivamente, ya que señalan regiones de interés donde la normalización divisiva tiene un efecto aparentemente pronunciado en la homogeneización de las respuestas. Estos círculos controlan la vecindad que define la región de agrupación, relajando así la respuesta en regiones donde el contraste es alto (círculos naranjas), mientras que aumenta la respuesta en regiones de bajo contraste (círculos blancos). De esta manera cuando la actividad local es pequeña, su valor se incrementa, y cuando dicha actividad local es grande, su valor se reduce. Esto permite maximizar el contraste local [103].

- Imagen con normalización divisiva (div. norm. image x'): Resultado final en el que la imagen se reconstruye a partir de las características normalizadas. Se espera que sea más uniforme en términos de contraste e iluminación, en comparación con la imagen original.

En la parte derecha de la Figura, se observan dos gráficos tridimensionales que reflejan la distribución de las propiedades de textura de la imagen:

- Textura (texture): Un espacio tridimensional donde se grafican las características de la textura de la imagen original. Se observa una amplia dispersión de los puntos, indicando una gran variabilidad en las características de textura debido a factores no informativos como la iluminación y la sombra.

- Textura con normalización divisiva (div. norm. texture): Se observa un gráfico 3D similar al anterior donde tras aplicar la transformación de normalización divisiva, las características texturales muestran una distribución más compacta. Esto implica que la normalización ha sido efectiva en atenuar las variaciones no informativas, haciendo que el modelo varíe menos a los distintos cambios no informativos que sufre la imagen.

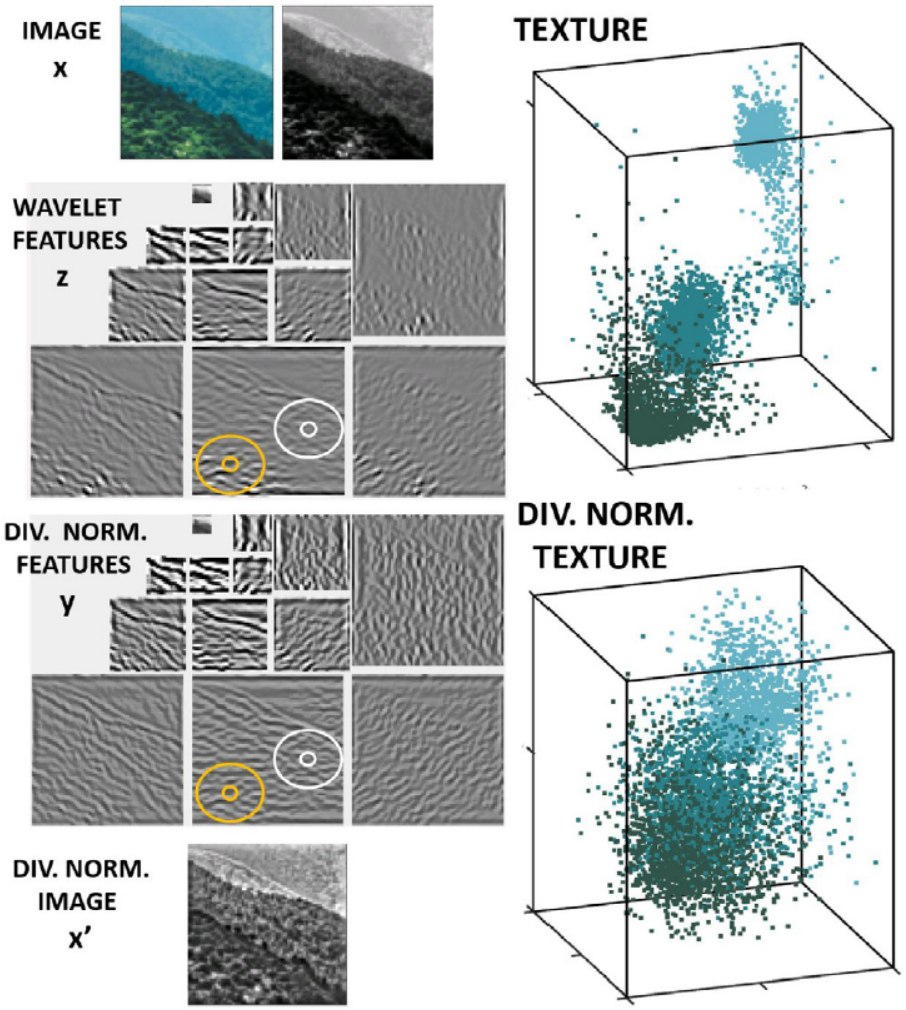


Figura 4.24: Procesamiento mediante normalización divisiva para mejorar la segmentación semántica de imágenes. La parte superior izquierda muestra la imagen original x. Esta forma parte de una escena natural con diferente iluminación y texturas, pero representando el mismo objeto. A continuación, se aplican filtros wavelet para extraer las características lineales z, y mediante la normalización divisiva se obtienen las características normalizadas y. Los círculos amarillos y blancos indican regiones de alto y bajo contraste afectadas por la normalización. En la parte inferior, se muestra la imagen reconstruida x' con características homogeneizadas. En la parte derecha, se observan dos gráficos 3D que muestran las distribuciones de la textura antes y después de la normalización, reflejando la compresión de la variabilidad inicial [32]. Figura extraída de .Autonomous driving", 2022.

Esta visualización detallada refleja la eficacia de la normalización divisiva para estabilizar las características visuales de los objetos que tienen distintas condiciones de luz y sombra. La aplicación de la normalización divisiva facilita la identificación y clasificación más precisa de los objetos en la imagen, ya que actúa como un ecualizador de las características visuales.

4.7.2 Formulación matemática de la normalización divisiva:

La normalización divisiva ajusta las respuestas de manera que se mantienen dentro de un rango operativo deseado, preservando al mismo tiempo sus valores relativos. Matemáticamente, para un conjunto de entradas

x, la transformación básica se define de la siguiente forma:

$$y_i = \gamma \frac{x_i^\alpha}{\beta^\alpha + \sum_j x_j^\alpha}, \tag{4.27}$$

donde α, β, γ son los parámetros que controlan la forma en la que se realiza la normalización. Esta formulación inicial ha sido extendida para abordar mejor la distribución de datos de imágenes naturales a través de la Normalización Divisiva Generalizada (GDN).

4.7.3 Normalización Divisiva Generaliazda (GDN):

La GDN redefine la normalización para considerar no solo la actividad local sino también las respuestas más distantes espacialmente. La transformación paramétrica de la GDN se expresa como una composición de una transformación lineal seguida de una forma generalizada de normalización divisiva del siguiente modo:

$$y = g(x;\theta) \qquad \text{s.a.} \quad y_i = \frac{z_i}{\left(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}}\right)^{\epsilon_i}} \qquad \text{y} \quad z = Hx, \tag{4.28}$$

con $z = Hx$, donde H es una matriz de transformación lineal. El objetivo es optimizar los parámetros β, ϵ, α, y γ de manera que la transformación H permita la gaussianización de los datos de la imagen en cuestión. La transformación estará bien definida si es diferenciable, continua e invertible [13].

Para la porción lineal de la transformación, necesitamos asegurar que la matriz H sea no singular. Para la porción de normalización, consideremos las derivadas parciales:

$$\frac{\partial y_i}{\partial z_k} = \frac{\delta_{ik}}{(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}})^{\epsilon_i}} - \frac{\alpha_{ik}\gamma_{ik}\epsilon_i z_i |z_k|^{\alpha_{ik}-1} \text{sgn}(z_k)}{(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}})^{\epsilon_i+1}}, \tag{4.29}$$

Para garantizar la continuidad, requerimos que todos los exponentes en la ec. 4.29 sean no negativos, así como la expresión entre paréntesis en el denominador sea positiva.
Una condición necesaria (aunque habitualmente no suficiente) para la invertibilidad se puede establecer de la siguiente manera. Primero, notemos que, como el denominador es positivo, cada vector z se mapea a un vector y en el mismo ortante. Los ejes cardinales de z se mapean a sí mismos, y para que este mapeo unidimensional sea continuo e invertible, debe ser monótono. A lo largo de los ejes cardinales, se mantiene la siguiente cota

$$|y_i| = \frac{|z_i|}{(\beta_i + \gamma_{ii}|z_i|^{\alpha_{ij}})^{\epsilon_i}} \leq \frac{|z_i|}{\gamma_{ii}^{\epsilon_i}|z_i|^{\alpha_{ii}\epsilon_i}} = \gamma_{ii}^{-\epsilon_i}|z_i|^{1-\alpha_{ii}\epsilon_i}. \tag{4.30}$$

La invertibilidad de la transformación garantizará que la función de densidad que describe los datos sea bien definida y útil para aplicaciones como la denoising (eliminación de ruido). Para que la transformación GDN sea continua e invertible, es necesario que la matriz Jacobiana que contiene las derivadas parciales sea definida positiva en todas partes. Esto se asegura imponiendo que el exponente el exponente $1 - \alpha_{ii}\epsilon_i$ sea positivo y que las expresiones en los denominadores sean positivas. Se introducen restricciones durante la optimización, tales como: $\alpha_{ij} \geq 1$, $\beta_i > 0$, $\gamma_{ij} \geq 0$, y $0 \leq \epsilon_i \leq \alpha_{ii}^{-1}$.
Inicializamos los parámetros de tal manera que $\frac{\partial y}{\partial z}$ sea definida positiva en todas partes (por ejemplo, dejando γ ser diagonal, de modo que el Jacobiano se aproxime a la matriz identidad, la transformación es separable, y la restricción necesaria en los ejes cardinales se vuelve suficiente). Finalmente, encontramos que la transformación GDN puede ser invertida eficientemente utilizando una iteración de punto fijo:

$$z_i^{(0)} = \text{sgn}(y_i)\left(\gamma_{ii}^{\epsilon_i}|y_i|\right)^{\frac{1}{1-\alpha_{ii}\epsilon_i}}, \quad z_i^{(n+1)} = \left(\beta_i + \sum_j \gamma_{ij}|z_j^{(n)}|^{\alpha_{ij}}\right)^{\epsilon_i} y_i. \tag{4.31}$$

Otras soluciones inversas iterativas han sido propuestas para este propósito (Malo et al., 2006; Lyu & Simoncelli, 2008), pero estas sólo aplican a casos especiales de la forma en la ec. 4.28.

4.7.4 Ventajas y aplicaciones de la GDN

La evolución a la normalización divisiva generalizada ha supuesto una mejora con respecto a la normalización divisiva, ya que presenta mejor rendimiento en cuanto a la distorsión de la imagen respecto al modelo JPEG estándar, si previamente se ha utilizado un sistema de compresión, mejorando además la calidad visual de las imágenes [14, 15].
Una de sus aplicaciones consiste en realzar detalles y contraste en imágenes afectadas por situaciones que generen dispersión óptica, como puede ser en imágenes que contienen niebla. Tambien aumenta el contraste en características de una imagen con un nivel bajo del mismo, sin la necesidad de introducir distorsiones demasiado significativas [93]. Tambien se ha demostrado que la incorporación de capas GDN en una red neuronal, mejoran de forma significaitiva la segmentación semántica de imágenes con condiciones metereológicas adversas como la niebla, mejorando de este modo la identificación de los objetos de la propia imagen, frente a las redes que no tienen capas GDN [32, 73]
Las GDN destacan frente a modelos tradicionales, como puede ser el modelo de Análisis de componentes principales (ACP) y Análisis de componentes independientes con gausianización marginal (ICA-MG), principalmente en la eficacia a la hora de reducir información mutua y minimizar de este modo la redundancia de la imagen en cuestión, dando lugar a una representación más eficiente. También aproxima captura mejor las distribuciones teóricas esperadas de los datos de las imágenes, tal y como se aprecia en sus histogramas. Además, los parches de imágenes creados por GDN para reconstruir imágenes, presentan un mayor realismo frente a los parches creados por ICA-MG y ACP, replicando con mayor éxito las distitnas texturas y patrones de la imagen [13].

Capítulo 5

Aproximaciones del momento segundo mediante kernel smoother con aplicación a la estimación de la volatilidad

La estimación de la volatilidad y la regresión cuantil son temas de investigación sumamente relevantes en áreas como la estadística, el aprendizaje automático y la econometría. En este trabajo, se proponen dos métodos para estimar las varianzas locales en problemas de regresión genéricos mediante el uso de kernel smoother. Las estrategias que presentadas pueden aplicarse en contextos multidimensionales (es decir, no solo para análisis de series temporales) y también de forma sencilla en estructuras de múltiples salidas. Además, estos métodos ofrecen la posibilidad de proporcionar estimaciones de incertidumbre utilizando una técnica de kernel smoother genérico. Diversos experimentos numéricos muestran los beneficios de los métodos propuestos, incluso en comparación con técnicas de referencia. Uno de estos experimentos incluye el análisis de un conjunto de datos reales.

5.1 Introducción

El análisis de regresión se puede considerar como un conjunto de metodologías para estimar las relaciones entre una variable dependiente y (a menudo denominada 'salida' o 'respuesta') y un vector $\mathbf{x}$ de variables independientes (comúnmente llamado 'entrada'). El objetivo principal en un problema de regresión es obtener una aproximación de la tendencia, definida como el valor esperado de y dado $\mathbf{x}$, es decir, $E[y|\mathbf{x}]$, que es el primer momento (no central) de la densidad condicional $p(y|\mathbf{x})$. Se puede afirmar que el problema de regresión más completo consiste en aproximar toda la densidad condicional $p(y|\mathbf{x})$, mientras que la tarea más simple en un problema de regresión consiste en estimar solo el primer momento $E[y|\mathbf{x}] = \int_{\mathcal{Y}} yp(y|\mathbf{x})dy$. Escenarios intermedios aparecen en diferentes aplicaciones, donde otros momentos (superiores a uno) son de interés y por lo tanto también son aproximados.
La estimación de la volatilidad (entendida como varianza local o desviación típica local) y el análisis de regresión cuantil son actualmente tareas importantes en estadística, aprendizaje automático y econometría. El problema de la estimación de la volatilidad tiene gran relevancia dentro del análisis de series temporales financieras, donde la volatilidad representa el grado de variación de la serie de precios de un activo financiero a lo largo del tiempo, generalmente medida por la desviación típica de los retornos logarítmicos del precio en cuestión. [1] Aún es un área importante y activa de investigación [58, 36, 51, 82].

[1] De hecho, es importante destacar que un problema inherente en la estimación de la volatilidad, es que la varianza condicional es latente, y por lo tanto no es directamente observable cuando se analizan datos reales.

En la literatura se han propuesto varios métodos. Entre dichos métodos destacan los modelos de heteroscedasticidad condicional autoregresiva (ARCH) y los modelos de heteroscedasticidad condicional autoregresiva generalizada (GARCH) [59, 24]. Ambos se consideran el punto de referencia para el análisis de volatilidad de series temporales. Otras clases importantes de técnicas son: los modelos de volatilidad estocástica [145] [108, Sección 7.4], y los modelos de media móvil ponderada exponencialmente [113]. También se han extendido para el escenario multivariante, pero la aplicación es mucho más compleja que el método propuesto aquí. Además, se han detectado otras críticas en la literatura: como afirman algunos autores [24, 100, 35], la mayoría de los modelos de volatilidad latente no logran describir satisfactoriamente algunos de los hechos que se observan en las series temporales financieras. Varios de los métodos se basan en modelos de espacio de los estados [108, Sección 7.4]. Algunos artículos de revisión importantes sobre este tema se pueden encontrar en trabajos relevantes como [9, 10, 69, 16] donde los autores también se han centrado en los fundamentos teóricos de los estimadores que se han propuesto recientemente. Se han considerado otros enfoques. Por ejemplo, los métodos de regresión lineal local (que pueden parecerse a un enfoque de kernel smoother) ya se han propuesto para la estimación de la volatilidad [62, 160, 135]. Comparten la noción de "localidadçon el enfoque de kernel smoothers, y realizan varias regresiones lineales (aplicadas a los errores residuales para estimar la varianza). En general, para todos estos métodos, las extensiones para los casos multivariantes y/o de múltiples salidas no son sencillas. Además, la mayoría de ellos son métodos paramétricos (aquí, se han considerado técnicas no paramétricas donde la complejidad del modelo está relacionada con el número de datos observados). En matemáticas financieras e ingeniería financiera, otra clase de métodos lo componen los modelos de volatilidad local, que son generalizaciones del modelo de Black–Scholes. Los modelos de volatilidad local son similares y están relacionados con los modelos de volatilidad estocástica, donde la volatilidad instantánea es en sí misma una variable aleatoria [52]. Otros autores estudian características de alta frecuencia de los datos que contienen ruido de microestructura del mercado para estimar la volatilidad instantánea (por ejemplo, [168]).

Los modelos de regresión cuantil estudian la relación entre un vector de entrada independiente $\mathbf{x}$ y algún cuantil o momento específico de la variable de salida y [40, 79, 104]. Por lo tanto, en el análisis de regresión cuantil el objetivo es estimar momentos de orden superior de la variable respuesta/salida, dada una entrada $\mathbf{x}$ [40, 79, 104, 18, 44, 80]. Por otro lado, es importante destacar que muchos métodos de regresión avanzados y de referencia en la literatura (como los Procesos Gaussianos, etc.) consideran una *varianza constante* como suposición inicial de la salida dada la entrada, var$[y|\mathbf{x}] = \sigma_e^2$, es decir, varía con la variable de entrada $\mathbf{x}$ [21, 106].

En este trabajo, se considera un enfoque extendido (con respecto a las suposiciones utilizadas en los métodos de regresión clásicos) donde var$[y|\mathbf{x}] = v(\mathbf{x})$, es decir, la varianza local varía con la entrada $\mathbf{x}$. Más precisamente, en este trabajo, se proponen dos procedimientos para estimar la varianza local en un problema de regresión y utilizando un enfoque de kernel smoothers [106, Sección 9]-[4]. Cabe destacar que los métodos de kernel smoothers contienen varias técnicas bien conocidas como casos especiales, como los *vecinos más cercanos de radio fijo*, como ejemplo [20]. La solución resultante es un método *no paramétrico*, por lo tanto, la complejidad y la flexibilidad de la solución crecen con el número de datos N. Esto asegura tener la flexibilidad adecuada para analizar los datos.

El primer método propuesto se basa en la derivación de Nadaraya-Watson de un kernel smoother lineal [21, 106]. El segundo enfoque propuesto se inspira en la *normalización divisiva*, una función basada en la actividad de las neuronas cerebrales [34]. Esta función tiene como objetivo estandarizar la actividad neuronal dividiéndola por la actividad de las neuronas vecinas. Este método ha demostrado propiedades estadísticas favorables [101, 13, 92] y ha sido utilizado en varias aplicaciones [93, 73]. Otros métodos importantes y relacionados que contienen resultados teóricos relevantes, se pueden encontrar en la literatura [67, 119, 41]. Los métodos propuestos se pueden aplicar al análisis de series temporales y/o en problemas de regresión más generales donde las variables de entrada son multidimensionales, y con ruido correlacionado. Por lo tanto, nuestro enfoque se puede implementar en la modelización estadística espacial y en cualquier otro problema inverso [104, 110]. Hay muchos conjuntos de datos con entradas multidimensionales (por ejemplo, que incluyen variables espaciales y temporales) cuya estructura implica un cambio sustancial de varianza. Por ejemplo, la descomposición de los datos

sísmicos. Además, otra ventaja del escenario propuesto, es que la generalización para escenarios de múltiples salidas se puede diseñar fácilmente. Dicho de otra forma, el enfoque propuesto es fácil de aplicar, de extender y de generalizar en términos de flexibilidad, es decir, se pueden considerar diferentes técnicas de regresión o diferentes funciones de kernel (posiblemente diferentes para la estimación de la tendencia y de la varianza), y el número de hiperparámetros puede ser decidido por el usuario.
Desde otro punto de vista, este trabajo aporta otra contribución relevante. Los modelos propuestos permiten realizar estimaciones de la incertidumbre con una técnica de kernel smoothers genérico. De hecho, un método de kernel smoothers genérico generalmente no está respaldado por una derivación probabilística generativa que también proporcione una estimación de la incertidumbre. Los métodos de regresión de procesos gaussianos (Gaussian processes, GPs por sus siglas en inglés) y máquinas de vectores de relevancia (relevance vector machines, RVMs por sus siglas en inglés) son excepciones bien conocidas y prácticamente únicas [131, 106]).
Se han probado los métodos propuestos en cinco experimentos numéricos. Cuatro de ellos involucran la aplicación de los diferentes modelos al análisis de series temporales y la comparación con modelos GARCH que son considerados técnicas de referencia para la estimación de la volatilidad en series temporales [23]. El último experimento numérico aborda un problema de regresión más general, con una variable de entrada multidimensional $\mathbf{x}$ de dimensión 122, considerando una base de datos real (emo-soundscapes) [61, 63].
El artículo está estructurado de la siguiente manera. La metodología principal empleada aquí para la regresión, basada en kernel smoothers, se describe en la Sección 5.2. También se remarcan varias características y aspectos relevantes. Los dos procedimientos propuestos para la estimación de la varianza se introducen en la Sección 5.3. Diferentes extensiones, generalizaciones y variantes se dan en la Sección 5.4. Numerosos experimentos numéricos que cubren diferentes escenarios (y con varias comparaciones) se proporcionan en la Sección 5.5. Finalmente, la Sección 5.6 está dedicada a discutir algunas conclusiones y algunos posibles trabajos futuros.

5.2 Aproximando la tendencia

Se considera un conjunto de N pares de datos, como $\{\mathbf{x}_i, y_i\}_{i=1}^N$, donde $\mathbf{x}_i \in \mathbb{R}^D$, con $D \geq 1$, and $y_i \in \mathbb{R}$. Cabe destacar el interés en obtener una función de regresión (también conocida como,"media local"– tendencia), es decir, en obtener un estimador eliminando el ruido de la señal $\widehat{f}(\mathbf{x})$ para todos los posibles $\mathbf{x}$ en el espacio de entrada. Una posibilidad es emplear un *kernel smoother lineal*. Más específicamente, se considera el uso del estimador de Nadaraya-Watson [21, 106] que tiene la siguiente forma,

$$E[y|\mathbf{x}] \approx \widehat{f}(\mathbf{x}) = \sum_{n=1}^{N} \frac{h(\mathbf{x}, \mathbf{x}_n)}{\sum_{j=1}^{N} h(\mathbf{x}, \mathbf{x}_j)} y_n = \sum_{n=1}^{N} \varphi(\mathbf{x}, \mathbf{x}_n) y_n, \tag{5.1}$$

donde $\varphi(\mathbf{x}, \mathbf{x}_n) = \frac{h(\mathbf{x}, \mathbf{x}_n)}{\sum_{j=1}^{N} h(\mathbf{x}, \mathbf{x}_j)}$ y $h(\mathbf{x}, \mathbf{z})$ es una función (que a menudo se llama "núcleo", Kernel) decidida por el usuario y definida en $\mathbb{R}^D \times \mathbb{R}^D$. Según esta definición, los pesos no lineales $\varphi(\mathbf{x}, \mathbf{x}_n)$ son normalizados, es decir,

$$\sum_{n=1}^{N} \varphi(\mathbf{x}, \mathbf{x}_n) = 1, \quad paratodo\mathbf{x}. \tag{5.2}$$

Como ejemplo, se podría considerar

$$h(\mathbf{x}, \mathbf{z}) = h(\mathbf{x}, \mathbf{z}|\lambda) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^2}{\lambda}\right),$$

donde λ es un hiperparámetro que debe ajustarse. Claramente, se obtiene

$$\varphi(\mathbf{x}, \mathbf{z}) = \varphi(\mathbf{x}, \mathbf{z}|\lambda).$$

La forma de este estimador es bastante general y contiene otros métodos diferentes bien conocidos, como casos especiales. Por ejemplo, incluye el algoritmo de los k-vecinos más cercanos (kNN) para regresión como un caso específico (para ser más específicos, el algoritmo de vecinos cercanos de radio fijo). De hecho, con una elección específica de $h(\mathbf{x}, \mathbf{x}_n|\lambda)$ (como una función rectangular), la expresión anterior puede representar el estimador de vecinos cercanos de radio fijo [20, 106].

Observación La función de regresión resultante es un método flexible no paramétrico. Tanto la complejidad como la flexibilidad de la solución crecen con el número de datos N ya que, en cada nuevo punto de datos $\mathbf{x}_{N+1}$, corresponde el uso de una función de kernel adicional $\varphi(\mathbf{x}, \mathbf{x}_{N+1})$ ubicada en $\mathbf{x}_{N+1}$ y la solución final $\widehat{f}$ será una combinación lineal de $N+1$ valores.

Observación Las características de la solución están determinadas por la elección del kernel por parte del usuario. Por ejemplo, la elección de una función de kernel gaussiano, $h(\mathbf{x}, \mathbf{x}_n)$, genera una solución suave diferenciable infinitamente. La elección de una función de kernel laplaciana conduce a una solución que no es diferenciable para todas las variables de entrada (inputs) $\mathbf{x}_n$ del conjunto de datos.

Observación Las variables de entrada $\mathbf{x}_n \in \mathbb{R}^D$ son vectores ($D \geq 1$), en general. Por lo tanto, los métodos descritos tienen una aplicabilidad mucho más amplia que las técnicas que solo se pueden emplear para series temporales (donde el índice temporal es un número escalar, $x = t \in \mathbb{R}$). Claramente, las metodologías descritas aquí también pueden emplearse para analizar series temporales. Además, incluso en un marco de series temporales, se puede obtener una predicción entre dos instantes de tiempo consecutivos. Por ejemplo, para una serie temporal con datos diarios, con un kernel smoother se puede tener una predicción para cada hora (o minuto) dentro de un día específico.

Observación Es importante destacar que la aplicación para el caso de entrada multidimensional, $\mathbf{x}_n \in \mathbb{R}^D$, es en general rápido y sencillo, a diferencia de otras soluciones locales polinómicas por partes (por ejemplo, [135, 160]) que requerirían la construcción de una cuadrícula multidimensional y costosa (para construir y evaluar adecuadamente el regresor por partes). Además, el marco propuesto contiene estas soluciones por partes como casos especiales, eligiendo una función de kernel que es distinta de cero solo en un subconjunto del soporte (como, por ejemplo, un kernel rectangular).

Aprendizaje de λ. Una posibilidad para ajustar los hiperparámetros de las funciones de kernel es utilizar un procedimiento de validación cruzada (CV). En este trabajo, se ha empleado una validación cruzada de *dejar uno fuera (leave-one-out)* (LOO-CV) [21].

5.3 Procedimientos de estimación de varianza

Se supone que la tendencia ya ha sido calculada (también conocida como "media local"), es decir, la función de regresión $\widehat{f}(\mathbf{x})$. Aquí se presentan dos métodos para obtener una estimación de la varianza local (o volatilidad) en cada punto $\mathbf{x}$, que teóricamente se define como

$$\begin{aligned} v(\mathbf{x}) = \mathsf{var}[y|\mathbf{x}] &= \int_{\mathcal{Y}} (y - E[y|\mathbf{x}])^2 p(y|\mathbf{x}) dy, \\ &= E[y^2|\mathbf{x}] - \Big(E[y|\mathbf{x}]\Big)^2. \end{aligned} \tag{5.3}$$

MÉTODO-1 (M1). Si los pesos $\varphi(\mathbf{x}, \mathbf{x}_n)$ son adecuados para combinar linealmente las salidas y_i y obtener una aproximación adecuada de $E[y|\mathbf{x}]$, se puede extender esta idea para aproximar el segundo momento no central $E[y^2|\mathbf{x}]$ como

$$E[y^2|\mathbf{x}] \approx \widehat{q}(\mathbf{x}) = \sum_{n=1}^{N} \varphi(\mathbf{x}, \mathbf{x}_n) y_n^2. \tag{5.4}$$

por lo tanto

$$\widehat{v}(\mathbf{x}) = \widehat{q}(\mathbf{x}) - \Big(\widehat{f}(\mathbf{x})\Big)^2 \approx \mathsf{var}[y|\mathbf{x}], \tag{5.5}$$

que es un estimador de la *varianza instantánea*. Nótese que $\widehat{q}(\mathbf{x})$ y $\widehat{f}(\mathbf{x})$ se obtienen con los mismos pesos

$$\varphi(\mathbf{x}, \mathbf{x}_n) = \varphi(\mathbf{x}, \mathbf{x}_n|\lambda),$$

con el mismo valor de λ (obtenido mediante un procedimiento LOO-CV). Así, una variante de este procedimiento consiste en aprender otro valor de λ_2, es decir, obtener otros coeficientes $\varphi_2(\mathbf{x}, \mathbf{z}) = \varphi_2(\mathbf{x}, \mathbf{z}|\lambda_2)$, en la ecuación (5.4) considerando la señal y_n^2 (en lugar de y_n).

MÉTODO-2 (M2). Se define la señal obtenida como los errores cuadrados estimados

$$v_n = \left(y_n - \widehat{f}(\mathbf{x}_n)\right)^2, \quad n = 1, ..., N.$$

Si $\widehat{f}(\mathbf{x}) \approx E[y|\mathbf{x}]$ como se asume, v_n es *una estimación de una sola muestra* de la varianza en $\mathbf{x}_n$. Entonces, el objetivo es aproximar la tendencia de esta nueva señal (es decir, nueva salida) v_n,

$$\widehat{v}(\mathbf{x}) = \sum_{n=1}^{N} \varphi_2(\mathbf{x}, \mathbf{x}_n) v_n, \tag{5.6}$$

donde se considera otro parámetro λ_2, es decir,

$$\varphi_2(\mathbf{x}, \mathbf{z}) = \varphi_2(\mathbf{x}, \mathbf{z}|\lambda_2),$$

que se ajusta nuevamente con LOO-CV pero considerando la nueva señal v_n. Nótese que $\widehat{v}(\mathbf{x})$ puede interpretarse como una estimación de la *varianza instantánea* de la señal subyacente. Como alternativa, también se pueden aplicar funciones de kernel completamente diferentes (como φ_2) que difieren en la forma funcional con respecto a φ, en lugar de solo por la elección de λ.

Observación. Nuevamente, para estimar la tendencia, nótese que los dos procedimientos anteriores pueden aplicarse para entradas multivariantes $\mathbf{x}_n \in \mathbb{R}^D$, y no solo para entradas escalares (como en las series temporales).
Observación. Si se divide $\mathbf{x}$ por $\widehat{v}(\mathbf{x})$, se obtiene una señal con varianza local uniforme, es decir, se ha eliminado la (posible) heteroscedasticidad. En [92], este procedimiento se utilizó para definir los kernels de relación en la normalización divisiva, y así igualar localmente la energía de las respuestas neuronales.

5.4 Extensiones y variantes

Esta sección está dedicada a describir varias generalizaciones y variantes de las ideas dadas en las secciones anteriores.

5.4.1 Uso de métodos de regresión genéricos

Las ideas descritas anteriormente pueden emplearse incluso aplicando diferentes métodos de regresión. A continuación, se proporcionan algunos pasos generales al estilo de un pseudo-código, para clarificar la aplicación de posibles técnicas de regresión diferentes:

1. Elegir un método de regresión. Obtener una función de tendencia $\widehat{f}(\mathbf{x}) \approx E[y|\mathbf{x}]$ dado el conjunto de datos $\{\mathbf{x}_i, y_i\}_{i=1}^N$.
2. Elegir un método para estimar la varianza instantánea (entre los dos siguientes):
   - **M1**. Elegir un método de regresión (el mismo del paso anterior o uno diferente). Considerando el conjunto de datos $\{\mathbf{x}_i, y_i^2\}_{i=1}^N$ y obtener $\widehat{q}(\mathbf{x}) \approx E[y^2|\mathbf{x}]$. Luego, calcular $\widehat{v}(\mathbf{x}) = \widehat{q}(\mathbf{x}) - (\widehat{f}(\mathbf{x}))^2$.
   - **M2**. Elegir un método de regresión (el mismo del paso anterior para la tendencia, o uno diferente). Considerando el conjunto de datos $\{\mathbf{x}_i, v_i\}_{i=1}^N$ donde $v_i = (y_i - \widehat{f}(\mathbf{x}_i))^2$, obtener la función $\widehat{v}(\mathbf{x})$.

Nótese que las técnicas de regresión en el paso 1 y paso 2 arriba pueden ser diferentes.

5.4.2 Estimación de la covarianza entre dos entradas genéricas

En esta sección, se describe cómo aproximar la covarianza entre dos entradas genéricas $\mathbf{x}$ y $\mathbf{z}$, extendiendo los procedimientos que se han propuestos previamente. Solo por simplicidad, se adapta el método M2 para estimar la covarianza $\widehat{c}(\mathbf{x}, \mathbf{z})$ entre dos entradas genéricas como

$$\begin{aligned}\widehat{c}(\mathbf{x}, \mathbf{z}) &= \sum_{n=1}^{N}\sum_{i=1}^{N} \frac{h(\mathbf{x}, \mathbf{x}_n)g(\mathbf{z}, \mathbf{x}_i)}{\sum_{j=1}^{N}\sum_{k=1}^{N} h(\mathbf{x}, \mathbf{x}_j)g(\mathbf{z}, \mathbf{x}_k)}(y_n - \widehat{f}(\mathbf{x}_n))(y_i - \widehat{f}(\mathbf{x}_i)), \\ &= \sum_{n=1}^{N}\sum_{i=1}^{N} \varphi(\mathbf{x}, \mathbf{z}, \mathbf{x}_n, \mathbf{x}_i)(y_n - \widehat{f}(\mathbf{x}_n))(y_i - \widehat{f}(\mathbf{x}_i)), \end{aligned} \tag{5.7}$$

donde

$$\varphi(\mathbf{x}, \mathbf{z}, \mathbf{x}_n, \mathbf{x}_i) = \frac{h(\mathbf{x}, \mathbf{x}_n)g(\mathbf{z}, \mathbf{x}_i)}{\sum_{j=1}^{N}\sum_{k=1}^{N} h(\mathbf{x}, \mathbf{x}_j)g(\mathbf{z}, \mathbf{x}_k)},$$

es un kernel normalizado bivariante (y h y g dos funciones de kernel genéricas). Nótese que $(y_n - \widehat{f}(\mathbf{x}_n))(y_i - \widehat{f}(\mathbf{x}_i))$ es un estimador de una sola muestra de la covarianza entre $\mathbf{x}_n$ y $\mathbf{x}_i$. Ideas similares han sido introducidas en [122]. También se puede estimar la correlación dentro del ruido en el conjunto de datos, siendo esto de interés en distintas aplicaciones [4, 104, 48].

Observación. Esta sección trata de aclarar cómo funciona en general el enfoque de estimación con kernels, para posiblemente estimar diferentes momentos y cantidades. La idea es utilizar estimadores de una sola muestra que involucran los puntos de datos $\{\mathbf{x}_n, y_n\}_{n=1}^N$ como 'anclas' y, para extender los resultados a entradas genéricas. Posteriormente se utilizan combinaciones lineales de estos valores ponderándolos según la distancia que hay entre las entradas genéricas y estos puntos ancla.

5.4.3 Escenario de múltiples salidas y otras extensiones

En un marco de múltiples salidas, dados N pares de datos, como $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$, donde $\mathbf{x}_i \in \mathbb{R}^D$, pero en este caso también las salidas son vectores $\mathbf{y}_i \in \mathbb{R}^{D_Y}$. Por lo tanto, para la tendencia local hay una función oculta vectorial $\widehat{\mathbf{f}}(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^{D_Y}$ para todos los posibles $\mathbf{x}$ en el espacio de entrada. Se puede escribir

$$\widehat{\mathbf{f}}(\mathbf{x}) = \sum_{n=1}^{N} \varphi(\mathbf{x}, \mathbf{x}_n)\mathbf{y}_n. \tag{5.8}$$

En cuanto a las estimaciones de las varianzas locales, se siguen los mismos procedimientos definiendo las siguientes cantidades vectoriales: $\widehat{\mathbf{q}}(\mathbf{x}) \approx E[\mathbf{y}^2|\mathbf{x}]$ para M1, $\widehat{\mathbf{v}}(\mathbf{x}) = \widehat{\mathbf{q}}(\mathbf{x}) - (\widehat{\mathbf{f}}(\mathbf{x}))^2$ y $\mathbf{v}_n = (\mathbf{y}_n - \widehat{\mathbf{f}}(\mathbf{x}_n))^2$ para M2.

Un λ para cada punto de datos. Además, también se podrían considerar diferentes hiperparámetros locales λ, por ejemplo λ_n con $n = 1, ..., N$, o más generalmente, con $\lambda = \lambda(\mathbf{x})$. En este caso, se tendrían diferentes funciones de coeficientes $\varphi_n(\mathbf{x}, \mathbf{x}_n) = \varphi_n(\mathbf{x}, \mathbf{x}_n|\lambda_n)$ (uno para cada input $\mathbf{x}_n$), de modo que la tendencia podría expresarse fácilmente como

$$\widehat{\mathbf{f}}(\mathbf{x}) = \sum_{n=1}^{N} \varphi_n(\mathbf{x}, \mathbf{x}_n)\mathbf{y}_n. \tag{5.9}$$

5.4.4 Añadiendo más flexibilidad al modelo inicial

El modelo inicial en la ec. (5.1) es un método no paramétrico flexible. Si se considera de nuevo la ec. (5.9) en un marco de salida única, se obtiene

$$\widehat{f}(\mathbf{x}) = \sum_{n=1}^{N} \varphi_n(\mathbf{x}, \mathbf{x}_n)y_n, \quad \text{con} \quad \varphi_n(\mathbf{x}, \mathbf{x}_n) = \varphi_n(\mathbf{x}, \mathbf{x}_n|\lambda_n).$$

Claramente, este es un modelo aún más flexible que la ec. (5.1), con N hiperparámetros para ajustar (donde N es el número de datos; es decir, el número de hiperparámetros crece con el número de datos). Claramente, se pueden diseñar escenarios intermedios donde la función de kernel única depende de dos hiperparámetros,

$$\varphi(\mathbf{x}, \mathbf{x}_n) = \varphi(\mathbf{x}, \mathbf{x}_n|\lambda_1, \lambda_2),$$

o más generalmente, de P hiperparámetros, $\varphi(\mathbf{x}, \mathbf{x}_n) = \varphi(\mathbf{x}, \mathbf{x}_n|\lambda_1, \lambda_2, ..., \lambda_P)$. Es importante señalar que todas estas ideas también pueden emplearse tanto en la estimación de la varianza como de la tendencia).

5.4.5 Uso conjunto de un kernel rectangular y una solución de mínimos cuadrados para el análisis de series temporales

Considerando los datos de entrada como series temporales, por medio de una instancia de tiempo escalar, $x = t \in \mathbb{R}$, donde el conjunto de datos está formado por los siguientes pares $\{t_i, y_i\}_{i=1}^N$. Considerando también que los intervalos, $t_i - t_{i-1}$, son constantes. En este caso, se puede omitir el índice t, y considerar el conjunto de datos $\{t_i, y_i\}_{i=1}^N$. En este escenario, como alternativa, se puede usar un modelo autorregresivo (AR),

$$y_i = a_1 y_{i-1} + a_2 y_{i-2} + ... + a_W y_{i-W} + \epsilon_y, \tag{5.10}$$

donde ϵ_y es una perturbación de ruido y los coeficientes a_i, para $i = 1, \ldots, W$, se obtienen mediante una minimización de mínimos cuadrados (LS) y la longitud de la ventana temporal W (es decir, el orden del filtro AR) se obtiene utilizando un criterio de información espectral (SIC) [107, 112]. Nótese que la ventana temporal W corresponde al uso de un kernel rectangular en un enfoque de kernel smoother (*pero* en este escenario solo considerando las muestras pasadas - ver la primera 'Observación' abajo). Por lo tanto, hasta ahora esta variante podría considerarse como un caso especial de la técnica genérica en la sección 5.2. Sin embargo, aquí los coeficientes que mezclan las muestras pasadas se obtienen mediante un procedimiento LS (no todos iguales al mismo valor). Luego, considerando los coeficientes estimados $\widehat{a}_i$ por LS, y $\widehat{W}$ por SIC, se obtiene

$$\widehat{f}(t_i) = \widehat{f}_i = \widehat{a}_1 y_{i-1} + \widehat{a}_2 y_{i-2} + ... + \widehat{a}_{\widehat{W}} y_{i-\widehat{W}}. \tag{5.11}$$

Como ejemplo, para aplicar M2 para estimar la varianza instantánea, se establece $v_i = (y_i - \widehat{f}_i)^2$, y se asume otro modelo AR sobre v_i con longitud de ventana ahora denominada H,

$$v_i = b_1 v_{i-1} + b_2 v_{i-2} + ... + b_H v_{i-H} + \epsilon_v, \tag{5.12}$$

donde nuevamente las estimaciones $\widehat{b}_i$ se pueden obtener por LS, y $\widehat{H}$ por SIC. La función de varianza instantánea resultante es

$$v(t_i) = v_i = \widehat{b}_1 v_{i-1} + \widehat{b}_2 v_{i-2} + ... + \widehat{b}_{\widehat{H}} v_{i-\widehat{H}}. \tag{5.13}$$

La aplicación de M1 podría realizarse de manera similar.

Observación. Los estimadores resultantes en esta sección siguen siendo una combinación lineal de (una parte de) las salidas. Por lo tanto, todavía se tienen kernel smoothers (o mejor *filtros lineales*, ya que solo consideran combinaciones de muestras pasadas). Sin embargo, nótese que en las ecuaciones (5.11)-(5.13) los coeficientes de las combinaciones lineales se obtienen por el método de mínimos cuadrados (LS). Por lo tanto, se tiene una ventana de longitud W (y luego H) pero esto difiere del uso de una función de núcleo rectangular por dos razones: (a) la ventana solo considera muestras pasadas; (b) los coeficientes no son todos iguales (a la relación 1 sobre el número de muestras incluidas en la ventana) sino que se ajustan por LS.

5.5 Experimentos numéricos

5.5.1 Aplicaciones de series temporales

En esta sección, se realizan 5 experimentos numéricos diferentes generados de distinta manera cada uno. El enfoque es el de series temporales ($x = t \in \mathbb{R}$) por dos razones principales: (a) en primer lugar, por simplicidad (se puede mostrar fácilmente una visualización de datos en un solo gráfico), (b) y no menos importante, hay competidores de referencia relevantes como los modelos GARCH [23, 68, 147], que también se pueden probar. Nótese que los modelos GARCH han sido diseñados específicamente para modelizar la volatilidad en series temporales.
En esta sección, se prueban las metodologías y los distintos modelos GARCH en cuatro ejemplos fáciles de reproducir. Estos cuatro ejemplos difieren en la forma en la que se han generado los datos. Estos lo han hecho como la suma de una función media, $f(t)$, y un término de error, $s(t)\varepsilon(t)$, es decir,

$$y(t) = f(t) + s(t)\varepsilon(t), \quad \varepsilon(t) \sim \mathcal{N}(0,1). \tag{5.14}$$

donde se ha utilizado el hecho de que se analizan series temporales, es decir, $x = t \in \mathbb{R}$. Las funciones $f(t)$ y $s(t)$ son deterministas y representan la tendencia y la desviación estándar de las observaciones $y(t)$. Mientras que $\varepsilon(t) \sim \mathcal{N}(0,1)$ representan un ruido aleatorio estándar Gaussiano, al igual que $v(t) = s(t)^2$. Para mantener la notación del resto del documento, el conjunto de datos $\{t_n, y_n\}_{n=1}^N$ estará formado por los N pares,

$$x_n = t_n, \quad y_n = y(t_n),$$

donde y_n será generado de acuerdo al modelo anterior. Los cuatro experimentos consideran:

1. $f(t) = \sin(0{,}5t), \quad s(t) = 0{,}2.$
2. $f(t) = t, \quad s(t) = 2|t|.$
3. $f(t) = 0{,}5t^2, \quad s(t) = 5\sin(0{,}3t).$

4. $f(t) = 0$ and $s(t)$ generado por un modelo GARCH.

En todos los casos, los datos generados $y(t)$ y la desviación estándar subyacente $s(t)$ (o varianza $v(t) = s(t)^2$) son conocidos, por lo tanto se pueden calcular los errores y evaluar el rendimiento de los algoritmos utilizados. Para la aplicación de los métodos propuestos, se considera una función kernel no lineal como

$$h(t, t_n) = \exp\left(-\frac{(t - t_n)^\alpha}{\lambda}\right), \tag{5.15}$$

con $\alpha \in \{1, 2, 10\}$.[2] El parámetro λ se ajusta mediante LOO-CV. Además, se comprueban siete combinaciones diferentes, cada una mezclando un método específico de aproximación de tendencia con una técnica específica de estimación de varianza:

- **Basado en kernel smoothers**: se examinan distintos métodos de regresión basados en las eqs. (5.1) y (5.15). Además, para la estimación de la varianza se consideran los procedimientos M1 y M2 descritos en la Sección 5.3. Más específicamente:
  - **KS-1-1**: $\alpha = 1$ y M1,
  - **KS-1-2**: $\alpha = 1$ y M2,
  - **KS-2-1**: $\alpha = 2$ y M1,
  - **KS-2-2**: $\alpha = 2$ y M2,
  - **KS-10-1**: $\alpha = 10$ y M1,
  - **KS-10-2**: $\alpha = 10$ y M2.
- **Basado en modelos AR**: se aplica el método descrito en la Sección 5.4.5, que emplea dos modelos autorregresivos, uno para la tendencia y otro para la varianza junto con el procedimiento M2.

La tabla 5.1 resume las siete implementaciones específicas consideradas anteriormente. Además, se prueban varios modelos GARCH(p,q): GARCH(1,1), GARCH(2,2), GARCH(5,5) y GARCH(10,10) (utilizando una implementación en Matlab de los modelos GARCH). **Observación.** Los modelos GARCH trabajan directamente con la señal sin tendencia, es decir, $|y_n - \widehat{f}_n|$. Para tener una comparación equivalente, se utiliza un kernel smoother con $\alpha = 2$, es decir, aplicando las ecs. (5.1)-(5.15) con $\alpha = 2$ (como en los métodos KS-2). Cabe recordar que los modelos GARCH han sido diseñados específicamente para realizar inferencias en series temporales heterocedásticas.

Tabla 5.1: Métodos propuestos (implementaciones específicas) probados en los experimentos numéricos.

Método	**KS-1-1**	**KS-1-2**	**KS-2-1**	**KS-2-2**	**KS-10-1**	**KS-10-2**	**Basado en AR**
Para la estimación de la tendencia	Ecs. (1)-(12), $\alpha = 1$		Ecs. (1)-(12), $\alpha = 2$		Ecs. (1)-(12), $\alpha = 10$		AR - Sección 5.4.5
Para la estimación de la varianza	M1	M2	M1	M2	M1	M2	AR and M2 - Sección 5.4.5

[2] El valor $\alpha = 1$ es el valor mínimo posible para tener una distancia L_p adecuada (con $p = \alpha$), cumpliendo con todas las propiedades de una distancia. La elección de $\alpha = 2$ es únicamente para considerar la distancia euclidea. Para valores mayores de α, se establece un $\alpha = 10$ que también evitará posibles problemas numéricos.

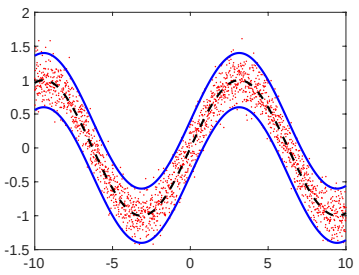
Experimento 1

Como se mencionó anteriormente, en este primer ejemplo se considera

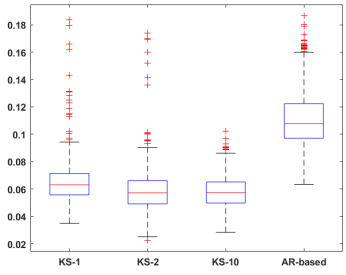
$$y(t) = \sin(0{,}5t) + 0{,}2\ \varepsilon(t), \quad \varepsilon(t) \sim \mathcal{N}(0,1),$$

es decir, donde $f(t) = \sin(0{,}5t)$ y $s(t) = 0{,}2$ (en este caso se obtiene una serie homocedástica). Se generan $N = 10^4$ datos con $t \in [-10, 10]$. Se aplican todos los algoritmos, se calcula el error absoluto medio (MAE), se repiten y promedian los resultados sobre 10^4 ejecuciones independientes. La Figura 5.1a muestra los datos del experimento, generados con el modelo anterior.

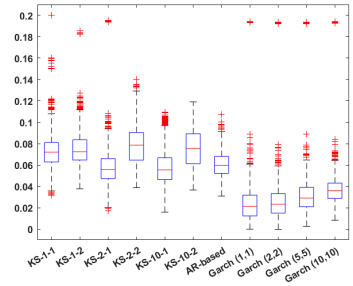
Resultados: Las Figuras 5.1b-5.1c proporcionan los resultados de este ejemplo. La Figura 5.1b muestra el MAE obtenido en la estimación de la tendencia, y la Figura 5.1c muestra el MAE en la estimación de la desviación estándar. En este caso, en lo que respecta a la tendencia, se observa que el método basado en AR tiene un MAE más alto que el resto al estimar la tendencia, mientras que el resto de valores MAE son bastante similares. También se obtienen unos resultados similares al aproximar con la desviación estándar, aunque los MAE menores son proporcionados por los modelos GARCH. En cuanto a la estimación de la desviación estándar para este ejemplo, el M1 parece proporcionar mejores resultados que el M2.



(a) Datos del ejemplo 5.5.1.



(b) MAE en la estimación de la tendencia. Ver la Tabla 5.1.



(c) MAE en la estimación de la desviación estándar.

Figura 5.1: (a) Datos del experimento (puntos rojos) generados por el modelo en la Sección 5.5.1; la función $f(t)$ se muestra como una línea discontinua negra y las dos líneas azules continuas representan $f(t) \pm 2s(t)$ (conteniendo aproximadamente el 95 % de la masa de probabilidad). **(b)-(c)** Box-plots de los MAEs de los diferentes algoritmos del experimento en la Sección 5.5.1.

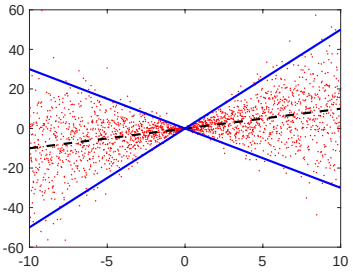
Experimento 2

En este segundo ejemplo, se considera

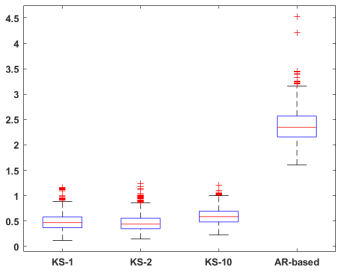
$$y(t) = t + 2|t|\ \varepsilon(t), \quad \varepsilon(t) \sim \mathcal{N}(0,1),$$

donde $f(t) = t$ y $s(t) = 2|t|$ (es decir, en este caso, se está ante un escenario heterocedástico). Como anteriormente, se generan $N = 10^4$ datos con $t \in [-10, 10]$. Se aplican todos los algoritmos, y se calcula el error absoluto medio (MAE) y nuevamente se repite y promedia los resultados en 10^4 ejecuciones independientes. La Figura 5.2a muestra los datos del experimento generado según el modelo anterior.

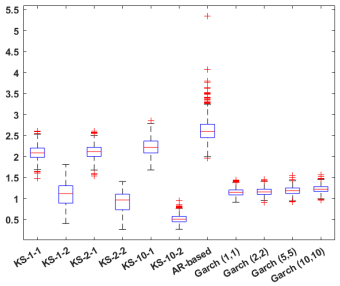
Resultados. Los resultados se presentan en las Figuras 5.2b-5.2c. Nuevamente, el MAE para el método basado en AR parece ser más alto que el de los demás algoritmos, al estimar la tendencia y la desviación estándar de la serie temporal. En cuanto a la estimación de la desviación estándar, los mejores resultados los proporcionan los métodos de kernel que utilizan el M2 (el MAE es particularmente pequeño con $\alpha = 10$). Por lo tanto, es notable que, incluso en este *escenario heterocedástico*, los métodos propuestos proporcionan resultados similares y, en algunos casos incluso mejores que los modelos GARCH.



(a) Datos del ejemplo 5.5.1.



(b) MAE en la estimación de la tendencia. Ver la Tabla 5.1.



(c) MAE en la estimación de la desviación estándar.

Figura 5.2: (a) Datos del experimento (puntos rojos) generados por el modelo en la Sección 5.5.1; la función $f(t)$ se muestra como una línea discontinua negra y las dos líneas azules continuas representan $f(t) \pm 2s(t)$ (conteniendo aproximadamente el 95 % de la masa de probabilidad). **(b)-(c)** Box-plots de los MAEs de los diferentes algoritmos del experimento en la sección 5.5.1.

Experimento 3

En este tercer ejemplo, se considera una desviación estándar que varía periódicamente y una función de tendencia polinómica de segundo orden, es decir,

$$y(t) = 0{,}5t^2 + 5\sin(0{,}3t)\ \varepsilon(t), \quad \varepsilon(t) \sim \mathcal{N}(0,1),$$

donde $f(t) = 0{,}5t^2$ y $s(x) = 5\sin(0{,}3t)$. Nuevamente, se está ante un escenario heterocedástico. Como se hizo anteriormente, se generan $N = 10^4$ datos con $t \in [-10, 10]$. Se calcula el error absoluto medio (MAE), y nuevamente se repite y promedia los resultados en 10^4 ejecuciones independientes. La Figura 5.3a muestra los datos del experimento generados según el modelo descrito.

Resultados. Las Figuras 5.3b-5.3c muestran los box-plots del MAE en la estimación de la tendencia y la desviación estándar. Como en ejemplos anteriores, el basado en AR da los peores resultados al calcular la tendencia de la serie temporal, mientras que el resto de los algoritmos tienen valores de error muy similares (recordando que los modelos GARCH emplean el método KS-2 para la estimación de la tendencia). Sin embargo, el método basado en AR proporciona un buen rendimiento en la aproximación de la desviación estándar. La técnica basada en AR utiliza M2 para la varianza. Además, para la estimación de la desviación estándar, los métodos KS-1-1-1, KS-2-1, KS-10-1 dan los errores menores. Por lo tanto, parece que en este caso M1 funciona mejor, incluso que los modelos GARCH.

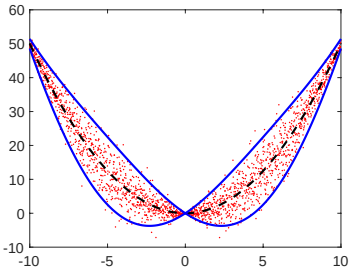
Experimento 4

Considerando:

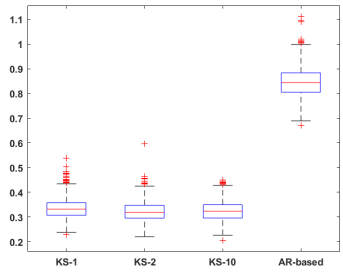
$$\begin{aligned} y(t) &= s(t)\ \varepsilon(t), \quad \varepsilon(t) \sim \mathcal{N}(0,1), \\ y_t &= s_t \epsilon_t = z_t, \end{aligned}$$

es decir, $f(t) = f_t = 0$, y habiendo denotado $z_t = s_t\epsilon_t$. El proceso para la desviación típica $s_t = \sqrt{v_t}$ será un modelo GARCH(P, Q), es decir,

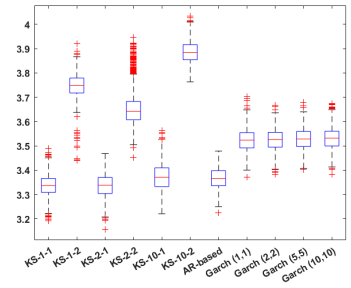
$$\begin{aligned} v_t =& \kappa + \gamma_1 v_{t-1} + \ldots + \gamma_P v_{t-P} + \\ &+ \alpha_1 z_{t-1}^2 + \ldots + \alpha_Q z_{t-Q}^2. \end{aligned}$$



(a) Datos del ejemplo 5.5.1.



(b) MAE en la estimación de la tendencia. Ver la Tabla 5.1.



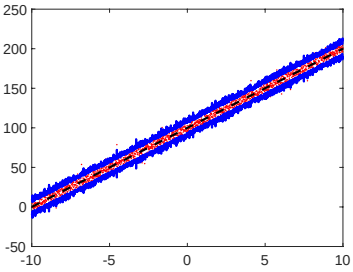
(c) MAE en la estimación de la desviación estándar.

Figura 5.3: (a) Datos del experimento (puntos rojos) generados por el modelo en la Sección 5.5.1; la función $f(t)$ se muestra como una línea discontinua negra y las dos líneas azules continuas representan $f(t) \pm 2s(t)$ (conteniendo aproximadamente el 95 % de la masa de probabilidad). **(b)-(c)** Box-plots de los MAEs de los diferentes algoritmos del experimento en la Sección 5.5.1.

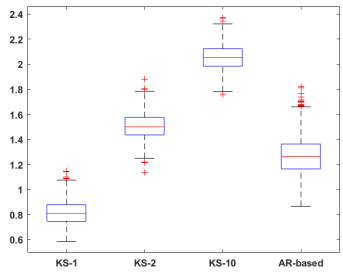
En este ejemplo, se genera la varianza a partir de un modelo GARCH$(2,3)$

$$\begin{aligned} v_t =& 2 + 0{,}2 \cdot v_{t-1} + 0{,}1 \cdot v_{t-2} + 0{,}2 \cdot z_{t-1}^2 + \\ &+ 0{,}2 \cdot z_{t-2}^2 + 0{,}2 \cdot z_{t-3}^2 \end{aligned} \tag{5.16}$$

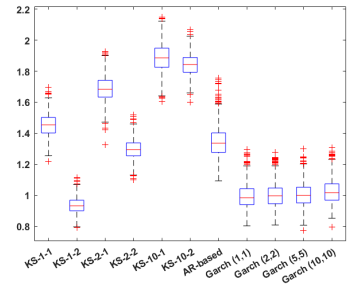
Claramente, se trata de un escenario heterocedástico.
Se generan $N = 10^4$ datos y se aplican las diferentes técnicas. Se calcula el error absoluto medio (MAE), y se repite y promedian los resultados en 10^4 ejecuciones independientes. Los datos del experimento generados se muestran en la Figura 5.4a. Nótese que la tendencia es lineal pero la volatilidad presenta cambios muy rápidos ya que es generada por un modelo GARCH(2,3) (ver línea azul en la Figura 5.4a).
Resultados. Los resultados de este ejemplo se presentan en las Figuras 5.4b-5.4c. El mejor rendimiento en la estimación de $s(t)$ lo obtienen KS-1-2 y todos los modelos GARCH (como se esperaba en este caso). Los basados en AR y KS-2-2 también proporcionaron valores cercanos respecto el MAE. En este ejemplo, claramente M2 supera a M1. Cabe destacar que, incluso este escenario es particularmente favorable para los modelos GARCH, todos los métodos propuestos proporcionan resultados competitivos. Los valores de MAE más altos (por lo tanto, peores) son dados por el KS con $\alpha = 10$ que es una elección/ajuste bastante extremo de este parámetro, especialmente cuando la volatilidad sigue una regla autorregresiva. De hecho, un kernel laplaciano con $\alpha = 1$ es más adecuado en este marco. Para más consideraciones ver [106, Sección 10].



(a) Datos del ejemplo 5.5.1.



(b) MAE en estimación de tendencia. Ver la tabla 5.1.



(c) MAE en estimación de desviación estándar.

Figura 5.4: (a) Datos del experimento (puntos rojos) generados por el modelo en la Sección 5.5.1. La tendencia lineal $f(t) = t$ se muestra como una línea negra discontinua, mientras que las dos líneas azules continuas representan $f(t) \pm 2s(t)$ donde $s(t)$ es generado por un modelo GARCH. **(b)-(c)** Box-plots de los MAE de los diferentes algoritmos del experimento en la Sección 5.5.1.

Comentarios finales sobre las aplicaciones a series temporales

Todos los métodos propuestos basados en kernel smoothers (KS), M1 y M2 proporcionan siempre resultados competitivos (con valores de MAE más cercanos o menores) en comparación con los algoritmos de referencia como los métodos GARCH, que han sido diseñados específicamente para la estimación de la desviación estándar en series temporales.
Estos experimentos numéricos muestran que los métodos propuestos M1 y M2 pueden incluso superar a los modelos GARCH en la estimación de la desviación típica en series temporales. Los modelos GARCH parecen proporcionar resultados más robustos cuando la señal ha sido generada por un modelo GARCH en la Sección 5.5.1 y, sorprendentemente, en el escenario homocedástico en la Sección 5.5.1. Los métodos basados en kernel smoothers superan virtualmente a los basados en AR debido al hecho de que incorporan las "muestras futuras" en sus estimadores. Una ventana rectangular, que incluye tanto muestras pasadas como futuras, debería mejorar el rendimiento del método basado en AR.

5.5.2 Aplicación en dimensiones de entrada superiores: datos reales sobre emociones de paisajes sonoros

Los modelos GARCH han sido diseñados específicamente para estimar la volatilidad en series temporales, por lo tanto $x = t \in \mathbb{R}$. Sin embargo, los métodos propuestos en este trabajo pueden aplicarse a problemas con entradas multidimensionales $\mathbf{x} \in \mathbb{R}^D$ con $D \geq 1$.
Más específicamente, aquí se trata de analizar una base de datos de emociones de paisajes sonoros. En la última década, los paisajes sonoros se han convertido en uno de los temas más activos en acústica. De hecho, el número de proyectos de investigación relacionados y artículos científicos está creciendo continuamente [3, 99, 97]. En planificación urbana y acústica ambiental, el esquema general consiste en (a) grabación de paisajes sonoros, (b) cálculo de indicadores acústicos y psicoacústicos de las señales, (c) inclusión de otros indicadores de contexto (por ejemplo, información visual [11]), y (d) clasificación de señales de audio de paisajes sonoros utilizando descriptores emocionales. Finalmente, el modelo puede ser desarrollado [110].
Aquí, se considera la base de datos de emo-paisajes sonoros (paisajes sonoros emocionales) (EMO) [61, 63], donde $N = 1200$ y $D = 122$, es decir, $\mathbf{x} \in \mathbb{R}^{122}$, que es la base de datos de paisajes sonoros más grande disponible con anotaciones de etiquetas emocionales, y también la más reciente hasta ahora [61, 63]. Entre las 122 características de audio que forman las entradas $\mathbf{x}$, se pueden distinguir tres grupos principales de características de las señales de audio: características psicoacústicas, características de dominio temporal y características de dominio de frecuencia. La salida y que se considera, se denomina *arousal*. Para más información, se puede encontrar la base de datos completa en `https://metacreation.net/emo-soundscapes/`.Se implementa una función kernel del tipo:

$$h(\mathbf{x}, \mathbf{z}) = h(\mathbf{x}, \mathbf{z}|\lambda) = \exp\left(-\frac{||\mathbf{x} - \mathbf{z}||^{\alpha}}{\lambda}\right),$$

con $\alpha = 2$. Por lo tanto, considerando la base de datos EMO $\{\mathbf{x}_i, y_i\}_{i=1}^{1200}$ se aplica M1, M2 y CV-LOO para ajustar λ. El MAE en la estimación de la tendencia es $0{,}0388$ obtenido con un parámetro óptimo $\lambda^* = 0{,}12$. Además, en ambos casos, se observa un aumento sensible de la volatilidad de la salida en la última mitad de las muestras, para i de 621 a 1213. Este podría ser un resultado interesante, que debería ser discutido con expertos en el campo y en la base de datos EMO.

5.6 Conclusiones

En este trabajo, se han propuesto dos métodos para estimar la varianza instantánea/local en problemas de regresión basados en kernel smoothers. Desde otro punto de vista, este trabajo también puede verse como una

forma de realizar regresión cuantílica (al menos, una regresión cuantílica de segundo orden) utilizando kernel smoothers.

En comparación con otros procedimientos mencionados en la literatura, los métodos propuestos tienen una aplicabilidad mucho más amplia que otras técnicas, ya que estas solo se pueden emplear para series temporales, como es el caso de los conocidos modelos de la familia GARCH (donde el índice de tiempo es un número escalar, $x = t \in \mathbb{R}$). De hecho, los modelos propuestos pueden emplearse con variables de entrada multidimensionales $\mathbf{x}_n \in \mathbb{R}^D$. Además, las técnicas propuestas pueden aplicarse a series temporales ($D = 1$), pudiendo también pueden obtener una predicción entre dos instantes de tiempo consecutivos (es decir, con una resolución más alta que los datos recibidos). En general, la estimación de la varianza local puede darse para cualquier dato de entrenamiento $\mathbf{x}_i$ pero también para cualquier entrada genérica de test $\mathbf{x}$.

Además, al analizar series temporales, las simulaciones numéricas han mostrado que los modelos propuestos proporcionan resultados competitivos incluso respecto a los modelos GARCH (que están específicamente diseñados para analizar series temporales). También se ha proporcionado una aplicación en datos reales y entradas multidimensionales (con dimensión 122).

Como línea de investigación futura, se plantea diseñar funciones de kernel smoother con parámetros locales λ, e incluso más generalmente con $\lambda = \lambda(\mathbf{x})$, para modelizar mejor la heteroscedasticidad. Además, se valorará la aplicación y diseño a escenarios de múltiples salidas.

Capítulo 6

Redes neuronales para el análisis de series temporales financieras

La predicción precisa de las distintas acciones del mercado financiero, es un área de la investigación que abarca mucha literatura. En todo ello, tanto los modelos de predicción como las series temporales, tienen mucho protagonismo. Dentro de los algoritmos de predicción, las redes neuronales están protagonizando un momento álgido, ya que al admitir funciones no lineales, permiten tener una precisión más alta [85, 39] respecto modelos más tradicionales, al ser capaces de captar una mayor cantidad de relaciones entre distintas variables y distintas temporalidades. Otras investigaciones hacen referencia a las distintas arquitecturas de redes neuronales utilizadas para la predicción. Es el caso de las LSTM [161], que al ser una arquitectura recurrente, funciona muy bien para series temporales. Destaca también la arquitectura GRU, perteneciente también a la familia de las redes recurrentes, como una buena alternativa para obtener predicciones precisas [139]. Algunos autores combinan incluso esta red con modelos clásicos como los ARIMA [5]. También ha habido experimentos con arquitecturas convolucionales, con el fin de encontrar el modelo que tenga una mayor precisión en su predicción [33].

Las investigaciones actuales están enfocadas en la combinación de distintos tipos de redes neuronales, tratando así de buscar un modelo más complejo, pero que a su vez recoja mejor las particularidades de los datos financieros. Arquitecturas como GRU-CNN [169], CNN-GRU-attention [38], CNN-LSTM y GRU-CNN [12, 141] y LSTM-CNN [88] son algunas de las combinaciones de redes encontradas en la literatura más actual. En linea a estas recientes investigaciones, este capítulo ha tratado de comparar el resultado del entrenamiento que tiene como alcance final, la predicción del precio de cierre de cotización de la empresa Apple, por medio de los precios de apertura de otros activos financieros.

6.1 Objetivo inicial

La hipótesis principal de este capítulo consiste en realizar una comparación de predicciones utilizando distintas redes neuronales, poniendo un énfasis especial en el impacto de la capa de Normalización Divisiva Generalizada (GDN). Se ha tratado de identificar cuál es el algoritmo que mejor predice los precios de cotización de cierre diarios de Apple, utilizando como variables independientes el precio de apertura del mismo día y de días anteriores de las acciones de Apple, Amazon, AMD, Microsoft y Google, se ha utilizado una ventana móvil de 10 periodos. Los algoritmos que se han utilizado en la comparación correspondiente, es el modelo lineal, GRU, LSTM y CNN. Todos ellos con su versión estándar y una versión mejorada que incorpora una capa GDN.

Es importante resaltar que la estructura de la capa GDN facilita su integración junto a otras redes neuro-

nales, donde puede ayudar a mejorar la robustez y el rendimiento del modelo al proporcionar una normalización adaptativa sobre las características extraídas de los datos. Por ello, se ha puesto especial atención en comprobar si las redes neuronales que tienen capa GDN, predicen mejor que aquellas redes homónimas que entrenan sin ella.

6.2 Metodología aplicada

Se comienza haciendo referencia a las librerías utilizadas, así como su versión, con el fin de facilitar la replicabilidad de los experimentos. A continuación se detalla cómo se han extraído los datos, así como una descripción detallada de los mismos. Se explica el método de normalización y validación cruzada que se ha utilizado, así como la partición de datos empleada. Posteriormente se explican cuáles han sido los algoritmos que se han entrenado, así como la métrica de error utilizada para evaluar la precisión de las predicciones.

6.2.1 Tabla de librerías, usos y versiones

Para una correcta reproducibilidad de los experimentos, se presenta a continuación una tabla con las versiones actuales de las librerías que se han utilizado para el desarrollo de experimentos, tal y como se puede apreciar en la tabla 6.1.

Librería	Uso	Versión
yfinance	Extracción de datos financieros en tiempo real y datos históricos.	0.2.38
pandas	Manipulación y análisis de datos estructurados.	2.0.3
tabulate	Formateo de tablas.	0.9.0
tensorflow	Aprendizaje automático y redes neuronales.	2.15.0
keras	API para construir y entrenar modelos en TensorFlow.	2.15.0
seaborn	Visualización de datos estadísticos.	0.13.1
scipy	Cálculo científico y técnico.	1.11.4
statsmodels	Modelado estadístico, exploración y estimación de datos.	0.14.2

Tabla 6.1: Versiones y usos de las librerías instaladas en el entorno de Python

6.2.2 Obtención y correlación de los datos

El primer paso en el análisis de series temporales, es la recopilación y el preprocesamiento de datos. Se ha utilizado la librería `yfinance`, para obtener datos históricos de precios de un total de 20 activos financieros, siendo estos: Apple Inc. (AAPL), Meta Platforms Inc. (anteriormente Facebook) (META), Microsoft Corporation (MSFT), Alphabet Inc. (Google) (GOOGL), Exxon Mobil Corporation (XOM), Chevron Corporation (CVX), JPMorgan Chase & Co. (JPM), Goldman Sachs Group Inc. (GS), Pfizer Inc. (PFE), Johnson & Johnson (JNJ), Advanced Micro Devices Inc. (AMD), NVIDIA Corporation (NVDA), Bank of America Corp (BAC), Citigroup Inc. (C), Amazon.com Inc. (AMZN), Netflix Inc. (NFLX), Merck & Co. Inc. (MRK), Boeing Company (BA),

General Electric Company (GE) y SPDR Gold Trust (GLD). Los datos han sido extraídos para el período que abarca desde el 10 de mayo de 2005 hasta el 10 de mayo de 2024, con frecuencia diaria. Una vez obtenido los datos de todos ellos, se ha realizado una matriz de correlación entre todos los activos, pero con especial énfasis en la correlación respectiva al activo Apple (para visualizar dicha matriz, ver el Anexo A). Como el objetivo es predecir el precio de cierre de Apple por medio de otros activos, se ha buscado únicamente aquellos 4 que tienen la correlación más alta, siendo estos activos Amazon (AMZN), Microsoft (MSFT), AMD, Apple (AAPL) y Google (GOOGL). Esto se aprecia en el siguiente análisis de correlación.

Análisis de la matriz de correlación: La Figura 6.1,contiene la matriz que muestra los coeficientes de correlación lineal entre los precios de cierre ajustados de las acciones de las empresas. Los coeficientes de correlación varían entre $+0.87$ y $+0.98$, indicando una fuerte correlación positiva entre los movimientos de los precios de estas compañías tecnológicas. El color en la matriz indica la fuerza de estas correlaciones, siendo los tonos más azules una correlación más débil, y los tonos más rojos una correlación más fuerte.

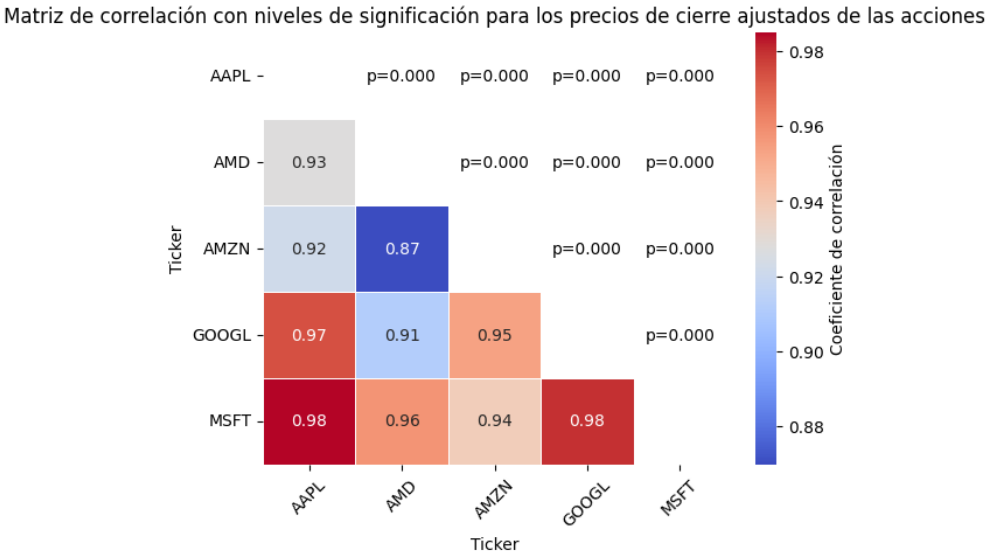


Figura 6.1: Matriz de correlación con niveles de significación para los precios de cierre ajustados de las acciones de Apple (AAPL), Amazon (AMZN), AMD (AMD), Google (GOOGL), y Microsoft (MSFT). Cada celda muestra el coeficiente de correlación entre los pares de acciones especificados. Los coeficientes de correlación varían entre 0.87 y 0.98, donde valores más altos indican una relación más fuerte. Todos los p-valores asociados con estas correlaciones son 0.000, lo que indica que las correlaciones son estadísticamente significativas.

Correlación entre Apple y el resto de empresas: Estos coeficientes sugieren que los precios de las acciones de Apple están muy alineados con los movimientos de las acciones de estas otras grandes tecnológicas. Las correlaciones particularmente altas con Google y Microsoft (0.97 y 0.98 respectivamente) ponen en relieve una evolución en la cotización de dichas compañías casi idéntica.

Es importante destacar que todos los p-valores de la matriz de correlación, son 0.000, lo que indica que las correlaciones son altamente significativas desde el punto de vista estadístico. Esto rechaza la hipótesis nula de que no hay correlación entre los precios de las acciones de estas empresas, o lo que es lo mismo, se descarta la idea de que estas fuertes correlaciones sean resultado del azar.

6.2.3 Descripción de los datos

Una vez que se han obtenido las series temporales con las que se va a trabajar, se ha realizado un análisis descriptivo de las mismas.

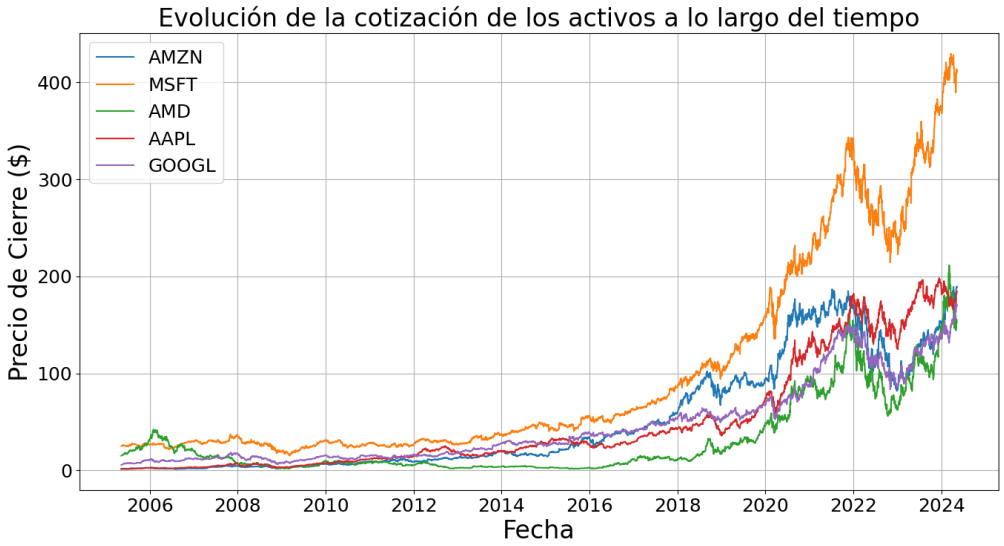


Figura 6.2: Evolución de la cotización a precio de cierre de los activos Apple (AAPL), Amazon (AMZN), AMD (AMD), Google (GOOGL), y Microsoft (MSFT)

Ticker	Media	Desv. típica	Min	50 %	Max
AAPL	59.93	55.44	6.86	32.34	198.11
AMZN	63.34	54.77	5.43	40.96	186.57
AMD	32.48	39.49	1.62	9.96	161.91
GOOGL	54.10	39.16	10.92	41.08	149.84
MSFT	115.39	101.84	23.01	62.66	382.70

Tabla 6.2: Estadística descriptiva de las acciones de Apple (AAPL), Amazon (AMZN), AMD (AMD), Google (GOOGL), y Microsoft (MSFT)

Ticker	Rendimiento anualizado	Volatilidad anualizada
AAPL	0.27	0.28
AMZN	0.27	0.33
AMD	0.35	0.56
GOOGL	0.19	0.27
MSFT	0.21	0.25

Tabla 6.3: Rendimiento y volatilidad anualizada de Apple (AAPL), Amazon (AMZN), AMD (AMD), Google (GOOGL), y Microsoft (MSFT)

Apple (AAPL): Fue creada en 1976 por Steve Jobs, Steve Wozniak y Ronald Wayne, es mundialmente reconocida principalmente por su innovación en productos electrónicos y software, con productos tan relevantes

como el iPhone, iPad y Mac. La cotización de Apple muestra en la Figura 6.2, un crecimiento constante y sostenido, lo que refleja la fuerte lealtad del consumidor y la innovación continua de la compañía. Tal y como se aprecia en la tabla 6.2, la cotización media es de 59.93$, con un rango de precios que va de 6.86$ a 198.12$. Por otro lado, la compañía se sitúa como una empresa sólida y estable, puesto que tal y como se aprecia en la tabla 6.3, el retorno anualizado robusto es del 27.08 %, junto a una volatilidad anualizada moderada, siendo esta del 28.21 %.

Amazon (AMZN): Empresa fundada en 1994 por Jeff Bezos. Comenzó siendo una librería online y ha evolucionado hasta convertirse hoy en día en un gigante del comercio electrónico y de servicios cloud, con AWS. En la Figura 6.2 se puede apreciar cómo se ha revalorizado la cotización de Amazon de forma significativa y consistente a lo largo del periodo analizado. Especialmente desde 2015, donde la cotización comenzó una clara tendencia alcista hasta 2020, coincidiendo con la pandemia de COVID-19. Tal y como se observa en la tabla 6.2, entre 2005 y 2024 el precio de la acción ha oscilado desde su valor mínimo en los 5.43$, hasta su valor máximo de 186.57$. El precio de cotización medio es de 63.35$, mientras que los precios han sido bastante dispersos respecto la media, tal y como indica la desviación típica de 54.77$. Por otra parte, la acción muestra un retorno anualizado alto, siendo este del 27 % tal y como indica la tabla 6.3, señal de un crecimiento robusto interanual. La volatilidad anualizada está en línea con la expansión y el crecimiento agresivo de la compañía, reflejando elevadas fluctuaciones de precios durante los periodos de crisis, mostrando a su vez una rápida recuperación.

AMD (AMD): Advanced Micro Devices (AMD), es una empresa de semiconductores fundada en 1969, y conocida por sus procesadores y tarjetas gráficas, la cual compite directamente con empresas como Intel en el mercado de CPUs y con Nvidia en el de GPUs. Este activo, ha sido un valor con tendencia muy bajista en lo que respecta a la evolución de su cotización, durante los 10 primeros años del periodo de estudio, siendo estos desde mayo de 2005 a mayo de 2015, tal y como se puede observar en la Figura 6.2. Fue a partir de ese momento cuando inició la tendencia alcista que le ha acompañado hasta 2024 incluido. En la tabla 6.2, se observa que la cotización media es de 32.50$, y su desviación típica de 39.49$. Por otro lado, los precios han fluctuado entre 1.62$ y 161.91$. siendo un valor con una elevada fluctuación. Cabe destacar que AMD es una empresa con un rendimiento anualizado bastante elevado, siendo este del 35.81 % tal y como se indica en la tabla 6.3, pero por otro lado, su volatilidad es muy alta, del 56.28 %. Todo ello lo posiciona como un valor muy arriesgado y volátil.

Google (GOOGL): Fue fundado en 1998 por Larry Page y Sergey Brin, ha crecido desde ser un motor de búsqueda hasta convertirse en Alphabet Inc., convirtiéndose en un conglomerado de compañías tecnológicas, que incluye empresas que dominan áreas como publicidad online, software, hardware y tecnología de inteligencia artificial. La Figura 6.2 sitúa a la acción con un crecimiento sostenido, acelerándose esta tendencia alcista después de 2014, hasta comienzos de 2022. Por otro lado la tabla 6.2 muestra una media de 54.13$, con los precios comprendidos entre 10.92$ y 149.39$. Estas cifras resaltan la solidez de Google como líder en tecnología y publicidad, junto con una diversificación de ingresos que ha permitido una expansión sostenida en nuevos mercados y tecnologías. En lo que respecta a los rendimientos anualizados, Google ofrece el retorno más bajo en comparación con el resto de compañías que se están utilizando para el estudio, siendo este del 19.36 % tal y como se muestra en la tabla 6.3. A su vez, la volatilidad anualizada es moderada, siendo esta del 27.82 %.

Microsoft (MSFT): Fue fundada por Bill Gates y Paul Allen en 1975. Microsoft es conocida por su sistema operativo Windows y su suite de productividad Office, además de una gama amplia de servicios de software y de cloud, todo ello de la mano de Azure. Microsoft ha demostrado durante todo el periodo estudiado, que ha sido la acción con una mayor revalorización tal y como se muestra en la Figura 6.2. Inició su senda alcista más a largo plazo, en torno a 2014, coincidiendo con la llegada de Satya Nadella como nuevo CEO de la compañía. La tabla 6.2, muestra que el rango de precios entre 23.01$ y 382.70$, mientras que su cotización media es de 115.40$. Todo ello indica tanto su posición consolidada en la industria, como su estabilidad y fortaleza como activo financiero, generando gracias a su capitalización, nuevas oportunidades de crecimiento, principalmente en el ámbito cloud. En comparación con el resto de valores de cotización, la volatilidad anualizada es relativamente media-baja, siendo esta del 25.96 %, mientras que por otro lado, la rentabilidad anualizada es del 21.43 % (tal y como indica la tabla 6.3, convirtiendo ambas métricas a Microsoft en un valor estable y con riesgo moderado.

A continuación se muestra el preprocesamiento básico realizado, que incluye la normalización de los precios de los activos, para garantizar que todas las características tengan una escala comparable. De este modo los algoritmos mejoran la estabilidad y su rendimiento, ya que así se evita que los precios altos dominen el proceso de entrenamiento.

6.2.4 Normalización de los datos

Para mejorar la estabilidad numérica y minimizar el problema de explosión del gradiente durante el entrenamiento de los algoritmos, se ha optado por normalizar los datos durante su preprocesado. La normalización de los datos no solo ayuda a prevenir problemas numéricos, sino que además acelera la convergencia del algoritmo de optimización durante el entrenamiento al proporcionar los datos homogéneos. Esta normalización se consigue mediante el ajuste de los precios a una escala común. Matemáticamente esta transformación se realiza de la siguiente manera: $z = \frac{(x-\mu)}{\sigma}$, donde x es el valor original, μ es la media del conjunto de datos, y σ es la desviación estándar del conjunto de datos. Esta transformación es conocida como normalización Z-score, y se puede apreciar cómo la normalización ajusta la distribución de los datos de forma que la media (μ) sea igual a cero y la desviación estándar (σ) igual a uno.

Aportes científicos como [65] han destacado la importancia de la aplicación de las técnicas de normalización durante el entrenamiento de redes neuronales, llegando a la conclusión de que las técnicas apropiadas tienen una gran influencia de cara a una correcta convergencia y rendimiento de las redes neuronales. Esto supone una mejora en lo que respecta a la eficiencia y la estabilidad del entrenamiento de los algoritmos.

6.2.5 Partición de los datos y validación cruzada

Para un entrenamiento robusto de los algoritmos, se ha realizado la partición de datos en conjuntos de entrenamiento, validación y test o prueba. Este proceso permite adaptar y optimizar los modelos, y además permite evaluar su capacidad de generalización de forma efectiva. Esto garantiza que el rendimiento del modelo sea generalizable para cualquier partición de datos que se realice, es decir, se busca que el modelo aprenda en lugar de memorizar. Esta partición debe de ser suficientemente representativa para los datos de entrenamiento, validación y test. A continuación se muestra cómo se han dividido los datos en los conjuntos de entrenamiento, validación y test, tanto en términos porcentuales como en valores numéricos absolutos basados en un total de 4783 observaciones. Tal y como se observa en la tabla 6.4, se ha realizado la partición de los datos test, que son aquellos datos para los cuales su finalidad es proporcionar una evaluación imparcial del modelo final con datos no observados durante el entrenamiento. De este modo se puede obtener un rendimiento real del algoritmo.

Conjunto de datos	Porcentaje del total de datos	Número de muestras
Test	20 %	954
Entrenamiento	65 %	3246
Validación	15 %	573

Tabla 6.4: Distribución de los conjuntos de datos para entrenamiento, validación y prueba.

Para ello se ha utilizado un 20 % del total de los datos de 4773 (4783 observaciones menos la ventana móvil de 10 observaciones que se usa para la realización de cada predicción de la serie temporal), resultando en 954 observaciones destinadas para pruebas.

Una vez separada la sección prueba (test), se dividen los datos restantes (el 80 %) en entrenamiento (train) y validación, siendo un total de 3819 observaciones. Durante el entrenamiento se van a utilizar grandes cantidades de datos para entrenar los algoritmos, con el objetivo de optimizar sus hiperparámetros con el objetivo de minimizar la función de coste. Para ello, la cantidad de datos debe de ser lo suficientemente representativa como para poder aprender ante todas las posibles casuísticas de los datos, siendo esta cantidad del 85 % de las observaciones restantes (un total de 3246). Por otra parte, se debe de reservar un pequeño conjunto de datos llamado validación para que, a medida que se van afinando los hiperparámetros del modelo, se vaya evaluando el estado del entrenamiento, de modo que se pueda ir evitando el sobreajuste del mismo. Para ello, se utilizarán el 15 % de los datos restantes para validar dicho entrenamiento, siendo un total de 573 observaciones para validación.

Cabe destacar que no se ha realizado una única partición de datos, sino que por medio de la validación cruzada se han creado 10 conjuntos de ensayo "train-validation-test" diferentes. De modo que para cada una de estas 10 divisiones, se entrena el modelo y posteriormente se evalúa con un conjunto de prueba diferente. Diversas investigaciones científicas remarcan la importancia de la partición de datos para una mejor precisión de los algoritmos. Entre ellas destaca el aporte de [91] donde se destaca las ventajas de la validación cruzada y las técnicas de partición de datos para una evaluación de modelos precisa. El objetivo de entrenar el algoritmo con cada uno de estos ensayos, radica en la necesidad de que los modelos sean lo suficientemente robustos, y que del mismo modo se pueda obtener una comprensión profunda respecto la capacidad de generalización del mismo. Una vez realizada la partición de datos de forma completa, se procede a entrenar los algoritmos y posteriormente a evaluar su rendimiento.

6.2.6 Entrenamiento

A continuación, se proporciona una breve descripción de cada uno de los modelos predictivos que se han empleado, y su relevancia en el contexto del análisis de series temporales financieras:

Modelo lineal El modelo lineal es una aproximación simple que asume una relación lineal entre las variables predictoras y la variable objetivo. Aunque es limitado en su capacidad para capturar patrones complejos entre los datos, sirve como un punto de referencia básico para evaluar el rendimiento de los modelos más avanzados 4.

Red GRU (Gated Recurrent Unit) Las redes GRU son una variante simplificada de las LSTM, diseñadas para capturar dependencias temporales en series temporales. Las GRU combinan las celdas de memoria y las puertas de entrada y olvido en una sola unidad, lo que reduce la complejidad computacional sin que la capacidad de modelización a largo plazo se vea afectada 4.

Red LSTM (Long Short-Term Memory) Las redes LSTM están diseñadas para ser capaces de captar dependencias a largo plazo en series temporales. Estas redes son especialmente útiles en series temporales financieras donde los patrones históricos pueden influir de forma significativa en los valores futuros 4.

Red convolucional (CNN) Las redes CNN son eficaces para detectar patrones locales en los datos de series temporales. Utilizan filtros convolucionales para extraer aquellas características que son relevantes, permitiendo de este modo capturar la estructura temporal de los datos y así mejorar la precisión de las predicciones 4.

Normalización divisiva generalizada, GDN La normalización divisiva generalizada (GDN), ha demostrado ser eficaz en la normalización de las características en el procesamiento de imágenes y señales, mejorando significativamente la calidad visual de las imágenes. Es especialmente útil en aplicaciones como la reducción de ruido y la mejora del contraste, lo cual se detalla en trabajos como el de Balle, Laparra y Simoncelli (2015) [13].
Se basa en la premisa de que las respuestas neuronales están sujetas a una normalización que depende del entorno local de las entradas, un fenómeno observado en diversos sistemas biológicos de percepción. Este método ajusta los coeficientes de la señal de entrada para normalizar la varianza de los datos de salida 4.

6.2.7 Evaluación de modelos

El entrenamiento de los algoritmos, implica entrenar el modelo usando únicamente el conjunto de datos de entrenamiento, mientras que por otro lado se valida dicho entrenamiento con los datos pertenecientes al conjunto de validación. Este proceso se repite 10 veces, donde en cada una de ellas, se utilizan distintas particiones aleatorias asegurando de este modo que cada modelo se evalúa en cada iteración bajo condiciones ligeramente diferentes, aumentando así la robustez de la misma. Una vez que se ha completado el proceso 'entrenamiento-validación' para cada modelo, se evalúa con su respectivo conjunto de datos de prueba, para determinar su rendimiento ante datos que no han sido visto antes por el algoritmo, es decir, no han sido incluidos ni durante el entrenamiento, ni para el proceso de validación. Esta evaluación proporciona una medida objetiva de cómo el modelo podría comportarse ante datos reales. Para ello, se ha aplicado la métrica del error absoluto medio (MAE), tal y como se explica a continuación. La evaluación de los modelos se realiza comparando las predicciones obtenidas para cada de datos, con los valores reales mediante la métrica del error absoluto medio (MAE), que se define matemáticamente como:

$$MAE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i|, \tag{6.1}$$

donde y es el valor real de la serie, y $\hat{y}$ es el valor de la predicción. Con ello, se calcula la media de las diferencias absolutas entre los valores verdaderos y los predichos. Esta métrica es altamente interpretable debido a su escala directamente comparable con los datos de entrada, lo cual facilita la evaluación del rendimiento y la comparativa de los distintos modelos [77].

6.3 Resultados obtenidos de los experimentos realizados

En esta sección se detallan los resultados obtenidos para todas las redes neuronales desarrolladas durante esta investigación, siendo estas: Modelo lineal, GRU, LSTM y CNN. Estos resultados están desglosados en gráficos boxplots, y en las distintas tablas del error absoluto medio para cada conjunto de datos.

6.3.1 Modelo Lineal & Lineal GDN

La siguiente sección proporciona un análisis de los boxplots donde se compara las funciones de pérdida para dos configuraciones del modelo lineal: una versión básica y otra que incluye una capa de Normalización Divisiva Generalizada (GDN). También se evalúan las métricas de Error Absoluto Medio de ambos modelos a través de los conjuntos de entrenamiento, validación y prueba.
Para el conjunto de entrenamiento, se observa en la tabla 6.5 que el modelo lineal básico presenta una pérdida media cercana a 0.93, con una desviación estándar relativamente baja. Por otro lado, el modelo que incluye una capa GDN muestra una pérdida media ligeramente mayor, próxima a 0.95, pero con una desviación estándar más alta tal y como se aprecia en la Figura 6.3. Esto indica que la pérdida del modelo Linear_GDN tiene más variabilidad durante el entrenamiento, lo que podría deberse a la complejidad adicional introducida por la capa GDN.
En lo que respecta a los datos de validación, las pérdidas medias para ambos modelos son muy similares tal y como se aprecia en la tabla 6.5, siendo estas en torno a 0.92. Sin embargo, el modelo Lineal_GDN muestra nuevamente una mayor variabilidad que se puede visualizar claramente en la Figura 6.3, en comparación con el modelo lineal básico, aunque el modelo lineal presenta una mayor cantidad de outliers. En cuanto al conjunto de prueba, ambos modelos muestran pérdidas medias cercanas a 0.97 tal y como se muestra en la tabla 6.5, si bien el modelo lineal_GDN presenta nuevamente una variabilidad mayor como así lo muestra la Figura 6.3.

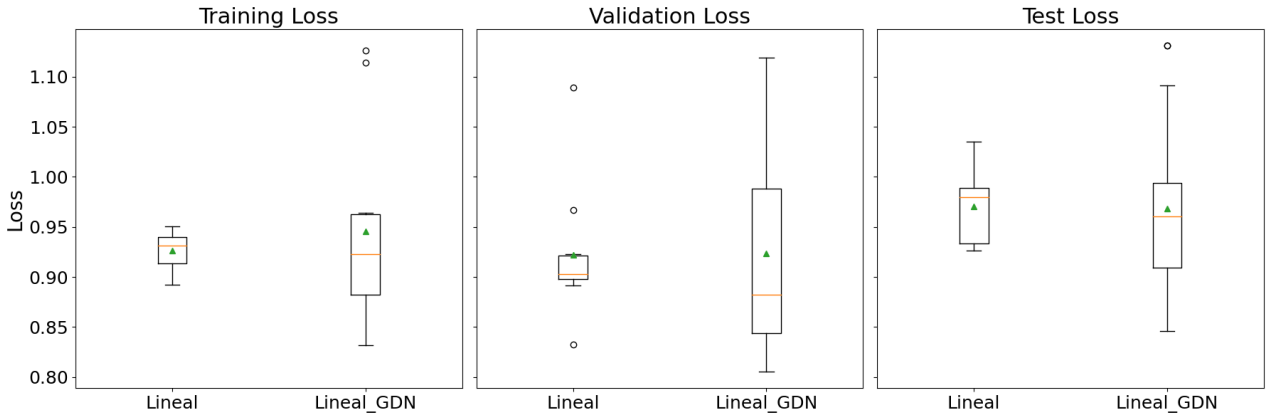


Figura 6.3: Comparación de pérdidas para modelos Lineal y Lineal_GDN en entrenamiento, validación y prueba

Métrica	Train MAE	Validation MAE	Test MAE
Resultados de Lineal			
Media	0.926	0.922	0.970
STD	0.018	0.064	0.034
Resultados de Lineal_GDN			
Media	0.945	0.923	0.968
STD	0.096	0.100	0.085

Tabla 6.5: Resultados del Error Absoluto Medio (MAE) para los modelos Lineal y Lineal_GDN

Desde un punto de vista estadístico, el modelo lineal básico parece ser más estable, mostrando menor variabilidad en la función de pérdidas de los conjuntos de entrenamiento, validación y prueba. Sin embargo, el modelo Lineal_GDN, a pesar de su mayor variabilidad, no presenta una mejora significativa en términos de pérdida media.

Curvas de entrenamiento del modelo Lineal & Lineal GDN

En lo referente al análisis de las curvas de entrenamiento para los modelos 'Lineal' y 'Lineal GDN' que se encuentran en el Anexo A, se observa que ambos modelos fueron entrenados cada uno durante 8500 épocas, con el objetivo de observar la evolución de la pérdida a lo largo del tiempo, así como evaluar la estabilidad y eficacia de la configuración de cada modelo, siendo al final modelos que convergen durante su entrenamiento.

6.3.2 Modelo GRU & GRU GDN

La siguiente sección presenta una comparativa de las funciones de pérdida por medio del gráfico boxplot para la red neuronal GRU, y otra enriquecida con una capa de Normalización Divisiva Generalizada (GDN). Además, se analiza el Error Medio Absoluto de ambos modelos, evaluándolos en los conjuntos de datos de entrenamiento, validación y prueba.
Para el conjunto de entrenamiento, se observa en la tabla 6.6 que el modelo GRU presenta una pérdida media cercana a 1.27, con una desviación estándar de 0.76. Por otro lado, el modelo que incluye una capa GDN muestra una pérdida media de 1.02, con una desviación estándar de 0.07. Esto indica que el modelo GRU_GDN no solo tiene una menor pérdida media, sino también una menor variabilidad durante el entrenamiento, tal y como se muestra en la Figura 6.4. Esto sugiere que los modelos GRU_GDN tienen una mayor estabilidad durante el proceso de aprendizaje.
Respecto a los datos de validación, las pérdidas medias para el modelo GRU son de aproximadamente 1.33, con una desviación estándar de 0.85 tal y como lo muestra la tabla 6.6. En comparación, el modelo GRU con una capa GDN muestra una pérdida media de 1.03 y una desviación estándar de 0.08. Esta reducción en la media y la variabilidad de la pérdida, tal y como se muestra en la Figura 6.4, sugiere que el modelo GRU con una capa GDN generaliza mejor a los datos no vistos durante el entrenamiento.
Por otra parte, en el conjunto de prueba, el modelo GRU muestra una pérdida media de 1.35, con una desviación estándar de 0.79 (ver tabla 6.6), mientras que el modelo GRU con una capa GDN presenta una pérdida media de 1.08 y una desviación estándar de 0.09. Nuevamente, el modelo GRU presenta una mayor dispersión y outliers tal y como se muestra en la Figura 6.4.

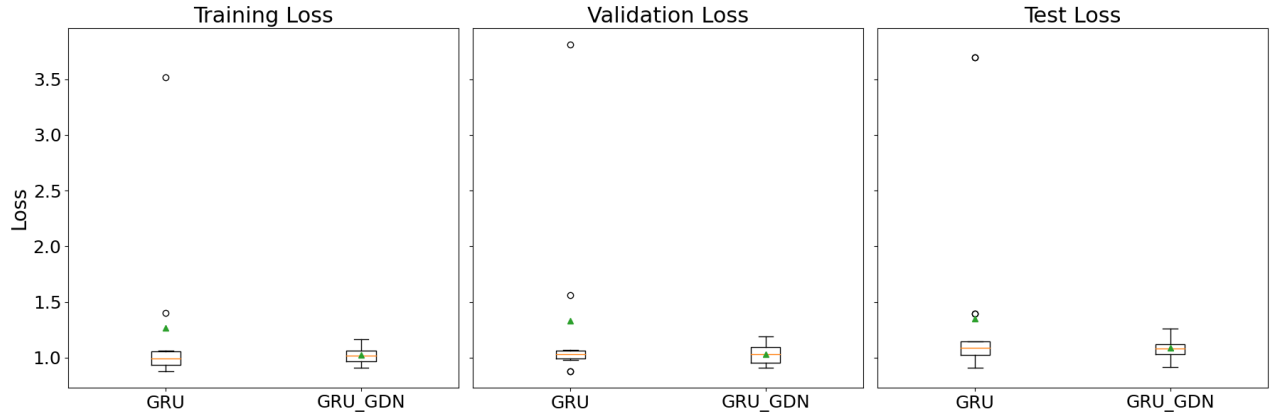


Figura 6.4: Comparación de pérdidas para modelos GRU y GRU_GDN en entrenamiento, validación y prueba

Métrica	Train MAE	Validation MAE	Test MAE
Resultados de GRU			
Media	1.267	1.330	1.352
STD	0.762	0.846	0.792
Resultados de GRU_GDN			
Media	1.020	1.028	1.084
STD	0.073	0.085	0.092

Tabla 6.6: Resultados del Error Absoluto Medio (MAE) para los modelos GRU y GRU con capa GDN

Desde un punto de vista estadístico, el modelo GRU con la capa GDN demuestra ser superior al modelo GRU básico en términos de pérdida media y estabilidad (variabilidad) en los conjuntos de entrenamiento, validación y prueba. La inclusión de la capa GDN parece mejorar la capacidad del modelo para aprender de los datos y generalizar a nuevos datos, tal y como se muestra en la tabla 6.6 con las menores pérdidas medias y desviaciones estándar en todas las fases de evaluación.

Curvas de entrenamiento del modelo GRU & GRU GDN

En el análisis de las curvas de entrenamiento para los modelos 'GRU' y 'GRU GDN' mostrados en el Anexo A, ambos modelos fueron entrenados durante 5500 épocas. Estos resultados muestran que ambos modelos logran adaptarse, converger y estabilizar su aprendizaje, adecuándose así a los patrones en los datos.

6.3.3 Modelo LSTM & LSTM GDN

Esta sección ofrece un análisis detallado de los boxplots que contrastan las funciones de pérdida de dos configuraciones de red diferentes: La primera es una red LSTM, y la segunda es dicha red pero modificada con una capa de Normalización Divisiva Generalizada (GDN). Adicionalmente, se discuten las métricas de Error Medio Absoluto para ambos modelos con los conjuntos de datos de entrenamiento, validación y prueba.

En la tabla 6.7, se observa que el modelo LSTM presenta una pérdida media de 1.267 en el conjunto de entrenamiento, con una desviación estándar de 0.107. En contraste, el modelo LSTM que incorpora una capa de Normalización Divisiva Generalizada (GDN) muestra una pérdida media más reducida, de 0.974 y con una desviación estándar de 0.047. Este modelo indica una consistencia mayor en las pérdidas durante el entrenamiento, tal y como se visualiza en la Figura 6.5.

Para los datos de validación, las pérdidas medias son 1.195 para el modelo LSTM básico y 0.957 para el modelo LSTM_GDN, según se refleja en la tabla 6.7. El modelo con la capa GDN tiene menos dispersión y variabilidad, lo cual es ve claramente en la Figura 6.5.
Por último, en el conjunto de prueba, tal y como se aprecia en la tabla 6.7,la pérdida media del modelo LSTM básico es de 1.223, mientras que para el modelo LSTM_GDN es de 0.988. La menor variabilidad del modelo LSTM_GDN, mostrada por una desviación estándar de 0.047 comparada con la del modelo básico de 0.110, sugiere una mejor generalización y robustez del modelo. Esto se puede apreciar en la Figura 6.5.

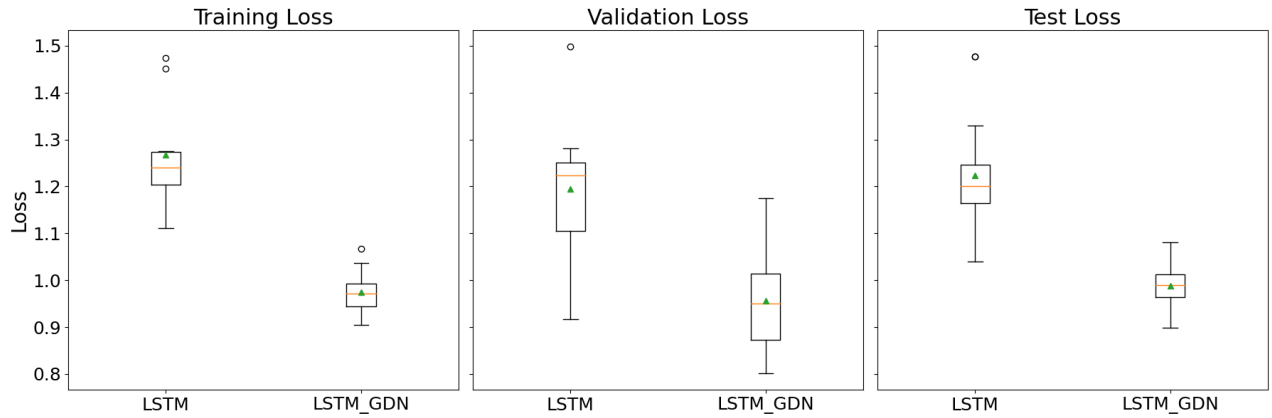


Figura 6.5: Comparación de pérdidas para modelos LSTM y LSTM_GDN en entrenamiento, validación y prueba

Métrica	Train MAE	Validation MAE	Test MAE
Resultados de LSTM			
Media	1.267	1.195	1.223
STD	0.107	0.143	0.110
Resultados de LSTM_GDN			
Media	0.974	0.957	0.988
STD	0.047	0.106	0.047

Tabla 6.7: Resultados del Error Absoluto Medio (MAE) para los modelos LSTM y LSTM con capa GDN

Desde un punto de vista estadístico, el modelo LSTM con la capa GDN demuestra ser superior al modelo LSTM básico en términos de pérdida media para los conjuntos de entrenamiento, validación y prueba. En este caso la inclusión de la capa GDN mejora significativamente la capacidad del modelo para aprender de los datos y generalizar a nuevos escenarios, como se refleja en la menor pérdida media y desviación estándar en todas las fases de evaluación de la Figura 6.5.

Curvas de entrenamiento del modelo LSTM & LSTM GDN

En esta sección se analizan las curvas de entrenamiento y validación para los modelos 'LSTM' y 'LSTM GDN', como se muestra en el Anexo A. Cada modelo ha sido entrenado durante 3500 épocas. Ambos modelos muestran una gran capacidad para adaptarse, aprender y converger de forma efectiva.

6.3.4 Modelo CNN & CNN GDN

A continuación, se realiza un análisis de los gráficos de boxplot, donde se comparan las funciones de pérdida entre dos configuraciones del modelo CNN: una primera versión estándar y otra que incorpora la Normalización Divisiva Generalizada (GDN). Posteriormente, se revisan las métricas del Error Medio Absoluto para ambas configuraciones a lo largo de las fases de entrenamiento, validación y prueba. Para los datos de entrenamiento, la tabla 6.8 muestra que el modelo CNN tiene una pérdida media de 1.711 con una desviación estándar de 0.458. El modelo CNN con la capa GDN, tiene una pérdida media de 1.750, aunque también tiene una desviación estándar algo superior, de 0.563 tal y como se visualiza en la Figura 6.6. En lo que respecta a los datos de validación, las pérdidas medias son relativamente más bajas para ambos modelos. El modelo CNN con GDN

muestra una leve mejora con una pérdida media de 1.044 en comparación con la del modelo básico que es de 1.081, tal y como se muestra en la tabla 6.8. Ambos modelos presentan la misma desviación estándar de 0.081, dando así lugar a una varianza igual de baja, lo cual es evidenciado en la Figura 6.6.

En cuanto al conjunto de datos de prueba, el modelo CNN registra una pérdida media de 1.112 tal y como muestra la tabla 6.8 mientras que el modelo con la capa GDN muestra una mejora marginal con una pérdida media de 1.095. La desviación estándar es ligeramente mayor en el modelo con GDN, siendo 0.077 comparada con 0.070 en el modelo básico. Bien es verdad que en ambos casos, la variabilidad es baja como bien se observa en la Figura 6.6.

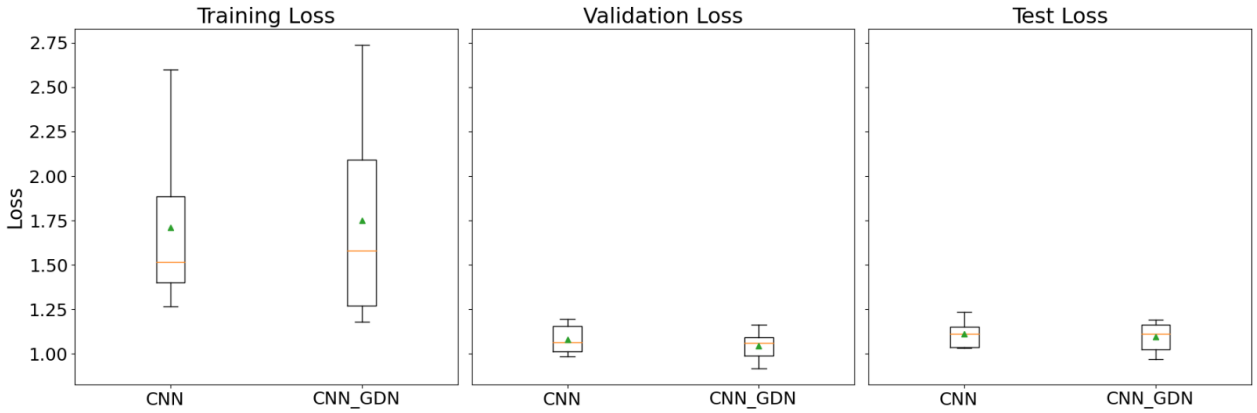


Figura 6.6: Comparación de pérdidas para modelos CNN y CNN_GDN en entrenamiento, validación y prueba

Métrica	Train MAE	Validation MAE	Test MAE
Resultados de CNN			
Media	1.711	1.081	1.112
STD	0.458	0.081	0.070
Resultados de CNN_GDN			
Media	1.750	1.044	1.095
STD	0.563	0.081	0.077

Tabla 6.8: Resultados del Error Absoluto Medio (MAE) para los modelos CNN y CNN con capa GDN

Aunque el modelo CNN con la capa GDN muestra una mayor variabilidad, su desempeño en términos de pérdida media es ligeramente mejor en las fases de validación y prueba, lo cual sugiere una ligera ventaja en la capacidad de generalización sobre el modelo básico.

Curvas de entrenamiento del modelo CNN & CNN GDN

Las curvas de pérdida para los modelos 'CNN' y 'CNN con capa de Normalización Divisiva Generalizada (GDN)', han sido entrenados durante 3500 épocas cada uno. Ambos modelos logran converger y estabilizar sus pérdidas, en un pequeño número de épocas.

6.4 Conclusiones

La tabla 6.9 presenta un resumen de los resultados obtenidos en términos del Error Absoluto Medio (MAE) acompañado de su desviación estándar (STD) para todos los modelos que se han utilizado durante este estudio, y para los tres conjuntos de datos: entrenamiento (Train), validación (Validation) y prueba (Test). El modelo Lineal muestra un MAE de 0.926 en el conjunto de entrenamiento, con una baja variabilidad de 0.018. En lo que respecta a validación y prueba, los MAE son ligeramente menores pero con una mayor variabilidad, especialmente en validación (0.922 ± 0.064) y menos en prueba (0.970 ± 0.034). En comparación con el modelo Lineal GDN, que tiene un MAE más alto en entrenamiento (0.945 ± 0.096), un valor similar en validación (0.923 ± 0.100) y algo inferior en el conjunto de prueba (0.968 ± 0.085), pero con una variabilidad considerablemente mayor en todas las fases. Para los modelos basados en redes recurrentes, el modelo GRU y su variante GRU_GDN muestran un comportamiento diferente. El modelo GRU registra un MAE significativamente alto de 1.267 en entrenamiento con una variabilidad extremadamente alta (0.762). Esta tendencia se mantiene para los conjuntos de datos de validación y prueba. El modelo GRU_GDN, en cambio, reduce notablemente el MAE en todas las fases y muestra una variabilidad considerablemente reducida, indicando una mayor estabilidad y rendimiento.
Similarmente, los modelos LSTM y LSTM_GDN muestran mejoras en MAE y STD al incorporar la capa GDN. El modelo LSTM con capa GDN tiene un MAE de 0.974 en entrenamiento, significativamente menor que el LSTM básico (1.267 ± 0.107), y muestra mejoras consistentes en validación y prueba.
Finalmente, los modelos CNN y CNN_GDN exhiben un comportamiento interesante donde, a pesar de un aumento en el MAE de entrenamiento en el modelo con GDN (de 1.711 a 1.750), se observa una mejora en los conjuntos de validación y prueba, reduciendo el MAE y manteniendo la variabilidad relativamente baja.

Modelo	Train	Validation	Test
Lineal	0.926 ± 0.018	0.922 ± 0.064	0.970 ± 0.034
Lineal GDN	0.945 ± 0.096	0.923 ± 0.100	0.968 ± 0.085
GRU	1.267 ± 0.762	1.330 ± 0.846	1.352 ± 0.792
GRU GDN	1.020 ± 0.073	1.028 ± 0.085	1.084 ± 0.092
LSTM	1.267 ± 0.107	1.195 ± 0.143	1.223 ± 0.110
LSTM GDN	0.974 ± 0.047	0.957 ± 0.106	0.988 ± 0.047
CNN	1.711 ± 0.458	1.081 ± 0.081	1.112 ± 0.070
CNN GDN	1.750 ± 0.563	1.044 ± 0.081	1.095 ± 0.077

Tabla 6.9: MAE para los distintos algoritmos

Cabe resaltar que los modelos que incorporan la capa de Normalización Divisiva Generalizada (GDN), generalmente muestran una reducción en la desviación típica en comparación con sus contrapartes sin GDN. Esto es especialmente notable en los modelos GRU y LSTM, donde la adición de GDN no solo mejora el MAE sino que también reduce considerablemente su variabilidad, lo que implica un aumento en la estabilidad y confiabilidad del modelo. El modelo GRU básico parece ser el peor, con el MAE más alto y una desviación típica muy alta en todas sus métricas. Esto indica no solo un rendimiento pobre en términos de error sino también una gran inconsistencia en los resultados. Llama especialmente la atención que la alta variabilidad que muestra el modelo GRU, sea aproximadamente diez veces menor cuando a dicho modelo se le añade una capa GDN. En contrapartida, destaca el modelo LSTM con GDN, con un rendimiento consistente y estable a través de todas las fases de evaluación (entrenamiento, validación y prueba), con el menor MAE y desviaciones típicas bajas.
El modelo que tiene un error medio absoluto menor y una variabilidad menor, es el modelo lineal simple. Esto lo convierte en el modelo más consistente, estable. Esto puede que sea debido a que la normalización de los datos haya dado lugar a una reducción de la complejidad de las series temporales. Esta complejidad puede deberse a diversos acontecimientos de las series temporales financieras como puede ser el reparto de dividendos, o la realización de split o contrasplit en sus acciones. Dichos eventos son capaces de provocar saltos o caídas abruptas en los precios. Esta normalización de los datos ha "suavizado" la relación de las series temporales

entre si y respecto a ellas mismas, eliminando a su vez variaciones no esenciales de los datos, siendo todo ello posiblemente el motivo por el que los modelos de redes neuronales más complejas resultan ser demasiado ambiciosos, aún cuando han tenido configuraciones sencillas pero no pudiendo evitar una posible tendencia al sobreajuste. Todo esto da lugar a que el modelo lineal sea lo suficientemente efectivo como para capturar la información relevante de las series de datos, sin caer en sobreajustes.
Una de los objetivos que se plantean como próximos pasos, de cara al futuro paper, fruto del actual trabajo de investigación realizado en este capítulo, será la realización de un contraste en cuanto a la calidad de los modelos respecto el uso de datos normalizados en comparación a usar datos sin normalizar. Se trata de valorar e identificar si este hecho tiene un papel protagonista a la hora de configurar redes más simples o más complejas. Además, se estudiarán otras arquitecturas y configuraciones de hiperparámetros con el fin de evaluar otros caminos que pudieran ser más óptimos respecto a los actuales.

Capítulo 7

Conclusiones

La ciencia de datos es un campo de investigación interdisciplinario que utiliza estadística, computación científica, métodos, procesos, algoritmos y sistemas científicos para obtener (recolectar o extraer), tratar, analizar y presentar informes a partir de datos ruidosos, estructurados y no estructurados. Actualmente, la inteligencia artificial y las innovaciones del machine learning han hecho que el procesamiento de datos sea más rápido y eficiente. La demanda del sector ha creado un ecosistema de cursos, grados académicos y puestos de trabajo en el campo de la ciencia de datos. Debido al conjunto de competencias multidisciplinarias y a la experiencia necesaria, la ciencia de datos promete un fuerte crecimiento también en las próximas décadas.

En esta tesis se ha hecho hincapié en el análisis de series temporales y, más en general, en la predicción de momentos de segundo orden en problemas de regresión. Las series temporales son conjuntos de datos cronológicamente ordenadas, que sirven para identificar tendencias, ciclos y patrones estacionales en áreas como economía, finanzas, meteorología y salud. Los avances tecnológicos en almacenamiento y procesamiento de datos han revolucionado la ciencia de datos y el análisis de series temporales, haciendo posible procesar grandes volúmenes de datos en tiempo real, así como responder de forma ágil a los cambios del mercado. Gracias a los grandes conjuntos de datos, al desarrollo de técnicas avanzadas como el machine learning, y a los modelos ARIMA y GARCH, este campo ha evolucionado hacia una mayor comprensión, convirtiéndose de este modo, en una herramienta capaz de transformar datos en información estratégica, ofreciendo a su vez una serie de ventajas competitivas en distintos entornos muy cambiantes y dinámicos.

El objetivo inicial de esta investigación era estudiar la viabilidad y eficacia de integrar las técnicas clásicas de estudio de análisis y modelización de series temporales, con nuevas tecnologías más modernas, basadas en machine learning (como las redes neuronales). Esta integración tiene como objetivo fortalecer el arsenal metodológico para analizar datos de series temporales, así como ampliar el alcance y la precisión de los modelos predictivos en estos campos. Al mismo tiempo, los métodos clásicos de predicción y análisis pueden mejorar la interpretabilidad de los métodos basados en machine learning. Entre los objetivos específicos de esta investigación, destaca la adaptación de la normalización divisiva generalizada al marco de series temporales. Es importante recordar que originalmente la normalización divisiva generalizada nació con el propósito de mejorar el procesamiento de imágenes, obteniendo además resultados muy prometedores [13].

La modalidad de trabajo llevada a cabo para la realización de esta investigación, ha estado compuesta de varias fases con el objetivo de abarcar todos los aspectos principales de este estudio. Cabe resaltar que el doctorando León Beleña, en todo el periodo de desarrollo de la tesis, ha trabajado siempre en el ámbito académico

como Profesor Asistente en la Universidad Francisco de Vitoria. Además, cabe resaltar el gran esfuerzo hecho por el estudiante para aprender nuevos enfoques, técnicas, y lenguajes de programación, así como la necesidad de cubrir las lagunas de conocimiento.

7.1 Conclusiones y contribuciones principales

Se ha logrado el objetivo principal de la tesis: se ha conseguido llevar el método llamado "normalización divisiva generalizada" (GDN), dentro de un contexto de series temporales a través de la metodología clásica de los kernel smoothers. Varias variantes y generalizaciones han sido diseñadas y probadas (considerando incluso varias extensiones, como los escenarios multi-outputs y otros enfoques). De una forma más especifica, en esta tesis se han explorado dos métodos para estimar la varianza local en problemas de regresión mediante la utilización de kernel smoothers. Se ha demostrado que ambos métodos (denominados MÉTODO-1, M1 y MÉTODO-2, M2), son muy efectivos en proporcionar estimaciones precisas y robustas de la tendencia y la varianza de los datos que son analizados, superando incluso a los modelos GARCH para ciertos tipos de datos. Estos métodos se aplican a series temporales y también a datos con entradas multidimensionales, como es el caso de los paisajes sonoros emocionales. Este hecho destaca su gran aplicabilidad así como su capacidad de adaptarse a diferentes estructuras y complejidades de los datos. Las contribuciones principales del estudio publicado son:

- Introducción de dos métodos robustos para estimar de la varianza local que van más allá de la aplicabilidad de la familia de los modelos GARCH.
- Las técnicas propuestas quedan demostradas como competitivas e incluso superiores en comparación con los modelos GARCH, a través de distintos experimentos que utilizan datos de distinta naturaleza.
- Los modelos propuestos destacan por su flexibilidad, capacidad de adaptación y generalización, tanto para datos de múltiples salidas como en el caso de las distintas series temporales que han sido creadas para este estudio, con distinta estructura de datos y distinta variabilidad.
- Validación exitosa en escenarios reales, concretamente en el análisis emocional de paisajes sonoros. Esto demuestra la efectividad y relevancia práctica de las metodologías desarrolladas.

Con esto se han conseguido los siguientes hitos:

- Publicación del artículo en una revista internacional JCR Q1:

  Beleña, L., Curbelo, E., Martino, L., and Laparra, V., "Second-Moment/Order Approximations by Kernel Smoothers with Application to Volatility Estimation". Mathematics, 12(9), 1406, 2024.

- Participación como ponente en un congreso internacional, de los resultados presentes en la publicación anterior. Machine Learning, Information Theory, Signal Processing and Communications Workshop, 2024, Universidad Rey Juan Carlos, Madrid.

7.2 Otros hallazgos

También se han producidos varios estudios y análisis relativos al uso de las redes neuronales para el análisis de series temporales financieras. Podemos afirmar que se han conseguido una serie de hitos, que esperamos puedan dar pié a futuras publicaciones. Se muestran estos hitos a continuación:

- La inclusión de la capa GDN a las distintas arquitecturas de redes neuronales, contribuye a reducir de forma general la variabilidad del modelo. Esto contribuye a una generalización más sólida y a una mejor capacidad predictiva.
- Entre los modelos evaluados, el modelo GRU básico mostró el peor rendimiento, con el MAE más alto y con una alta variabilidad en todas sus fases de evaluación. Este resultado remarca la necesidad de incorporar a la red ciertas modificaciones como la capa GDN para reducir la inestabilidad de este tipo de arquitecturas.
- El modelo lineal simple mostró un rendimiento comparativamente alto, siendo el modelo más estable y con menor MAE entre todos los modelos evaluados. Este fenómeno puede atribuirse a la capacidad de los modelos lineales para capturar la dinámica esencial de los datos sin incurrir en sobreajustes, especialmente en contextos donde la complejidad de las series temporales se reduce mediante la normalización previa de los datos.
- Por otro lado, el modelo LSTM con GDN destacó como la segunda arquitectura de red más consistente y estable, ofreciendo un MAE muy reducido y unas desviaciones típicas muy controladas en todas las fases de evaluación, por detrás del modelo lineal simple.

El autor de la tesis ha también conseguido (durante el periodo de desarrollo la tesis):

- Publicación del artículo en una revista internacional JCR Q1:

  Sandubete, J. E., Beleña, L., and García-Villalobos, J. C., "Testing the efficient market hypothesis and the model-data paradox of chaos on top currencies from the foreign exchange market (FOREX)". Mathematics, 11(2), 286, 2023.
- Presentación de dicho artículo en congreso: León Beleña, P., Hidalgo, P., Lazcano, A., and Sandubete, J. E. (24, Noviembre 2023 - Universidad de la Rioja, Logroño). Análisis de la Hipótesis de Mercados Eficientes, la Triple Hora Bruja y el Efecto del Día de Vencimiento, en las Principales Monedas del Mercado de Divisas (FOREX). VI Congreso Iberoamericano de Jóvenes Investigadores en Ciencias Económicas y Dirección de Empresas.
- Ponencia en congreso: "Hidalgo, P., Lazcano, A., and Beleña, L. (16 Diciembre 2021 - Universidad Pontificia Comillas ICADE, Madrid). Consecuencias en el Índice de Confianza Económico del análisis del sentimiento de tweets asociados a la pandemia. IV Congreso Iberoamericano de Jóvenes Investigadores en Economía y Empresa".

7.3 Posibles líneas de investigación futuras

El presente trabajo sugiere diversas direcciones para futuras publicaciones y futura investigación, entre las que se incluyen:

- Investigar el desarrollo de funciones kernel que ajusten de forma dinámica sus parámetros, de forma que respondan a características locales de los datos para mejorar la precisión de las estimaciones de tendencias y varianzas. Esto puede recordar los conocidos "Warped Gaussian Processes" o un deep Gaussian Process en dos etapas.
- Profundizar en la combinación de los modelos clásicos de predicción y la técnicas avanzadas de machine learning.

- Profundizar en la aplicación de estos modelos propuestos (previa adaptación) en campos como la economía y las finanzas, donde los modelos GARCH tienen un gran protagonismo debido a la importancia de una correcta modelización de la volatilidad en dichas series temporales.
- Estudio más detallado del análisis teórico de las propiedades estadísticas de los estimadores que han sido propuestos, haciendo especial hincapié en los conceptos de sesgo, varianza y consistencia.

Bibliografía

[1] A. F. Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] H. Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723, 1974.

[3] F. Aletta and J. Xiao. What are the current priorities and challenges for (urban) soundscape research? *Challenges*, 9(1):16, 2018.

[4] N. S. Altman. Kernel smoothing of data with correlated errors. *Journal of the American Statistical Association*, 85(411):749–759, 1990.

[5] Z. An and Z. Feng. A stock price forecasting method using autoregressive integrated moving average model and gated recurrent unit network. In *2021 International conference on big data analysis and computer science (BDACS)*, pages 31–34. IEEE, 2021.

[6] T. G. Andersen and T. Bollerslev. Answering the skeptics: Yes, standard volatility models do provide accurate forecasts. *International economic review*, pages 885–905, 1998.

[7] D. Anderson and K. Burnham. Model selection and multi-model inference. *Second. NY: Springer-Verlag*, 63(2020):10, 2004.

[8] W. Anggraeni, R. A. Vinarti, and Y. D. Kurniawati. Performance comparisons between arima and arimax method in moslem kids clothes demand forecasting: Case study. *Procedia Computer Science*, 72:630–637, 2015.

[9] M. Asai and M. McAleer. Dynamic asymmetric leverage in stochastic volatility models. *Econometric Reviews*, 24(3):317–332, 2005.

[10] M. Asai, M. McAleer, and J. Yu. Multivariate stochastic volatility: a review. *Econometric Reviews*, 25(2-3):145–175, 2006.

[11] O. Axelsson, M. E. Nilsson, and B. Berglund. A principal components model of soundscape perception. *The Journal of the Acoustical Society of America*, 128(5):2836–2846, 2010.

[12] A. M. Bagde et al. Predicting stock market time-series data using cnn-lstm neural network model. *arXiv preprint arXiv:2305.14378*, 2023.

[13] J. Ballé, V. Laparra, and E. P. Simoncelli. Density modeling of images using a generalized normalization transformation. *arXiv preprint arXiv:1511.06281*, 2015.

[14] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE, 2016.

[15] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016.

[16] O. E. Barndorff-Nielsen and N. Shephard. Variation, jumps and high frequency data in financial econometrics. *Advanced in Economics and Econometrics. Theory and Applications*, 2006.

[17] D. S. Bates. Jumps and stochastic volatility: Exchange rate processes implicit in deutsche mark options. *The Review of Financial Studies*, 9(1):69–107, 1996.

[18] D. G. Baur and T. Dimpfl. A quantile regression approach to estimate the variance of financial returns. *Journal of Financial Econometrics*, 17(4):616–644, 2019.

[19] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[20] J. L. Bentley, D. F. Stanat, and E. Hollins Williams. The complexity of finding fixed-radius near neighbors. *Information Processing Letters*, 6(6):209–212, 1977.

[21] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[22] F. Black. Studies of stock market volatility changes. *Proceedings of the American Statistical Association, Business & Economic Statistics Section, 1976*, 1976.

[23] T. Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of econometrics*, 31(3):307–327, 1986.

[24] T. Bollerslev. A conditionally heteroskedastic time series model for speculative prices and rates of return. *The review of economics and statistics*, pages 542–547, 1987.

[25] A. Borovykh, S. Bohte, and C. W. Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[26] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[27] A. W. Bowman. An alternative method of cross-validation for the smoothing of density estimates. *Biometrika*, 71(2):353–360, 1984.

[28] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[29] G. E. P. Box and D. A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[30] P. J. Brockwell and R. A. Davis. *Introduction to time series and forecasting*. Springer, 2002.

[31] K. P. Burnham and D. R. Anderson. Multimodel inference: understanding aic and bic in model selection. *Sociological methods & research*, 33(2):261–304, 2004.

[32] PABLO HERNÁNDEZ CÁMARA, VALERO LAPARRA PÉREZ-MUELAS, and JESÚS MALO LÓPEZ. Autonomous driving. 2022.

[33] J. Cao and J. Wang. Stock price forecasting model based on modified convolution neural network and financial time series analysis. *International Journal of Communication Systems*, 32(12):e3987, 2019.

[34] M. Carandini and D. J. Heeger. Normalization as a canonical neural computation. *Nature Reviews Neuroscience*, 13(1):51–62, 2012.

[35] M. A. Carnero, D. Peña, and E. Ruiz. Persistence and kurtosis in garch and stochastic volatility models. *Journal of financial econometrics*, 2(2):319–342, 2004.

[36] C. Chang, M. McAleer, and R. Tansuchat. Modelling long memory volatility in agricultural commodity futures returns. *Annals of Financial Economics*, 7(02):1250010, 2012.

[37] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[38] C. Chen. Stock price prediction based on the fusion of cnn-gru combined neural network and attention mechanism. In *2023 6th International Conference on Electronics Technology (ICET)*, pages 1166–1170. IEEE, 2023.

[39] CH. Chen. Neural networks for financial market prediction. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 2, pages 1199–1202. IEEE, 1994.

[40] H. Cheng. Second order model with composite quantile regression. *Journal of Physics: Conference Series*, 2437(1):012070, jan 2023.

[41] S. Chib and E. Greenberg. On conditional variance estimation in nonparametric regression. *Statistics and Computing*, 23(2):261–270, Mar 2013.

[42] B. Chidester, M. N. Do, and J. Ma. Rotation equivariance and invariance in convolutional neural networks. *arXiv preprint arXiv:1805.12301*, 2018.

[43] A. A. Christie. The stochastic behavior of common stock variances: Value, leverage and interest rate effects. *Journal of financial Economics*, 10(4):407–432, 1982.

[44] I. C. Chronopoulos, A. Raftapostolos, and G. Kapetanios. Forecasting value-at-risk using deep neural network quantile regression. *Journal of Financial Econometrics*, 2023.

[45] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[46] R. Coen-Cagli and O. Schwartz. The impact on midlevel vision of statistically optimal divisive normalization in v1. *Journal of vision*, 13(8):13–13, 2013.

[47] R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative finance*, 1(2):223, 2001.

[48] E. Curbelo, L. Martino, F. Llorente, and D. Delgado-Gomez. Adaptive posterior distributions for uncertainty analysis of covariance matrices in bayesian inversion problems for multioutput signals. *preprint viXra:2310.0032*, pages 1–28, 2023.

[49] S. Das, C. D. Hollander, and S. Suliman. Automating visual inspection with convolutional neural networks. In *Annual Conference of the PHM Society*, volume 11, 2019.

[50] El Laberinto de Falken. Visión artificial: Redes convolucionales (cnn). El Laberinto de Falken, Octubre 2019.

[51] L. Dedi and B. Yavas. Return and volatility spillovers in equity markets: An investigation using various garch methodologies. *Cogent Economics & Finance*, 4(1):1266788, 2016.

[52] E. Derman, I. Kani, and J. Z. Zou. The local volatility surface: Unlocking the information in index option prices. *Financial Analysts Journal*, 52(4):25–36, 1996.

[53] D. A. Dickey and W. A. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American statistical association*, 74(366a):427–431, 1979.

[54] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[55] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[56] M. P. Eckstein, A. J. Ahumada, and A. B. Watson. Visual signal detection in structured backgrounds. ii. effects of contrast gain control, background variations, and white noise. *JOSA A*, 14(9):2406–2419, 1997.

[57] R. Engle. Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of Business & Economic Statistics*, 20(3):339–350, 2002.

[58] R. Engle. Risk and volatility: Econometric models and financial practice. *American Economic Review*, 94(3):405–420, 2004.

[59] R. F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica: Journal of the econometric society*, pages 987–1007, 1982.

[60] R. F. Engle, D. M. Lilien, and R. P. Robins. Estimating Time Varying Risk Premia in the Term Structure: The ARCH-M Model. *Econometrica*, 55(2):391–407, 1987.

[61] J. Fan, M. Thorogood, and P. Pasquier. Emo-soundscapes: A dataset for soundscape emotion recognition. In *2017 Seventh international conference on affective computing and intelligent interaction (ACII)*, pages 196–201. IEEE, 2017.

[62] JIANQING FAN and QIWEI YAO. Efficient estimation of conditional variance functions in stochastic regression. *Biometrika*, 85(3):645–660, 09 1998.

[63] E. Fonseca, J. Pons Puig, X. Favory, F. Font Corbera, Dm. Bogdanov, A. Ferraro, S. Oramas, A. Porter, and X. Serra. Freesound datasets: a platform for the creation of open audio datasets. In *Hu X, Cunningham SJ, Turnbull D, Duan Z, editors. Proceedings of the 18th ISMIR Conference; 2017 oct 23-27; Suzhou, China.[Canada]: International Society for Music Information Retrieval; 2017. p. 486-93.* International Society for Music Information Retrieval (ISMIR), 2017.

[64] G. M Fung, O. L Mangasarian, and A. J. Smola. Minimal kernel classifiers. *Journal of Machine Learning Research*, 3(Nov):303–321, 2002.

[65] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[66] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[67] P. Hall and R. J. Carroll. Variance function estimation in regression: the effect of estimating the mean. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 51(1):3–14, 1989.

[68] P. R. Hansen and A. Lunde. A forecast comparison of volatility models: does anything beat a garch (1, 1)? *Journal of Applied Econometrics*, 20(7):873–889, 2005.

[69] P. R. Hansen and A. Lunde. Realized variance and market microstructure noise. *Journal of Business & Economic Statistics*, 24(2):127–161, 2006.

[70] T. Hastie, R. Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[71] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[72] D. J. Heeger. Normalization of cell responses in cat striate cortex. *Visual neuroscience*, 9(2):181–197, 1992.

[73] P. Hernández-Cámara, J. Vila-Tomás, V. Laparra, and J. Malo. Neural networks with divisive normalization for image segmentation. *Pattern Recognition Letters*, 173:64–71, 2023.

[74] S. L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *The review of financial studies*, 6(2):327–343, 1993.

[75] G. Hinton. Neural networks for machine learning. Coursera lecture slides, 2012.

[76] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[77] T. O. Hodson. Root-mean-square error (rmse) or mean absolute error (mae): When to use them or not. *Geoscientific Model Development*, 15(14):5481–5487, 2022.

[78] C. C. Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International journal of forecasting*, 20(1):5–10, 2004.

[79] Alex YiHou Huang, Sheng-Pen Peng, Fangjhy Li, and Ching-Jie Ke. Volatility forecasting of exchange rate by quantile regression. *International Review of Economics and Finance*, 20(4):591–606, 2011.

[80] Q. Huang, H. Zhang, J. Chen, and M. He. Quantile regression models and their applications: a review. *Journal of Biometrics & Biostatistics*, 8(3):1–6, 2017.

[81] J. C. Hull and S. Basu. *Options, futures, and other derivatives*. Pearson Education India, 2016.

[82] B. Ibrahim, A. Elamer, T. Alasker, M. Mohamed, and H. Abdou. Volatility contagion between cryptocurrencies, gold and stock markets pre-and-during covid-19: evidence using dcc-garch and cascade-correlation network. *Financial Innovation*, 10(1):104, 2024.

[83] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[84] K. Jarrett, K. Kavukcuoglu, Marc'Aurelio R., and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision*, pages 2146–2153. IEEE, 2009.

[85] F. Kamalov. Forecasting significant stock price changes using neural networks. *Neural Computing and Applications*, 32(23):17655–17667, 2020.

[86] B. Karlik and A. V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[87] S. Kim and M. Kang. Financial series prediction using attention lstm. *arXiv preprint arXiv:1902.10877*, 2019.

[88] T. Kim and H. Y. Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. *PloS one*, 14(2):e0212320, 2019.

[89] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[90] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

[91] R. Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[92] V. Laparra, J Ballé., A Berardino, and E. Simoncelli. Perceptual image quality assessment using a normalized laplacian pyramid. In *Proc. Human Vis. Elect. Im.*, volume 2016, pages 1–6, 2016.

[93] V. Laparra, A. Berardino, J. Ballé, and E. P. Simoncelli. Perceptually optimized image rendering. *JOSA A*, 34(9):1511–1525, 2017.

[94] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[95] Y. LeCun, L. Bottou, G. B. Orr, and K-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[96] H. Li, K. Fu, M. Yan, X. Sun, H. Sun, and W. Diao. Vehicle detection in remote sensing images using denoizing-based convolutional neural networks. *Remote Sensing Letters*, 8(3):262–270, 2017.

[97] M. Lionello, F. Aletta, and J. Kang. A systematic review of prediction models for the experience of urban soundscapes. *Applied Acoustics*, 170:107479, 2020.

[98] G. Liu, K. J Shih, T.-C. Wang, F. A. Reda, K. Sapra, Z. Yu, A. Tao, and B. Catanzaro. Partial convolution based padding. *arXiv preprint arXiv:1811.11718*, 2018.

[99] P. Lundén and M. Hurtig. On urban soundscape mapping: A computer can predict the outcome of soundscape assessments. In *INTER-NOISE and NOISE-CON Congress and Conference Proceedings*, volume 253, pages 2017–2024. Institute of Noise Control Engineering, 2016.

[100] H. Malmsten. Evaluating exponential garch models. Technical report, SSE/EFI Working paper Series in Economics and Finance, 2004.

[101] J. Malo and V. Laparra. Psychophysically tuned divisive normalization approximately factorizes the pdf of natural images. *Neural computation*, 22(12):3179–3206, 2010.

[102] B. B. Mandelbrot. *The variation of certain speculative prices*. Springer, 1997.

[103] M. Martinez-Garcia, P. Cyriac, T. Batard, M. Bertalmío, and J. Malo. Derivatives and inverse of cascaded linear+ nonlinear neural models. *PloS one*, 13(10):e0201326, 2018.

[104] L. Martino, F. Llorente, E. Curbelo, J. López-Santiago, and J. Míguez. Automatic tempered posterior distributions for bayesian inversion problems. *Mathematics*, 9(7), 2021.

[105] L. Martino and J. Read. A joint introduction to gaussian processes and relevance vector machines with connections to kalman filtering and other kernel smoothers. *Information Fusion*, 74:17–38, 2021.

[106] L. Martino and J. Read. Joint introduction to Gaussian Processes and Relevance Vector Machines with connections to Kalman filtering and other kernel smoothers. *Information Fusion*, 74:17–38, 2021.

[107] L. Martino, R. San Millan-Castillo, and E. Morgado. Spectral information criterion for automatic elbow detection. *Expert Systems with Applications*, 231:120705, 2023.

[108] L. Martino, H. Yang, D. Luengo, J. Kanniainen, and J. Corander. A fast universal self-tuned sampler within Gibbs sampling. *Digital Signal Processing*, 47:68 – 83, 2015.

[109] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[110] R. San Millan-Castillo, L. Martino, E. Morgado, and F. Llorente. An exhaustive variable selection study for linear models of soundscape emotions: Rankings and gibbs analysis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:2460–2474, 2022.

[111] A. Moghar and M. Hamiche. Stock market prediction using lstm recurrent neural network. *Procedia Computer Science*, 170:1168–1173, 2020.

[112] E. Morgado, L. Martino, and R. San Millan-Castillo. Universal and automatic elbow detection for learning the effective number of components in model selection problems. *Digital Signal Processing*, 140:104103, 2023.

[113] JP Morgan et al. Creditmetrics-technical document. *JP Morgan, New York*, 1:102–127, 1997.

[114] E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[115] K. Nakamura and B-W Hong. Adaptive weight decay for deep neural networks. *IEEE Access*, 7:118857–118865, 2019.

[116] I. Namatēvs. Deep convolutional neural networks: Structure, feature extraction and training. *Information Technology and Management Science*, 20(1):40–47, 2017.

[117] D. B. Nelson. Conditional Heteroskedasticity in Asset Returns: A New Approach. *Econometrica: Journal of the Econometric Society*, pages 347–370, 1991.

[118] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[119] W. K. Newey. Kernel estimation of partial means and a general variance estimator. *Econometric Theory*, 10(2):1–21, 1994.

[120] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

[121] C. Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2023-07-28.

[122] A. Pallini. Kernel methods for estimating covariance functions from curves. In *Classification and Data Analysis*, pages 319–326, 1999.

[123] E. Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[124] D. Peña. *Análisis de series temporales*. El Libro Universitario - Manuales. Alianza Editorial, 2010.

[125] R. J. Prazenica and A. J. Kurdila. Volterra kernel identification using triangular wavelets. *Journal of Vibration and Control*, 10(4):597–622, 2004.

[126] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 2002.

[127] F.r Provost and T. Fawcett. *Data Science for Business: What you need to know about data mining and data-analytic thinking*. OŔeilly Media, Inc., 2013.

[128] R. Putra, S. W. Tyas, and M. G. Fadhlurrahman. Geographically weighted regression with the best kernel function on open unemployment rate data in east java province. *Enthusiastic: International Journal of Applied Statistics and Data Science*, pages 26–36, 2022.

[129] H. Qiao, X. Xi, Y. Li, W. Wu, and F. Li. Biologically inspired visual model with preliminary cognition and active attention adjustment. *IEEE transactions on cybernetics*, 45(11):2612–2624, 2014.

[130] N. C. Rabinowitz, B. DB. Willmore, J. WH. Schnupp, and A. J. King. Contrast gain control in auditory cortex. *Neuron*, 70(6):1178–1191, 2011.

[131] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006.

[132] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[133] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[134] D. E. Rumelhart, G. E. Hinton, and R. J. W. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[135] D. Ruppert, M. P. Wand, U. Holst, and O. Hössjer. Local polynomial variance-function estimation. *Technometrics*, 39(3):262–273, 1997.

[136] G. Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[137] S. Selvin, R. Vinayakumar, E.A. Gopalakrishnan, V. K. Menon, and K.P. Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 international conference on advances in computing, communications and informatics (icacci)*, pages 1643–1647. IEEE, 2017.

[138] S.J. Sheather and M.C. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society: Series B (Methodological)*, 53(3):683–690, 1991.

[139] G. Shen, Q. Tan, H. Zhang, P. Zeng, and J. Xu. Deep learning with gated recurrent unit networks for financial sequence predictions. *Procedia computer science*, 131:895–903, 2018.

[140] B. W. Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[141] H. Song and H. Choi. Forecasting stock market indices using the recurrent neural network based hybrid models: Cnn-lstm, gru-cnn, and ensemble models. *Applied Sciences*, 13(7):4644, 2023.

[142] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[143] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

[144] Z. Tang, C. De Almeida, and P. A. Fishwick. Time series forecasting using neural networks vs. box-jenkins methodology. *Simulation*, 57(5):303–310, 1991.

[145] S. J. Taylor. Modeling stochastic volatility: A review and comparative study. *Mathematical finance*, 4(2):183–204, 1994.

[146] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[147] J. R. Trapero, M. Cardos, and N. Kourentzes. Empirical safety stock estimation based on kernel and garch models. *Omega*, 84:199–211, 2019.

[148] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th conference on business informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017.

[149] S. I. Vagropoulos, G. I. Chouliaras, E. G. Kardakos, C. K. Simoglou, and A. G. Bakirtzis. Comparison of sarimax, sarima, modified sarima and ann-based models for short-term pv generation forecasting. In *2016 IEEE international energy conference (ENERGYCON)*, pages 1–6. IEEE, 2016.

[150] M. García Villanueva and L. R. Muñoz. Diseño de una arquitectura de red neuronal convolucional para la clasificación de objetos. *Ciencia Nicolaita*, (81):46–61, 2020.

[151] H. Wang, X. Li, D. Bi, X. Xie, and Y. Xie. A robust student's t-based kernel adaptive filter. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(10):3371–3375, 2021.

[152] Z. Wang and A. C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, 26(1):98–117, 2009.

[153] G. S. Watson. Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372, 1964.

[154] M. West. Time series decomposition. *Biometrika*, 84(2):489–494, 1997.

[155] S. S. Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. *The annals of mathematical statistics*, 9(1):60–62, 1938.

[156] B. M. Williams. Multivariate vehicular traffic flow prediction: evaluation of arimax modeling. *Transportation Research Record*, 1776(1):194–200, 2001.

[157] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[158] A. Yadav, C.K. Jha, and A. Sharan. Optimizing lstm for time series prediction in indian stock market. *Procedia Computer Science*, 167:2091–2100, 2020.

[159] J. Yepez and S.-B. Ko. Stride 2 1-d, 2-d, and 3-d winograd for convolutional neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(4):853–863, 2020.

[160] K. Yu and M. C Jones. Likelihood-based local linear estimation of the conditional variance function. *Journal of the American Statistical Association*, 99(465):139–144, 2004.

[161] P. Yu and X. Yan. Stock price prediction based on deep neural networks. *Neural Computing and Applications*, 32(6):1609–1628, 2020.

[162] G. U. Yule. Vii. on a method of investigating periodicities disturbed series, with special reference to wolfer's sunspot numbers. *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, 226(636-646):267–298, 1927.

[163] A. Zafar, M. Aamir, N. Mohd Nawi, A. Arshad, S. Riaz, A. Alruban, A. K. Dutta, and S. Almotairi. A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 12(17):8643, 2022.

[164] P. W. Zaki, A. M. Hashem, E. A. Fahim, M. A. Masnour, S. M. ElGenk, M. Mashaly, and S. M. Ismail. A novel sigmoid function approximation suitable for neural networks on fpga. In *2019 15th International Computer Engineering Conference (ICENCO)*, pages 95–99. IEEE, 2019.

[165] J-M Zakoian. Threshold Heteroskedastic Models. *Journal of Economic Dynamics and Control*, 18(5):931–955, 1994.

[166] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[167] H. Zhou and Q. Sun. Research on principle and application of convolutional neural networks. In *IOP Conference Series: Earth and Environmental Science*, volume 440, page 042055. IOP Publishing, 2020.

[168] Y. Zu and H. P. Boswijk. Estimating spot volatility with high-frequency financial data. *Journal of Econometrics*, 181(2):117–135, 2014.

[169] M. Zulqarnain, R. Ghazali, M. G. Ghouse, Y. M. M. Hassim, and I. Javid. Predicting financial prices of stock market using recurrent convolutional neural networks. *International Journal of Intelligent Systems and Applications (IJISA)*, 12(6):21–32, 2020.

Anexo A

Matriz de correlación de todos los activos seleccionados

A continuación se presenta un análisis detallado de la matriz de correlación para los precios de cierre ajustados de las distintas empresas que se han tenido en cuenta en el estudio de la predicción de Apple. La matriz de correlación muestra que los valores cercanos a 1 pertenecen a movimientos de precios altamente sincronizados entre dichos pares de acciones. Por ejemplo, la correlación entre las acciones de Apple (AAPL) y Microsoft (MSFT) es de 0.98, lo que indica que estos valores tienden a moverse juntos de manera muy consistente, probablemente debido a factores de mercado compartidos o similares influencias sectoriales.

Los valores negativos indican movimientos de precios en direcciones opuestas. Este es el caso, de General Electric (GE) respecto a Google (GOOGL) con una correlación de -0.47, posiblemente debido a las diferencias fundamentales sectoriales. Por otro lado, los valores de correlación muy próximos a cero representan una falta de relación lineal entre ambos precios. Por ejemplo, Exxon Mobil (XOM) muestra correlaciones bajas con varias empresas tecnológicas, como Apple, Google o AMD, subrayando la independencia de sus movimientos de precios respecto a las fluctuaciones en el sector tecnológico.

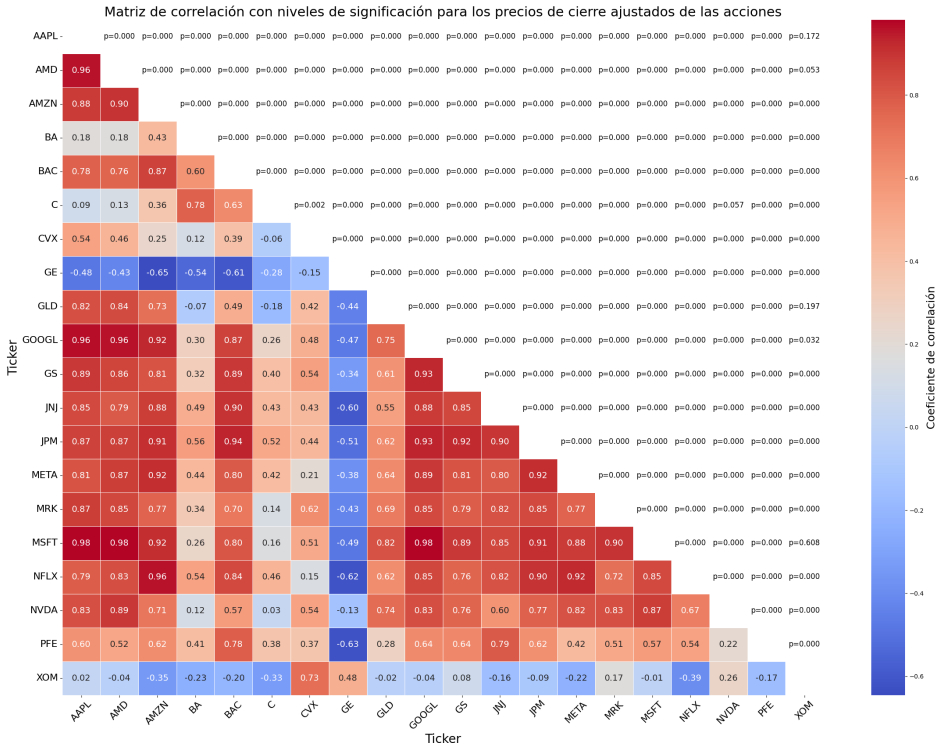


Figura 1: Matriz de correlación con todos los activos tenidos en cuenta en el análisis. Estos son: Apple Inc. (AAPL), Meta Platforms Inc. (anteriormente Facebook) (META), Microsoft Corporation (MSFT), Alphabet Inc. (Google) (GOOGL), Exxon Mobil Corporation (XOM), Chevron Corporation (CVX), JPMorgan Chase & Co. (JPM), Goldman Sachs Group Inc. (GS), Pfizer Inc. (PFE), Johnson & Johnson (JNJ), Advanced Micro Devices Inc. (AMD), NVIDIA Corporation (NVDA), Bank of America Corp (BAC), Citigroup Inc. (C), Amazon.com Inc. (AMZN), Netflix Inc. (NFLX), Merck & Co. Inc. (MRK), Boeing Company (BA), General Electric Company (GE) y SPDR Gold Trust (GLD))

Modelo Lineal y Lineal GDN

La Figura 2 muestra una red neuronal lineal simple, iniciando con la entrada de datos en la *capa de entrada*, seguida por una *capa flatten* que transforma los datos multidimensionales en un formato unidimensional. A continuación, una *capa densa* realiza transformaciones lineales y no lineales sobre los datos aplanados, finalizando en una *capa de salida* con los resultados del modelo.

La Figura 3 muestra la inclusión de una *capa de Normalización Divisiva Generalizada (GDN)* inmediatamente después de la *capa de entrada*. Esta capa ajusta las activaciones neuronales, normalizando las entradas de forma adaptativa. Una vez normalizados los datos, pasan a través de una *capa flatten* seguida por una *capa densa*, finalizando en la *capa de salida*.

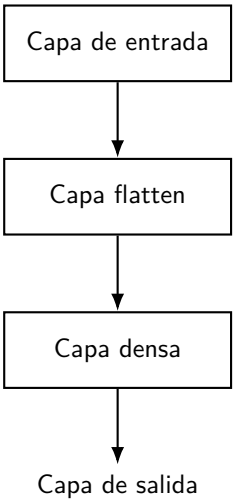


Figura 2: Arquitectura del modelo Lineal

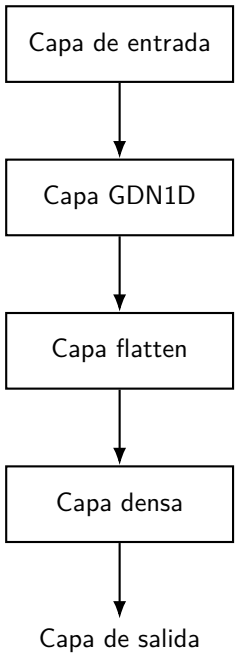


Figura 3: Arquitectura del modelo Lineal, incluyendo una capa unidimensional GDN

Modelo GRU y GRU GDN

La Figura 4 inicia con una *capa de entrada* seguida de una *capa GRU*, que procesa series temporales. Una *capa LeakyReLU* introduce no linealidad, seguida por una *capa densa*. Antes de la *capa de salida*, una *capa Dropout* ayuda a reducir el posible sobreajuste.

En la Figura 5, se introduce una *capa de Normalización Divisiva Generalizada (GDN)* después de la *capa de entrada*, y antes de la *capa GRU*. La arquitectura se completa con las *capas LeakyReLU*, *densa* y *Dropout*, finalizando en la *capa de salida*.

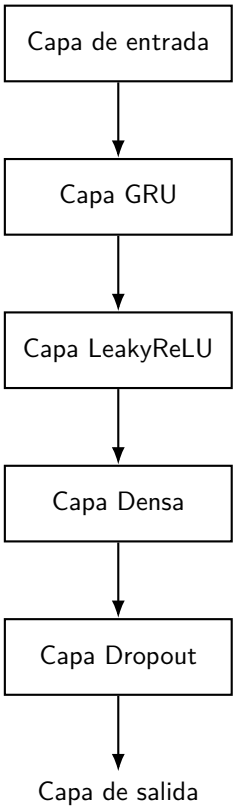


Figura 4: Arquitectura del modelo GRU

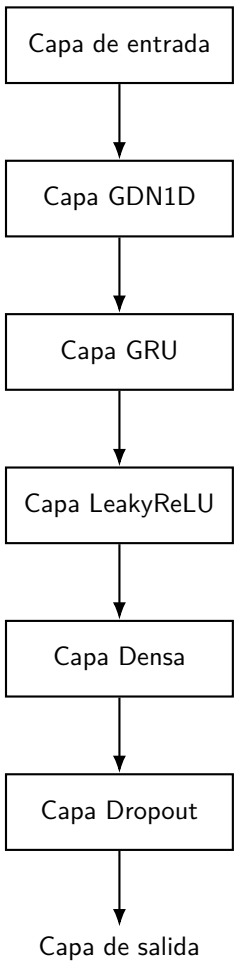


Figura 5: Arquitectura del modelo GRU, incluyendo una capa unidimensional GDN

Modelo LSTM y LSTM GDN

La Figura 6 presenta un modelo basado en la arquitectura LSTM, comenzando con una *capa de entrada* seguida de una *capa LSTM*, adecuada para el procesamiento de secuencias temporales. Una *capa Dropout* se incorpora para reducir el sobreajuste, seguida de una *capa Flatten* que prepara los datos para el procesamiento denso. Finalmente, una *capa densa* conduce a la *capa de salida*, donde se generan las predicciones finales del modelo.

En la Figura 7, se introduce una *capa de Normalización Divisiva Generalizada (GDN)* justo después de la *capa de entrada*, optimizando el procesamiento inicial de las entradas. Esta modificación busca mejorar la capacidad del modelo de manejar características entrantes al normalizarlas antes de pasarlas a la *capa LSTM*. La estructura sigue con una *capa Dropout* y una *capa Flatten*, antes de finalizar en una *capa densa* y la *capa de salida*.

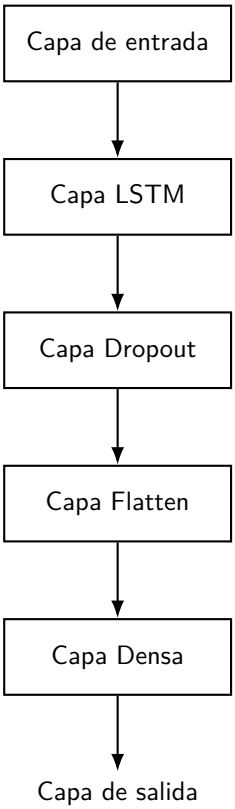


Figura 6: Arquitectura del modelo LSTM

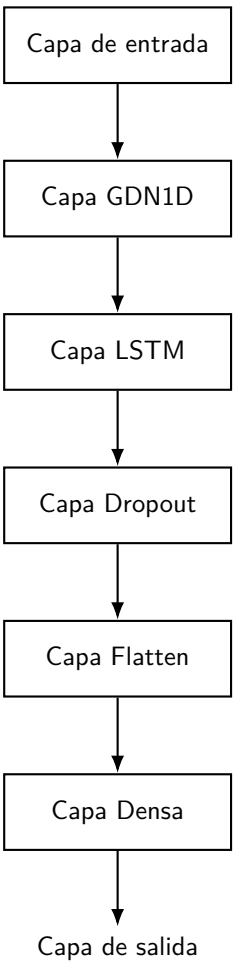


Figura 7: Arquitectura del modelo LSTM, incluyendo una capa unidimensional GDN

Modelo CNN y CNN GDN

La Figura 8 muestra una arquitectura de red neuronal convolucional (CNN) que comienza con una *capa de entrada* y continúa con una *capa Conv1D*. Tras la convolución, los datos pasan por una *capa de Normalización por Lotes* y una *capa de Max Pooling* para reducir la dimensionalidad. Una *capa Dropout* se emplea para mitigar el sobreajuste antes de seguir con una *capa Flatten*, seguida de una *capa densa* y dar lugar finalmente, a la *capa de salida*.

En la Figura 9, se añade una *capa GDN* inmediatamente después de la *capa de entrada*. El resto de la arquitectura incluye las mismas capas de *Conv1D*, *Normalización por Lotes*, *Max Pooling*, *Dropout*, *Flatten*, *Densa*, y finaliza con la *capa de salida*.

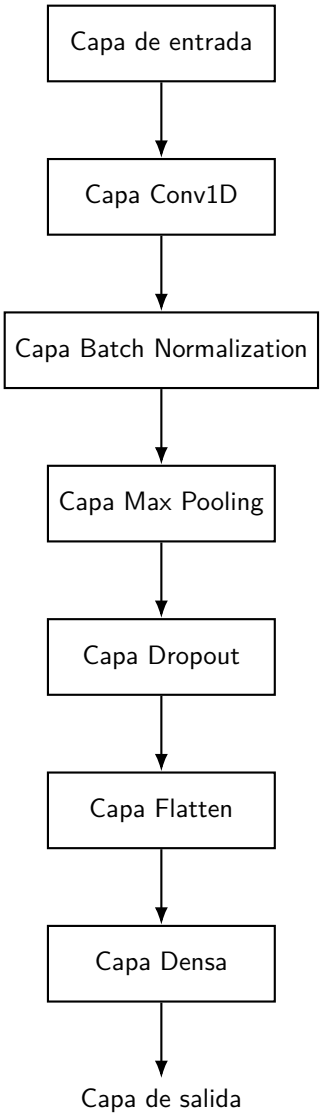


Figura 8: Arquitectura del modelo CNN

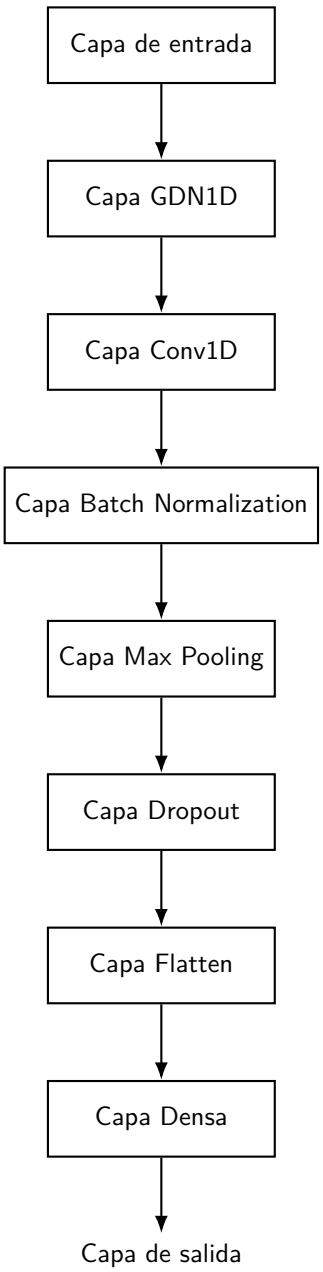


Figura 9: Arquitectura del modelo CNN, incluyendo una capa unidimensional GDN

Curvas de entrenamiento de los modelos

Entrenamiento del modelo Lineal y Lineal GDN

El modelo Lineal, tal y como se aprecia en la Figura 10 muestra una disminución rápida de la función de pérdida, tanto en el entrenamiento como en la validación durante las primeras épocas, seguido de una estabilización cerca de un valor de pérdida mínima.

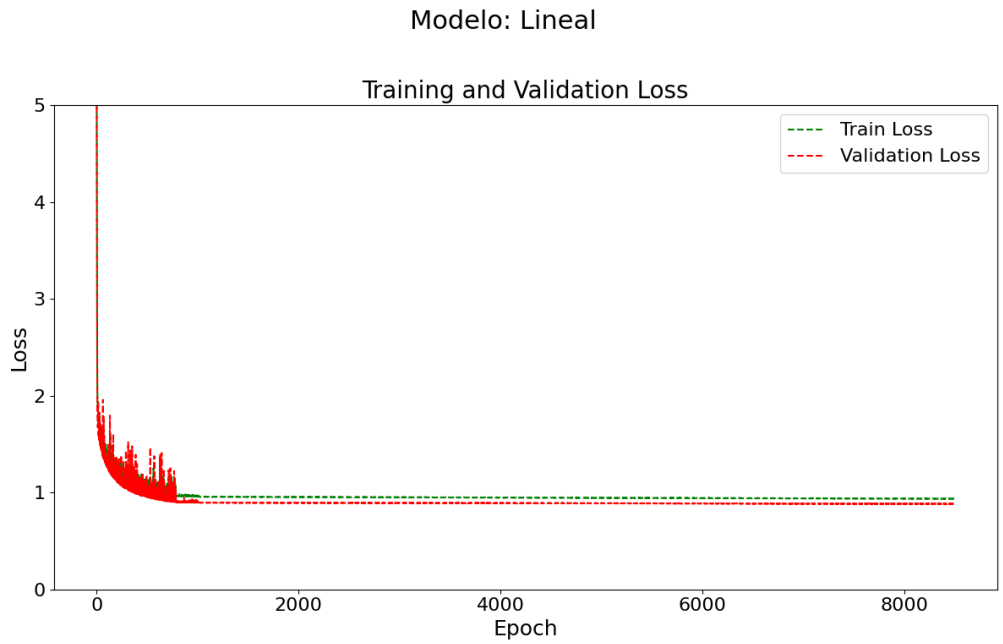


Figura 10: Curva de entrenamiento para el modelo lineal

El modelo Lineal GDN, que aparece en la Figura 11, incorpora una capa de Normalización Divisiva Generalizada. Presenta una variabilidad inicial más alta en las pérdidas de entrenamiento y validación. Aunque esta variabilidad se reduce y las pérdidas se estabilizan después de las primeras épocas. Con el paso de las épocas, la pérdida se estabiliza a un nivel comparable al del modelo lineal, demostrando que la adición de la capa GDN no impide la convergencia del modelo.

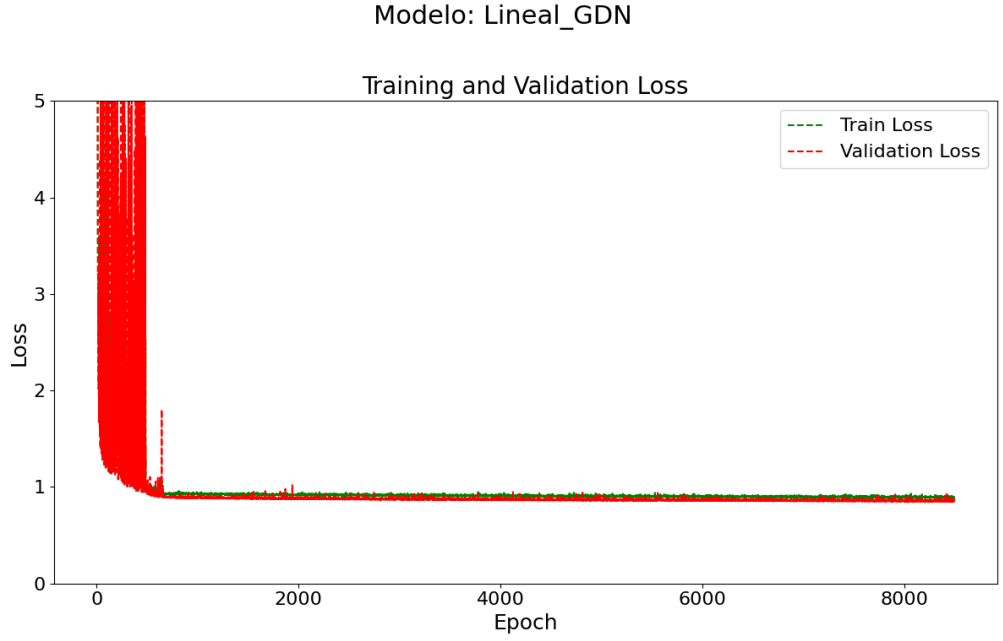


Figura 11: Curva de entrenamiento para el modelo lineal con una capa GDN

Entrenamiento del modelo GRU y GRU GDN

El modelo GRU, representado en la Figura 12, muestra una rápida disminución en la función de pérdida tanto en el entrenamiento como en la validación en las etapas iniciales. La pérdida de entrenamiento se reduce significativamente en un corto periodo y luego se estabiliza, lo que indica un aprendizaje eficaz y una buena generalización sin signos evidentes de sobreajuste.

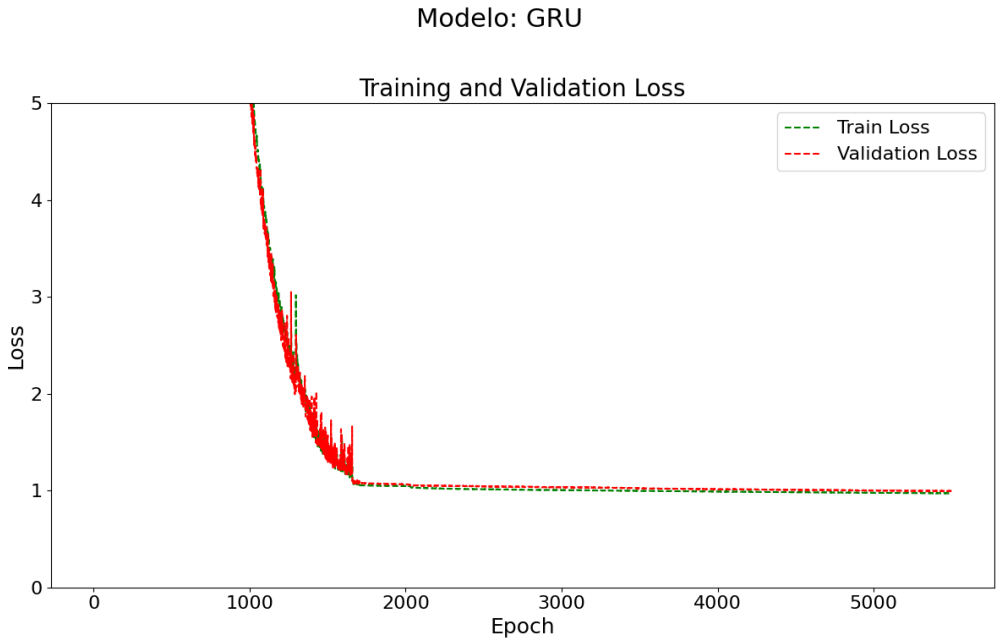


Figura 12: Curva de entrenamiento para el modelo GRU

Por otra parte, el modelo GRU GDN, como se puede observar en la Figura 13, incluye una capa de Normalización Divisiva Generalizada y muestra una mayor variabilidad en las pérdidas de entrenamiento y validación en las primeras épocas. Esta mayor variabilidad podría reflejar el ajuste inicial más complejo requerido por la presencia de la capa GDN. Sin embargo, con el avance de las épocas, tanto la pérdida de entrenamiento como la de validación se reducen y alcanzan una convergencia y estabilización.

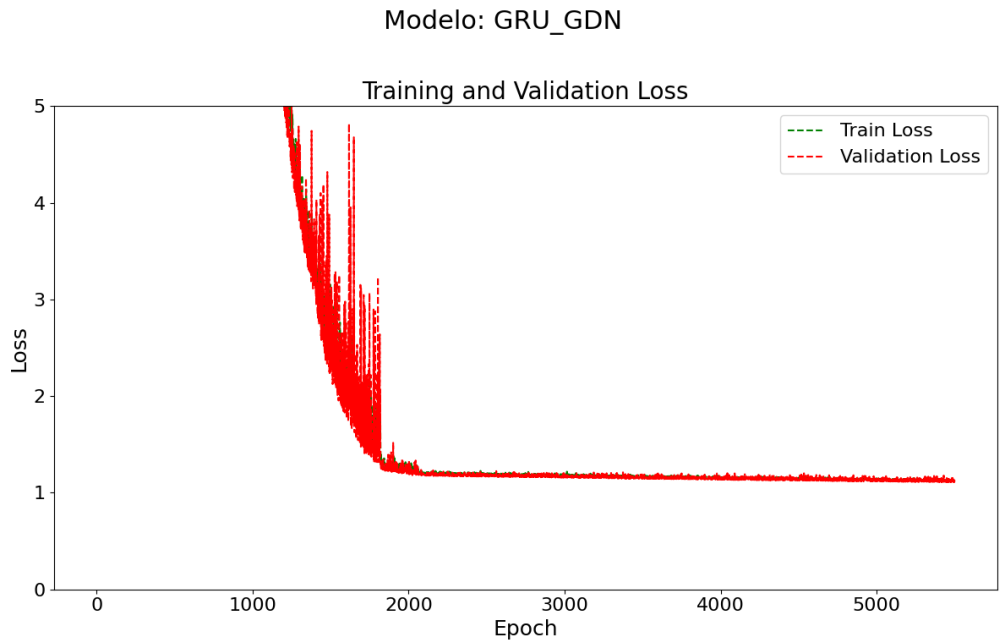


Figura 13: Curva de entrenamiento para el modelo GRU con una capa GDN

Entrenamiento del modelo LSTM y LSTM GDN

El modelo LSTM ilustrado en la Figura 14, muestra una notable eficiencia en la reducción de la pérdida, decreciendo de forma agresiva (tanto en entrenamiento como en validación), hasta llegar a la época 1200 aproximadamente, donde posteriormente se estabiliza. Esto indica un aprendizaje eficiente y estable sin indicios de sobreajuste significativo. La proximidad de las curvas de entrenamiento y validación a lo largo del proceso sugiere que el modelo generaliza bien a nuevos conjuntos de datos.

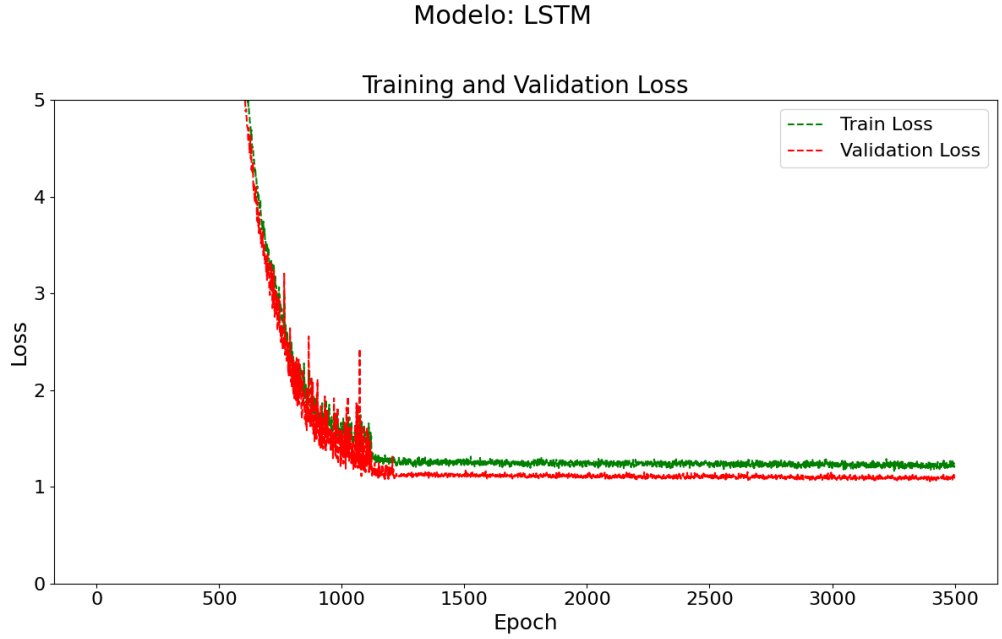


Figura 14: Curva de entrenamiento para el modelo LSTM

El modelo LSTM GDN, que se muestra en la Figura 15, comienza con una mayor variabilidad en las pérdidas de entrenamiento y validación. Esto podría deberse a la complejidad adicional que la capa GDN introduce. Sin embargo, tanto la pérdida de entrenamiento como la de validación convergen y se mantienen en un nivel bajo similar al del modelo LSTM.

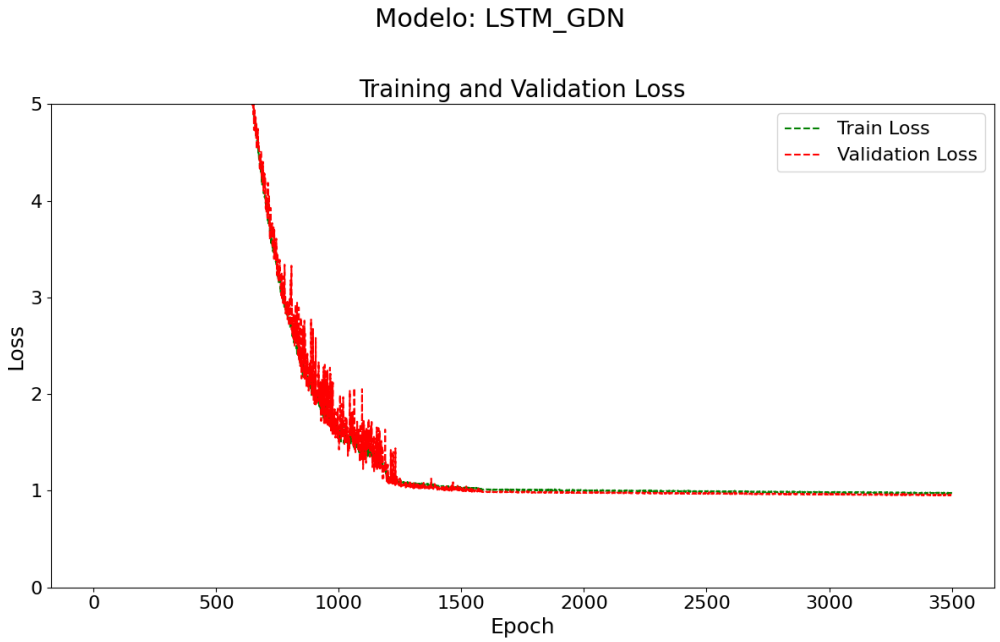


Figura 15: Curva de entrenamiento para el modelo LSTM con una capa GDN

Entrenamiento del modelo CNN y CNN GDN

La Figura 16 muestra la evolución de las pérdidas de entrenamiento y validación para el modelo CNN. Este modelo demuestra una rápida disminución de la pérdida al comienzo del entrenamiento, lo que indica una eficiente capacidad de aprendizaje. Cabe resaltar que después de la caída inicial, las pérdidas se estabilizan y mantienen un nivel bajo y constante.

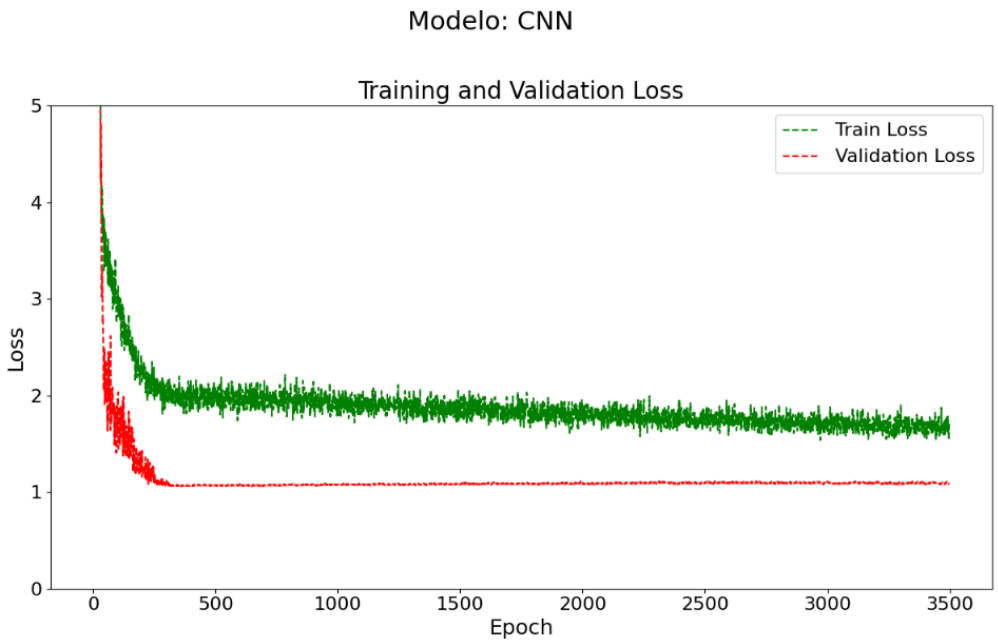


Figura 16: Curva de entrenamiento para el modelo CNN

Por otro lado, el modelo CNN GDN, representado en la Figura 17, consigue reducir la pérdida a un nivel comparable al modelo CNN después de las primeras épocas, demostrando así su capacidad para ajustarse de forma efectiva a los datos, a pesar de la complejidad añadida.

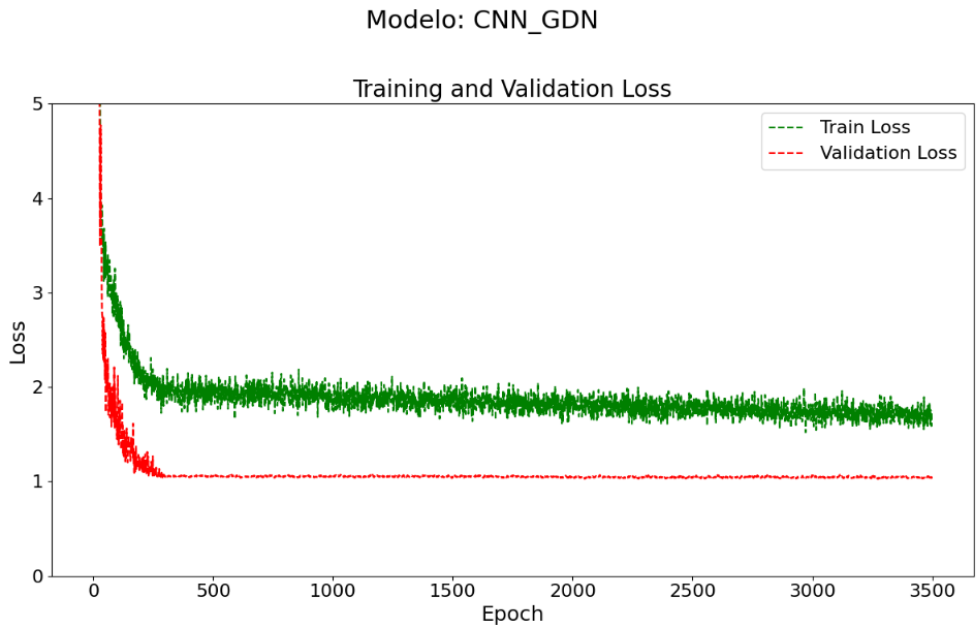


Figura 17: Curva de entrenamiento para el modelo CNN con una capa GDN