Novel efficient deep learning architectures for time series forecasting

TESIS DOCTORAL

Manuel Jesús Jiménez Navarro

Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Diciembre 2022

Novel efficient deep learning architectures for time series forecasting

Memoria que presenta para optar al título de Doctor en Informática con mención de Doctorado Internacional

Manuel Jesús Jiménez Navarro

Dirigida por los Doctores

Dr. Gualberto Asencio Cortés
Dra. María del Mar Martínez Ballesteros

Departamento de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

Diciembre 2022

A mi familia
por ser los pilares de mi vida.

A mis amigos/compañeros
por su infinita ayuda y paciencia.

A aquellas personas que me han llevado a ser lo que soy.

A mi Roy.

The most important step a man can take. It's not the first one, is it?
It's the next one. Always the next step.
Brandon Sanderson, Oathbringer

Tesis doctoral subvencionada por el Ministerio de Ciencia e Innovación bajo el proyecto PID2020-117954RB, el Fondo Europeo de Desarrollo Regional y la Junta de Andalucía para los proyectos PY20-00870 and UPO-138516.

Agradecimientos

Esta tesis ha sido el resultado de una intensa carrera que comencé hace ya unos cuatro años cuando comencé a interesarme por algo que sonaba tan bien como Inteligencia Artificial o aprendizaje profundo. Todo el esfuerzo y los muchísimos golpes (de verdad que han sido muchos) he llegado a poder realizar esta tesis. Sin embargo, este mérito no es solo mío ni mucho menos. He tenido la inmensa suerte de tener a las personas adecuadas a mi lado que me han apoyado en el ámbito personal y profesional, si he llegado hasta este punto es gracias a ellos. Intentaré comprimir todo el agradecimiento que siento por ellos en pocas líneas.

En primer lugar me gustaría agradecer a Gualberto y María, no me imagino a unos directores más pacientes, atentos y comprensibles como ellos. Gualberto y María son una inspiración para mí. Gualberto me motiva a ser creativo desde que lo conocí hace ya tres años, para mí es un gran amigo, profesor e investigador. María me motiva a ser perseverante, no conozco persona más trabajadora, atenta y amable que ella. Además, me gustaría dar infinitas gracias a Paco por todos sus consejos y todo el apoyo que he recibido. Si Gualberto me inspira para ser creativo, Paco me hace ser más práctico y mantener los pies en la tierra. Con la unión de ambos trato de lograr un equilibrio y gracias a María no me he rendido en seguir intentándolo.

Por supuesto, tengo que dar las gracias a todos mis compañeros de la Universidad de Sevilla y la Universidad Pablo de Olavide. Gracias a Pepe Riquelme por las charlas que me inspiraron a comenzar en este mundo con su mítico: "La predicción es la propósito más antiguo de la humanidad". Gracias a José María por decirle a un grupo de chavales: "¿queréis asistir a unas charlas sobre Inteligencia artificial?". Gracias a David por todos los consejos que me da y perdón por ser tan pesado. Gracias también a Manuel, Pedro y Belén por su ayuda en el ámbito profesional y emocional. Gracias a Cristina y Jorge por su paciencia y ayuda. Gracias a Pepe Torres por la época que estuve en la Universidad Pablo de Olavide, si me sentí cómodo fue en parte gracias a ti. Gracias a Alicia por toda su ayuda y por confiar en mí. Gracias a Laura, Andrés y Ángela por aguantarme. También me gustaría agradecer a todos los profesores que ahora son compañeros que, además de enseñarme como alumno, ahora me enseñan en mi época como profesor.

Me gustaría agradecer también a Isabel y a toda al Instituto Politécnico de Beja por acogerme durante la estancia en Portugal. Me he sentido apoyado en todo momento y no me han dado más que facilidades desde el principio.

Gracias a todos mis amigos. Tengo la inmensa suerte de tener unos amigos que no me merezco para lo pasota y desastre que soy. Gracias a Javi, Julio, Reyes, Rueda, Fonso, Diego, Manue, Tamara, Alba, Paula, Sandra, Juanmi, Eva, Jorge, Antonio, Ale, Amelia, Manuela, Juanjo. Y seguro que me he dejado alguno, como digo, no me merezco a amigos con los que comparto ideas y que mehan ayudado tanto durante toda mi vida.

Gracias a mi familia, es gracias a ellos que sigo adelante y que me esfuerzo en ser mejor cada día. Gracias a mi madre, todo el amor que puedo expresar por ella no es ni la mitad del amor que me da ella cada día. Gracias a mi Padre, él fue el origen de mi curiosidad por la ciencia y lo que soy hoy. Gracias a mis hermanos, no les digo suficientes veces lo que les quiero realmente. Gracias a mis tías y tíos por todos los buenos momentos que he pasado con ellos (y los que pasaré). Gracias a mis primos y primas. Gracias a Javier. Gracias a mi Roy.

Resumen

La presente tesis se centra en el estudio de la predicción de series temporales mediante el uso de la técnica conocida como *deep learning* (aprendizaje profundo en español) o redes neuronales. A su vez, se realizan una serie de nuevas propuestas metodológicas, que mejoran la eficiencia de las arquitecturas existentes, aplicadas a una serie de conjunto de datos reales que presenta un reto en la sociedad actual.

La técnica conocida como *deep learning* ha adquirido gran popularidad en los últimos años debido a sus increíbles resultados en áreas como la visión artificial, procesamiento del lenguaje natural y predicción de series temporales, entre otras. Esta técnica se inspira en el funcionamiento de la célula básica del cerebro, la neurona. Las neuronas se organizan en capas formando una red neuronal, procesando la información de entrada y propagando su salida hacia otras capas de neuronas hasta obtener la salida final. Esta técnica ha sido adaptada en múltiples ocasiones a la predicción de series temporales desarrollando arquitecturas con unos resultados con resultados competitivos con el estado del arte actual. Sin embargo, aunque la eficacia ha sido un gran punto a favor, en ocasiones estas arquitecturas han degradado su eficiencia impidiendo su aplicación en escenarios reales. Existen diversas formas de mejorar la eficiencia, reduciendo algunos de los aspectos que toman gran cantidad de recursos como: memoria necesaria para almacenar la arquitectura, tiempo de inferencia o tiempo de entrenamiento, entre otros. Esta tesis se centra en mejorar el tiempo de entrenamiento, pues, resulta el cuello de botella a la hora de experimentar con nuevas arquitecturas, optimizar las arquitecturas existentes o reentrenar arquitecturas en ciertos escenarios reales.

Ante el problema de eficiencia que presentan las arquitecturas dentro del ámbito del *deep learning* o las redes neuronales, se han desarrollado cuatro propuestas diferentes con el objetivo de obtener una eficacia igual o superior que otras arquitecturas de la literatura requiriendo una menor cantidad de recursos computacionales.

La primera de las propuestas introduce dentro del diseño de la arquitectura la idea de un aprendizaje incremental. Esta idea establece diferentes objetivos a las capas de la red neuronal, estableciendo al principio un obje-

tivo muy sencillo e incrementando la dificultad del objetivo asignado a las capas. De esta manera, se acelera el proceso de aprendizaje al ser capaz de aprender rápidamente los conceptos necesarios para el objetivo más sencillo y propagar este conocimiento a las capas posteriores.

La segunda propuesta parte de la primera propuesta y establece una hipótesis adicional. En lugar de que los diferentes objetivos se optimicen sin que los más complejos puedan influir en los más sencillos, se permite que exista influencia. De esta manera, el conocimiento adquirido de los objetivos más sencillos puede ser modificado parcialmente por los subsiguientes objetivos más complejos.

La tercera propuesta surge como idea de las dos primeras propuestas. En este caso la idea fundamental es similar, separar la responsabilidad del proceso de predicción. En esta propuesta se separa la responsabilidad descomponiendo la serie temporal usando un proceso de suavizado. La primera capa, por lo tanto, recibe la entrada suavizada y es encargada de obtener una predicción parcial. La siguiente capa recibe el "residuo" resultante de restar la versión original a la versión suavizada. La siguiente capa, por lo tanto, repite el proceso de suavizado y obtiene una nueva predicción parcial. Tras procesar todas las capas, las predicciones parciales son sumadas para obtener la salida final. La idea intuitiva, por lo tanto, es que cada capa tenga un rol diferente, centrándose en diferentes aspectos de la serie temporal a través de la descomposición. A su vez, las capas deben colaborar para obtener la predicción final.

La cuarta y última propuesta integra la selección de atributos dentro de la arquitectura de una las redes neuronales, con el objetivo de reducir la dimensionalidad del problema y mejorar la eficiencia de las técnicas de selección de atributos aplicadas a *deep learning*. Otras propuestas de selección de atributos aplicadas a *deep learning* tienen problemas de eficacia, eficiencia y/o interpretabilidad. Esta propuesta describe una nueva capa conectada a la entrada que sirva de puerta a las diferentes características de entrada, de esta manera se elimina la influencia de aquellas características que resultan irrelevantes para el problema. Gracias a esta capa se puede determinar las características de forma eficiente, sin disminuir la eficacia de la arquitectura en gran medida. Además, esta capa sirve como ventana a las características que la arquitectura ha establecido como irrelevantes, dando una idea del comportamiento aprendido.

Palabras Clave

Series temporales, predicción, aprendizaje profundo, selección de atributos, eficiencia

Abstract

This thesis focuses on the study of time series prediction using the technique known as deep learning or neural networks. At the same time, a series of new methodological proposals are made, which improve the efficiency of existing architectures, applied to a series of real data sets that present a challenge today.

The technique known as deep learning has gained great popularity in recent years due to its incredible results in areas such as computer vision, natural language processing and time series prediction, among others. This technique is inspired by the functioning of the basic brain cell, the neuron. Neurons are organized in layers forming a neural network, processing the input information and propagating its output to other layers of neurons until the final output is obtained. This technique has been adapted on multiple occasions to the prediction of time series, developing architectures with results that are competitive with the current state of the art. However, although effectiveness has been a great advantage, sometimes these architectures have degraded their efficiency, preventing their application in real scenarios. There are several ways to improve efficiency, reducing some of the aspects that take a large amount of resources such as: memory needed to store the architecture, inference time or training time, among others. This thesis focuses on improving training time, since it is the bottleneck when experimenting with new architectures, optimizing existing architectures or retraining architectures in certain real scenarios.

Faced with the problem of efficiency presented by architectures in the field of deep learning or neural networks, four different proposals have been made, whose main objective is to obtain greater efficiency by obtaining equal or superior effectiveness with respect to the architectures used in the comparative analysis.

The first of the proposals introduces the idea of incremental learning into the design of the architecture. This idea establishes different objectives to the layers of the neural network, establishing at the beginning a very simple objective and increasing the difficulty of the objective assigned to the layers. In this way, the learning process is accelerated by being able to quickly learn the concepts needed for the simplest objective and propagate this knowledge

to the subsequent layers.

The second proposal builds on the first proposal and makes an additional assumption. Instead of the different objectives being optimized without the more complex ones being able to influence the simpler ones, influence is allowed to exist. In this way, the knowledge gained from the simpler objectives can be partially modified by the subsequent more complex objectives.

The third proposal arises as an idea from the first two proposals. In this case the fundamental idea is similar, separating responsibility from the prediction process. In this proposal the liability is separated by decomposing the time series using a smoothing process. The first layer, therefore, receives the smoothed input and is responsible for obtaining a partial prediction. The next layer receives the "residue" resulting from subtracting the original version from the smoothed version. The next layer, therefore, repeats the smoothing process and obtains a new partial prediction. After processing all layers, the partial predictions are summed to obtain the final output. The intuitive idea, therefore, is that each layer has a different role, focusing on different aspects of the time series through decomposition. In turn, the layers must collaborate to obtain the final prediction.

The fourth and last proposal integrates an attribute selection method into the neural network architecture, with the objective of reducing the dimensionality of the problem and improving the efficiency of attribute selection techniques applied to deep learning. Other attribute selection proposals applied to deep learning have problems of effectiveness, efficiency and/or interpretability. This proposal describes a new layer connected to the input that serves as a gateway to the different input features, thus eliminating the influence of those features that are irrelevant to the problem. Thanks to this layer, the features can be determined efficiently, without decreasing the efficiency of the architecture to a large extent. In addition, this layer serves as a window to the features that the architecture has established as irrelevant, giving an idea of the learned behavior.

Keywords

Time series, forecasting, deep learning, feature selection, efficiency.

Índice

Índice de figuras

Índice de Tablas

Parte I

Introducción

Capítulo 1

Introducción

They say "doubt everything", but I disagree. Doubt is useful in small amounts, but too much of it leads to apathy and confusion. No, don't doubt everything. QUESTION everything. [...] Questioning is progress, but doubt is stagnation.

Tom Jubert / Jonas Kyratzes

Vivimos en una era donde grandes cambios se producen en intervalos cada vez más cortos. Uno de los cambios que la humanidad siempre ha buscado ha sido, en cierta manera, optimizar su tiempo. La creación de herramientas que nos asistan en nuestras tareas, o incluso las automaticen, está en nuestra naturaleza. En algún momento de nuestra historia descubrimos que era esencial para nuestra supervivencia poder transmitir, procesar y almacenar la información de una forma eficiente, creando herramientas dedicadas a ello. Ya entre el 8000 y 7500 a.C se usaban fichas de arcilla llamadas "bullae" como forma de realizar el registro de intercambios comerciales. La informática es uno de los resultados del proceso evolutivo de estas herramientas, originando la era de la información. Por supuesto, la informática no fue la única responsable del comienzo de esta era, todo fue consecuencia de múltiples avances científicos en diversas áreas.

La informática abrió el camino a la automatización masiva del acceso y transmisión de la información, ya no es necesario esperar largas horas a encontrar o transferir los datos necesarios. La optimización del almacenamiento nos permite tener acceso a una ingente cantidad de datos de forma sencilla que nunca antes en la historia había sido posible. El procesamiento automático nos permite transformar grandes cantidades de datos en información y,

además, automatizar diversas tareas usando esa información. La informática dio lugar a herramientas que nos asisten y automatizan el procesamiento de la información. Sin embargo, quisimos ir un poco más allá. Para poder realizar una tarea, era necesario tener un conocimiento sobre el problema en cuestión. A más complejo el problema a resolver, más conocimiento era necesario usar para procesar los datos. Esto dio lugar a preguntarnos si sería necesario resolver un problema sin necesidad de poseer un gran conocimiento del problema o invertir largas horas para encontrar una solución. A raíz de esta idea nació la inteligencia artificial, delegando la búsqueda de una solución a una máquina.

La inteligencia artificial intenta replicar la inteligencia, con el objetivo de resolver problemas. Este campo contiene muchas ramas que abordan la inteligencia de varias formas. La rama en la que se centrará esta tesis es la llamada: aprendizaje automático (*machine learning*) [127]. Esta rama basa su funcionamiento en la extracción de conocimiento a partir de un conjunto de datos. Gracias a la automatización del almacenamiento y transmisión de la información, el aprendizaje automático ha revolucionado diversos campos al ser capaz de automatizar tareas donde no era posible obtener resultados suficientemente buenos o era necesario un conocimiento experto demasiado profundo.

Esta tesis se centra en una de las técnicas dentro de la rama del aprendizaje automático conocida como redes neuronales. Esta técnica intenta replicar la inteligencia imitando el comportamiento del cerebro, cuyo componente principal es una célula llamada neurona. Estas neuronas se conectan entre ellas transmitiendo información a partir de ciertos estímulos de entrada. En algún punto, dentro de toda esta red de neuronas, surge la inteligencia que usas para poder leer este mismo texto. Este tipo de técnica también se conoce como *deep learning* [29] y resulta ser bastante versátil dado que puede ser aplicada a múltiples problemas tales como: reconocimiento de imágenes [18], generación de texto [69] o predicción de series temporales [93], entre otros.

De entre todos los problemas posibles, se ha escogido la predicción de series temporales. Este problema consiste en conocer los sucesos que se van a producir en el futuro antes de que estos ocurran. Como asumimos que el mundo en el que vivimos es causal, todo suceso del pasado influye en los sucesos del futuro. Por lo tanto, para realizar predicciones, es necesario almacenar una serie de datos ordenados en el tiempo que se usarán para determinar el futuro. Este conjunto de datos ordenados se conoce normalmente como serie temporal, y se trata de uno de los tipos de datos más comunes en la industria hoy en día.

1.1. Motivación

Esta sección se centrará en analizar las razones por las que se ha elegido estudiar cada uno de los aspectos de los que se compone la tesis. Para ello, se desglosará el título y se detallarán las motivaciones que han llevado a escribir esta tesis.

1.1.1. ¿Por qué predicción de series temporales?

Predecir el futuro ha sido una de las preocupaciones de la humanidad desde la antigüedad. Poder predecir cuándo llovería o cuando era el momento adecuado para cultivar era esencial para nuestra supervivencia. A la predicción siempre se le ha atribuido cierto misticismo, y muchas personas se han atribuido el crédito de poder "ver el futuro"basándose en poca o ninguna base científica. Métodos más arcaicos para intentar predecir el futuro se basaban en la naturaleza estacionaria de ciertos elementos de la naturaleza. Los solsticios son un claro ejemplo donde, cada año, existía un momento donde el sol estaba a más altura, dando lugar a una época cálida y de abundancia. En la actualidad, múltiples modelos matemáticos han sido desarrollados con el objetivo de modelar el comportamiento de eventos naturales como la temperatura, lluvia o terremotos, entre otros.

La predicción es un problema complejo, donde el comportamiento puede llegar parecer hasta caótico en ciertos casos. En este tipo de problemas, la cantidad de factores que pueden influir en el futuro es enorme. Para poder tener en cuenta todos estos factores es necesario medir, almacenar y procesar una o varias series temporales requiriendo grandes cantidades de recursos económicos y humanos. Debido a las limitaciones en estos recursos, la incertidumbre aumenta considerablemente como consecuencia de no poseer los datos de todos los factores implicados.

Sin embargo, pese a la dificultad de este tipo de problemas, los múltiples beneficios sociales y económicos de poder predecir el futuro hacen que esta área adquiera un gran interés. Poder conocer el riesgo de padecer una enfermedad en el futuro, de conocer el número de ventas del siguiente mes u optimizar el flujo de tráfico para reducir la contaminación, son algunos de los ejemplos que motivan centrarse en el uso de la predicción.

1.1.2. ¿Por qué *deep learning*?

La introducción de la técnica conocida como *deep learning* ha revolucionado diversas áreas dentro de la informática. Hasta hace pocos años, reconocer a un perro o a un gato era un problema bastante complejo con pobres resultados. Sin embargo, gracias a la aparición de un nuevo método dentro del contexto del *deep learning* que procesa la información de forma similar a como procesa el cerebro las imágenes, los resultados obtenidos en la clasifica-

ción de imágenes obtuvo una mejora significativa. Otros campos como, por ejemplo, el procesamiento de audio y texto, también han sufrido una mejora de calidad considerable gracias a los nuevos avances dentro del *deep learning* a lo largo del tiempo.

Existen numerosos estudios que aplican métodos dentro del *deep learning* a la predicción de series temporales, produciendo muy buenos resultados. Aun así, esta técnica aún no ha demostrado ser la predominante en este tipo de problemas. Es por ello que es necesaria la exploración de nuevos métodos dentro del *deep learning* que permitan mejorar el estado del arte actual.

Además, la ingente cantidad de datos disponibles hoy en día motiva el uso del *deep learning* que, como casi toda técnica dentro del aprendizaje automático (*machine learning*), mejora su efectividad cuantos más datos disponibles posee.

Por último, dentro de la inteligencia artificial, el *deep learning* reduce aún más la necesidad de poseer un conocimiento profundo del problema a resolver. Esto se debe a que los datos tabulares, datos que normalmente se representan en forma de tabla como un registro de inventario o un registro de la demanda eléctrica, necesitan ser procesados previamente. De esta forma, se facilita la extracción de conocimiento de los distintos métodos dentro del *machine learning*. Este procesamiento implica la necesidad de tener un conocimiento profundo sobre el problema, resultando costoso o imposible en muchas ocasiones. Para evitar la necesidad de realizar ese procesamiento sobre los datos, el *deep learning* es capaz de realizar ese procesamiento de forma automática. Sin embargo, esta ventaja viene con un coste, el *deep learning* suele tomar más recursos y necesitar más datos para poder llegar a obtener buenos resultados.

1.1.3. ¿Por qué eficiente?

Los buenos resultados obtenidos mediante el uso del *deep learning* llevaron a un aumento de investigaciones con el objetivo de mejorar lo máximo posible esta técnica. Diversos investigadores observaron que para poder obtener cada vez un mejor procesamiento de los datos que llevara a una mejor extracción de conocimiento, era necesario aumentar el coste computacional [84]. Actualmente existen métodos que obtienen resultados impresionantes, sin embargo, el coste computacional de estos métodos no es asumible en muchas ocasiones. Al igual que ha sucedido con otros inventos, diversas herramientas se han ido optimizando para reducir la cantidad de recursos necesarios. El ordenador es un claro ejemplo de esta optimización, los primeros ordenadores podían llegar a ocupar una sala entera, pero con el tiempo el espacio y consumo se redujo hasta los ordenadores que usamos hoy en día.

Desde el punto de vista académico, el avance dentro del *deep learning* se está viendo entorpecida debido a las limitaciones computacionales. No todos

los expertos pueden permitirse el coste que conlleva trabajar con métodos del *deep learning*. Esto nos lleva a las siguientes cuestiones: ¿es necesario realmente este enorme coste de recursos para obtener resultados que compitan con el estado del arte? ¿realmente existe una necesidad inherente de la técnica que precise de este cómputo? En la presente tesis se intenta abordar estas cuestiones desde una perspectiva práctica, que fomente el uso de esta técnica de una forma más eficiente.

La eficiencia también cobra gran interés desde el punto de vista industrial. Aplicar el *deep learning* a un problema y ponerlo en producción es un problema complejo que requiere una monitorización constante. Es común que las soluciones obtenidas mediante *deep learning* pierdan eficacia con el tiempo debido a un fenómeno conocido como "concept drift" [3]. A grandes rasgos, este fenómeno provoca un cambio el comportamiento de los datos. Al cambiar el comportamiento de los datos, el conocimiento adquirido por la solución resulta cada vez menos útil degradando la eficacia. Una solución común a este fenómeno es extraer el conocimiento de los nuevos datos almacenados regularmente, creando nuevas soluciones con un conocimiento actualizado. También es común usar múltiples soluciones a la vez, donde cada una se centra en resolver un problema de los múltiples existentes (por ejemplo, predecir las ventas en varias tiendas). Esto produce un coste computacional enorme complicando la escalabilidad. Por lo tanto, es necesario usar métodos que requieran menos recursos facilitando su puesta en producción.

Existen diversas formas de mejorar la eficiencia dentro del deep learning. Esta tesis abordará la eficiencia creando nuevos métodos dentro de la técnica del *deep learning* que permitan extraer el conocimiento de los datos con mayor rapidez requiriendo menos recursos computacionales. El motivo por el que se aborda la eficiencia centrada en el proceso de extracción de conocimiento, se debe a que este proceso es el más costoso computacionalmente normalmente.

1.1.4. ¿Por qué desarrollar nuevas arquitecturas?

Las arquitecturas son aquellos elementos dentro del *deep learning* que contienen todo el conocimiento extraído. Una posible opción para mejorar la eficiencia podría haberse centrado en optimizar los componentes de una o varias arquitecturas dentro del *deep learning* lo máximo posible. Sin embargo, en el campo del *machine learning* las mayores mejoras se han producido cuando se han propuesto nuevas arquitecturas. Existen diversas arquitecturas que han llegado a dominar campos de aplicación como el procesamiento del lenguaje natural o el procesamiento de imágenes [156].

El estudio de un problema puede ser pensado como un problema de optimización, donde cada individuo explora un trozo de espacio de soluciones buscando la mejor solución. Limitarse a buscar muy cerca de un grupo de individuos que han encontrado un trozo del espacio prometedor puede ser

tentador, pero también limita y sesga la exploración de nuevas soluciones que puedan obtener mejores resultados. También es necesario definir qué solución es mejor ¿consideramos solo la eficacia como medida? Por este motivo, en la presente tesis apostamos por explorar nuevas arquitecturas que se alejen un poco de las arquitecturas ya establecidas creando arquitecturas que consideren en mayor medida la eficiencia.

1.2. Objetivos

Los objetivos de esta tesis son los siguientes:

- O.1. Estudiar exhaustivamente la eficiencia y la eficacia de las arquitecturas existentes dentro del estado del arte, aplicadas principalmente a la predicción de series temporales.
- O.3. Desarrollar nuevas arquitecturas que permitan obtener una eficacia parecida o mejor, reduciendo el coste computacional.
- O.4. Desarrollar nuevos métodos de selección de atributos aplicado a arquitecturas de *deep learning*, que permitan obtener el mínimo conjunto de atributos relevantes de forma eficiente y sin perder eficacia.
- O.5. Aplicar las arquitecturas desarrolladas a distintos campos de aplicación para verificar las mejoras.

1.3. Estructura de la memoria

El Capítulo 2 describe el contexto general en el que se enmarca la tesis. Para ello, se presentan los conceptos básicos para comprender los principales aspectos expuestos durante la tesis. Estos conceptos se dividen en tres: *deep learning*, predicción de series temporales y selección de atributos.

El Capítulo II presenta una recopilación de trabajos realizados recientemente en el contexto de la predicción de series temporales mediante el uso del *deep learning*.

La Parte 5.4.2 presenta las distintas propuestas de arquitecturas aplicadas a la predicción de series temporales y selección de atributos. Cada una de estas propuestas se basan en una o varias hipótesis que sirven como las bases en las que se apoya el método. Posteriormente, se explica la metodología implementada en base a las hipótesis. Por último, se discuten los resultados obtenidos y las conclusiones.

El Capítulo 10 describe las principales conclusiones obtenidas para cada una de las propuestas descritas, mientras que el Capítulo 11 analiza los trabajos futuros.

Por último, se incluyen los apéndices donde se introduce el glosario de términos usado durante todo el presente trabajo y resultados adicionales de las distintas propuestas.

1.4. Contribuciones

Las contribuciones principales de esta tesis tanto publicadas como en proceso de publicación son listadas a continuación:

- M. J. Jiménez-Navarro, F. Martínez-Álvarez, A. Troncoso y G. Asencio-Cortés. HLNet: A Novel Hierarchical Deep Neural Network for Time Series Forecasting. In proceedings of 16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021), Advances in Intelligent Systems and Computing, vol 1401, pp 717–727, 2021.
- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez, A. Troncoso y G. Asencio-Cortés. HLNet: A Novel Hierarchical Deep Neural Network for Time Series Forecasting. Logic Journal of the IGPL, 2022 (aceptado).
- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. PHILNet: A Novel Efficient Approach for Time Series Forecasting using Deep Learning. Information Sciences, 2022 (en proceso de revisión).
- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. A New Deep Learning Architecture with Inductive Bias Balance for Oil Temperature Forecasting. Journal of Big Data, 2022 (en proceso de revisión).
- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. Feature-Aware Drop Layer (FADL): A Nonparametric Neural Network Layer for Feature Selection. In proceedings of 17th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2022), Advances in Intelligent Systems and Computing, 2022.

1.5. Resumen

En este capítulo se ha presentado de una forma general el contexto de la tesis doctoral. La presente tesis se enmarca dentro de la técnica llamada *deep learning*, una subrama del *machine learning*. De todas las posibles aplicaciones, esta técnica se ha centrado en la predicción de series temporales.

El objetivo es mejorar la eficiencia reduciendo el tiempo de extracción de conocimiento. Para ello, se propone el uso de nuevas arquitecturas que permitan explorar nuevas soluciones sin sesgarse a soluciones ya establecidas. Se han descrito las principales motivaciones de la presente tesis, los objetivos principales y, por último, las principales contribuciones realizadas.

Capítulo 2

Contexto

Words are inaccurate pointers to reality and should by no means be trusted.

Frederick Lenz.

La predicción de series temporales mediante *deep learning* posee una gran cantidad de conceptos que son necesarios describir para entender la presente tesis. En primer lugar, se detallará el concepto de *deep learning*, sus componentes principales y los mecanismos necesarios para poder extraer el conocimiento de los datos. En segundo lugar, se detallará el concepto de predicción de serie temporal, su composición, los tipos de serie temporal existentes y los tipos de predicción. Por último, es necesario establecer los mecanismos por los cuales se evalúan las arquitecturas aplicadas a series temporales, sus métricas y su interpretación.

2.1. Deep learning

En la técnica conocida como *deep learning*, es la arquitectura la que se encargará de extraer y contener toda la información de los datos. Las propuestas descritas en la presente tesis se centran en modificar los elementos más básicos de esta técnica. Por este motivo, es necesario describir sus componentes más básicos y la forma en la que se extraerá la información del problema.

2.1.1. Perceptrón

La neurona es el componente principal del cerebro, cuya principal tarea es la de recibir ciertos estímulos en forma de entrada, procesar esas esos estímulos y producir una serie de salidas. Con el objetivo de simular el comportamiento de esta célula, en 1957 Frank Rosenblatt realizó una formalización matemática que dio lugar al perceptrón [118].

Los estímulos de entrada se representan como datos en nuestro contexto, estos datos serán procesados para resolver el problema al extraer conocimiento de ellos. Se observó que las neuronas solo empiezan a producir su salida a partir de cierta intensidad de los estímulos de entrada, el valor de intensidad que determina si se producirá la salida o no se denomina umbral. Este umbral es independiente para cada neurona y, además, los estímulos no tienen la misma influencia sobre la neurona. Por lo tanto, se formalizó el procesamiento que realiza una neurona como una suma ponderada de las entradas. Además, se incluyó un umbral (*bias*) que determina si el resultado de la suma ponderada es suficiente para que el perceptrón comience a transmitir la información. El proceso de comenzar a transmitir la información en base al umbral se conoció como activación, determinando cuando el perceptrón pasaría de no activada a activada. El proceso de activación puede modificarse para dar lugar a comportamientos distintos. Por ejemplo, si es necesario que la salida sea binaria, se puede establecer una operación que discretice la salida a valor 0 o 1. Esta operación realizada tras el procesamiento, se conoce como función de activación y representa el comportamiento de la salida del perceptrón.

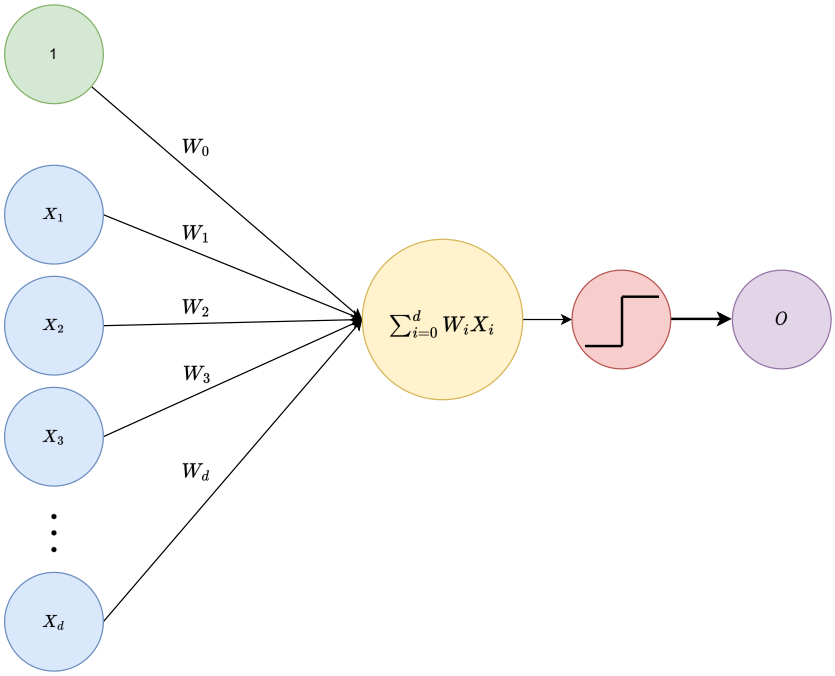


Figura 2.1: Estructura de un perceptrón con vector de entrada X y salida O.

La Figura 2.1 muestra la estructura de un perceptrón clásico donde los d estímulos de entrada son representados con $X_i, i \in 0..d$, mientras que los pesos asignados a cada estímulo se representa con $W_i, i \in 0..d$, siendo W_0

el *bias* que recibe un valor de uno como constante. Tras ello, se aplica la función de activación escalón de Heaviside (H) de la siguiente forma:

$$H(\textstyle\sum_{i=0}^{d} W_i X_i) = \begin{cases} 1, \ si \ \displaystyle\sum_{i=0}^{d} W_i X_i >= 0, & (2.1) \\ 0, \ si \ \displaystyle\sum_{i=0}^{d} W_i X_i < 0. & (2.2) \end{cases}$$

Por lo tanto, si el resultado de la suma ponderada es positiva o negativa se asignará un valor de uno o zero a la salida respectivamente.

Formalmente, el perceptrón puede ser definido matemáticamente de la siguiente forma:

$$O = F(\sum_{i=0}^{D} W_i \cdot X_i) \tag{2.3}$$

donde X_i representa cada una de las entradas que recibe el perceptrón y W_i representa la ponderación (los pesos) que determinan la influencia de esa entrada sobre la activación de la entrada. Hay que tener en cuenta que el umbral está representado con el peso W_0 y la entrada X_0. Existen, por lo tanto, tantos pesos como número de entradas D y, tras realizar la suma ponderada, se aplica la función de activación F para producir la salida O.

2.1.2. Redes neuronales

En el cerebro las neuronas se conectan entre ellas, de manera que intercambian información formando redes que son capaces de colaborar para aumentar la capacidad de procesar la información. De manera similar, los perceptrones pueden unirse, aumentando la capacidad de resolver problemas más complejos gracias a la colaboración de los distintos perceptrones. La conexión de varios perceptrones se denomina perceptrón multicapa. El perceptrón multicapa organiza los perceptrones en capas, donde la información de la capa anterior es propagada a los perceptrones de la capa siguiente sin propagar la información a los perceptrones dentro de una misma capa.

Una red neuronal, en nuestro contexto, es una generalización del perceptrón multicapa. Se estructura normalmente con una capa de entrada, una (o varias) capa oculta y una capa de salida. La capa de entrada representa los datos que se usarán para determinar la salida, mientras que la capa de salida obtiene todas las salidas de la última capa intermedia y los transforma en la salida de la red. La Figura 2.2 muestra un ejemplo de red neuronal donde las neuronas están representadas como nodos sin nombre, unidas por líneas que representan las conexiones entre ellas con un peso asociado. Por simplicidad, el umbral (*bias*) se ha obviado y la función de activación, junto a la suma ponderada, se incluye dentro de los nodos.

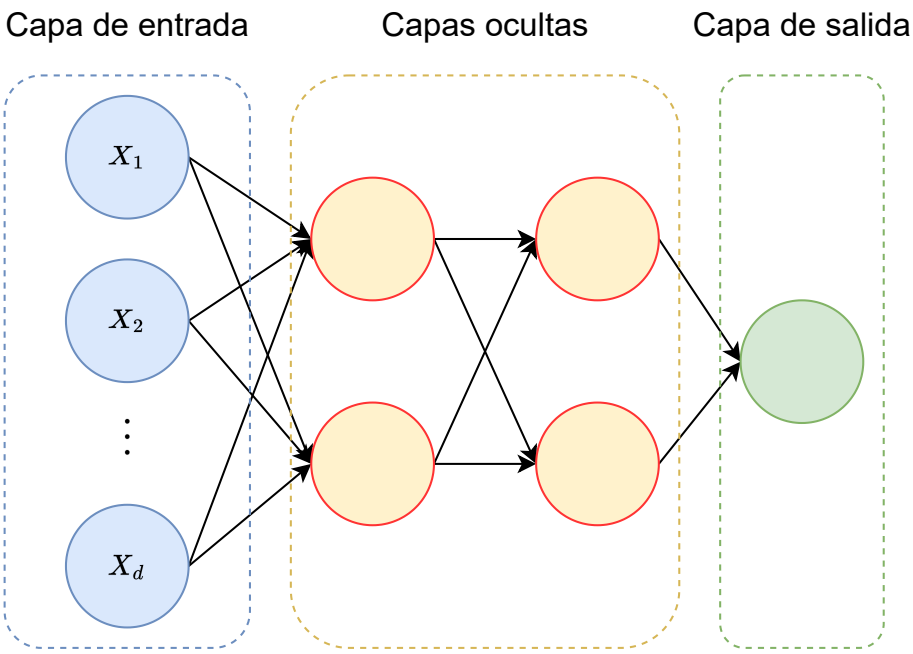


Figura 2.2: Red neuronal con d elementos en la capa de entrada, dos capas ocultas y su capa de salida.

En 1989 Kurt Hornik et al. [64] establecieron el teorema de aproximación universal. Este teorema establece que es posible aproximar cualquier función con una red neuronal usando un número determinado de perceptrones en la capa intermedia con los pesos adecuados. Sin embargo, para poder realizar esta aproximación con funciones más complejas, es necesario incrementar el número de perceptrones hasta un número computacionalmente inoperable. En su lugar, se optó por otra aproximación que requería un menor cómputo con excelentes resultados. En vez de aumentar la "anchura", se aumentaría la "profundidad" (*depth*) de la red. Es decir, en lugar de aumentar el número de perceptrones de la capa intermedia, se añadieron más capas intermedias una tras otra con un menor número de perceptrones. Esto permite a la red neuronal a extraer el conocimiento de forma jerárquica, de forma que conceptos más abstractos se extraen en capas más cercanas a la entrada y conceptos más concretos en capas más cercanas a la salida de la red. Gracias a esto, Rina Dechtner [29] le acuño el nombre de *deep learning* a las redes neuronales con varias capaz ocultas por primera vez en 1986. El impacto de la profundidad en las redes neuronales sigue siendo de interés y su impacto se sigue estudiando en la actualidad [145].

2.1.3. Tipos de capas

Los resultados obtenidos por una red neuronal (su salida), dependen de las distintas transformaciones aplicadas por los pesos de las distintas capas intermedias y la capa final. Las transformaciones realizadas por las capas de las primeras redes neuronales (perceptrón multicapa) han evolucionado, creando distintas variedades que incluyen distintos sesgos sobre las transformaciones que realizan. Es decir, se han desarrollado capas que realizan una transformación orientada a, por ejemplo, extraer patrones espaciales de la entrada o codificar información secuencial. Generalmente, las capas pueden separarse en tres tipos: totalmente conectadas, convolucionales y recurrentes. En esta sección se detallarán los tres tipos incluyendo las transformaciones realizadas por cada una de ellas.

2.1.3.1. Totalmente conectadas

La capa totalmente conectada (*Fully-Connected*) fue la primera capa desarrollada para el perceptrón multicapa. Dentro de esta capa, todas las neuronas (perceptrones) están conectadas con las salidas de la capa anterior. Algebraicamente, esta capa es equivalente a realizar una multiplicación de dos matrices X^D y $W^{D,O}$, donde X representa el vector obtenido como salida de la capa anterior con D dimensiones, W representa la matriz de pesos de la capa totalmente conectada con dimensión DxO. De esta manera, al realizar la multiplicación matricial se transforman el vector Y^O con dimensión O.

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} + \begin{bmatrix} b_1 & b_2 \end{bmatrix} = \begin{bmatrix} o_1 & o_2 \end{bmatrix} \tag{2.4}$$

$$X^3 \times W^{3,2} + B^2 = Y^2 \tag{2.5}$$

La Ecuación 2.4 describe la transformación que realiza una capa con dos perceptrón internamente. La entrada X con tres elementos se multiplica matricialmente con la matriz W con tres filas y dos columnas añadiendo, finalmente, el "bias" de cada perceptrón. Como resultado se obtienen dos salidas, una por cada perceptrón en la capa. Hay que tener en cuenta que no se ha aplicado ninguna función de activación, o lo que es lo mismo, se ha aplicado la función de activación identidad. La siguiente ecuación describe el mismo proceso, pero de forma simplificada.

2.1.3.2. Convolucionales

La capa convolucional fue desarrollada originalmente en 1988 por Kunihiko Fukushima [46] con el objetivo de reconocer patrones en imágenes. El funcionamiento de esta capa se basa en el uso de filtros que aplican una

convolución a la entrada, extrayendo patrones de esta que ayudan a estimar la salida. A diferencia de la capa totalmente conectada, la transformación no se aplica sobre toda la entrada de dimensión D directamente. En su lugar, el filtro es una matriz con un tamaño fijo K, tal que $K < D$. De esta manera, el filtro es aplicado a cada segmento de tamaño K de la entrada, "deslizándose" hacia los demás segmentos de la entrada. Originalmente, esta capa fue aplicada a imágenes extrayendo propiedades como bordes, puntas o gradientes, entre otros. Sin embargo, las capas convolucionales también han demostrado excelentes resultados en otros tipos de datos, por ejemplo, las series temporales.

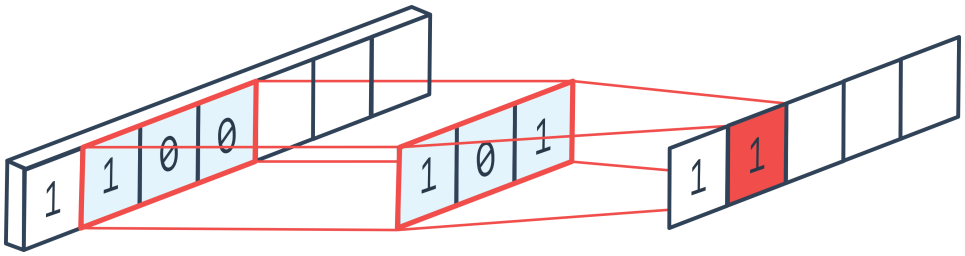


Figura 2.3: Ejemplo de filtro aplicado a un vector unidimensional. A la izquierda se representa la entrada de la capa, en el centro el kernel y a la derecha la salida que produce el kernel.
Fuente: Peltarion.

Algebraicamente, cada uno de los filtros de las capas convolucionales puede verse como un vector F^K, donde la matriz de pesos del filtro se rellena con ceros los lugares de la entrada donde no va a aplicarse la transformación. De esta manera se define la matriz $W^{D,D-K+1}$ donde cada columna representan los mismos pesos del filtro con tamaño K. Posteriormente, esta matriz se multiplica con el vector de entrada X^D con tamaño D, dando lugar a un vector dimensión O que es igual a $D - K - 1$. Posteriormente, cada una de las salidas para cada uno de los filtros se unen para formar la salida de la capa convolucional.

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_D \end{bmatrix}^T \times \begin{bmatrix} f_1 & 0 & 0\cdots & 0 \\ f_2 & f_1 & 0\cdots & 0 \\ f_3 & f_2 & f_1\cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0\cdots & f_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{D-K-+1} \end{bmatrix} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ \vdots \\ o_{D-K-1} \end{bmatrix}^T \tag{2.6}$$

$$X^D \times W^{D,O} + B^O = Y^O \tag{2.7}$$

La Ecuación 2.6 muestra el proceso realizado por capa convolucional aplicado a una entrada X con D elementos multiplicado por una matriz de

pesos W con D filas y $D - K - 1$ columnas, donde se encuentran el filtro F con $K = 3$ componentes. Al igual que en cualquier capa, se incluye el "bias" correspondiente.

2.1.3.3. Recurrentes

Las capa recurrente fue desarrollada originalmente en 1986 por David E. Rumelhart et al. [120] en las redes neuronales de Hopfield [63]. Esta capa fue principalmente desarrollada para aprovechar la naturaleza secuencial de las series temporales. El comportamiento es similar al de una capa totalmente conectada añadiendo información adicional sobre el "estado" (H_t^D) anterior. Este estado es la salida que produjo la capa (H_{t+1}^D) para el evento anterior, retroalimentándose y transformándose a través de sus pesos $W_h^{D,D}$. De forma intuitiva, puede pensarse en el estado como la memoria de la red con información sobre los eventos anteriores, de esta manera, el pasado puede influir en el comportamiento del futuro. Finalmente, para producir la salida, existen pesos ($W_y^{D,0}$) específicos que transforman el estado en la salida, que servirán de entrada para la siguiente capa. Algebraicamente, las operaciones realizadas en esta capa son similares a las capas totalmente conectadas, pero incluyendo el cálculo del estado.

$$X^D \times W^{D,D} + H_t^D \times W_h^{D,D} + B_h^D = H_{t+1}^D \tag{2.8}$$

$$H_t^D \times W_y^{D,O} + B^O = Y^O \tag{2.9}$$

La Ecuación 2.8 muestra el proceso realizado por capa recurrente aplicado a una entrada X con D para actualizar el estado actual y obtener el estado siguiente. En este caso existen dos pesos distintos, uno aplicado la entrada y otro aplicado al estado para calcular el estado siguiente. La Ecuación 2.9 multiplica una matriz de pesos diferente al estado actual, con el objetivo de calcular la salida de la capa.

2.1.4. Descenso del gradiente

Tal y como se ha definido en la sección anterior, las redes neuronales aplican una serie de transformaciones a los datos de entrada hasta obtener la salida. Estas transformaciones son dependientes principalmente de los pesos W ya que son los que principalmente determinarán cuando las neuronas se activarán. Sin embargo, aparecen una serie de preguntas ¿cuáles son los pesos óptimos que son capaces de resolver nuestro problema? el descenso del gradiente es la respuesta a esta pregunta.

El descenso del gradiente se trata de un algoritmo que busca la solución óptima para una serie de variables de entrada. A raíz de esto aparecen nuevas cuestiones ¿cómo podemos saber cuál es la solución óptima? ¿qué es el gradiente y qué significa descender por él?.

En primer lugar, necesitamos una forma objetiva para poder evaluar cómo de buena es la solución propuesta. La medida que habitualmente usada es conocida como error. Esta medida indica a grandes rasgos la diferencia entre la salida esperada y la obtenida. Existen diversas formas de medir el error donde cada una de las formas tiene unas diferentes propiedades, todas estas formas serán detalladas en la Sección 2.3. Por ahora es suficiente con saber que, cuanto mayor sea la diferencia entre el valor predicho y el esperado, mayor será el error. Este error dependerá en gran medida de los pesos que transforman la entrada por lo que, cada combinación posible de pesos tendrá asociada un error formando un espacio con N dimensiones (tantos como pesos) más una (error).

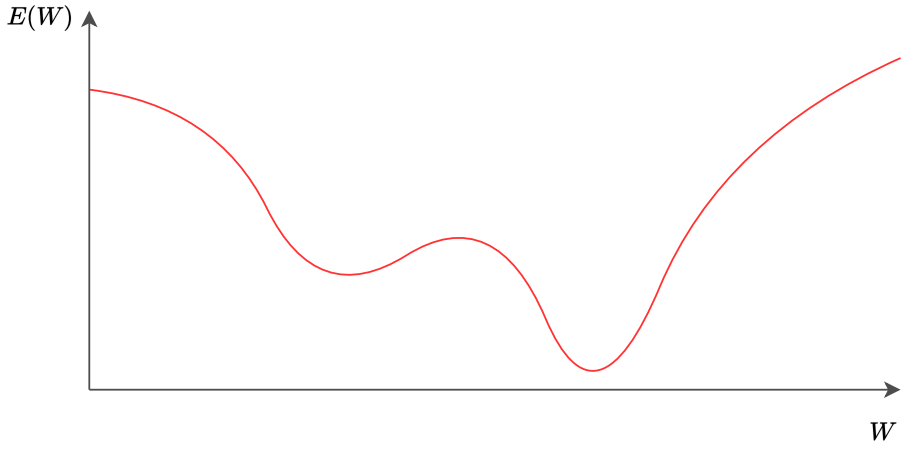


Figura 2.4: Ejemplo de espacio del error en función del peso. Por simplicidad se ha considerado solo un peso W que influye en el error $E(W)$ formando un espacio de dos dimensiones.

La dimensión que representa el error contiene zonas más altas y zonas más bajas, indicando combinaciones de parámetros con mayor o menor error respectivamente. El objetivo, por lo tanto, es acercarse a zonas donde el error es más bajo para obtener una mejor solución. Como explorar todo el espacio de soluciones es computacionalmente muy costoso, partimos de un punto inicial P_0 con un error E_0 e intenta viajar a un punto P_1 con un error E_1 esperando que $E_1 < E_0$. Sin embargo, ¿cómo sabe en qué dirección se encuentra un lugar con menos error? El gradiente de una función de error L obtiene un vector de tantas dimensiones como pesos que tiene la propiedad de dirigirse a lugares donde el error es mayor. Este gradiente se obtiene mediante el cálculo de la derivada parcial de la función de pérdida con respecto a todos los pesos. Como el gradiente apunta a zonas con mayor error, los pesos deben modificarse para seguir el sentido contrario descendiendo por la dimensión del error.

Una vez establecida la dirección y el sentido del vector que nos indica hacia dónde se encuentran zonas con menor error, solo es necesario conocer la magnitud del vector. Esta magnitud es crucial para poder encontrar un mínimo de forma eficiente. Por un lado, pequeñas magnitudes implican explorar el espacio de soluciones de forma más lenta con pasos hacia zonas con menor error muy pequeños. Por otro lado, magnitudes mayores implican pasos hacia zonas con menor error muy grandes por lo que la exploración puede ser más "rápida". Una exploración más lenta normalmente no es deseable puesto que requeriría de más recursos, sin embargo, dar pasos mayores es un riesgo al existir la posibilidad de saltarse o impedir la convergencia a ese mínimo. Por tanto, es necesario establecer un parámetro que regule la magnitud de los pasos que se van a realizar durante la exploración de soluciones. A este parámetro apodado "ratio de aprendizaje" ya que se trata de un valor que regula la velocidad en la que los pesos aprenderán (extraerán el conocimiento) de los datos.

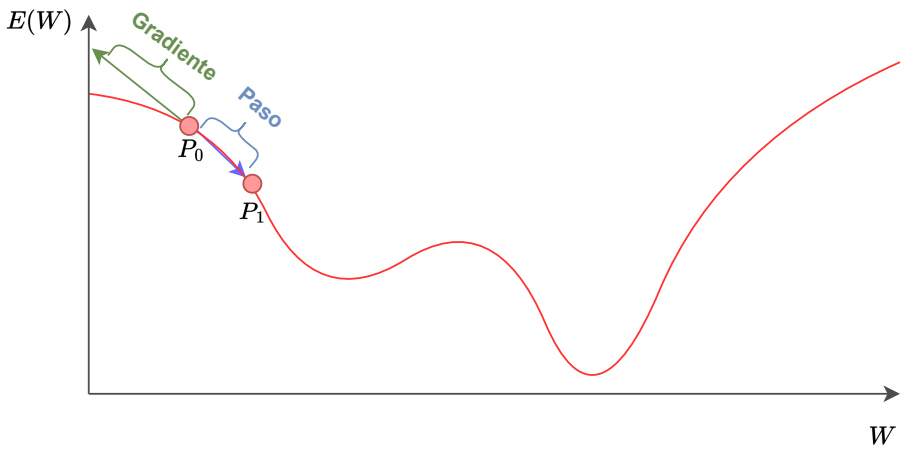


Figura 2.5: Ejemplo de paso realizado en el proceso del descenso del gradiente. Ten en cuenta cómo el paso es menor que el gradiente al reducirse su magnitud por el efecto del ratio de aprendizaje.

El ratio de aprendizaje normalmente establece un valor que comúnmente da lugar a buenos resultados con un tiempo aceptable. Sin embargo, aún existe un problema de eficiencia durante el descenso del gradiente con la dimensión del error. Cuando se exploran soluciones, es necesario realizar el cálculo del error para cada una de ellas. Este cálculo del error se realiza normalmente entre todos los datos que hay almacenados. Sin embargo, el coste para realizar un solo paso puede ser demasiado grande. A raíz de este problema, se estableció una forma alternativa de poder calcular el error para una solución. En lugar de calcular el error con todos los datos, se calcula el error con un pequeño conjunto de datos llamado "batch" con un coste

computacional mucho menor y se realizan los pasos en base a ese error. Este método dificulta el aprendizaje ya que la dirección y sentido de zonas con menor error son menos precisas. Sin embargo, el coste se reduce considerablemente permitiendo el aprendizaje para cualquier cantidad de datos. Este método fue apodado como descenso del gradiente estocástico (SGD por sus siglas en inglés) [116], donde el término estocástico proviene del hecho de que el subconjunto de datos para calcular el error se obtenía aleatoriamente.

2.1.5. Retropropagación

El descenso del gradiente permite optimizar los pesos en base al error que se obtiene tras evaluar los datos. Sin embargo, en una red neuronal ¿todos los pesos influyen de la misma forma en el error? Normalmente, cada peso tiene una influencia diferente sobre el error final. Además, debido a la estructura de las redes neuronales, la salida de las neuronas con sus correspondientes pesos tiene una influencia sobre las neuronas de la capa siguiente y sus pesos. Por este motivo, es necesario un algoritmo que permita obtener la aportación al error de cada uno de los pesos de una forma eficiente. Este método se apodó como retropropagación (*backpropagation*) debido a que propaga la contribución del error desde la salida de la red hasta la primera capa. Este método fue por primera vez aplicado a la optimización de redes neuronales en 1986 por D. Rumelhart et al. [121].

Para asignar la contribución al error, se realiza el cálculo de la derivada parcial del error para cada peso, de manera que se obtiene el gradiente específico para cada peso. Dado que existe una dependencia de ese peso con otros, se usa la regla de la cadena. Esta regla matemática, permite realizar el cálculo de la derivada de una función que se compone a su vez de un conjunto de funciones. Gracias a la retropropagación, cada peso tiene asignado su gradiente con el objetivo de mejorar el error global hasta converger hacia un mínimo. Normalmente es necesario observar varios subconjuntos en múltiples iteraciones, cada iteración sobre todos los subconjuntos de los que se componen los datos suele apodarse época. A todo este proceso por el que se ajustan los pesos usando el descenso del gradiente y la retropropagación se conoce como la fase de entrenamiento.

Tal y como se definió en la Ecuación 2.3, la salida de una neurona se compone de la suma ponderada de las entradas y, posteriormente, la aplicación de una función de activación. Por lo tanto, en una red neuronal con L capas la salida $\hat{y}$ se define como:

$$\hat{y} = F_L(W_L(F_{L-1}(W_{L-1}(\cdots)))), \tag{2.10}$$

donde F^L y W^L denotan la función de activación y pesos de la capa L.

Asimismo, para poder realizar el cálculo del gradiente se debe tomar la derivada parcial de esta función usando la regla de la cadena sobre la función

de error $\mathcal{E}$ para todos los pesos W:

$$\frac{\partial \mathcal{E}(y,\hat{y})}{\partial W} = \frac{\partial E(y,\hat{y})}{\partial F_L(Z_L)} \circ \frac{\partial F_L(Z_L)}{\partial Z_L} \circ \frac{\partial Z_L}{\partial W_L} \circ \frac{\partial F_L(Z_{L-1})}{\partial Z_{L-1}} \circ \frac{\partial Z_{L-1}}{\partial W_{L-1}} \cdots \quad (2.11)$$

donde $\frac{\partial E(y,F^L(Z^L))}{\partial F^L(Z^L)}$ representa la derivada parcial de la función de error con respecto la salida de la última capa L, Z^L representa la entrada de la capa L y $\frac{\partial Z^L}{\partial W^L}$ representa la derivada parcial de la entrada con respecto los pesos de la capa. Es importante tener en cuenta que para calcular la derivada parcial de Z_{L-1} y propagar el error es necesario usar la derivada calculada en la capa anterior. Una de las grandes ventajas de este algoritmo es la eficiencia debido a que, independientemente de lo profunda que sea la red, los cálculos de las derivadas parciales aumentan solo linealmente.

Algoritmo 2.1.5.1 Algoritmo de retropropagación para un red con L capas.

A. Dado un ejemplo con entrada X y valor real y calcular predicción $\hat{y}$.

B. Obtener las salidas de cada una de las capas $(Z_L, \cdots, Z_1)$

C. Computar el error $E(y,\hat{y})$ y su gradiente con respecto la salida de la red $F(Z_L)$

$$\frac{\partial \mathcal{L}_n}{\partial F(Z_L)}. \quad (2.12)$$

D. Por cada capa $l = L, \ldots, 1$ calcular

$$\frac{\partial \mathcal{E}_n}{\partial W_l} = \frac{\partial \mathcal{L}_n}{\partial Z_j} \prod_{k=l+1}^{L} \frac{\partial F_k(Z_k)}{\partial Z_k} \circ \frac{\partial Z_k}{\partial W_k}, \quad (2.13)$$

donde W_j hace referencia a los pesos de la capa j.

E. Actualizar los pesos de la red neuronal por cada capa l $l = L, \ldots, 1$

$$W_l = W_l - \eta \frac{\partial \mathcal{E}_n}{\partial W_l}, \quad (2.14)$$

donde η representa el ratio de aprendizaje.

2.1.6. Funciones de activación

La función de activación es un componente esencial para las redes neuronales que puede tener un gran impacto en la efectividad y eficiencia [17] [88]. En caso de que la función de activación sea no lineal, la red tendrá la capacidad de aplicar transformaciones no lineales y resolver problemas que no pueden ser resueltos con una combinación lineal.

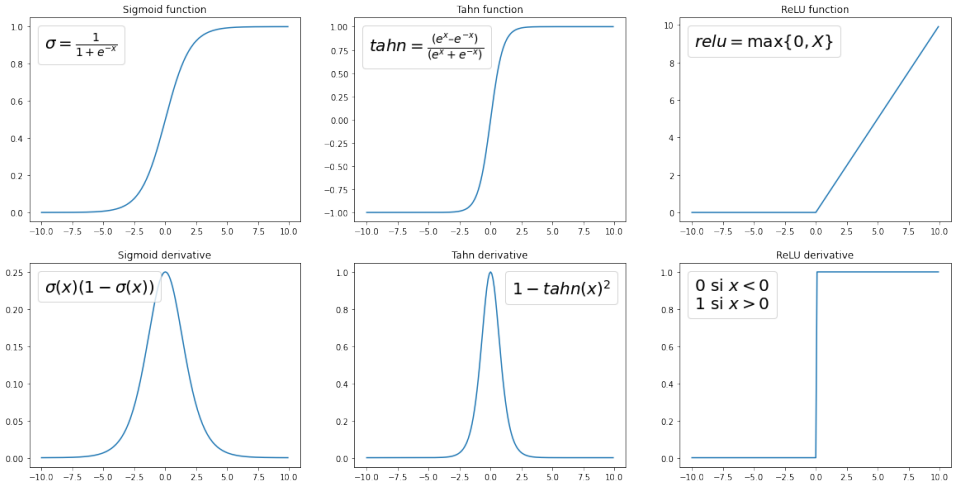


Figura 2.6: Representación de las funciones sigmoide, *tahn* y *ReLU* junto a sus derivadas.

Existen diversos tipos de funciones de activación que han ido evolucionando a lo largo del tiempo para mejorar la eficacia y eficiencia de las redes neuronales. Dependiendo del tipo de problema que se quiera resolver, se debe escoger una función de activación asignándole una semántica específica.

La función lineal o identidad es la más simple ya que no aplica ningún tipo de transformación a la suma ponderada de la entrada de la neurona. Esta función de activación es útil en situaciones donde la salida de la neurona represente un valor numérico real como, por ejemplo, en el caso de la predicción del consumo eléctrico. Formalmente, la función para cualquier entrada X se define como:

$$F(X) = X \tag{2.15}$$

La función sigmoide es una función no lineal donde valores de entrada muy grandes positivos se acercan asintóticamente a uno y valores de entradas muy pequeños negativos se acercan asintóticamente a cero. Este tipo de función de activación fue usada tradicionalmente en capas intermedias o en capas de salida problemas de clasificación multiclase. En clasificación multiclase, la salida de la neurona está entre cero y uno representando la probabilidad de pertenecer a una o varias clases. Formalmente, para cualquier entrada X esta función se define como:

$$F(X) = \frac{1}{1 + e^{-X}} \tag{2.16}$$

La función de activación Tanh (*Tangent Hiperbolic*) es una función de activación similar a la función sigmoide pero centrada en cero y con rango

(-1, 1). Esta función de activación suele ser usada normalmente en capas intermedias al igual que la sigmoide. La diferencias entre la función sigmoide y tahn se ve clara en la comparación de sus derivadas en la Figura 2.6. La derivada de la función tahn es varios órdenes mayores comparada con la sigmoide, lo que da lugar a gradientes con una magnitud varios órdenes mayores. Otra propiedad beneficiosa de la función tahn es que se encuentra centrada en cero de forma simétrica, lo que facilita la convergencia de la red al no existir un sesgo hacia una dirección específica. Formalmente, para cualquier entrada X esta función se define como:

$$F(X) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \tag{2.17}$$

La función *ReLU* (*Rectified Linear Unit*) es una de las funciones de activación no lineales más usadas actualmente en las capas intermedias. Esta función definida a trozos establece un valor de cero para valores negativos mientras que el comportamiento es lineal para valores positivos. Esta función tiene un conjunto de propiedades matemáticas que permiten a la red obtener una mejor eficacia y eficiencia [50]. Múltiples funciones de activación se han definido a raíz de esta red mejorando a la función original, algunos ejemplos son: *SELU* (*Scaled Exponential Linear Units*) [82], *Leaky ReLU* [100] o *ELU* (*Exponential Linear Units*) [23], entre otros. Formalmente, para cualquier entrada X esta función se define como:

$$F(X) = \text{máx}\{0, X\} \tag{2.18}$$

2.1.7. Desvanecimiento de gradientes

El desvanecimiento de gradientes es un problema común que se originó como consecuencia de aumentar la profundidad de las redes neuronales. Este problema se origina como consecuencia de usar funciones de activación como la sigmoide o *tahn* que establecen un rango entre cero y uno como salida. Como se discutió en la Sección 2.1.5, para actualizar los pesos de la red es necesario hallar las derivadas parciales de cada capa. Dado que las capas se conectan entre ellas, forman una composición de funciones compleja que se calculan mediante la regla de la cadena multiplicando las derivadas parciales de cada capa. Cuando nos encontramos en capas más cercanas a la entrada, el error debe propagarse por todas las capas desde la salida multiplicando las derivadas parciales de las funciones de activación. La multiplicación de funciones de activación que transforman los valores en el rango (0, 1) produce que los valores tiendan al cero al tratarse de valores menores a uno. Por lo tanto, cuantas más capas tenga la red, más lejos debe propagarse el error y más pequeño es la magnitud del gradiente. Esto causa que las capas más cercanas a la entrada no lleguen a actualizarse y la red no llegue a converger.

Este problema ha sido estudiado en múltiples ocasiones [61] y se han estudiado soluciones en forma de arquitecturas como la LSTM descrita en la Sección 5.1 y funciones de activación como $ReLU$ mencionada anteriormente.

2.1.8. Inicialización de pesos

La inicialización es un aspecto fundamental en cualquier algoritmo y, por supuesto, de las redes neuronales. Los pesos iniciales determinan el punto inicial en el espacio formado por los parámetros de la red y por el que el gradiente comienza a modificar los pesos hasta llegar a un lugar óptimo. Una mala inicialización puede dificultar el proceso de optimización siendo incapaz de obtener un óptimo.

Las propiedades que normalmente deben tener los pesos de la red neuronal son: usar valores pequeños y sin simetría entre los pesos de neuronas.

Valores pequeños para los pesos son necesarios para evitar el caso contrario al del desvanecimiento de gradientes conocido como explosión de gradientes. En este caso, al contrario que en el desvanecimiento, los gradientes se vuelven cada vez más grandes hasta que llegan a valores mayores a los que un ordenador es capaz de representar. Valores demasiado pequeños hace que los gradientes se acerquen a cero, sin embargo, ese problema ha sido estudiado y solucionado (Sección 2.1.7).

La simetría entre los pesos de diferentes neuronas debe evitarse para lograr que la red neuronal no se atasque en soluciones subóptimas. Si todos los pesos de entrada de una neurona son exactamente iguales a los pesos de entrada de otra neurona, el comportamiento que tendrán (su salida) será exactamente igual. Si dos neuronas tienen la misma salida, el algoritmo de retropropagación le asignará el mismo error. Por lo tanto, el gradiente obtenido durante el descenso por el gradiente será el mismo y los pesos de ambas neuronas sufrirán los mismos cambios. Si esto se repite por toda la red, toda la red asignará los mismos pesos por cada una de sus capas. Esto limitaría en gran medida la capacidad de explorar combinaciones de pesos distintas que permitan llegar a óptimos.

Existen distintas propuestas comúnmente usadas hoy en día que cumplen las propiedades establecidas para inicializar los pesos. La opción más simple es asignar los pesos de forma aleatoria usando una distribución normal. Otras propuestas más sofisticadas como la de Xavier Glorot et al. [49] establece que la distribución de la salida de cada neurona sea igual. Sin embargo, la propuesta de Xavier Glorot et al. fue desarrollada para funciones de activación como sigmoide o *tahn*, pero presentaba problemas para funciones de activación como *ReLU*. Como propuesta de inicialización para funciones de activación como *ReLU*, Kaiming He et al. [60] que establece la misma propiedad adicional pero demuestra que la varianza de la distribución de salida de cada neurona debe ser distinta.

2.1.9. Generalización

Converger hacia un mínimo no siempre implica que la solución encontrada sea la mejor para todos los casos, es posible que la solución encontrada se encuentre en un mínimo local. Por ello, es necesario conocer dos conceptos importantes en el contexto del aprendizaje automático: la generalización y el sobreajuste. La generalización, es la capacidad que tienen las soluciones a extrapolar la solución aplicada a eventos fuera del conjunto usado para ajustar los pesos (entrenamiento). El sobreajuste (*overfitting*), por el contrario, es un fenómeno por el que una solución se ha ajustado demasiado a un conjunto de eventos específico sin ser capaz de extrapolar a eventos fuera de este conjunto. El sobreajuste puede deberse incluso a que las arquitecturas han "memorizado" el comportamiento de los casos durante la fase de entrenamiento.

Existen diversas formas de evitar el sobreajuste aumentando la capacidad de generalizar que están relacionadas con la arquitectura. La forma más habitual de evitar el sobreajuste es aplicando algún tipo sobre distintos factores como puede ser el tamaño y/o pesos del modelo o la dimensionalidad del problema.

Limitar el tamaño del modelo se relaciona con la hipótesis de la navaja de Ockam. Esta hipótesis establece que, a igualdad de condiciones, la solución más sencilla es mejor. Este hecho se relaciona con la generalización debido a que soluciones más complejas y que, por lo tanto, usan más pesos tienen más posibilidades establecer más sesgos sobre los datos. Un modelo más simple, al tener una capacidad más limitada suele generalizar mejor a casos fuera del conjunto de entrenamiento.

La regularización es otra forma de restringir la complejidad de las redes neuronales. Las regularizaciones más comunes aplicadas a las redes neuronales son las conocidas como lasso (L1) y ridge regression (L2). La regularización L1 fue popularizada por Robert Tibshirani en 1996 [151], como un método para la reducción de características mediante la minimización del valor absoluto de los pesos de la arquitectura. La regularización L2 fue propuesta por primera vez en 1963 por Andrey Nikolayevich Tikhonov [152], como un método de regularización similar a L1 mediante la minimización del cuadrado de los pesos de la arquitectura con el objetivo de simplificar la arquitectura. La hipótesis fundamental de la regularización es incentivan a los pesos de la red a acercarse a cero, dando lugar a que no todos los pesos puedan influir. De esta manera, al realizar el entrenamiento, se asignará un error menor a aquellas redes neuronales que usen menos pesos (pesos con valor cero) para que no todas las neuronas tengan que activarse. Este comportamiento es similar al del cerebro, habitualmente no se activan todas las neuronas dando lugar una activación "dispersa" (sparse). Al usar promover que se usen menos pesos, se simplifica la red neuronal aumentando la capacidad de generalización.

La regularización lasso (L1) y ridge (L2) se formalizan como:

$$L1(W) = \sum_{i=1}^{n} |W_i| \tag{2.19}$$

$$L2(W) = \sum_{i=1}^{n} W_i^2 \,, \tag{2.20}$$

donde W representa los n pesos implicados en la regularización.

En la misma línea que la regularización, existe una propuesta creada específicamente para las redes neuronales fácilmente integrable en cualquier tipo de arquitectura como una capa más. La capa de Dropout fue desarrollada en 2014 por Nitish Srivastava et al. [141] como una forma de aumentar la generalización de las redes neuronales. La idea es eliminar el impacto de ciertas neuronas y sus pesos de forma aleatoria durante el entrenamiento de la red. De esta manera, durante el entrenamiento, se obliga a que la red neuronal contenga varias subredes que sean capaces de obtener una buena solución aumentando la dispersión de los pesos. A la hora de realizar el testeo esta capa no elimina ninguna neurona, de esta forma, la predicción producida por el modelo es la combinación de todas las subredes que han sido entrenadas durante la fase de entrenamiento produciendo mejores resultados.

En las propuestas descritas anteriormente se pretende mejorar la generalización reduciendo la complejidad del modelo, sin embargo, otra posible aproximación es simplificar el problema que se intenta resolver. La simplificación se obtiene reduciendo la dimensionalidad del problema eliminando las características que aporten menos a la predicción. Estas técnicas pueden tener en cuenta el modelo, evaluándolo por cada subconjunto de características encontrado por algún algoritmo de búsqueda, o pueden basarse en algún tipo de heurística o estadístico que establezca la importancia de cada característica. La selección de características reduce el impacto de la maldición de la dimensionalidad, introducida por primera vez en 1961 por Richard E. Bellman [12], que influye en la cantidad de datos necesarios para poder obtener una buena eficacia.

2.1.10. Hiperparámetros

A la hora de construir una arquitectura basada en una red neuronal, es necesario establecer ciertos parámetros que la definan. Estos parámetros no se optimizan como los pesos de la red, en su lugar, definen la estructura y controlan aspectos del entrenamiento sin cambiar. Por este motivo, estos parámetros son conocidos como los hiperparámetros.

Los principales hiperparámetros que definen a una red neuronal son el número de capas, número de neuronas por cada capa y la función de activación. También pueden existir parámetros específicos para ciertas capas como:

tamaño del filtro en las redes convolucionales o probabilidad de eliminar la influencia de una neurona en la capa Dropout, entre otros.

Los hiperparámetros que suelen definir el proceso de entrenamiento de la red son: el número de instancias que tendrá cada batch, el número de épocas máximas que durará el entrenamiento o el ratio de aprendizaje, entre otros.

Los hiperparámetros tienen una gran influencia en el desempeño final de la arquitectura. Por lo tanto, normalmente es necesario usar un algoritmo de optimización que busque la combinación adecuada de entre todo el espacio de hiperparámetros establecido en base a una métrica a minimizar. Existen diversos algoritmos de optimización de hiperparámetros usados en el contexto del aprendizaje automático, en esta sección se analizarán tres algoritmos: búsqueda en rejilla, búsqueda aleatoria y búsqueda bayesiana.

La búsqueda en rejilla establece una serie valores por cada uno de los hiperparámetros, generando el espacio de búsqueda a partir de todas las combinaciones posibles. Una vez generado el espacio, se realiza una búsqueda exhaustiva entrenando y evaluando la arquitectura por cada combinación de hiperparámetros. Una vez finalizado el proceso de búsqueda se obtiene la combinación con mejor métrica. Este algoritmo, aunque asegura que se obtiene el mejor resultado para el espacio de búsqueda generado, puede resultar ineficiente para un conjunto de combinaciones elevado. Además, si el proceso de entrenamiento y evaluación de la arquitectura es costoso, este método puede no ser posible. Sin embargo, este algoritmo asegura que se ha evaluado la arquitectura con unas combinaciones específicas y que se ha abarcado todo el espacio de forma uniforme.

La búsqueda aleatoria establece los rangos de cada uno de los hiperparámetros numéricos y una serie de valores para los categóricos. A partir de estos rangos y valores se crea el espacio de búsqueda de hiperparámetros. Posteriormente, el algoritmo genera distintas combinaciones de hiperparámetros dentro del espacio de búsqueda de forma aleatoria, entrenando y evaluando la arquitectura por cada combinación. Dado que en el caso de los hiperparámetros numéricos reales existen infinitos posibles valores, es necesario establecer un número de iteraciones máximas para parar el proceso de generación, entrenamiento y evaluación. Este método suele encontrar mejores resultados que la búsqueda en rejilla ya que no se limita a los valores establecidos para generar las combinaciones. Sin embargo, es posible que no se explore todo el espacio de búsqueda de forma uniforme, lo que puede ser un problema si se requiere analizar la sensibilidad de una arquitectura ante los hiperparámetros.

La búsqueda bayesiana [154] inicializa los hiperparámetros y genera las combinaciones al igual que la búsqueda aleatoria. La diferencia radica en que esta búsqueda establece un modelo probabilístico por cada hiperparámetro. Tras el entrenamiento y evaluación, se actualiza el modelo usando la teoría de Bayes [73] de manera que se ajusta el espacio de búsqueda a zonas más

prometedoras. La principal ventaja con respecto la búsqueda aleatoria es que permite obtener mejores resultados usando menos iteraciones, mientras que las desventajas son las mismas.

2.2. Predicción de series temporales

Las series temporales son uno de los tipos de datos más comunes en la industria, los cuales están compuestos por una serie de eventos (valores numéricos, nominales o lógicos, entre otros) ordenados por una componente temporal. De entre todas las tareas posibles, la predicción es una de las más desafiantes. La predicción intenta estimar el comportamiento de la serie temporal en el futuro. Para ello, debido a la naturaleza causal de muchos problemas, se usan los datos provenientes del pasado para predecir eventos o cualidades del futuro. En esta sección se describirán una serie de conceptos claves de la predicción de las series temporales en el contexto de la tesis.

2.2.1. Tipos de series temporales

Existen diversos tipos de series temporales dependiendo de ciertos aspectos, donde cada uno de estos aspectos representa un problema y/o arquitectura diferente.

Las series temporales pueden clasificarse según el tipo de dato que esté ordenado según la componente temporal. Un video es una serie temporal donde las imágenes (*frames*) están ordenados con respecto el tiempo. De la misma manera, el sonido es otro tipo de serie temporal donde una señal se registra a lo largo del tiempo. El tipo que se estudiará mayormente en esta tesis son valores numéricos muestreados en periodos regulares como, por ejemplo, las pulsaciones por minuto de una persona registradas durante un día.

Los datos de la serie temporal contienen atributos que representan cada una de las variables que se miden a lo largo del tiempo. Por ejemplo, para un mismo instante de tiempo puede medirse la temperatura, la humedad y la velocidad del viento. Si se mide una sola variable, la serie temporal se define como univariante. Sin embargo, si se miden varias variables, la serie temporal se define como multivariante.

Es posible que se obtenga información de una sola variable pero que haya cierta característica que las separe e influya en su comportamiento. Se denomina serie temporal agrupada a varias series temporales que miden la misma variable. Un tipo de agrupamiento que pueden presentar las series temporales es la espacial cuando se mide la temperatura en diferentes estaciones, se obtienen ventas de diferentes tiendas o se mide el tráfico en diferentes ciudades, entre otros ejemplos. En la literatura, a veces se diseña este problema como una serie temporal multivariante donde la entrada de la red son cada una

de las variables a lo largo del tiempo. Sin embargo, el coste computacional aumenta conforme aumenta el número de grupos. Además, esta aproximación establece una serie de requisitos: todas las series temporales deben estar presentes en el mismo instante de tiempo con la misma frecuencia (ver Sección 2.2.3) y al incrementar el número de grupos es necesario reentrenar la red. No siempre es posible asumir estos requisitos, por lo que se toman como series temporales univariantes donde en una ventana de entrada no pueden encontrarse eventos de grupos diferentes.

Cuando existe un problema donde los datos cambian solo en ciertos momentos o ante ciertas circunstancias, es posible que se decida solo almacenar los datos tras el cambio para duplicar la información. Por este motivo, las series temporales se pueden clasificar en regulares e irregulares determinando cuando se almacenan los datos continuamente y cuando no respectivamente.

2.2.2. Tipos de predicción

La predicción puede realizarse de distintas formas, cada una de estas formas influyen determinan un problema y/o arquitectura diferente. Normalmente, la predicción obtiene una serie de eventos futuros llamados horizonte, a partir de unos eventos pasados llamados ventana.

Al igual que los tipos de serie temporal, cuando se realiza una predicción se puede realizar sobre uno o varios atributos. Cuando el objetivo es predecir varios atributos a la vez, se conoce como predicción multivariante. Por otro lado, cuando se predice solo un atributo, se conoce como predicción univariante.

Existen problemas donde se requiere predecir un solo instante de tiempo después, mientras que existen otros donde es necesario predecir varios eventos correspondientes a varios instantes de tiempo futuros para observar la evolución del comportamiento de la serie temporal. Dependiendo de si la cantidad de eventos futuros a predecir es uno o varios, se define la predicción como unihorizonte o multihorizonte, respectivamente.

Cuando la predicción es multihorizonte, existen principalmente dos aproximaciones para obtener los eventos futuros: directa o recursiva. La aproximación directa obtiene todos los eventos futuros a partir de una sola arquitectura o varias arquitecturas donde cada una se especializa en un instante de tiempo (horizonte) futuro. La aproximación recursiva obtiene las predicciones una a una aprovechando las anteriores predicciones, de esta manera, para predecir el instante $t + 2$ se usan los eventos de la ventana $t0 \dots t - w$ y la predicción $t + 1$.

La predicción del futuro puede realizarse para clasificar un evento futuro u obtener una serie de valores numéricos. Es posible predecir la cantidad de tráfico que tendrá una zona para una hora específica o predecir si el volumen de tráfico es alto, normal o bajo.

Varios problemas no requieren predecir un valor exacto, en su lugar es preferible predecir un rango donde es probable que los eventos futuros se encuentren. Para este tipo de casos existen varios tipos de arquitecturas cuyo objetivo es predecir un intervalo para cada instante de tiempo futuro con una cierta confianza. En algunos casos, se establece algún tipo de premisa sobre la distribución en la que se mueven los datos futuros o realizar la predicción de los percentiles de los eventos futuros.

2.2.3. Componentes

Normalmente, las series temporales contienen una serie de componentes que la definen. Los componentes que representan a una serie temporal son: el nivel, la estacionalidad, la tendencia, el ruido y la frecuencia de muestreo.

El nivel representa el valor medio para toda la serie temporal, este valor normalmente se mantiene constante independientemente del instante de tiempo.

La estacionalidad se trata de patrones que se repiten a lo largo del tiempo. Por ejemplo, la temperatura representa una estacionalidad diaria donde el pico es a las horas centrales del día y el valle está en la madrugada. Pueden existir múltiples estacionalidades en una misma serie temporal. Siguiendo con el ejemplo de la temperatura, existe también una estacionalidad anual al elevarse en verano y disminuir en invierno.

La tendencia representa el incremento o decremento lineal a largo plazo que presenta la serie a lo largo del tiempo. Por ejemplo, es posible que de forma incremental la temperatura media aumente más con el paso de los años indicando una tendencia positiva.

Es posible construir la serie temporal mediante una fórmula que usa el nivel, la estacionalidad y la tendencia. Existen dos tipos de fórmulas: la aditiva, donde todos los elementos son sumados, y la multiplicativa, donde todos los elementos son multiplicados. Dependiendo del tipo de problema y serie temporal la fórmula aditiva o multiplicativa puede ser mejor opción para construir la serie temporal. Sin embargo, en el mundo real una serie temporal siempre contiene un componente que no sigue un comportamiento bien definido. Este componente normalmente se llama ruido o residuo y representa la incertidumbre que contiene la serie temporal que puede ser causa de factores caóticos.

Un factor importante en una serie temporal es la frecuencia de muestreo. Este componente representa cada cuanto intervalo de tiempo se almacenarán los eventos. Un ejemplo puede tratarse de un sensor que mide los decibelios en un lugar cada hora, minuto o segundo. Una frecuencia de muestreo alta implica usar más recursos para almacenar todos los datos provenientes del sensor. Sin embargo, una frecuencia baja impide poder modelar el comportamiento de la serie temporal a frecuencias más altas, ya que normalmente no

es posible predecir el comportamiento a nivel de segundo con datos a nivel de hora.

2.2.4. Procesamiento

Para poder permitir o facilitar que se pueda realizar el entrenamiento de una arquitectura de *deep learning*, es necesario realizar una serie de procesamientos. Estos procesamientos son de gran importancia en muchos casos puesto que tienen una gran influencia en la eficacia.

Las arquitecturas usan una ventana W de eventos pasados para realizar la predicción de los eventos futuros (horizonte) H. Dado que normalmente las series temporales se tratan de secuencias con N eventos donde $N > W$, es necesario dividir todos los eventos de la secuencia en subsecuencias de tamaño W. Como muestra la Figura 2.7, el proceso llamado enventanado transforma la serie temporal en subsecuencias de eventos correlativos de tamaño W y su correspondiente subsecuencia de eventos futuras H. A partir de estas dos subsecuencias, la arquitectura usa cada una de las ventanas para producir las predicciones que, posteriormente, son comparadas con los eventos futuros reales. Este proceso viene determinado por el tamaño de ventana, el tamaño de horizonte y el desplazamiento que determina la frecuencia con la que se van a construir las subsecuencias.

Existen atributos dentro de los eventos que representan valores categóricos como el día de la semana, tipo de coche o país de procedencia, entre otros. Estos valores necesitan ser convertidos a valores numéricos para poder ser interpretados por técnicas como las redes neuronales. Hoy en día, existen diversos tipos de procesamiento para estos casos [57].

El rango de los atributos puede influir en el proceso de aprendizaje de las redes neuronales y afectar tanto en la eficacia como en la eficiencia [4]. Por ello, es necesario realizar un proceso que escale los datos a un rango/distribuciones comunes. Algunos de estos métodos escalan los datos a un rango entre cero y uno (Normalización) o modifican la distribución de los datos para que la media sea cero y la desviación típica uno (Estandarización).

Una gran variedad de series temporales se toma mediante sensores. Los sensores tienden a fallar con el tiempo cuando existe algún problema fuera de lo común o se acercan al fin de su vida útil. Es por ello por lo que es común que existan intervalos de tiempo donde ciertos eventos o atributos no se han medido correctamente, dando lugar a lo que se conoce como valores perdidos. Normalmente, los métodos de aprendizaje automático no pueden trabajar con valores perdidos por lo que se usa un procedimiento para imputar ese valor aproximándolo al valor que debería tener. Existen diversas técnicas [39], desde imputar esos valores con el valor más común hasta usar una arquitectura para predecir ese valor.

Al igual que los sensores pueden no dar uno o varios valores, es posible

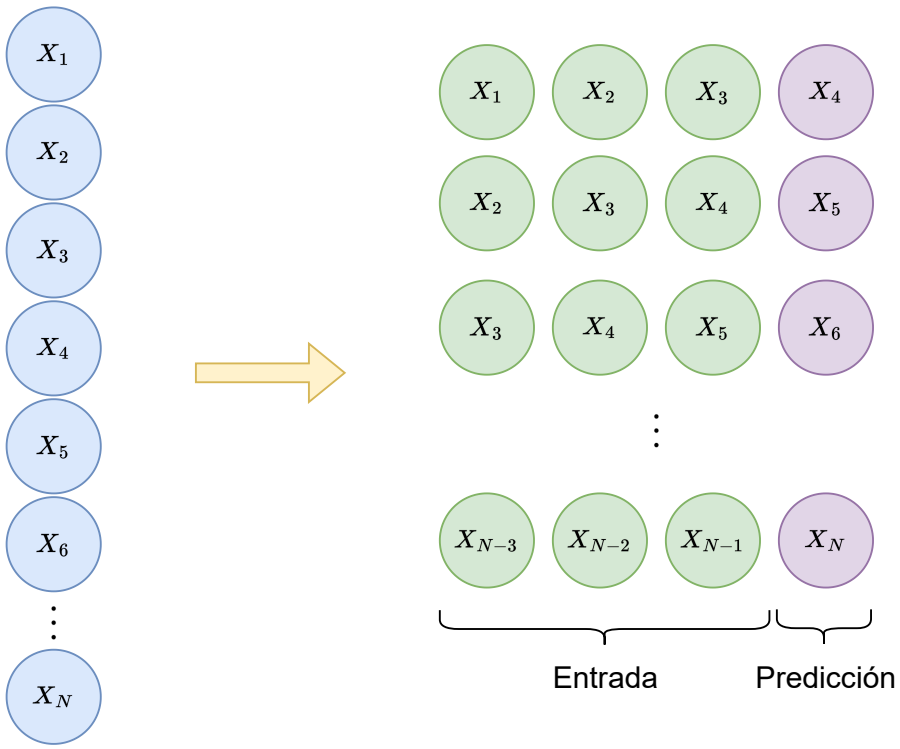


Figura 2.7: Proceso de enventanado aplicado a una serie temporal univariante $X_i, i \in 1..N$ con N eventos para un tamaño de ventana igual a tres y una predicción de un paso en el futuro.

que los valores sean incorrectos. Cuando el valor de un atributo se encuentra fuera del rango de valores normales, a este valor se le conoce como valor atípico. Cuando se detecta un valor atípico se puede invalidar y usar algún procedimiento para imputarlo como en el caso de los valores perdidos. Sin embargo, este caso posee una dificultad adicional ¿cómo de fuera del rango debe estar un valor para considerarlo atípico? La solución no siempre es sencilla y depende del contexto del problema. Actualmente existen diversos métodos que detectan los valores atípicos, estos métodos pueden ser muy simples mediante el uso de umbrales o más sofisticados mediante el uso de una arquitectura que detecte los valores atípicos. Hay que tener en cuenta que, aunque los métodos más sofisticados pueden dar mejores resultados, normalmente repercute en la eficiencia del proceso por el que se obtienen las predicciones de la arquitectura en producción.

2.3. Evaluación

Para poder conocer cómo de buena es una solución con respecto otra solución, es necesario establecer una serie de métodos que permitan medir de forma fiable las características de la solución. Esta sección se centrará en describir las métricas, sus propiedades e introducir el concepto de generalidad que resulta de vital importancia para evaluar las soluciones en el contexto de la predicción de series temporales.

2.3.1. Métricas

Las métricas estudiadas se centrarán en dos aspectos: eficacia y eficiencia. La eficacia mide la calidad de los resultados con respecto el valor real mientras que la eficiencia mide el consumo de recursos usado por la solución.

2.3.1.1. Eficacia

Antes de definir las métricas que se describirán a continuación, es necesario definir una serie de elementos:

- Se define al elemento t, como al instante de referencia en el que se realizan las predicciones.
- Se define al elemento N, como al número de instancias pertenecientes al conjunto de datos usadas durante la evaluación de la métrica. Cada instancia tendrá asociado un instante de tiempo t.
- Se define al elemento H, como al número de elementos futuros a predecir, el horizonte.
- Se define al elemento y_{t+h}, como a la predicción para el instante futuro $t + h$.
- Se define al elemento $\hat{y}_{t+h}$, como al valor real tomado en el instante futuro $t + h$.

El error absoluto medio (*Mean Absolute Error* o MAE) es de las métricas más fundamentales, mide la media de las diferencias entre los pares compuesto por las predicciones obtenidas por la arquitectura y los eventos reales. Cuanto más cercano es el MAE a cero, mejor es la solución evaluada. La métrica se representa de la siguiente forma:

$$MAE = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{H}\sum_{h=1}^{H}\left|y(t)_h - \hat{y}(t)_h\right|, \tag{2.21}$$

donde $|y(t)_h - \hat{y}(t)_h|$ se define como el valor absoluto de la diferencia entre el valor predicho y el valor real.

El error cuadrático medio (*Mean Squared Error* o MSE) mide el cuadrado de las diferencias entre los pares compuesto por las predicciones obtenidas por la arquitectura y los eventos reales. Esta métrica es la más usada para la evaluación de soluciones aplicadas a predicción. Debido a que toma el cuadrado de las diferencias, esta métrica se caracteriza por dar un mayor peso a errores mayores. La métrica se representa de la siguiente forma:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{H}\sum_{h=1}^{H}(y(t)_h - \hat{y}(t)_h)^2\,, \tag{2.22}$$

donde $(y(t)_h - \hat{y}(t)_h)^2$ se define como el cuadrado de las diferencias entre el valor predicho y el valor real.

El error cuadrático medio realiza una estimación de la varianza de los errores. Si se toma la raíz cuadrada se realiza una estimación de la desviación típica. Esta métrica es conocida como la raíz del error cuadrático medio (*Root Mean Squared Error* o RMSE) y se representa de la siguiente forma:

$$RMSE = \frac{1}{N}\sum_{t=1}^{N}\sqrt{\frac{1}{H}\sum_{h=1}^{H}(y(t)_h - \hat{y}(t)_h)^2} \tag{2.23}$$

Con el objetivo de facilitar la interpretación de los valores aportados, existen métricas que miden la eficacia de las soluciones de forma porcentual. El error porcentual absoluto medio (*Mean Absolute Percentile Error* o MAPE) mide la desviación los errores en valor absoluto con respecto el valor real. La métrica se representa de la siguiente forma:

$$MAPE = \frac{1}{N}\sum_{t=1}^{N}\frac{1}{H}\sum_{h=1}^{H}\left|\frac{y(t)_h - \hat{y}(t)_h}{y(t)_h}\right| \tag{2.24}$$

El error porcentual absoluto medio es una métrica comúnmente usada, sin embargo, dependiendo del contexto puede existir escenarios donde los valores aportados por la métrica no reflejen la realidad. Un problema bien conocido es causado por la división por el valor real que, cuando el valor real toma valores cercanos a cero, se obtienen valores de error muy por encima de lo que realmente sucede. El error porcentual absoluto medio ponderado (*Weighted Absolute Percentile Error* o WAPE) evita este escenario usando el sumatorio de todos los valores reales como divisor, de esta forma, pequeños valores puntuales en el conjunto de datos no afectan significativamente sobre la métrica. La métrica se representa de la siguiente forma:

$$WAPE = \frac{1}{H}\sum_{h=1}^{H}\frac{\sum_{t=1}^{N}|y(t)_h - \hat{y}(t)_h|}{\sum_{t=1}^{N}|y(t)_h|} \tag{2.25}$$

2.3.1.2. Eficiencia

La eficiencia se centrará en medir los recursos computacionales necesarios para todo el ciclo de vida de las arquitecturas, desde su entrenamiento hasta su puesta en producción. Los elementos principales que se analizarán son: tiempo de entrenamiento, tiempo de inferencia y número de parámetros.

El tiempo de entrenamiento mide el tiempo necesario para ajustar los parámetros de la arquitectura hasta obtener la convergencia. La convergencia, se define como la solución donde el mínimo error posible ha sido encontrado. Es de suma importancia minimizar el tiempo de esta métrica debido a que el entrenamiento es el proceso que más recursos suele necesitar, tomando días o incluso meses en muchas ocasiones para una sola arquitectura.

El tiempo de inferencia mide el tiempo desde que se alimenta a la arquitectura con ciertas ventanas de entrada y obtiene las predicciones una vez ya entrenado. Esta métrica adquiere gran importancia en problemas donde el tiempo de respuesta debe ser muy rápido y las predicciones deben obtenerse a nivel de milisegundo o menos.

El número de parámetros mide la cantidad de parámetros que contiene la arquitectura que modela la solución. A diferencia del tiempo de inferencia y tiempo de entrenamiento, esta métrica mide los recursos computacionales en términos de memoria. Esta métrica puede implicar la imposibilidad de integrar las soluciones en dispositivos con recursos muy limitados como móviles, relojes inteligentes o dispositivos IoT, entre otros.

2.3.2. Tests estadísticos

Evaluar la efectividad o eficiencia de los algoritmos normalmente no es tan sencillo con usar las métricas para un modelo y ver cuál es el valor más bajo de error o tiempo de ejecución. Normalmente se usan varias configuraciones para una misma arquitectura, se evalúan varias arquitecturas sobre varios conjuntos de datos o se repite el proceso de entrenamiento varias veces obteniendo resultados diferentes debido a la aleatoriedad. Si, por ejemplo, una arquitectura obtiene un error menor a otra arquitectura en solo un caso, pero es similar en todos los otros casos ¿podemos afirmar realmente que la primera arquitectura es mejor? Es necesario poder evaluar que las diferencias entre varias arquitecturas y entre varios conjuntos de datos son estadísticamente significativas, esto quiere decir que las conclusiones obtenidas mediante una muestra de resultados pueden generalizarse a la distribución real.

Para ello, el método estadístico normalmente usado para poder establecer conclusiones es el contraste de hipótesis. Este método parte de una hipótesis inicial (hipótesis nula) que se desea contrastar, esta hipótesis será la que se mantendrá a no ser que se demuestre lo contrario. En nuestro contexto, la hipótesis nula establecerá la siguiente hipótesis: la distribución de los resul-

tados obtenidos para dos arquitecturas a comparar es igual. En caso de que no se demuestre lo contrario, se aceptará al no encontrar ninguna prueba que sea capaz de rechazarla. Sin embargo, si se encuentran pruebas que rechacen la hipótesis nula se aceptará la hipótesis alternativa: la distribución de los resultados obtenidos para dos arquitecturas a comparar no es igual. Para demostrar o rechazar la hipótesis nula, se establece un valor que ayude a medir el grado de significancia de la hipótesis. Esta medida se conoce como p-valor y mide la probabilidad de aceptar la hipótesis alternativa pero que en la realidad no sea cierta. Si el p-valor es muy bajo, significa que podemos rechazar la hipótesis nula y aceptar la alternativa. Si el p-valor es muy alto, no existen evidencias suficientes para rechazar la hipótesis nula por lo que se acepta. Para medir cuán bajo o alto es el p-valor, se suele establecer un rango entre el 95 % de probabilidad, es decir, si el p-valor es menor al 5 % se considerará que el valor es bajo y se rechazará la hipótesis nula con un 95 % de confianza.

Los tests estadísticos pueden dividirse de varias formas dependiendo de las características del propio método o de los resultados usados para obtener las métricas. Los tests pueden dividirse en tests pareados o no pareados, muestras independientes o dependientes, paramétricos o no paramétricos.

Los tests pareados establecen el contraste de hipótesis sobre los mismos escenarios para dos individuos. En nuestro contexto, los individuos son las arquitecturas y los escenarios los conjuntos de datos sobre los que se han entrenado/evaluado obteniendo sus métricas. Por otro lado, los tests no pareados pueden establecer comparaciones entre distintos algoritmos para distintos escenarios. Dependiendo del test escogido para realizar el contraste de hipótesis se puede requerir una muestra pareada o no pareada.

Los individuos/arquitecturas a comparar pueden considerarse independiente o no independientes. Se consideran individuos independientes como aquellos individuos que no tienen ninguna relación entre ellos, mientras que los individuos dependientes comparten algún tipo de relación o existe una influencia el uno en el otro. Por ejemplo, comparar dos arquitecturas con los mismos parámetros y entrenadas con dos subconjuntos solapados del mismo conjunto de datos, se considera una muestra dependiente ya que los datos compartidos entre los dos subconjuntos influyen en el rendimiento de ambas arquitecturas. Sin embargo, dos arquitecturas distintas evaluadas sobre el mismo conjunto de datos se consideran muestras independientes. Dependiendo del conjunto de datos, se asume la dependencia o independencia de los resultados usados para realizar el contraste.

Dependiendo de las premisas establecidas por el test estadístico, se puede considerar los tests como paramétricos o no paramétricos. Los tests paramétricos establece como premisa que los resultados usados para realizar el contraste deben pertenecer a una distribución de algún tipo, mientras que los tests no paramétricos no realizan ningún tipo de premisa sobre la distri-

bución. Para determinar la distribución a la que pertenecen los resultados, es necesario usar otro test estadístico que permita establecer unas conclusiones significativas que extrapolen a la problación. Por ejemplo, el test de Saphiro-Wilk [129] permite contrastar si los datos pertenecen a una distribución normal.

Cada tests, asume uno o varios escenarios dependiendo del número de arquitecturas y conjuntos de datos para obtener los resultados con el objetivo de realizar los tests estadísticos. Se pueden diferenciar cuatro escenarios:

A. Dos arquitecturas evaluadas sobre el mismo conjunto de datos.

B. Dos arquitecturas sobre varios conjuntos de datos distintos.

C. Varias arquitecturas evaluadas sobre el mismo conjunto de datos.

D. Varias arquitecturas evaluadas sobre conjuntos de datos distintos.

Método	Pareado	Paramétrico	Dependiente	Escenario
T de Student [30]	Sí	Sí	Sí	A
McNemar [104]	Sí	No	Sí	A
Wilcoxon [160]	Sí	No	Sí	A,B
ANOVA [48]	No	Sí	No	C,D
Friedman [45]	No	No	No	C,D

Tabla 2.1: Tests estadísticos usados para contraste de hipótesis entre arquitecturas. Para cada método se describe sus características junto al escenario que cubre.

En el caso de evaluar varias arquitecturas, los tests estadísticos normalmente establecen que existen diferencias entre las distribuciones de resultados. Sin embargo, no indica entre qué modelos se presentan esas diferencias significativas. Por ello, se usan otros tests conocidos como post-hoc [59] [58] [36] que aportan esta información.

La Tabla 2.1 muestra distintos tipos de métodos usados comúnmente y sus características.

Los tests estadísticos son muy útiles, pero tienen algunos problemas que dificultan su uso e interpretación. Por ejemplo, algunos problemas encontrados en el contraste de hipótesis son:

- Normalmente resulta de interés la probabilidad de que la hipótesis nula sea cierta. Sin embargo, el p-valor no responde a esta pregunta.

- El tamaño del efecto y de la muestra no son distinguibles.

- En caso de no rechazar la hipótesis nula, no se puede afirmar con seguridad que sea cierta. Si no que no existen evidencias suficientes como para rechazarla.

Debido a estos problemas existentes, Alessio Benavoli et al. [14] propusieron una alternativa bayesiana para el contraste de hipótesis. Esta alternativa ofrece versiones alternativas para métodos existentes como el test de rango de signos de Wilcoxon y test de Friedman.

2.3.3. Generalidad

Para medir la capacidad que tiene un modelo de generalizar a otros casos, la aproximación común es realizar una separación de los datos usados para el entrenamiento de la red en dos conjuntos diferentes llamados: conjunto de entrenamiento y conjunto de prueba. El conjunto de entrenamiento se usa para extraer el conocimiento y ajustar los pesos sin usar en ningún momento el conjunto de prueba. Una vez que la solución termina el proceso de aprendizaje, se aplica alguna métrica sobre el conjunto de prueba y entrenamiento. Si la efectividad sobre el conjunto de prueba es peor que la efectividad sobre el conjunto de entrenamiento es posible que se haya producido un sobreajuste. Por otro lado, si las métricas son similares se considera que la solución es capaz de generalizar correctamente.

Es una buena práctica monitorizar la arquitectura mientras se ajustan sus pesos para observar su evolución o parar el entrenamiento en el momento que se detecte un deterioro en la generalidad. Por ello, normalmente se divide el conjunto de datos en otro conjunto conocido como conjunto de validación. Este tipo de separación suele llamarse *hold-out* al mantener fuera del proceso de entrenamiento una parte del conjunto de datos. Sin embargo, es necesario que el conjunto de validación y prueba sean unas muestras que representen fielmente a todo el conjunto de datos. En caso contrario, la evaluación no sería efectiva y se obtendrían conclusiones que no reflejarían la realidad. Un ejemplo de este problema puede ocurrir en problemas de clasificación si en los conjuntos de prueba y validación la distribución de las clases no es similar a la del conjunto. En el caso de las series temporales, una mala elección del conjunto de validación o prueba puede ocurrir si se escoge un periodo que no represente toda una estación completa en series temporales estacionales. Por ejemplo, escoger solo el periodo correspondiente a primavera para evaluar una solución para una serie temporal de demanda eléctrica no sería una representación fiel de la eficacia de la solución.

En ocasiones, es posible que una solución obtenga muy buenos resultados para el conjunto de prueba, pero al obtener nuevos eventos la eficiencia se deteriora mostrando unos resultados peores de los esperados. Esto puede ocurrir porque la solución encontrada se ha sobreajustado al conjunto de pruebas, mostrando una eficacia mucho mejor de la que realmente tiene. Co-

mo solución de este fenómeno se propuso otro método de evaluación llamado validación cruzada. El método divide el conjunto de datos en N subconjuntos iguales, usando N-1 partes para entrena y para las pruebas. El proceso se repite N veces cambiando el conjunto de pruebas por un subconjunto diferente cada vez. Como resultado final se obtienen N métricas evaluadas sobre todo el conjunto de datos. Esta métrica suele ser una medida más fiable de la generalidad que otros métodos de validación. Sin embargo, el coste computacional de realizar esta validación es mucho mayor que la validación mediante *hold-out*. Además, en escenarios como las series temporales, resulta de mayor utilidad evaluar la generalidad con un conjunto de pruebas que se encuentre en un instante de tiempo mayor al conjunto de pruebas al ser el escenario más realista.

2.3.4. Resumen

En este capítulo se han descrito los conceptos más importantes en los que se centra esta tesis. El primer concepto es el de las redes neuronales o *deep learning*, se trata de una arquitectura pensada para modelar el conocimiento a partir de una serie de datos imitando los componentes básicos del cerebro, la neurona. Además, se ha descrito el proceso por el que la red neuronal adquiere el conocimiento ajustando los parámetros que transforman la entrada en la salida deseada mediante el uso del descenso por el gradiente y la retropropagación. El segundo concepto importante es el de la predicción de series temporales, una tarea que obtiene un conjunto de eventos ordenados por una componente temporal (serie temporal) y obtiene un os eventos futuros (predicción). Se ha caracterizado las series temporales describiendo una clasificación y componentes que la conforman. Se ha tipificado las distintas formas de realizar la predicción y se ha descrito el procesamiento habitual sobre los datos para permitir y facilitar la extracción de conocimiento de las redes neuronales. Por último, se ha descrito las principales métricas para evaluar la calidad de las soluciones modeladas a partir de la red neuronal en tres aspectos fundamentales: eficacia, eficiencia y generalidad.

2.4. Selección de atributos

La presente tesis presenta una propuesta centrada en el uso de la selección de atributos [32], debido a sus ventajas en la eficiencia de arquitecturas *deep learning*. Por lo tanto, es necesario introducir y comprender la aplicación de esta técnica.

La selección de atributos o reducción de la dimensionalidad consiste en reducir el número de características que contiene un conjunto de datos admitiendo solo aquellos relevantes para el problema y, dependiendo del tipo, la arquitectura usada. Huan Liu et al. [96] en una de sus divisiones de los

diferentes métodos de selección de atributos establece las siguientes tipos: filtro, *wrapper* e integrado.

En esta sección se describirá las principales ventajas de usar la selección de atributos en el contexto de la tesis y las principales técnicas usadas para la selección de atributos.

2.4.1. Ventajas e incovenientes

La selección de atributos posee una serie de beneficios sobre diferentes aspectos como: eficacia, eficiencia, interpretabilidad y obtención de datos. Durante esta sección se analizarán cada uno de los aspectos enmarcado dentro del contexto del *deep learning* y predicción de series temporales.

Uno de los aspectos fundamentales por los que se ha estudiado la selección de atributos es debido a su impacto en la eficacia en muchos algoritmos de aprendizaje automático. Normalmente una buena selección mejora la eficacia obtenida en la evaluación debido a varios factores. Uno de los factores es que el problema se simplifica gracias a la reducción de la dimensionalidad de este lo que, a su vez, permite que la solución generalice mejor. Otro factor importante es la presencia de atributos que no tienen una relevancia en el problema, introduciendo ruido que degrada la eficacia de la solución. La selección de atributos permite eliminar aquellos atributos que no tienen una relevancia sobre el problema o que directamente su comportamiento es ruidoso. Por último, el fenómeno conocido como la maldición de la dimensionalidad [13] es un factor importante que afecta a la mayoría de técnicas de aprendizaje automático. Este fenómeno establece que, a mayor dimensionalidad, más datos son necesarios para obtener unos buenos resultados. Esto se debe a que el espacio del problema aumenta exponencialmente con el número de atributos y los datos se "dispersan". Aunque las redes neuronales no tienen por qué verse afectada su eficacia por la selección de atributos, diversos estudios demuestran que una selección puede mejorar la eficacia en múltiples problemas como se demostrará en la propuesta de la Sección 9.

La selección de atributos es de suma importancia en contextos donde los recursos computacionales son limitados, debido a la mejora de eficiencia que se produce en todos los aspectos del desarrollo de aplicaciones basada en datos. En ocasiones, la eficiencia es mucho más importante que la eficacia al ser preferible una solución menos eficaz pero capaz de ajustarse a los requisitos propuestos por el cliente. Gracias a la selección de atributos, es posible eliminar aquellos atributos no relevantes del medio de almacenamiento usado. El procesamiento aplicado a los datos también se acelera al procesar menos información y la solución necesitará menos tiempo para obtener las predicciones. El ahorro en recursos computacionales se hace especialmente importante en las series temporales donde, dependiendo de la frecuencia de muestreo, eliminar una característica reduce considerablemente el consumo de almacenamiento. Por último, la eficiencia durante el entrenamiento de las

redes neuronales puede mejorar. Esto es gracias a que las transformaciones que debe aprender la red neuronal son más sencillas al centrarse en menos información más relevante.

La interpretabilidad es un aspecto de gran importancia para las redes neuronales debido a su naturaleza como "caja negra". Es decir, no es posible obtener una serie de indicadores que permitan al usuario establecer un razonamiento que responda a porqué se ha tomado una decisión o cual es el conocimiento que la solución se ha adquirido. La interpretabilidad también es importante para la predicción de series temporales al permitir aumentar la confianza en las predicciones obtenidas, sabiendo que no se ha dejado llevar por sesgos o conclusiones erróneas. La selección de atributos es una manera sencilla de obtener una interpretación de las predicciones al proporcionar el conjunto de atributos que son relevantes e irrelevantes para el problema en general. Gracias a ello, es posible establecer hipótesis sobre la naturaleza del problema y sacar conclusiones sobre las predicciones realizada. Si además, la selección de atributos se ve influenciada por la arquitectura, se puede relacionar la relevancia con el conocimiento extraído.

El último de los aspectos que consideramos de importancia en nuestro contexto es la obtención de datos. Es común usar sensores o algún sistema informático para obtener los datos de forma automática. Sin embargo, existen contextos donde la recolección debe realizarse manualmente o es muy costoso. Por ejemplo, consideramos una serie temporal sobre los datos clínicos de un paciente a lo largo del tiempo, donde los datos se recolectan a través de varias pruebas que un especialista debe llevar a cabo. Cada una de las pruebas puede tomar un tiempo y esfuerzos enormes sin conocer con exactitud la relevancia de estos en el riesgo de padecer ciertas enfermedades. La selección de atributos, al reducir la dimensionalidad del problema, también permite evitar recolectar datos que no son relevantes para el problema.

2.4.2. Filtrado

La técnica conocida como filtrado establece una valoración de las diferentes características basándose en una medida de relevancia. Estas medidas analizan cada par de atributos dentro del conjunto de datos de forma supervisada (teniendo en cuenta la variable objetivo) o no supervisada y establecen un umbral que separa aquellas variables relevantes de las irrelevantes. La Tabla 2.2 muestra un conjunto de las medidas de relevancia más comunes junto a varias propiedades. Las propiedades interesantes incluyen el tipo de atributos que puede procesar (numéricos o categóricos) y el tipo de relación que se asume (lineal o no lineal). Adicionalmente, aunque no son considerados medidas de relevancia, es posible usar otros métodos para valorar la relevancia de un atributo. Estos métodos incluyen usar la varianza del atributo o los coeficientes de un modelo lineal entrenado con el conjunto de datos como medida de relevancia.

Medida	Tipo de variable	Relación
Coeficiente de Pearson [44]	Numérica	Lineal
Coeficiente de Spearman [34]	Numérica	No lineal
Coeficiente ANOVA [48]	Numérica y Categórica	Lineal
Coeficiente de Kendall [78]	Numérica y Categórica	No lineal
Tablas de contingencia [138]	Categórica	-
Ganancia de información [172]	Numérica y Categórica	-
Información mutua [38]	Numérica y Categórica	-

Tabla 2.2: Medidas de relevancia usadas comúnmente en la selección de atributos. Incluye los tipos de atributos que son capaces de procesar y el tipo de relación que asume entre los atributos.

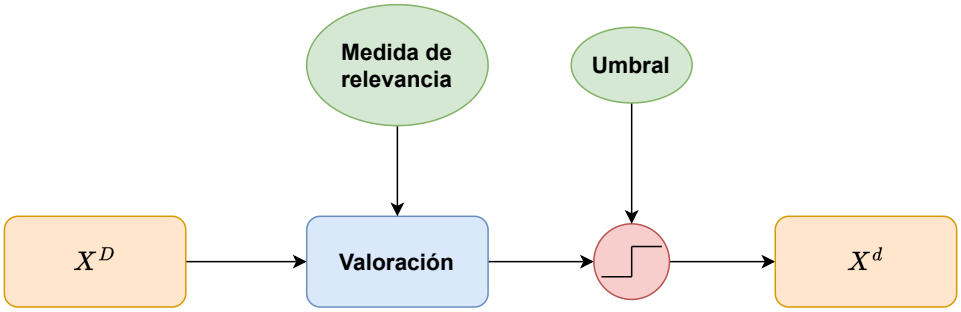


Figura 2.8: Proceso de selección de atributos para la técnica de filtro con un conjunto de datos X con D atributos de entrada y d atributos de salida siendo $d < D$.

La Figura 2.8 muestra el proceso completo que se lleva a cabo durante el filtrado. Los hiperparámetros, mostrados en color verde con una elipse, son la medida de relevancia escogida y el umbral por el que se determinará qué atributos mantener.

Una de las desventajas de esta técnica es que la arquitectura que posteriormente se usará para modelar la solución, no tiene ninguna influencia en el proceso de selección. Debido a ello, cualquier interpretación obtenida tras la selección es solo dependiente de los datos y la medida de relevancia, por lo que no es posible relacionar las características seleccionadas con el comportamiento de la arquitectura.

Entre las ventajas se encuentra que, comparada con las otras técnicas que se describirán posteriormente, el filtrado es muy eficiente ya que el cuello de botella se encuentra normalmente en el entrenamiento y evaluación de la arquitectura. Debido a ello, es una técnica idónea para las redes neuronales al no necesitar repetir el proceso de aprendizaje.

2.4.3. Wrapper

La técnica de filtro no considera a la arquitectura durante el proceso de selección. Esto puede afectar a la efectividad obtenida, por lo que la técnica conocida como *wrapper* integra a la arquitectura en el proceso de selección.

A grandes rasgos la técnica mediante filtro no es muy diferente a la de *wrapper*, existen dos diferencias fundamentales. La primera diferencia radica en que la medida de relevancia es la propia arquitectura. Sin embargo, a diferencia del filtro, la arquitectura no obtiene la relevancia de los atributos. En su lugar, la arquitectura evalúa el subconjunto de atributos seleccionados obteniendo una valoración general de todo el subconjunto y no de cada atributo independientemente. Esto nos lleva a la segunda diferencia fundamental, en lugar de valorar la relevancia de los atributos es necesario buscar el subconjunto óptimo con respecto la métrica escogida. La búsqueda se realiza de forma iterativa, entrenando y evaluando la arquitectura en cada iteración hasta la obtener el conjunto óptimo o cumplir con una condición que termine la búsqueda (ejemplo: el error es menor a un umbral).

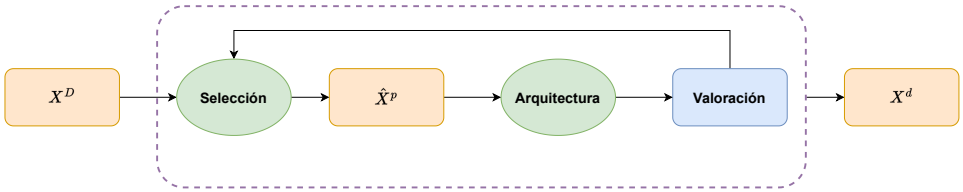


Figura 2.9: Proceso de selección de atributos para la técnica *wrapper*.

La Figura 2.9 muestra el proceso llevado a cabo por el método *wrapper*, donde recibe un conjunto de datos X con D atributos de entrada y d atributos óptimos de salida siendo $d < D$. Ten en cuenta que, durante el proceso iterativo de búsqueda se obtienen soluciones parciales $\hat{X}$ con p atributos donde $p < D$.

Los principales hiperparámetros del método son el método que realiza la selección del subconjunto de atributos y la arquitectura que se usará para entrenar y posteriormente evaluar el subconjunto. Finalmente, tras evaluar todos los posibles subconjuntos o terminar el proceso tras la condición de parada, se obtiene el mejor subconjunto de atributos.

Existen diversas estrategias para generar el subconjunto de atributos, las principales son: selección exhaustiva, selección aleatoria, selección hacia delante, selección hacia atrás y bidireccional. La selección exhaustiva es la estrategia más ineficiente ya que genera todas las posibles combinaciones, sin embargo, la estrategia asegura que se obtenga el subconjunto óptimo. La selección aleatoria es una aproximación más eficiente que la exhaustiva y, tal y como indica su nombre, genera los subconjuntos de forma aleatoria hasta alcanzar la condición de parada. La selección hacia delante parte de un

conjunto vacío de atributos, en cada iteración, añade el atributo con mejor eficacia obtenida durante la evaluación hasta llegar a la condición de parada. La selección hacia atrás realiza lo contrario que la selección hacia delante, parte de todos los atributos y, en cada iteración, elimina el atributo que degrade menos o mejore más la eficacia durante la evaluación. Por último, la selección bidireccional realiza la selección hacia delante seguida de la selección hacia atrás combinándolas. Normalmente, en este tipo de técnicas se aplica una condición de parada en base al número de atributos que se desea seleccionar.

La principal ventaja de esta técnica es que es posible relacionar la selección con la arquitectura y, además, llegar a hacer interpretable esa selección. Sin embargo, la principal desventaja de este tipo de método es su eficiencia ya que necesita reentrenar la arquitectura con cada iteración. En ocasiones, la eficiencia no es un impedimento ya que el modelo es lo suficientemente simple como para que el entrenamiento no suponga un gran incremento del tiempo. Este no es el caso de las redes neuronales, ya que el tiempo de entrenamiento puede ser muy alto incluso para una sola arquitectura.

2.4.4. Integrada

Las técnicas de selección integradas obtienen lo mejor de la técnica de filtrado y *wrapper* al incluir el proceso de selección de atributos dentro de la arquitectura. Cada técnica realiza la integración del proceso de selección de forma distinta. Las principales arquitecturas que integra la selección se pueden dividir en dos tipos: regularización y árboles. La Figura 2.10 representa el proceso llevado a cabo por la selección de atributos integrada. Tal y como se observa, la arquitectura obtiene la selección de atributos como resultado del proceso de entrenamiento de todo el conjunto de datos.

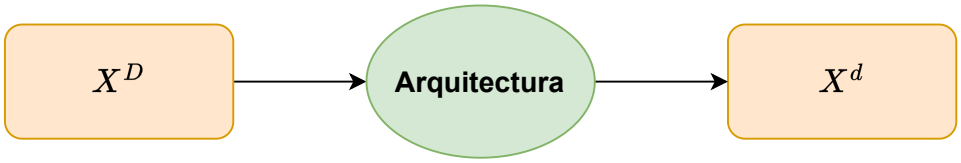


Figura 2.10: Proceso de selección de atributos para la técnica integrada con un conjunto de datos X con D atributos de entrada y d atributos de salida siendo $d < D$.

Las arquitecturas que integran la selección de atributos mediante regularización se basan normalmente en un modelo lineal con regularización lasso (L1). Este tipo de regularización fue analizada en la Sección 2.1.9, cuya principal característica era que se penalizaba el valor absoluto de los pesos de la arquitectura. Gracias a esto, los pesos de una arquitectura lineal asociados a cada uno de los atributos tienden a acercarse a cero. Por lo tanto, si el peso

asociado a un atributo se vuelve cero, significa que la propia arquitectura ha decidido que ese atributo no es relevante produciendo la selección de atributos. Normalmente la regularización añade un hiperparámetro λ que ajusta la influencia de la penalización. En el contexto de la selección de atributos, este parámetro puede usarse para relajar o reforzar la selección de atributos. De esta manera, un valor de λ alto tenderá a eliminar más atributos que uno bajo.

La arquitectura Elastic Net propuesta por Hui Zou et al. en 2005 [174] combina la regularización lasso (L1) y ridge (L2) para obtener las ventajas de ambos métodos. La regularización ridge reduce la complejidad de la arquitectura penalizando los pesos muy grandes. Sin embargo, la penalización introducida por en ridge no es capaz de asignar un valor de cero a los pesos, por lo que no se puede realizar la selección de atributos. ElasticNet aprovecha por lo tanto la selección de atributos y la reducción de complejidad usando tanto lasso como ridge. La combinación se realiza de la siguiente manera:

$$R(W) = \lambda \frac{1}{2} L1(W) + (1 - \lambda) \frac{1}{2} L2(W) \tag{2.26}$$

donde W representan los pesos implicados en la regularización. Las regularizaciones $L1(W)$ y $L2(W)$ hace referencia a las ecuaciones 2.19 y 2.20 respectivamente presentes en la Sección 2.1.9. Por último, el hiperparámetro λ ajusta el impacto de la regularización aplicada por lasso (L1) y ridge (L2).

Las arquitecturas basadas en árboles son, por naturaleza, integran una selección de atributos. El objetivo de esta arquitectura consiste construir un árbol que separa el conjunto de datos en subconjuntos con menor entropía (clasificación) o varianza (regresión). Para realizar la separación, cada nodo selecciona uno de los atributos y aplica una condición que divide el conjunto en dos o más ramas. Una explicación más detallada del funcionamiento de este tipo de arquitecturas se realizará en la Sección 4.4. La clave está en que el árbol construido no tiene porqué usar todos los atributos para separar el conjunto de datos de forma óptima, por lo que aquellas características no usadas se consideran no relevantes. Para ello, es necesario establecer una condición de parada adecuada como la la entropía o varianza mínima aceptada por cada división.

Otras arquitecturas se basan en la unión de varios árboles donde cada árbol se ha aplicado a un subconjunto de atributos distinto. A causa la selección de este subconjunto, se pueden haber seleccionado características que no serían consideradas relevantes para todo el conjunto de datos. En estos casos normalmente se aplica en primer lugar un ranking de los atributos considerando el decremento de varianza o entropía media tras aplicar la división en cada atributo. Tras ello, se establece un umbral para obtener aquellos atributos considerados relevantes al igual que el caso de la selección mediante filtro.

Todas estas técnicas son eficientes y, a la vez, eficaces y es posible relacionar la selección con la arquitectura. Sin embargo, las redes neuronales no se encuentran dentro del conjunto de arquitecturas con selección de atributos integrada. Por este motivo, es necesario usar las técnicas de *wrapper* o filtro para realizar la selección en redes neuronales.

2.4.5. Resumen

La selección de atributos es una técnica con múltiples beneficios en el área del aprendizaje automático. Normalmente, una buena selección de atributos mejora la eficacia obtenida por la arquitectura, la eficiencia del almacenamiento y procesamiento de los datos añade cierta interpretabilidad general sobre los atributos relevantes para el problema y agiliza el proceso de recolección de datos.

Los principales tipos de selección de atributos son: filtro, *wrapper* e integrado. La selección mediante filtro usa una medida de relevancia para valorar cada atributo y posteriormente realizar la selección mediante un umbral. La selección mediante *wrapper* realiza una búsqueda sobre las distintas combinaciones de atributos posibles usando la arquitectura para evaluar cada combinación. Por último, la selección integrada añade el proceso de selección de atributos dentro de la propia arquitectura. La selección integrada suele ser la preferible ya que obtiene la eficiencia de la selección mediante filtro y la eficiencia e interpretabilidad de la selección mediante *wrapper*. Sin embargo, no todas las arquitecturas son capaces de integrar la selección.

Parte II

Estado del arte

Introducción

The man who never alters his opinion is like standing water, and breeds reptiles of the mind.

William Blake, The Marriage of Heaven and Hell.

En esta parte se analizarán las principales arquitecturas usadas en la literatura para la predicción de series temporales durante los últimos diez años. Muchas de las arquitecturas desarrolladas desde hace más de diez años siguen siendo ampliamente usadas, a diferencia de otras áreas como la visión artificial o el procesamiento del lenguaje natural donde las redes neuronales se han convertido en el estándar. Arquitecturas empleadas tradicionalmente siguen presentes al igual que métodos recientemente desarrollados, es por ello por lo que el estado del arte incluye estas arquitecturas más clásicas y desarrolladas recientemente.

Con el objetivo de analizar el amplio conjunto de arquitecturas usadas para la predicción de series temporales, se han tenido en cuenta diversas propuestas publicadas en el periodo de 2012-2022. Las propuestas se han analizado y clasificado en tres tipos diferentes: algoritmos clásicos de predicción de series temporales, algoritmos clásicos de regresión y arquitecturas *deep learning*.

Los algoritmos clásicos de predicción de series temporales se analizan en el Capítulo 3, donde se detallan las familias de algoritmos que tradicionalmente fueron desarrolladas específicamente para la predicción de series temporales. Las familias de algoritmos aquí estudiadas incluyen a la familia Box-Jenking y *Exponential Smoothing*. Dentro de cada una de estas familias, se desarrollaron una gran variedad de algoritmos con el objetivo de abordar problemas de distinta naturaleza.

Los algoritmos clásicos de regresión se analizan en el Capítulo 4. Este capítulo incluye aquellos algoritmos cuya idea base no fue principalmente desarrollada para la predicción de series temporales y, posteriormente, fueron adaptadas para este tipo de problemas. Los algoritmos detallados comprenden a los modelos lineales, las máquinas vector soporte, los vecinos más cercanos, los árboles de decisión y los algoritmos basados en *ensemble*.

Las arquitecturas *deep learning* son las últimas analizadas en el Capí-

tulo 5. Este tipo de arquitecturas, al igual que los algoritmos clásicos de regresión, no fueron concebidas específicamente para la predicción de series temporales, pero fueron adaptadas posteriormente. Esta arquitectura es el principal objeto de estudio en la presente tesis, por ese motivo se realiza un análisis independiente. Se analizarán las principales familias de arquitecturas aplicadas a la predicción de series temporales, su consumo de recursos y sus limitaciones.

Capítulo 3

Algoritmos clásicos de predicción de series temporales

Good times become good memories, but bad times become good lessons.

Iroh.

Las familias de algoritmos Box-Jenking y *Exponential Smoothing* normalmente parten de una hipótesis sobre la naturaleza de los componentes de la serie temporal, definiendo una ecuación que modela dicha hipótesis. La sencillez y la interpretabilidad de este tipo de modelos hace que siga siendo usado hoy en día en múltiples aplicaciones. Además, suelen necesitar pocos recursos en la mayoría de los casos para calcular la ecuación, almacenarla o realizar predicciones (inferencia). Sin embargo, este tipo de algoritmos poseen algunos problemas para obtener una buena eficacia en casos complejos. Además, muchos de estos algoritmos están limitados a series temporales univariantes con datos numéricos. Estas limitaciones reducen la eficacia de las soluciones aportadas por los algoritmos debido a que, comúnmente, existen datos numéricos y no numéricos adicionales que influyen enormemente en el comportamiento de la serie temporal.

3.1. Box-Jenkins

3.1.1. Metodología

Para explicar esta familia de algoritmos se llevará a cabo una explicación desde los algoritmos más simples hasta los más complejos.

Los algoritmos más simples de esta familia son conocidos como *Auto Regressive* (AR) y *Moving Average* (MA). Ambos se componen de una ecuación lineal que predice el comportamiento futuro. La ecuación lineal usa la

información del pasado ponderado por su correspondiente peso. La diferencia entre estos dos métodos radica en la información del pasado que usan. Mientras que el método AR usa los eventos pasados, MA usa los errores cometidos en instantes de tiempo pasados. Ambos métodos reciben un único hiperparámetro que establece el tamaño de la ventana a pasado que tendrá en cuenta en cada método, estos hiperparámetros son p y q para el método AR y MA respectivamente.

Para ajustar los pesos de las ecuaciones en AR y MA, se usa una función de autocorrelación parcial. Esta función mide la relación entre el evento en el instante t y otro evento en un instante $t-k$ donde toda la influencia de los eventos entre los instantes $[t-1, t-k+1]$ han sido eliminados.

Los algoritmos Auto Regressive (AR) y Moving Average (MA) se combinaron dando lugar a Auto Regressive Moving Average o ARMA. La combinación se realizó mediante la suma de las ecuaciones usadas en estos dos algortimos.

Como extensión de ARMA, se creó el algoritmo más conocido de esta familia: *Auto Regressive Integrated Moving Average* o ARIMA. El nuevo componente introducido añade una transformación aplicada a la serie temporal para convertir series temporales no estacionarias en estacionarias. Para ello, la transformación aplica una diferenciación d veces. Esta diferenciación transforma la serie temporal mediante la resta de todos los eventos con respecto el anterior. De esta manera, el evento en el instante de tiempo t se resta al evento con instante de tiempo $t-1$. Este modelo fue extendido añadiendo una componente estacional (*seasonal*) dando lugar al modelo conocido como SARIMA y, posteriormente, añadiendo la posibilidad de añadir otras variables independientes (exógenas) dando lugar a SARIMAX.

3.1.2. Trabajos relacionados

Adhistya Erna Permanasari et al. [40] usan SARIMA con el objetivo de predecir casos de Malaria desde 1993 hasta 2009 en Estados Unidos, obtenida por el Centro de control de enfermedades y prevención (CDC por sus siglas en inglés). Para ello, se usó una serie temporal univariante con el objetivo de predecir tres años en el futuro. Como conclusión se obtiene que el mejor algoritmo es SARIMA con un 21,6 % de MAPE.

Qais Abdulqader [1] usa algoritmos de tipo Box-Jenkins con el objetivo de predecir el censo iraquí obtenido desde 1950 hasta 2010. La serie temporal constaba de un solo atributo (censo) y el objetivo era predecir un año en el futuro. Como conclusión, se obtuvo que el mejor modelo en general era una combinación de AR e Integrated (I) con un RMSE de 3,34.

Nari Sivanandam Arunraj et al. [6] usa SARIMAX con el objetivo de predecir la demanda de plátanos en Lower Bavaria (Alemania). La serie temporal se compone del número de ventas y algunas variables relacionadas con

la demanda de plátanos. Como conclusión se obtiene que SARIMAX obtiene una predicción lo suficientemente competente estableciendo un intervalo de confianza para las predicciones del 95 %.

Jamal Fattah et al. [42] usa ARIMA con el objetivo de predecir la demanda de alimentos en una fábrica marroquí obtenidas desde 2010 hasta 2015. Como conclusión se obtuvo que el modelo ARIMA obtiene unos resultados aceptables para diferentes métricas relacionadas con el contexto del problema.

Sima Siami-Namini et al. [132] establece una comparativa entre ARIMA y una red neuronal recurrente LSTM (ver Sección 5.1). En el estudio se usan varias series temporales univariantes del ámbito financiero y económico con el objetivo de realizar predicciones de un solo paso en el futuro y multihorizonte. Como conclusión se obtuvo que la red neuronal recurrente obtiene un 85 % de mejora con respecto ARIMA.

Peter T. Yamak et al. [165] establece una comparativa entre ARIMA y dos redes neuronales recurrentes conocidas como LSTM y GRU (ver Sección 5.1). El estudio se centra en la predicción de una serie temporal univariante compuesta de registros sobre el precio del Bitcoin. Como conclusión se obtuvo que ARIMA mejoraba a ambas redes neuronales con una mejora del 30 % con respecto GRU.

Rishi Raj Sharma et al. [130] realiza la predicción de una serie temporal no estacionaria usando una descomposición en autovalores mediante la matriz de Hankel junto a ARIMA. En una primera fase, se descompone la serie temporal en varios subcomponentes con el objetivo de reducir el efecto de la no estacionalidad. Posteriormente, cada subcomponente es modelado mediante ARIMA generando una predicción por cada uno. Finalmente, la predicción final se produce sumando cada una de las predicciones por cada subcomponente. Esta metodología fue evaluada usando series temporales univariantes de infecciones de COVID-19 en India, Estados Unidos y Brasil.

3.2. Exponential Smoothing

3.2.1. Metodología

La idea principal de este tipo de algoritmo es similar a la familia Box-Jenkins. Esta familia de métodos establece una combinación lineal de los eventos pasados ponderados, en este caso, por uno o varios parámetros.

El algoritmo más simple se conoce como simple *Exponential Smoothing* y recibe un solo parámetro α. Este parámetro establece la influencia de los eventos pasados sobre el evento futuro. Además, la influencia de los eventos se reduce exponencialmente conforme más lejanos en el tiempo se encuentran.

A raíz del algoritmo más simple se creó el algoritmo conocido como *Double Exponential Smoothing*. Este algoritmo añade un parámetro β que regula

la influencia de la tendencia sobre la predicción. Esta tendencia puede ser modelada de distintas formas, dependiendo del tipo de tendencia que presente la serie temporal. Por ejemplo, una serie temporal puede presentar un crecimiento lineal o exponencial.

Por último, para modelar el último de los componentes principales de una serie temporal se creó el *Triple Exponential Smoothing*. Este algoritmo incluye la influencia de la estacionalidad sobre las predicciones. Esta influencia, al igual que la tendencia, puede tener un comportamiento diferente dependiendo de si se trata de una estacionalidad aditiva o multiplicativa.

3.2.2. Trabajos relacionados

Baihaqi Siregar et al. [135] usa varios algoritmos basados en *Exponential Smoothing* con el objetivo de predecir la producción de aceite de palma recolectado entre 2010 y 2014. Como conclusión se obtuvo que la técnica de triple exponential smoothing con estacionalidad aditiva obtuvo los mejores resultados con un RMSE de 0,1.

Sourabh Shastri [131] aplica *Exponential Smoothing* con el objetivo de predecir el porcentaje de nacidos que recibieron el bacilo de Calmette y Guérin en India desde 1980 hasta 2014. Como conclusión, los resultados muestran resultados competitivos para los siguientes cinco años.

Fajar Sidqi et al. [133] usa *Exponential Smoothing* y *Double Exponential Smoothing* con el objetivo de predecir las ventas de un producto entre 2017 y 2018. Como conclusión *Exponential Smoothing* obtuvo los mejores resultados con un MAPE del 20 %.

Slawek Smyl [137] describe la solución ganadora para la competición conocida como M4 [122]. El objetivo de esta competición es realizar la predicción de series temporales de múltiples ámbitos y características. La solución consiste en la combinación de *Exponential Smoothing* simplificado con estacionalidad multiplicativa y una red neuronal LSTM. La combinación de ambos modelos se realizó mediante la técnica definida como *ensemble of specialist* [136].

Devon Barrow et al. [11] extiende el algoritmo *Exponential Smoothing* con el objetivo de incrementar la robustez ante outliers. Para ello, estudia los beneficios del uso de M-Estimadores [28] y propone el algoritmo conocido como *inverse boosting* (ver Sección 4.5). Los resultados demuestran la mejora del uso de *inverse boosting* comparado con el *boosting* y se observa que el uso de M-Estimadores obtiene los mejores resultados en general.

Samir K. Safi [124] combina *Exponential Smoothing*, ARIMA y redes neuronales en un solo método con el objetivo de predecir datos de infecciones producidas por el COVID-19. Las predicciones de todos los modelos fueron combinados usando el mismo peso para todos obteniendo resultados competitivos.

Capítulo 4

Algoritmos clásicos de regresión

Sic Parvis Magna.
Francis Drake.

Los algoritmos clásicos de regresión inicialmente fueron desarrollados para resolver problemas de regresión. Posteriormente, fueron adaptados a problemas de predicción de series temporales sin añadir ningún tipo de hipótesis adicional. Este tipo de técnicas siguen siendo de gran utilidad debido su sencillez y a la buena eficacia que demuestran en varios escenarios. En cuanto a la eficiencia, los algoritmos que obtienen la mejor eficacia son normalmente aquellos que requieren de un mayor coste computacional en cuanto al tiempo de entrenamiento, inferencia y consumo de memoria.

4.1. Regresión lineal

4.1.1. Metodología

La hipótesis inicial de este método es que existe una ecuación lineal de ciertas variables que es capaz de estimar un evento en el futuro. Esta aproximación es similar a la descrita en los algoritmos clásicos, pero con menos asunciones. Las variables que definen la predicción pueden ser eventos pasados del atributo que se quiere predecir o de otras variables que influyan en la predicción, siendo un método más genérico admitiendo cualquier tipo de información. Por otro lado, este tipo de métodos no introduce ningún tipo de restricción sobre los pesos de los eventos, independientemente de si son más o menos alejados en el tiempo, ni tampoco impone ningún tipo de separación entre los distintos componentes de la serie temporal. Además, el método es interpretable al poder relacionar los pesos para cada variable con

la influencia sobre la predicción.

El ajuste de los pesos que regulan la influencia de las variables de la ecuación lineal usa la minimización de los errores producidos por el método. Este proceso de entrenamiento de pesos se denomina estimación de mínimos cuadrados [47].

4.1.2. Trabajos relacionados

Goce Ristanoski et al. [115] mejora el rendimiento del modelo lineal modificando la función de error a optimizar. Para ello, en primer lugar, divide la serie temporal en varios grupos y establece como objetivo optimizar el error medio por cada grupo y la varianza. Como conclusión se obtuvo una mejora de hasta el 40 % con respecto un algoritmo lineal estándar.

Jiahan Li [89] aplica un modelo lineal junto a la regularización lasso (L1) a series temporales del ámbito macroeconómico. Gracias a ello, el método fue capaz de realizar una selección de los atributos relevantes (ver Sección 2.4.4) obteniendo resultados superiores a anteriores técnicas habitualmente usadas en el mismo ámbito.

Tingting Fang et al. [41] realiza una comparativa entre el método SARIMAX y un modelo lineal aplicado demanda del sistema de calefacción urbana en Espoo (Finlandia). Los datos incluyen variables meteorológicas y consumo de calefacción con el objetivo de predecir dos días en el futuro. Los resultados indican que el modelo lineal obtiene los mejores resultados que SARIMAX en la mayoría de los casos.

Ningkai Tang et al. [149] usa un modelo lineal con regularización lasso (L1) aplicado la predicción de generación eléctrica en paneles solares en Estados Unidos. Los resultados muestran buenos resultados comparado con propuestas anteriores siendo más robusto a las anomalías presentes en la serie temporal.

Yanpeng Zhang et al. [169] desarrolla un proceso basado en cuatro fases aplicado a series temporales fuzzy [139]. En la primera fase transforma la serie temporal fuzzy en un conjunto de series temporales. Dado que es posible que haya conjuntos con poca representación, durante la segunda fase se usa la técnica de sobremuestreo conocida como SMOTE [19]. Durante la tercera fase se realiza un clustering basado en la similitud entre las relaciones lineales de cada conjunto de series temporales. Por último, usan un modelo lineal y una red neuronal para obtener las predicciones finales.

4.2. Vecinos más cercanos

4.2.1. Metodología

Aunque no se trata de una método que represente la información contenida en los datos mediante una función a diferencia de los otros métodos propuestos, este método es de gran importancia debido a sus múltiples aplicaciones en el contexto de las series temporales.

La hipótesis principal de este método se basa en la suposición de que eventos parecidos tendrán un comportamiento parecido en el futuro. Esta aproximación se usa tanto para clasificación como para regresión, dividido en tres pasos muy simples.

En el primer paso se obtiene el evento que se desea predecir/clasificar y se calcula la distancia con respecto todos los otros eventos almacenados. Esta distancia es un hiperparámetro de esta aproximación, algunos ejemplos incluyen a la distancia euclídea, manhattan, hamming, entre otras funciones de distancia.

El segundo paso consiste en obtener los k eventos con menor distancia, es decir, los eventos (vecinos) más parecidos (cercanos) al evento que se desea predecir/clasificar. El hiperparámetro k se trata, por lo tanto, del segundo hiperparámetro de esta aproximación y suele ser el parámetro más importante y con mayor influencia.

Por último, a partir de los eventos más parecidos se obtiene la salida, cuyo cálculo de la salida depende de si nos encontramos en un problema de clasificación o regresión. Para clasificación se obtiene el voto mayoritario, es decir, si $k = 3$ la salida será la clase mayoritaria de los tres vecinos más cercanos. En el caso de la regresión, una aproximación común es obtener la media de los eventos futuros a los vecinos más cercanos.

4.2.2. Trabajos relacionados

Tao Ban et al. [10] establece un procedimiento de dos pasos usando los vecinos más cercanos y aplicándolo a la predicción de datos financieros. El primer paso consiste en obtener los k índices más parecidos, conocidos como los vecinos más cercanos de referencia, usando la correlación de Pearson es usada como medida de distancia. Posteriormente, se aplica un ventaneo con tamaño de ventana z a los k vecinos más cercanos de referencia y se obtiene la predicción basada en los m vecinos más cercanos. Los resultados muestran errores competitivos comparado con métodos como redes neuronales, AR o vecinos más cercanos.

Konstantinos I. Nikolopoulos et al. [106] usan los vecinos más cercanos aplicados a la demanda de productos esporádicos. Se usaron varios métodos específicos para la predicción de demanda en productos con largos periodos sin ventas como comparativa. Como conclusión, se obtuvo que los vecinos

cercanos presentaban resultados competitivos en casos donde se tiene la certeza de que existen patrones repetitivos en el conjunto de datos del cual los vecinos cercanos pueden beneficiarse.

Mirko Kück et al. [83] aplica los vecinos más cercanos aplicado a la competición M3 [101]. En el artículo se presentan variaciones de los vecinos más cercanos correspondientes al tercer paso principalmente. Se considera el procedimiento de los vecinos más cercanos habitual y el uso de un modelo lineal para predecir el futuro basado en la información de los k vecinos más cercanos. Además, se presentan varias regularizaciones al modelo lineal con el objetivo de mejorar su eficacia. La metodología implementada se comparó con varias aproximaciones del estado del arte como ARIMA y dos redes neuronales con distintas configuraciones. Los resultados mostraron que la metodología usando un modelo lineal con regularización de tipo ridge obtuvo los mejores resultados.

Guancen Lin et al. [94] extiende a los vecinos más cercanos creando la metodología llamada: EEMD–MKNN–TSPI. Esta metodología basa su funcionamiento en tres pasos fundamentales. El primer paso realiza una descomposición de la serie temporal mediante la técnica conocida como *Ensemble Empirical Mode Decomposition* (EEMD) [162]. El segundo paso usa el algoritmo de vecinos más cercanos multidimensional propuesto en el propio artículo para extender la funcionalidad de los vecinos cercanos obteniendo las predicciones parciales por cada una de las componentes usando el peso propuesto en el algoritmo TSPI desarrollado por Parmezan [123]. El último paso recompone las predicciones sumando todas las predicciones parciales formando la predicción final. Como conclusión, la metodología implementada en el artículo muestra unos resultados competitivos comparado con métodos similares usando la misma descomposición.

4.3. Máquinas de vectores soporte

4.3.1. Metodología

Las máquinas de vectores soporte [24] son una de las técnicas más usadas hoy en día, consiguiendo un buen equilibrio entre eficacia y eficiencia en muchos casos. Al igual que los métodos lineales, se establece una ecuación lineal que intenta ajustarse lo mejor posible a los datos usando una serie de atributos como entrada. La diferencia principal con respecto los métodos lineales es que este tipo de método no ajusta los pesos en base a minimizar la diferencia del error. En su lugar, se minimiza la norma euclidiana de los pesos sujeto a una restricción sobre el error. La restricción establece que el error no tiene por qué minimizarse para aproximarse a cero, en su lugar debe ser menor a un umbral que establece la tolerancia del método.

Un problema conocido de esta, y todos los métodos que se basan en una

ecuación lineal, es que existen problema no-lineales que impiden llegar a una solución lo suficientemente eficaz. Dentro del contexto de las máquinas de vectores soporte, para superar esta dificultad, se definió una transformación sobre los datos apodado como el "truco" del kernel. Este "truco" consiste en aumentar la dimensionalidad mediante los denominados kernels, que calculan nuevas dimensiones al aplicar operaciones matemáticas sobre las dimensiones iniciales. De manera que, tras aumentar la dimensionalidad, el problema pase de ser lineal.

4.3.2. Trabajos relacionados

Deepak Gupta et al. [54] modifica a la máquina vector soporte habitual dando lugar a las máquinas de vectores soporte gemelas [109]. Este método, usa dos hiperplanos (ecuaciones lineales) en lugar de uno, usando la técnica conocida como optimización cuadrática [5]. El método fue entrenado y evaluada en 44 conjuntos de datos de diferentes datos financieros de mercados internacionales. Los resultados mostraron una mejora en cuanto a la eficacia y eficiencia comparado con una máquina vector soporte clásica.

Sebastián Maldonado et al. [102] añade un proceso automático para la selección del tamaño de ventana modificando el comportamiento de la máquinas de vectores soporte. Para ello, usa el descenso del gradiente para eliminar aquellos intervalos de tiempo no relevantes ajustando los parámetros del kernel presentes en la máquina vector soporte. El nuevo método fue evaluado sobre cuatro conjuntos de datos cuyo objetivo es predecir la carga eléctrica en varios países. La propuesta obtiene una buena eficacia comparado con métodos como Box-Jenkins y redes neuronales. Además, al seleccionar los intervalos relevantes es posible establecer una interpretabilidad sobre el comportamiento aprendido por el modelo.

Vijander Singh et al. [134] aplica máquina vector soporte a datos de infecciones de la pandemia COVID-19. Los datos recolectados fueron desde enero hasta abril de 2020 y el objetivo era predecir el ratio de mortalidad tres semanas en el futuro. Como conclusión se obtiene que el método es adecuado para predecir este tipo de datos, obteniendo buenos resultados mediante una comparación subjetiva.

Ping Jiang et al. [72] propone un nuevo método basado en máquinas de vectores soporte. Este método es evaluado sobre datos de demanda eléctrica procedentes del mercado eléctrico de Nueva Gales del Sur y Singapur durante cuatro meses de 2014. El método divide el procedimiento en tres pasos. En el primer paso se extrae el ruido presente en la serie temporal usando la técnica conocida como *variational mode decomposition* [35]. Durante la segunda fase, la transformada rápida de Fourier (FFT por sus siglas en inglés) [16] es usada para obtener las componentes estacionales. Tras ello, durante la tercera fase, una máquina vector soporte clásica es entrenada para obtener las predicciones y una máquina vector soporte de mínimos cuadrados de

salida doble [147] es usada para obtener los intervalos con la varianza de las predicciones. Como conclusión, se obtuvo que la metodología implementada mejora considerablemente a métodos habituales como ARIMA o máquinas de vectores soporte.

Min Ding et al. [33] usa un método basado en máquinas de vectores soporte de mínimos cuadrados con kernel híbrido (HKLSSVM por sus siglas en inglés) aplicado a la predicción de la energía eléctrica. El método divide su proceso en tres fases principales. La primera fase usa una descomposición mediante el proceso denominado como *maximal wavelet decomposition* [110]. Usando la descomposición de la fase anterior, durante la segunda fase se aplica un algoritmo de *clustering* formando tres grupos distintos. La tercera, y última fase, entrena varios HKLSSVM por cada uno de los grupos. Posteriormente, todas las predicciones son reconstruidas para formar la predicción final.

4.4. Árboles de decisión

4.4.1. Metodología

Este tipo de método, también denominada árboles de clasificación y regresión (CART por sus siglas en inglés), es la más popular usada hoy en día para una gran variedad de problemas. La hipótesis inicial resulta de imitar como las personas tomamos decisiones en base a los hechos. Normalmente las personas analizamos ciertos hechos y establecemos ciertas condiciones para tomar las decisiones necesarias. Este tipo de método imita este comportamiento, representando las decisiones como un árbol. Estos árboles contienen ciertos nodos y ramas, los nodos representan las condiciones que dan lugar a las ramas, distintas posibilidades resultantes de las condiciones. De esta manera, el árbol separa los eventos moviéndolos por las distintas ramas en función de sus atributos hasta que llegan a un nodo hoja donde no hay más condiciones. Las hojas se encargan de resolver el subproblema generado tras separar el conjunto de datos. La forma que pueden tomar estas hojas es diversa, puede ser desde un valor constante hasta un método más complejo. Una variante conocida llama M5'[114] [158] se establece una ecuación lineal en las hojas para dar solución al subproblema.

En este tipo de métodos, los pesos de las usados en las hojas se ajustan de acuerdo con el proceso de optimización seleccionado. Por ejemplo, un método lineal usará el ajuste por mínimo cuadrados para cada una de las hojas. Hay que tener en cuenta que la eficiencia se degrada conforme aumenta el número de hojas y, en mayor medida, con el tipo de método usado en las hojas. A más hojas, más parámetros son necesarios optimizar y más almacenamiento debe usarse para contenerlos. Este hecho se vuelve especialmente poco eficiente cuando los métodos usados en las horas son muy complejas,

por lo que normalmente se opta por usar métodos simples.

Se conoce como se optimizan las hojas, pero ¿cómo se eligen qué condiciones usar en cada uno de los nodos del árbol? El objetivo del árbol es separar los eventos usando condiciones de forma que faciliten la toma de decisiones. Para ello, se escoge un atributo del evento, se establece un umbral para dividir los casos y se evalúa la calidad de esa división. Para escoger un atributo y establecer un umbral se realiza mediante una búsqueda aleatoria que prueba una serie de umbrales por cada atributo dentro del rango de valores posible. A la hora de evaluar el umbral aplicado a un atributo se usa una métrica como, por ejemplo, la varianza o el error cuadrático medio del método ajustado a ese subproblema. A más compleja a la forma de evaluar más ineficiente se vuelve la optimización de las condiciones.

Sin embargo, es necesario tener en cuenta la generalidad de este tipo de métodos. Un ajuste perfecto de los datos posible es separar todos los eventos del conjunto de datos de forma independiente en cada hoja. De esta forma, la métrica usada para evaluar las hojas sería mínima en todas, sin embargo, la capacidad de generalización sería nula al ajustarse demasiado a los datos. Para evitar estos escenarios, una aproximación es limitar el crecimiento del árbol de alguna mediante, por ejemplo, limitar la profundidad (el número de condiciones anidadas) máxima, limitar el número mínimo de eventos en una hoja, entre otras limitaciones.

4.4.2. Trabajos relacionados

Alicia Troncoso et al. [153] usa ocho tipos diferentes de árboles con distintos métodos en las hojas, aplicados a la predicción de la velocidad del viento en Guadalajara, España. Los resultados muestran que los árboles cuyas hojas se basan en funciones no lineales basadas en vecinos más cercanos obtienen mejores resultados comparado con otros algoritmos como: redes neuronales, máquinas de vectores soporte, redes bayesianas, entre otros algoritmos. Además, las métodos basados en árboles mostraban una eficiencia mayor que la mayoría de las usados en la comparativa.

Bahram Choubin et al. [22] compara árboles de decisión con ARIMA y *adaptive neuro-fuzzy inference system* (ANFIS) [71] aplicado a la predicción de precipitaciones tomadas en Kerman, Irán. Los resultados muestran que los árboles de decisión muestran mejores resultados en general que las otras propuestas.

Snezhana Georgieva Gocheva-Ilieva et al. [51] usa árboles de decisión para predecir el contaminante PM_10 en Ruse y Pernikm, Bulgaria. El método fue comparado con ARIMA obteniendo mejores resultados en general.

Mukesh K. Tiwari et al. [74] realiza una comparación entre M5, *extreme learning machines* [66] y redes neuronales aplicado a la predicción de inundaciones para 1, 5 y 10 horas en el futuro. Los resultados muestran una

efectividad competitiva por parte de M5 sobre todo para los horizontes de predicción de 1 y 5 horas.

4.5. Métodos de ensemble

4.5.1. Metodología

Este tipo de técnica se basa en la hipótesis de la "sabiduría de las masas" (*wisdom of crowds*) [146] y la navaja de Ockham . La hipótesis de la sabiduría de las masas establece que la unión de las estimaciones realizadas por varias personas obtiene mejor eficacia normalmente comparado con la estimación de una sola persona. La hipótesis de la navaja de Ockham establece que, a igual de condiciones, la solución más simple es la mejor debido a que soluciones más complejas suelen tener en cuenta más sesgos afectando a la capacidad de generalización. Con estas dos hipótesis en mente, los *ensemble* combinan varios métodos (normalmente simples) para obtener una mejor eficacia. La eficacia depende del número de métodos, lo que afecta en gran medida a la eficiencia ya que el coste computacional aumenta.

Dentro de este tipo de técnica existen distintas formas de implementar la combinación de los distintos métodos, existen cuatro principales aproximaciones llamadas: *voting*, *bagging*, *stacking* y *boosting*.

La aproximación *voting* une las predicciones de los distintos métodos estableciendo como la predicción final aquella cuyo valor se repita más en los distintos métodos entrenados con el mismo conjunto de datos. Por ejemplo, si tres métodos de dos obtienen que mañana estará soleado y el tercer método predice que estará nublado, la predicción final será soleado simulando una votación. En caso de tratarse de un problema de regresión, la salida puede ser considerada como la media o mediana de las predicciones obtenidas por cada una de los métodos.

La aproximación *bagging* es similar al *voting* pero donde los métodos no se han entrenado con el mismo conjunto de datos, si no de un subconjunto. De esta manera, cada modelo parte de una premisa diferente con sus propios sesgos y a la hora de realizar la predicción se suele mejorar la generalidad.

La aproximación *stacking*, al igual que *voting*, parte de varios métodos entrenados con el mismo conjunto de datos. Sin embargo, en lugar de votar la predicción final, se usa un método adicional que recoge todas las predicciones de los distintos métodos como entrada y realiza la predicción final. Este método adicional permite ajustar la importancia de cada modelo de forma automática.

La aproximación *boosting* parte de una hipótesis adicional que permite colaborar entre los distintos métodos del *ensemble* cubriendo los errores las unas de las otras. Para permitir esta colaboración se establece un peso por cada evento en el conjunto de datos que marca la importancia de cada even-

to. De forma secuencial, se entrenan una serie métodos que tienen en cuenta la importancia de los eventos a la hora de evaluarlos y se actualizan los pesos en base al error cometido de forma que los eventos que mejor se han estimados tienen un peso menor. De esta forma, el siguiente método dará más importancia a los eventos que las no hayan podido estimarse correctamente anteriormente. Dentro de este tipo de técnica se han creado varias aproximaciones basadas principalmente en el uso de árboles de decisión, estos métodos son: XGBoost [20], LightGBM [77] y CatBoost [112].

4.5.2. Trabajos relacionados

Este tipo de técnica es una de las más usados debido a la eficacia que presentan en múltiples escenarios para la predicción de series temporales.

Igor Ilic et al. [68] proponen una versión interpretable de la aproximación de *ensemble* basado en *boosting*. Para ello, proponen un método iterativo en el que se comienza con un método base entrenado de forma habitual. Tras ello, se entrena un árbol de regresión sobre los errores cometidos por el método base. Por último, se forma un vector binario indicando el conjunto de condiciones que llevan a la hoja de la hoja con mayor error, se añaden esos atributos al conjunto inicial y se repite el mismo proceso hasta unas iteraciones máximas. Gracias a la generación de atributos, es posible conocer aquellas hojas que presentan un comportamiento distinto al conjunto de datos incrementando la interpretabilidad. La propuesta fue comparada con varias aproximaciones basadas en *ensemble* y ARIMAX sobre conjuntos de datos reales y sintéticos. Se obtuvieron resultados competitivos comparados con otra aproximación de interpretabilidad.

Mikel Larrea et al. [87] usan un la aproximación voting usando máquinas de aprendizaje automático extremo [66] como método base. La propuesta establece un funcionamiento habitual para el entrenamiento de las distintos métodos que compondrán el *ensemble* y, posteriormente, para realizar la predicción final se usa una media ponderada. Los pesos de la media ponderada se establecen mediante el uso de optimización por enjambre de partículas (PSO por sus siglas en inglés). La propuesta fue evaluada usando un conjunto de demanda eléctrica comparada con aproximaciones más simples para realizar la predicción final como usar la media o mediana. Los resultados demuestran una considerable mejora de efectividad usando la propuesta.

Pyae-Pyae Phyo et al. [111] realiza una comparativa entre 17 aproximaciones basadas en *ensemble* y propone una combinación de las cinco mejores aproximaciones en un solo método usando *voting*. La propuesta fue evaluada usando un conjunto de datos de consumo eléctrico y comparada con las aproximaciones de *ensemble* individuales y ARIMA. Los resultados obtenidos, muestran una mejora de efectividad considerable en general.

Donghwan Kim et al. [79] usa la aproximación *stacking* y *bagging* modifi-

cado sobre una serie de redes neuronales aplicado a varios conjuntos de datos de series temporales univariantes. El procedimiento de *bagging* se realiza mediante la generación de diferentes versiones de la serie temporal usando *maximum overlap discrete wavelet transform* (MODWT) [31]. Posteriormente, cada una de las diferentes versiones es usada para entrenar una red neuronal independiente. Por último, un proceso de *stacking* habitual es usado sobre las predicciones de las distintas redes neuronales entrenadas. La propuesta fue comparada con varias aproximaciones basadas en *ensemble*, regresión y Box-Jenkins. Los resultados mostraron que la eficacia del método propuesto superaba a las otros métodos en 14 de los 20 conjuntos de datos usados.

Capítulo 5

Arquitecturas *deep learning*

Si actúas como si supieras lo que estás haciendo, puedes hacer lo que quieras.

Frida Kahlo.

Este tipo de arquitecturas es la base en la que se centrará la tesis. A diferencia de los algoritmos presentados anteriormente, estos métodos permiten modelar series temporales que presentan comportamientos no lineales. Además se imponen menos sesgos sobre la arquitectura, dando más libertad a aprender transformaciones que otras arquitecturas no podrían por limitaciones en el propio diseño. Además, esta libertad permite reducir el preprocesamiento necesario sobre los datos conocido como "ingeniería de características" (*feature engineering*), encargado de transformar los datos obteniendo más características relevantes que permitan a las arquitecturas alcanzar una mejor eficacia. Sin embargo, delegar parte de responsabilidad sobre la arquitectura tiene un coste computacional asociado que es necesario asumir. Existen una gran variedad de arquitecturas que se han aplicado a la predicción de series temporales, usadas en varias aplicaciones de forma independiente o junto a algoritmos clásicos/multipropósito.

5.1. Redes recurrentes

5.1.1. Metodología

Las redes *Long Short Term Memory* o LSTM, son una las redes recurrentes más populares usadas aplicadas a series temporales. Su funcionamiento se basa en las capas con el mismo nombre, desarrolladas en 1997 por Sepp Hochreiter y Jürgen Schmidhuber [62] con el objetivo de solucionar el problema con la dependencia temporal entre los diferentes eventos a largo plazo. Este problema originaba un problema en la eficacia en las redes neuronales

que usaban capas recurrentes (Sección 2.1.3) cuando la predicción dependía de eventos que se encontraban muy alejados del tiempo debido al problema conocido como desvanecimiento de gradientes (Sección 2.1.7).

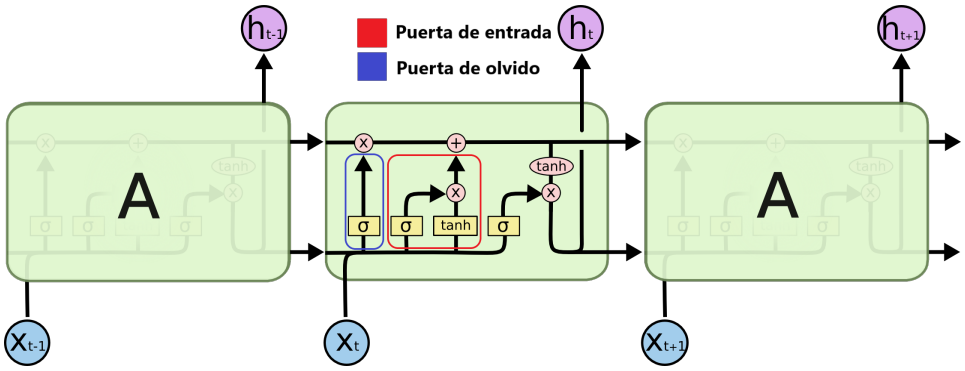


Figura 5.1: Estructura de la LSTM.
Fuente: Christopher Olah

La idea principal de este tipo de red es que filtra la información de todos los eventos en la ventana de entrada eliminando los eventos que tienen poca relevancia. Para ello, establece una serie de "puertas" que hacen de filtros de la codificación de los eventos en el *cell state* obteniendo como salida el *hidden state*.

El *cell state* se trata de un resumen de todos los datos que han pasado por la red, y que sirve para obtener un contexto general de la información filtrado por la puerta de olvido y la puerta de entrada. Por un lado, la puerta de olvido calcula la importancia a partir los eventos de entrada (x_t) y el *hidden state* (h_t) del instante anterior estableciendo una ponderación donde cero implica que no tiene relevancia y uno implica que tiene mucha relevancia. Por otro lado, la puerta de entrada decide qué información nueva proveniente del evento de entrada y del *hidden state* anterior se añade al *cell state*.

El *hidden state* obtiene una representación de los datos más recientes que sirve para dar un contexto más corto en el tiempo, a su vez, el *hidden state* servirá de entrada a la red neuronal en el siguiente instante de tiempo junto a los nuevos eventos. A partir de la LSTM surgieron varias modificaciones que intentaban mejorar este tipo de redes. Entre ellas, una de las más famosas fue la *Gated Recurrent Unit* o GRU desarrollada en 2014 por Kyunghyun Cho et al. [21]. Este tipo de red simplificaba la LSTM uniendo la puerta de olvido y de entrada en una mejorando la eficiencia.

5.1.2. Trabajos relacionados

Una extensión de esta arquitectura llamada DeepAR fue propuesta en 2020 por David Salinas et al. [126]. Esta arquitectura, aborda la predicción

de series temporales de una forma distinta a la nombrada anteriormente. En lugar de realizar predecir el comportamiento futuro con valores numéricos, establece una distribución con un rango donde los posibles valores futuros se pueden encontrar. La arquitectura usa las capas LSTM como base para ello, con la inclusión de un modelo probabilístico que determina los parámetros de la distribución del ruido de la predicción (Sección 2.2.3). Este modelo es dependiente de la naturaleza de los datos, en el artículo original proponen dos modelos probabilísticos: Gaussiano para valores reales o Binomial Negativa para valores positivos naturales. Los parámetros del modelo probabilístico son entrenados junto a las capas LSTM obteniendo la distribución de los valores futuros.

Renato R. Maaliw et al. [99] realiza una comparativa entre varias capas LSTM anidadas frente a *Exponential Smoothing* y ARIMA. Los tres modelos fueron evaluados con el conjunto de datos de COVID-19 para un horizonte de predicción de 15 días. Los resultados mostraron que el uso de varias capas LSTM anidadas obtiene mejores resultados que als otras arquitecturas comparadas.

Benjamin Lindemann et al. [95] realiza un estudio de distintas variaciones de la red LSTM aplicado a distintos escenarios del área de la predicción de las series temporales. Las variaciones de las redes estudiadas incluyen: LSTM bidireccionales [125], LSTM jerárquicas [157] [144], LSTM de rejilla [76], entre otras variaciones. Los escenarios usados para analizar las distintas arquitecturas incluyen relaciones no-lineales en el conjunto de datos, dependencias a corto y largo plazo, series temporales multivariantes, entre otros. Los resultados mostraron que la arquitectura Seq2Seq LSTM parcialmente condicionada [70] obtiene los mejores resultados para todos los escenarios presentados.

Xuechen Li et al. [91] aplican una arquitectura basada en GRU donde la optimización de hiperparámetros se realiza mediante el *Sparrow Search Algorithm* (SSA) [163] aplicado a la producción proveniente de datos de exploración de yacimientos petrolíferos. La arquitectura usa la arquitectura GRU bidireccional. Esta arquitectura procesa la ventana de entrada en el sentido habitual, de eventos más antiguos a más recientes, y también en sentido contrario, de eventos más recientes a más antiguos. Esto permite obtener una representación de la serie temporal en ambos sentidos, mejorando en ocasiones la capacidad predictiva de la arquitectura. La arquitectura se comparó con otras arquitecturas recurrentes y Box-Jenkins obteniendo una mejora significativa.

5.2. Redes convolucionales

5.2.1. Metodología

Las redes convolucionales son uno de los tipos de red neuronal más común, principalmente por su aplicación a la visión artificial. Estas redes se basan en el uso de las capas convolucionales explicadas en la Sección 2.1.3. A través de estas capas son capaces de transformar los datos extrayendo características de estos de forma jerárquica, donde las características más generales se encuentran en las primeras capas y las más específicas en las últimas.

Debido a su popularidad, en 2015 Serkan Kiranyaz et al. desarrollaron la capa convolucional unidimensional [81]. Esta capa se propuso como solución para la clasificación de electrocardiogramas.

El funcionamiento de una capa convolucional unidimensional es similar a la original, la diferencia es que la convolución se aplica a una sola dimensión. En el caso de las series temporales, la dimensión sobre la que los filtros se desplazan realizando las convoluciones es la temporal.

5.2.2. Trabajos relacionados

Este tipo de redes también ha sido usada en conjunto a otras arquitecturas, como LSTM o GRU [98], debido a su diversidad y eficiencia obteniendo grandes resultados. La idea principal es unir la capacidad de extraer características temporales de las redes convolucionales con la capacidad de filtrar los eventos de las capas recurrentes.

La arquitectura *Temporal Convolutional Network* o TCN, fue desarrollada en 2018 por Shaojie Bai et al. [8]. Esta arquitectura introduce el concepto de una mejora de las convoluciones unidimensionales orientadas a la predicción de series temporales llamada: convoluciones causales.

La convolución causal establece que para una predicción en un instante de tiempo t solo los eventos anteriores a t deben considerarse. En la convolución unidimiensional, la salida de una capa se obtiene aplicando un filtro sobre los eventos de cada instante de tiempo $1 \cdots t$. Si el filtro tiene tamaño 3, para calcular la salida en el primer instante de tiempo se aplicará el filtro sobre los instantes $1, 2, 3$. Es decir, para obtener la salida de un instante t se considera información posterior. Para establecer una causalidad, se realiza un "relleno" sobre las entradas de las capas convolucionales a la izquierda. De esta manera, el filtro siempre tendrá en cuenta datos del pasado. Por ejemplo, si se rellenaran los eventos de entrada con un valor de cero representando instantes de tiempo anteriores al de la entrada en el caso anterior, el filtro para el primer instante de tiempo obtendría como entrada los eventos $-1, 0, 1$, siendo los instantes -1 y 0 relleno. La Figura 5.2 muestra el campo receptivo de la neurona $O_{2,t}$ en el instante t en la segunda capa de profun-

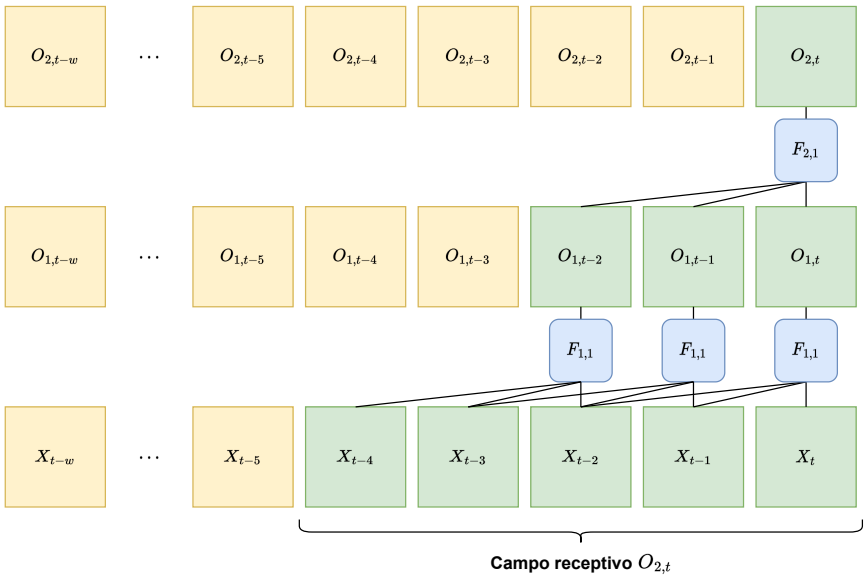


Figura 5.2: Campo receptivo para una neurona la segunda capa $O_{2,t}$.

didad sobre las entradas $X_{t-w}, \cdots, X_t$ para un tamaño de ventana w. Los filtros usados para computar las salidas de cada capa se denotan como $F_{l,i}$. Donde $l \in [1, L]$ representa la capa de las L en la red, e $i \in [1, P]$ representa cada uno de los P filtros.

En esta red se aborda también un problema de eficiencia que surge para redes con una ventana de entrada grande. El problema ocurre porque, para obtener información de todas las entradas, las redes convolucionales deben aumentar el número de capas. Por ejemplo, la salida de la primera capa convolucional en el momento t solo tendrá en cuenta los eventos en los instantes $t-2, t-1$ y t, los eventos de entrada abarcados por una salida en las capas convolucionales se conoce como campo receptivo. Si se añaden más capas, se aumenta el campo receptivo al conectarse con la salida de filtros de instantes anteriores. Sin embargo, cuando el tamaño de entrada es grande, el número de capas necesarias aumenta linealmente produciendo redes demasiado profundas. Como solución se propone "dilatar" los filtros considerando como entrada no los instantes de tiempo correlativos, si no aquellos instantes de tiempos cada d instantes. De esta manera, se aumenta el campo receptivo muestreando los eventos de entrada. La Figura 5.3 muestra el campo receptivo dilatado de la neurona $O_{1,t}$ en el instante t en la primera capa de profundidad sobre las entradas $X_{t-w}, \cdots, X_t$ para un tamaño de ventana w.

Modelos más recientes como el propuesto por Minhao Liu et. al. [97], obtienen resultados del estado del arte basándose en el uso de capas convolucionales que extraen características de la serie temporal a distinta frecuencia de muestreo. Esto permite extraer una serie de características con gran di-

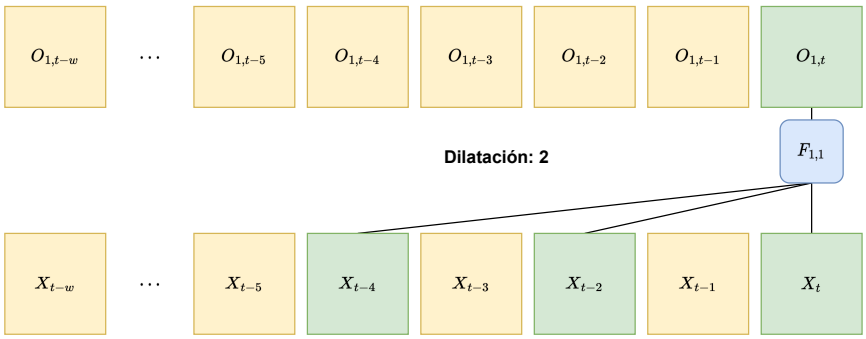


Figura 5.3: Campo receptivo dilatado para una salida $O_{1,t}$.

versidad, que mejora considerablemente la efectividad de la arquitectura en varias aplicaciones. La arquitectura usa las conexiones residuales propuestas por He et. al. [75] y Huang et. al. [67] obteniendo unos resultados del estado del arte en varios conjuntos de datos.

5.3. Transformers

5.3.1. Metodología

Tras la aparición de la arquitectura Transformer presentada por Minhao Liu et al. [156] en 2017, se introdujo un gran avance en el área del procesamiento del lenguaje natural. Debido a las similitudes entre el procesamiento del lenguaje natural y las series temporales en ciertas aplicaciones como el texto predictivo, se estudiaron diversas arquitecturas aplicadas a la predicción de series temporales.

El componente principal de los Transformers es el mecanismo conocido como Attention, este mecanismo ha sido el principal motivo del éxito que ha tenido este tipo de métodos. Normalmente, el cerebro no procesa todos los estímulos que recibe, el cerebro puede filtrar estos estímulos prestando atención solo a los importantes [105]. El mecanismo Attention, intenta imitar el comportamiento del cerebro y establece una ponderación que mide el grado de importancia de cada instante de tiempo. Este mecanismo recibe tres entradas conocidas como: *Query*, *Key* y *Value*. La idea intuitiva de estos valores está relacionada con las tecnologías de recuperación de información donde a partir de una consulta (*Query*), se realiza la comparación con la información asociada a los elementos a consultar (*Key*) estableciendo una puntuación para cada elemento (*Value*). En el caso específico de las series temporales estas tres entradas son la misma, lo cual permite a la secuencia de entrada atenderse a sí misma dando lugar a lo que se conoce como Self-Attention. Siguiendo con la idea de un sistema de recuperación de información, cada evento de la ventana tendría asociada una puntuación en base

a la consulta realizada con cada uno de los eventos de la misma ventana de entrada. Formalmente, este mecanismo se formula como:

$$Attention(Q, K, V) = softmax\left(\frac{Q \times K^T}{\sqrt{D_K}}\right) \times V\,, \tag{5.1}$$

donde Q, Ky V representan las entradas *Query*, *Key* y *Value* respectivamente, $softmax$ representan la función de activación Softmax y D representa el número de características de K. Hay que tener en cuenta que Q, K y V no tienen por qué ser los eventos de entrada. Es común establecer una transformación específica para cada entrada que permita representar las entradas para aumentar su expresividad. Por este motivo, es normal tener asociado unos pesos W_Q, W_K y W_V para cada entrada dando lugar a Q, K y V respectivamente.

En el artículo original se propone una extensión del mecanismo de Attention conocido como MultiHeadAttention. A partir de las características obtenidas por la red al propagar la información a través de las capas, este mecanismo divide las características en varios subespacios y calcula aplica el mecanismo de atención a cada subespacio. De esta manera, cada subespacio es procesado por una cabeza que se centra en atender a varios aspectos de forma paralela en lugar de intentar atender a todo.

5.3.2. Trabajos relacionados

Esta propuesta sirvió como inspiración originando diversas arquitecturas de predicción de series temporales.

El método propuesto por B. Lim et al. en 2021 apodado como *Temporal Fusion Transformer* (TFT) [92], introduce el mecanismo de Attention como uno de sus componentes esenciales. Este método se aplica a problemas de predicción multihorizonte de percentiles, por lo tanto, en lugar de realizar la predicción de los eventos futuros, realiza una estimación de los percentiles para generar un rango en el que la predicción se encontrará con una cierta confianza. La entrada se compone de una ventana de eventos pasados, una ventana de eventos futuros conocidos y valores estáticos. Los eventos futuros conocidos incluyen información temporal como la fecha y hora de los eventos futuros que se van a predecir, mientras que los valores estáticos incluyen información que no depende del tiempo como la localización, tipo de tienda, zona horaria, entre otros. Los componentes principales de esta arquitectura incluyen: un componente para seleccionar las características importantes, capas LSTM para procesar la información, un componente para permitir ignorar información de capas anteriores en caso de ser necesario y el mecanismo de MultiHeadAttention para establecer la importancia de cada intervalo de tiempo.

Existen varios problemas conocidos del mecanismo Self-Attention que

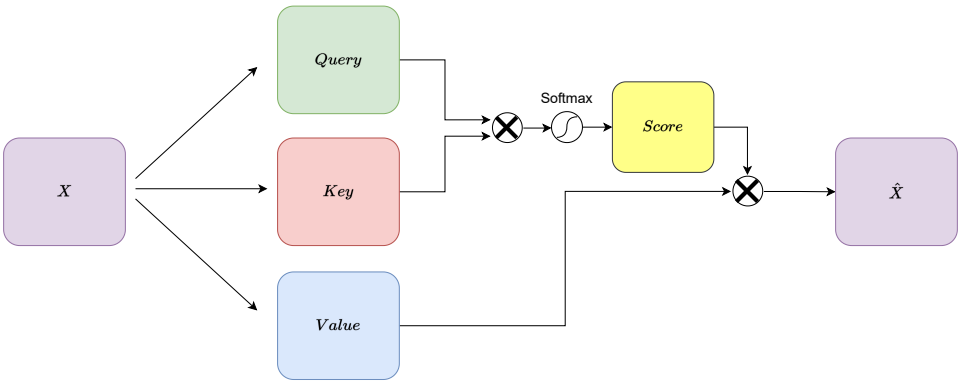


Figura 5.4: Mecanismo Self-Attention aplicado a una entrada X.

dificultan su aplicación a series temporales. El mecanismo de Self-Attention necesita de realizar el cálculo cada par de eventos, este hecho implica que el tiempo de ejecución y almacenamiento incrementa cuadráticamente con el tamaño de la ventana. Como solución a este problema, se han propuesto varias arquitecturas que solucionan este problema. La mayoría de estas soluciones se basan en el hecho de que es posible obtener la misma eficacia atendiendo sin necesidad de calcular la atención por cada par. En su lugar, se seleccionan un número limitado de pares basados en alguna heurística dando lugar a lo que se conoce como una atención dispersa (sparse). Dentro de esta línea, se encuentran los trabajos aplicados a la predicción de series temporales: LogSparse [90] propuesto por Shiyang Li et al., Informer [171] propuesto por Haoyi Zhou et al., Informer conocida como QuerySelector propuesto por Jacek Klimek et al. como una extensión de Informer, entre otros trabajos.

5.4. NBEATS

5.4.1. Metodología

Neural Basis Expansion Analysis for interpretable Time Series forecasting o también conocido como NBEATS, es una arquitectura basada en capas totalmente conectadas desarrollada en 2020 por Attilio Sbranaet al. [107]. Esta arquitectura introduce el mecanismo llamado "Doubly Residual Stacking" basado en las conexiones residuales propuestas por He et. al. [75] y Huang et. al. [67].

La idea intuitiva de esta arquitectura es que se divida el problema en diferentes subproblemas que colaboren para obtener una mejor eficacia. Los bloques *Doubly Residual Stacking* predice dos tipos de salidas llamados *backcast* y *forecast*. El *backcast* producido por el modelo transforma los eventos de entrada reduciendo su complejidad restando el *backcast* a la entrada. El resultado de la transformación es un residuo que servirá de entrada al siguien-

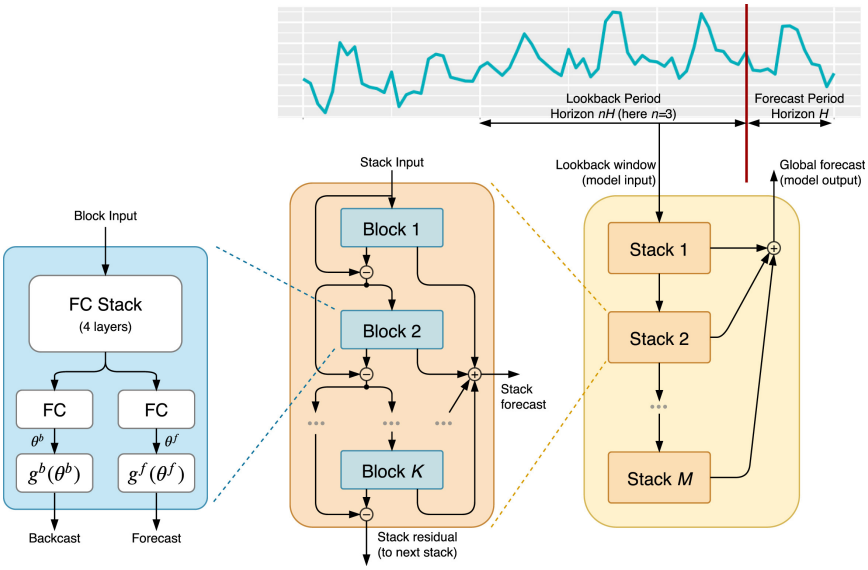


Figura 5.5: Diagrama de la arquitectura propuesta en NBEATS. *Fuente [128].*

te bloque, de manera que cada bloque se centra en un aspecto diferente con diferente complejidad. Cada bloque obtiene una predicción parcial llamada *forecast* que finalmente se suman para obtener la salida final.

En el artículo original establecen dos maneras de diseñar la arquitectura. En un diseño se impone un sesgo sobre el *backcast* producido por los bloques que descompone la serie temporal en tendencia y estacionalidad. Para ello, se modeló la tendencia mediante el uso de una función que varia lentamente a lo largo de la predicción y la estacionalidad se restringió para pertenecer a una función periódica. Este diseño tiene como objetivo mejorar la interpretabilidad del modelo al conocer la tarea correspondiente a cada bloque de la red neuronal.

5.4.2. Trabajos relacionados

Attilio Sbrana et al. [128] extienden a NBEATS añadiendo una red recurrente que recibe todos los residuales generados por cada uno de los niveles y los combina para generar un único residual final que es añadido a la predicción como una salida más. Esta arquitectura fue evaluada sobre el mismo conjunto de datos que NBEATS, obteniendo unos resultados similares pero con unos tiempos de entrenamiento nueve veces más rápido que NBEATS.

Emma Stevenson et al. [143] aplica la arquitectura NBEATS a la predicción de la radiación solar. Los predicción se realizó con un tamaño del horizonte de 27 días en el futuro. La arquitectura se comparó con varios métodos de predicción estadísticos usados comúnmente en aplicaciones simi-

lares. Los resultados muestran que NBEATS mejora considerable a todas las arquitecturas usadas dentro de la comparación.

Jente Van Belle et al. [155] mejoran la arquitectura NBEATS añadiendo al error una métrica de inestabilidad. Esta métrica mide la diferencia entre predicciones en un mismo intervalo de tiempo producida por la disponibilidad de nuevos datos. Los resultados fueron probados en los conjuntos de datos M3 [101] y M4 [122] comparado con el modelo original, obteniendo una mejora significativa de los resultados en cuanto a la estabilidad de las predicciones.

Parte III

Propuestas

Introducción

Nullius addictus iurare in verba magistri, quo me cumque rapit tempestas, deferor hospes.

Horacio.

Esta parte describe las contribuciones realizadas durante la tesis dividida en 4 capítulos.

El Capítulo 6 describe la propuesta nombrada como Hierarchical Learning Network o HLNet. La arquitectura desarrollada reduce el tiempo de convergencia introduciendo un proceso de aprendizaje jerárquico similar al proceso de aprendizaje humano. Gracias a esto, tras la evaluación sobre diversos conjuntos de datos se comprobó que se mejoraba la eficiencia de la arquitectura LSTM sin degradar la eficiencia. Las publicaciones relacionadas con esta propuesta son:

- M. J. Jiménez-Navarro, F. Martínez-Álvarez, A. Troncoso y G. Asencio-Cortés. HLNet: A Novel Hierarchical Deep Neural Network for Time Series Forecasting. In proceedings of 16th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2021), Advances in Intelligent Systems and Computing, vol 1401, pp 717–727, 2021.
- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez, A. Troncoso y G. Asencio-Cortés. HLNet: A Novel Hierarchical Deep Neural Network for Time Series Forecasting. Logic Journal of the IGPL, 2022 (aceptado).

El Capítulo 7 presenta la propuesta nombrada como Propagated HIerarchical Learning Network o PHILNet. Esta arquitectura nace como una mejora de HLNet al permitir los diferentes niveles en la jerarquía intercambien información. La arquitectura fue evaluada sobre los mismos conjuntos de datos que HLNet. La publicación relacionada con la propuesta es:

- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. PHILNet: A Novel Efficient Approach for Time

Series Forecasting using Deep Learning. Information Sciences, 2022 (en proceso de revisión).

El Capítulo 8 muestra la propuesta nombrada como Smooth Residual Connection Network o SRCNet. La arquitectura descompone la secuencia de entrada en diferentes componentes introducidos en sendas capas independientes que colaboran para formar la predicción. La publicación relacionada con la propuesta es:

- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. A New Deep Learning Architecture with Inductive Bias Balance for Oil Temperature Forecasting. Journal of Big Data, 2022 (en proceso de revisión).

El Capítulo 9 desarrolla la propuesta nombrada como Feature-Aware Drop Layer o FADL. Esta propuesta introduce una nueva capa que integra la selección de atributos sin necesidad de reentrenar la red neuronal. La propuesta fue evaluada sobre diversos conjuntos de datos de clasificación y regresión obteniendo una eficiencia competitiva con una eficiencia superior. La publicación relacionada con la propuesta es:

- M. J. Jiménez-Navarro, M. Martínez-Ballesteros, F. Martínez-Álvarez y G. Asencio-Cortés. Feature-Aware Drop Layer (FADL): A Nonparametric Neural Network Layer for Feature Selection. In proceedings of 17th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2022), Advances in Intelligent Systems and Computing, 2022.

Capítulo 6

Hierarchical Learning Network (HLNet)

La persona que se levanta es aún más grande que la que no ha caído.

Concepción Arenal.

6.1. Resumen

Esta sección detalla la propuesta llamada HLNet. Esta arquitectura permite agilizar el tiempo de convergencia durante el entrenamiento gracias a la introducción de un sesgo inductivo dentro del diseño de la arquitectura. El diseño incrementa la dificultad de la tarea original permitiendo adquirir conocimiento de forma más rápida. Un proceso de suavizado se ha realizado para reducir la complejidad del problema original, debido a que reduce el ruido presente en los eventos dentro de la ventana. A cada uno de los niveles en los que se divide la arquitectura se le asigna la ventana con un suavizado diferente. A más profunda la arquitectura, menor suavizado se aplica, incrementando la dificultad al permitir más ruido. Los resultados demuestran que existe una considerable mejora de eficiencia sin degradar la eficacia comparado con una red LSTM en la mayoría de conjuntos de datos probados.

6.2. Introducción

Esta sección describe la propuesta de arquitectura aplicada principalmente a la predicción de series temporales univariantes multihorizonte conocida como *Hierarchical Learning Network*, de aquí en adelante HLNet. El nombre hace referencia a la forma de aprendizaje que realiza esta arquitectura, realizando un aprendizaje siguiendo una jerarquía establecida en la red. Para ello, se establece una serie de tareas adicionales relacionadas con la tarea de

predicción original a cada capa dentro de la arquitectura. Cada una de estas capas tendrá una dificultad asociada diferente, de manera que las primeras capas tendrán menos dificultad que las capas siguientes.

En áreas como la visión artificial, se ha analizado el comportamiento aprendido por las capas convolucionales que permiten obtener tan buenos resultados. Algunos estudios [166] apuntan a que los filtros dentro de las capas convolucionales presentan un comportamiento jerárquico incremental. Los filtros pertenecientes a las primeras capas aplican transformaciones que obtienen características más generales como líneas laterales, gradientes, esquinas, entre otras. A medida que la profundidad de la red aumenta, los filtros se vuelven más específicos con el objetivo de obtener características como formas de cara, mano, objetos, entre otras. Es decir, las redes convolucionales organizan jerárquicamente las capas para aumentar la complejidad de las características obtenidas. Este comportamiento no ha sido estudiado en el caso de la predicción de series temporales debido a las diferencias existentes entre la clasificación y regresión.

El hecho de establecer una jerarquía de aprendizaje ordenada es común en las personas. Normalmente, cuando aprendemos nuevos conceptos o habilidades no empezamos resolviendo el problema más complejo. En su lugar, establecemos un aprendizaje incremental en el que la información de las tareas más sencillas sirve de apoyo para conseguir resolver tareas más complejas. Un ejemplo puede ser el de aprender a conducir una motocicleta. Normalmente una persona no comenzaría aprendiendo a montar en moto directamente o, si lo hiciera, le costaría más que una persona que ha adquirido unos conocimientos a través de una tarea similar. Por ejemplo, una persona que sepa montar en bicicleta podría aprovechar conocimientos y habilidades como saber cómo pulsar los frenos, sentido del equilibrio, señales, entre otras habilidades. Estos conocimientos agilizan el proceso de aprendizaje de tareas similares más complejas como montar en motocicleta al ahorrar conocimientos y establecer tareas más sencillas que permitan comprender las tareas más complejas. Este proceso de transferencia de conocimiento de distintas tareas, también es estudiado en el área del aprendizaje automático conocida como *transfer learning* [173].

En la Sección 6.3 se definirá la hipótesis fundamental de la arquitectura y sus implicaciones. En la Sección 6.4 se definirá formalmente la arquitectura y sus componente principales. En la Sección 6.5 se presentan y discuten los resultados en cuanto a la eficacia y eficiencia de la arquitectura. En la Sección 6.7 se establecen las principales conclusiones obtenidas.

6.3. Hipótesis

En esta sección definimos las hipótesis fundamentales usadas a partir de las ideas intuitivas para definir posteriormente la arquitectura.

Hipótesis 1 (H1): *Es posible acelerar el proceso de aprendizaje mediante la asignación de tareas relacionadas que aumenten la dificultad al aumentar la profundidad de la red neuronal.*

La Hipótesis 1 es causado por el hecho de aprender tareas más sencillas normalmente toman menos tiempo para converger, aprendiendo las características necesarias para resolver su tarea más rápidamente. Debido a que las tareas más simples están relacionadas con las más complejas, muchas de las características aprendidas por las tareas simples pueden ser transferidas a las tareas posteriores ahorrando así parte del aprendizaje.

Hipótesis 2 (H2): *Aplicar un suavizado a la serie temporal simplifica la tarea de predicción a la vez que mantiene características comunes con la versión original.*

La Hipótesis 2 es la usada para definir las tareas asignadas a cada capa. La idea es que suavizar una serie temporal elimina el ruido que presentan las series temporales habitualmente. Esto facilita la tarea de predicción y mantiene características comunes que pueden transferirse a las otras capas. Por ejemplo, es posible que se aprenda más rápidamente a extraer la tendencia o estacionalidad de la tarea más sencilla que será similar a la de tareas posteriores. Otra buena propiedad que posee el suavizado es que permite definir el grado de suavizado deseado, siendo capaz de aumentarlo o disminuirlo a partir de un hiperparámetro. Esta propiedad permite graduar la dificultad para cada capa definiendo un hiperparámetro diferente.

Hipótesis 3 (H3): *Es posible explicar parcialmente el comportamiento de las capas al conocer las tareas asignadas.*

Tal y como se ha explicado en el Sección 6.2, no existe un trabajo aplicado a predicción de series temporales que intente explicar el comportamiento de las redes neuronales de forma jerárquica como en las redes convolucionales. Como consecuencia de establecer tareas jerárquicas definidas, la Hipótesis 3 establece que el comportamiento de la red neuronal recurrente puede ser explicado de forma similar al de las redes convolucionales usados en visión artificial.

6.4. Metodología

En esta sección se define la arquitectura desarrollada a partir de las hipótesis. En primer lugar, se establecerá la nomenclatura definida para comprender todas las variables usadas. Posteriormente, se describirá la arquitectura junto a todos sus componentes.

6.4.1. Nomenclatura

- Sea m el número de características que posee el conjunto de datos.
- Sea w el tamaño de la ventana que servirá de entrada a la arquitectura.
- Sea t el instante de tiempo de referencia a partir del cual se han tomado los eventos pasados y se realizará la predicción a futuro.
- Sea h el número de eventos futuros a predecir.
- Sea $S_{m,w}(t)$ la ventana de entrada con m características y w eventos pasados para un instante de tiempo t.
- Sea n el número de capas o grupos de capas en la arquitectura.
- Sea G_i cada una de las capas o grupo de capas para cada nivel de profundidad $i \in 1..n$.
- Sea σ_i el factor de suavizado aplicado a cada nivel de profundidad $i \in 1..n$.
- Sea $\hat{S}^i_{m,w'}(t)$ el resultado del suavizado aplicado a las entradas para cada nivel de profundidad i, donde el tamaño de la ventana se ha visto reducida a w' siendo $w' < w$.
- Sea $O^i_{1..h}$ la salida de cada uno de los grupos i para cada horizonte $1..h$.

6.4.2. Descripción

Cada capa en HLNet contiene siempre una subcapa encargada de realizar el suavizado apodada *SmoothLayer*, una subcapa recurrente LSTM (Sección 5.1) y una capa totalmente conectada apodada $FeedForward$. A continuación, se describirá cada una de estas subcapas, sus interacciones internas y con otras capas.

La arquitectura HLNet comienza tomando la ventana $S_{m,w}(t)$ que sirve como entrada para cada uno de los niveles. Normalmente, las redes neuronales obtienen como entrada la salida de las capas anteriores como entrada de las capas posteriores (Sección 2.1.2). Sin embargo, el objetivo es que las capas se independicen parcialmente de las capas anteriores, centrándose en su tarea específica. Por esa razón, cada capa recibirá la misma entrada y realizará una transformación específica.

La subcapa *SmoothLayer* establece un suavizado para la entrada específico para cada capa. El suavizado se realiza aplicando una media móvil sobre la serie temporal. La media móvil obtiene una serie de subconjuntos a los que aplica la media, estos grupos se solapan entre ellos funcionando como una pila que extrae el valor más antiguo para añadir al más nuevo. Esta subcapa

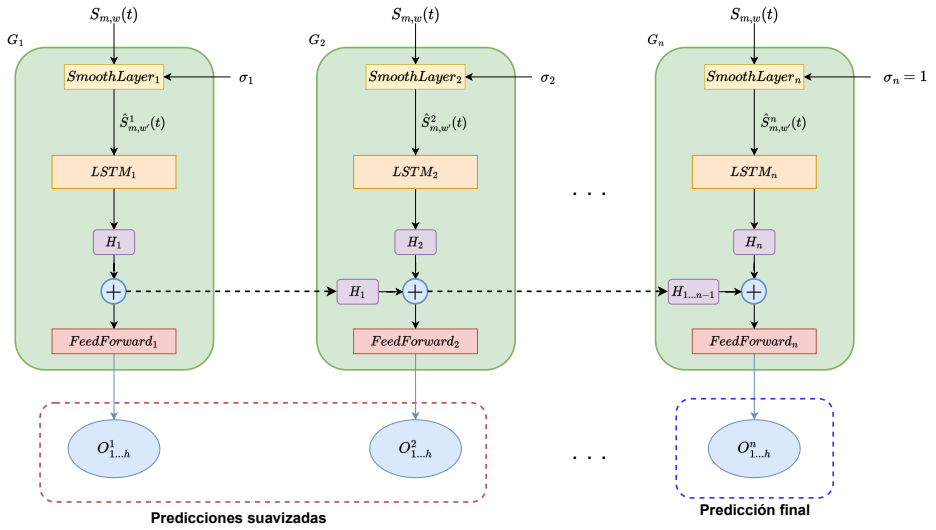


Figura 6.1: Arquitectura HLNet completa con n capas. Las líneas punteadas que conectan las distintas capas representan la propagación de los *hidden state* sin propagación de gradientes.

recibe un hiperparámetro σ o factor de suavizado que establece el tamaño de los subconjuntos que se usará para realizar la media móvil. Como salida, se obtiene la versión suavizada de la entrada donde el número de eventos ha cambiado ya que, a consecuencia del agrupamiento se reduce la ventana en $\sigma - 1$. Disminuir el factor de suavizado introduce más ruido al tener más en cuenta las variaciones locales de la serie temporal. Por ese motivo, al aumentar la profundidad de la red, se aumentará la dificultad de la tarea disminuyendo el factor de suavizado de manera que $\sigma_1 > \sigma_2 > \cdots > \sigma_n$. Hay que tener en cuenta que para la última capa n siempre tiene como factor de suavizado 1, esto quiere decir que no se realiza ninguna transformación sobre la entrada manteniéndola invariante. El objetivo de la última capa es realizar la predicción sin ningún tipo de suavizado aplicado a la ventana de entrada. La Figura 6.2 muestra el proceso que aplica la media móvil.

Una vez obtenida la entrada suavizada específica para cada capa, la versión simplificada $\hat{S}_{m,w}(t)$ se introduce dentro de una capa LSTM y obtiene el *hidden state* H de salida. Por lo tanto, la función de la capa LSTM es producir una serie de características que codificarán el conocimiento obtenido de la ventana de entrada. El *hidden state* tiene dos propósitos en HLNet, el primero es ser transferido a las capas siguientes para ser aprovechado, mientras que el segundo es servir de entrada para la capa $FeedForward$.

La transferencia de conocimiento se realiza concatenando las características obtenidas por la capa LSTM de cada una de las capas anteriores. De esta manera, la capa i recibe toda la información aprendida por las capas

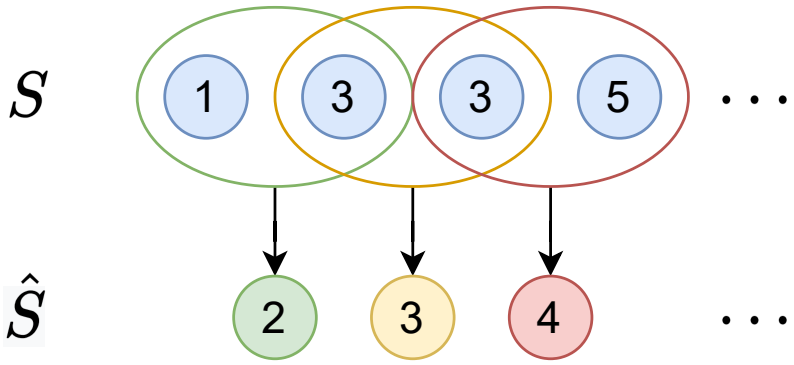


Figura 6.2: Proceso de media móvil donde el tamaño de cada subconjunto es dos, aplicado a una ventana S.

$1...i$. Sin embargo, el conocimiento de las capas anteriores se aísla para que las capas más profundas y con tareas más complejas no influyan en las capas encargadas de aprender tareas más sencillas. Para lograr esto, se elimina la propagación de gradientes a la hora de transferir los *hidden state* entre capas. Esto ocasiona que durante el proceso de optimización mediante descenso por el gradiente (Sección 2.1.4), no se tengan en cuenta los gradientes provenientes de la propagación de error de capas posteriores.

La concatenación del conocimiento obtenido en una capa y las anteriores se introduce en la última subcapa totalmente conectada que es el encargado de producir las salidas. Cada salida será encargada de producir la predicción a futuro de su versión suavizada, de esta manera establecemos a cada capa un objetivo diferente donde cada capa tiene su error asignado. Formalmente, la salida de cada capa se define como:

$$O^i_{1,\dots,h} = F_i(G_i(S_{m,w}(t)) \oplus \dots \oplus G_1(S_{m,w}(t))), \tag{6.1}$$

donde F_i representa la capa $FeedForward$ de la capa i, G_i representa el *hidden state* obtenido de cada capa a partir de la entrada y el símbolo $\oplus$ representa la concatenación de los *hidden state*.

Los errores de cada capa son calculados usando su predicción y los valores futuros reales suavizados con el mismo hiperparámetro que la capa. Los errores de cada capa son promediados generando el error final de la arquitectura.

6.5. Resultados

En esta sección se analizarán los resultados obtenidos tras la evaluación realizada sobre una serie de conjuntos de datos dentro de un marco de experimentación. Para ello, en primer lugar, se describirá brevemente cada uno

de los datos usados durante la experimentación. Tras ello, se detallará el marco experimental usado para entrenar y evaluar las arquitecturas. Una vez descrito los datos y el marco experimental se presentarán y discutirán los resultados obtenidos por cada una de las arquitecturas. Por último, se evaluará la evolución de la eficiencia con respecto el tamaño de los datos.

6.5.1. Conjuntos de datos

Para evaluar la arquitectura desarrollada, se han escogido 13 conjuntos de datos públicos de una diversa variedad de campos de aplicación. Los conjuntos de entrenamiento escogidos son univariantes y agrupados, donde cada uno presenta un horizonte de predicción y ventana pasada diferentes. Algunas de las series temporales no eran agrupadas, pero para mantener la homogeneidad en el proceso de evaluación, se ha llevado a cabo un proceso de agrupamiento. En la Sección 6.5.2 se detallará más el proceso de agrupamiento y ventaneo realizado.

La Tabla 6.1 muestra un resumen de cada uno de los conjuntos de datos involucrados y sus características. En total, se han usado cerca de 50000 series temporales, algunas de ellas obtenidas de competiciones públicas [56].

Conjunto de datos	**N**	**h**	**w**	**W**
Demand	72	144	144	432
CIF16o12	57	12	15	36
CIF16o6	15	6	7	18
ExchangeRate	8	6	7	18
M3	1428	18	22	36
M4	48000	18	22	36
NN5	111	56	70	168
SolarEnergy	137	6	7	18
Tourism	336	24	30	48
Traffic	862	24	30	72
Traffic-metr-la	207	12	15	36
Traffic-perms-bay	325	12	15	24
WikiWebTraffic	997	59	73	118

Tabla 6.1: Número de series temporales (N), horizonte de predicción (h), tamaño de ventana mínimo (w) y máximo (W) por cada uno de los conjuntos de datos.

El conjunto de datos Demand [27] contiene medidas de demanda eléctrica para el año 2015. Los datos fueron obtenidos para un edificio desde la página web de la Red Eléctrica Española (REE) muestreados cada diez minutos. La serie temporal fue agrupada generando 72 grupos con 576 eventos cada uno. El objetivo propuesto es predecir 24 horas en el futuro, 144 eventos al

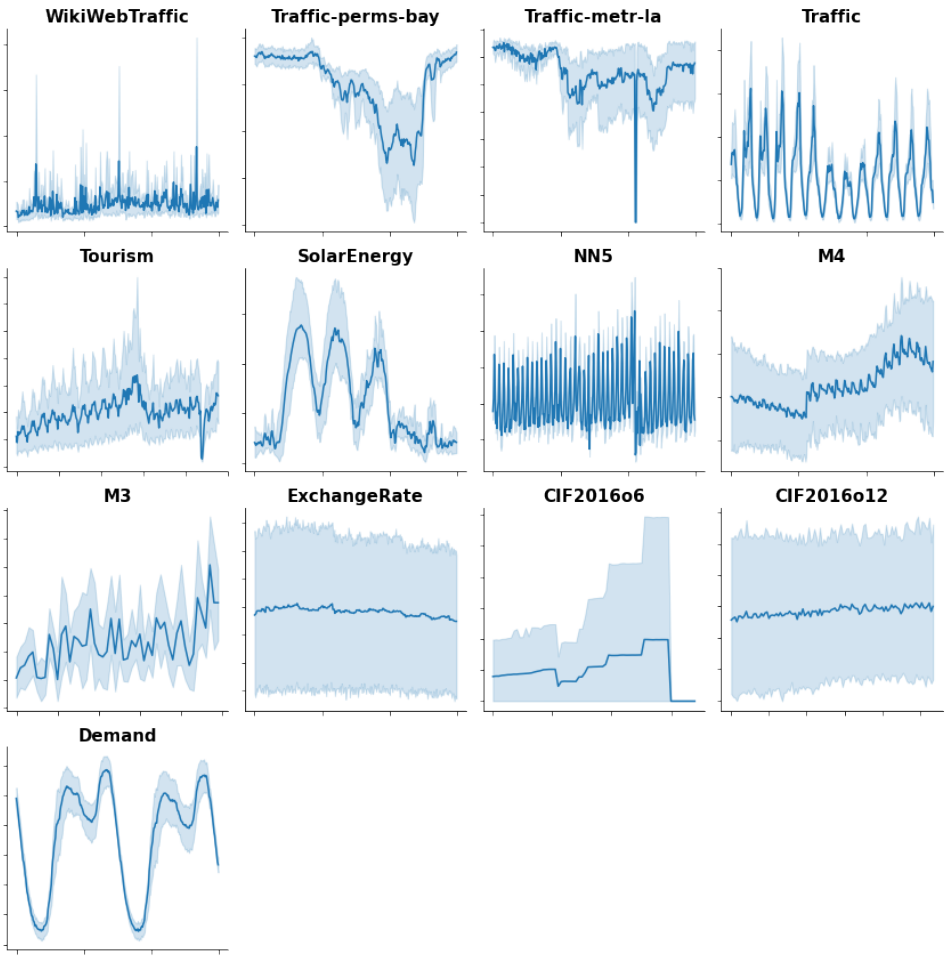


Figura 6.3: Visualización de los diferentes conjuntos de datos usados durante la experimentación.

muestrearse cada 10 minutos.

Los conjunto de datos CIF16o12 y CIF16o6 provienen de la competición *Computational Intelligence in Forecasting* 2016 [142]. La competición establece contiene un conjunto de datos con 72 series temporales muestreadas de forma mensual. En 15 de las 72 series temporales se establece un horizonte de predicción a 6 meses vista (CIF1606), mientras que en 57 series temporales se establece un horizonte de 12 meses vista (CIF16o12). El conjunto de datos contiene indicadores de riesgo bancario reales mientras que otros han sido generados de forma artificial.

El conjunto de datos Exchange Rate (ER) [86] contiene ratios de conversión entre distintas monedas tomados de ocho países: Australia, Reino Unido, Suiza, China, Japón, Nueva Zelanda y Singapur. Las series temporales tienen una frecuencia diaria obtenidas desde 1990 hasta 2016, dado lugar

a 7588 eventos por cada país. El horizonte de predicción establecido para este conjunto de datos son 6 días.

Los conjunto de datos M3 [101] y M4 [122] provienen de las competiciones Makridakis 3 y Makridakis 4 propuesto por el investigador Spyros Makridakis. Estas competiciones contienen un gran conjunto de series temporales con múltiples frecuencias y ámbitos incluyendo el industrial, financiero y demográfico. El número de series temporales por cada ámbito se distribuye de forma similar a como se distribuyen realmente, dando lugar a un escenario realista. Se ha limitado a las series temporales con una frecuencia mensual ya que se trata de las series temporales con más eventos, además, se han reducido las series temporales para reducir los requisitos computacionales. En total, el conjunto de datos M3 y M4 contiene 1428 y 48000 series temporales con 58 eventos cada una.

El conjunto de datos NN5 [25] proviene de una competición con el mismo nombre. Los datos fueron tomados de las retiradas de un cajero automático de Reino Unido. Se trata de 137 series temporales con una frecuencia diaria tomadas a lo largo de dos años dando lugar a alrededor de 238 eventos por cada serie temporal. El objetivo es predecir 56 días en el futuro.

El conjunto de datos SolarEnergy [168] contiene medidas obtenidas de la producción de 137 placas solares diferentes. Los datos fueron tomados con una frecuencia de 10 minutos durante el año 2006 en el estado de Alabama. El objetivo es predecir una hora en el futuro, 6 eventos dado que la frecuencia de muestreo es cada 10 minutos.

El conjunto de datos Tourism [7] contiene datos provenientes de organismos turísticos en Australia, Hong Kong y Nueva Zelanda. Los datos miden el volumen total de turismo a nivel nacional. El conjunto de datos se compone de 336 series temporales tomadas mensualmente. El objetivo es predecir dos años en el futuro, 24 eventos.

El conjunto de datos Traffic [2] contiene datos de ocupación de carreteras (en el rango [0, 1]) obtenidos por el departamento de transporte de California a través de 325 sensores en la Bahía de San Francisco. La frecuencia de muestreo de cada serie temporal es de diez minutos tomados entre 2015 y 2016, el objetivo es predecir cuatro horas en el futuro, 24 eventos dado que la frecuencia es cada diez minutos.

Los conjuntos de datos Traffic-metr-la (TML) y Traffic-perms-bay (TPB) [164] son dos datasets que miden la velocidad provenientes de varios sensores en dos zonas de Estados Unidos. Los datos de Traffic-metr-la (TML) mide la velocidad con una frecuencia de cinco minutos en varias carreteras de Los Ángeles a través de 207 sensores durante cuatro meses. Los datos de Traffic-perms-bay (TPB) obtiene las mismas medidas, pero a través de 207 sensores en la Bahía de San Francisco durante seis meses. El objetivo es predecir una hora en el futuro, 12 eventos dado que la frecuencia es cada cinco minutos.

El conjunto de datos WikiWebTraffic (WWT) [52] proviene de una com-

petición con el mismo nombre y contiene datos del número de visitas en 997 páginas de Wikipedia. La frecuencia de muestreo de cada página es diaria recogidas entre 2015 y 2016. El objetivo es predecir 59 días en el futuro.

6.5.2. Marco experimental

Con el objetivo de asegurar una comparación justa entre todos los conjuntos de datos, se ha desarrollado un marco experimental común para todas las arquitecturas con sus diferentes configuraciones aplicados a los distintos conjuntos de datos.

Dado que la arquitectura HLNet se basa en capas LSTM, se ha usado una red neuronal con capas LSTM como referencia para comprobar la mejora en eficiencia.

Cada una de las arquitecturas establece una serie de hiperparámetros, algunos comunes y otros específicos. Dado que la eficiencia y eficacia de los modelos puede variar dependiendo de los hiperparámetros, es necesario establecer un conjunto de experimentos que para un amplio rango de hiperparámetros. Además, para establecer una justa comparación, es necesario que los hiperparámetros comunes a las distintas arquitecturas usadas sean iguales para asegurar que cada arquitectura se ha evaluado sobre condiciones similares. Para ello, se ha realizado la búsqueda en rejilla (*grid search*) para encontrar la mejor arquitectura para una configuración de hiperparámetros.

Hiperparámetro	Valores
Número de capas (L)	2, 4
Número de neuronas por capa (U)	32, 64, 128
Factor de suavizado (σ)	2, 4, 6, 8
Ratio de aprendizaje (LR)	0.001, 0.0001

Tabla 6.2: Hiperparámetros usados durante la búsqueda en rejilla. Ten en cuenta que el factor de suavizado solo aplica a HLNet.

La Tabla 6.2 muestra los valores posibles para el número de capas, número de neuronas por cada capa y el ratio de aprendizaje. El horizonte a predecir es el establecido en la Sección 6.5.1 y el tamaño de ventana es el 125 %, 200 % o el 300 % del horizonte. El tamaño del *batch* se mantiene fijo a 32 ya que es un valor que suele funcionar bien en general.

La separación entre los conjuntos de entrenamiento, validación y test se han realizado usado la técnica de origen fijo [56]. Esta técnica realiza la separación teniendo en cuenta los diferentes grupos dentro de la serie temporal. Por cada grupo, se realiza el proceso de ventaneo y la última ventana de cada grupo se deja para el testeo, mientras que los eventos restantes se separan en entrenamiento y validación al 80 % y 20 % respectivamente.

Para asegurar que los tiempos de eficacia obtenidos son justos, los dos

modelos se han probado con el mismo hardware. Los componentes principales del hardware son: una tarjeta gráfica NVIDIA RTX 3070, un procesador AMD Ryzen 7 5800X 3.8GHz y 32 GB de RAM.

6.5.3. Discusión de resultados

La Tabla 6.3 muestra los mejores parámetros encontrados tras realizar la búsqueda en rejilla para LSTM y HLNet. En ambos casos se muestran los hiperparámetros comunes: el número de capas y las neuronas por cada capa. En caso de HLNet se muestra el factor de suavizado (σ), que es el hiperparámetro no compartido asignado a $SmoothLayer$. En cuanto al factor de aprendizaje, el mejor valor ha sido 0.0001 para todas las arquitecturas.

	LSTM		HLNet		
	Capas	Neuronas	Capas	Neuronas	σ
Demand	2	32	4	64	2
CIF16o12	4	64	4	64	6
CIF16o6	4	32	2	64	2
ExchangeRate	2	64	4	32	2
M3	2	128	2	32	6
M4	2	32	4	32	8
NN5	2	128	4	64	4
SolarEnergy	2	32	4	32	2
Tourism	2	128	2	32	6
Traffic	2	128	2	64	4
Traffic-metr-la	2	64	4	32	2
Traffic-perms-bay	4	64	4	128	4
WikiWebTraffic	4	32	4	128	6

Tabla 6.3: Mejores hiperparámetros encontrados por la búsqueda en rejilla.

En general, no hay un patrón aparente en el número de neuronas o capas entre la LSTM o HLNet. Aparentemente, HLNet no necesita tener más capas o neuronas que la arquitectura LSTM para converger a la mejor solución. En cuanto al factor de suavizado, tampoco parece haber un valor predominante, por lo que dependiendo de la naturaleza del problema un factor de suavizado parece más adecuado que otros. El único caso atípico es el conjunto de datos M4, que es el único que presenta un factor de suavizado igual a ocho.

La Tabla 6.4 muestra los resultados obtenidos con respecto la eficacia y eficiencia de la arquitectura LSTM y HLNet para los mejores hiperparámetros encontrados por la búsqueda en rejilla.

Si se define al modelo más eficaz como el modelo que obtiene el menor

	Modelo							
	LSTM				HLNet			
	MAE	MSE	WAPE (%)	TIME	MAE	MSE	WAPE (%)	TIME
Demand	1055	2.21e06	3.83	**0'49"**	**893**	**1.77e06**	**3.27**	3'40"
CIF16o12	**12,633**	**7.52e09**	17.20	0'30"	14,003	9.06e09	**15.32**	**0'17"**
CIF16o6	2.34e06	2.56e13	32.23	0'05"	**2.14e06**	**2.37e13**	**19.93**	**0'04"**
ER	**1.77e-3**	**5.71e-6**	0.30	2'05"	1.83e-3	8.00e-6	**0.28**	**0'52"**
M3	691	1.55e06	**15.30**	7'22"	**683**	**1.50e06**	**15.28**	**2'49"**
M4	**592**	**1.94e06**	**14.57**	**9'15"**	598	**1.94e06**	14.78	12'18"
NN5	**3.47**	30.5	18.56	**20'09"**	3.49	**29.49**	**18.34**	23'23"
SE	**1.96**	**9.54**	**222.00**	9'55"	2.16	12.40	255.00	**7'21"**
Tourism	2,349	1.21e08	19.69	3'41"	**2,217**	**9.61e07**	**18.95**	**1'40"**
Traffic	**0.0012**	6.53e-4	**34.92**	41'02"	**0.0012**	**5.77e-4**	**34.40**	**21'12"**
TML	**1.99**	**8.77**	**3.33**	14'14"	2.00	8.82	3.34	**12'16"**
TPB	**0.91**	**2.10**	**1.38**	48'27"	0.92	2.12	**1.38**	**33'41"**
WWT	**11.97**	**8,834**	**46.89**	46'49"	11.99	9,010	**47.38**	**45'24"**

Tabla 6.4: MAE, MSE, WAPE y tiempo de entrenamiento por cada conjunto de datos y arquitectura.

error en más métricas, se obtiene que ambos modelos son bastantes similares en términos generales. Se obtienen seis conjuntos de datos donde LSTM obtiene menor error que HLNet y cinco conjuntos de datos donde HLNet obtiene menor error que LSTM. Las diferencias de error entre ambas arquitecturas no son muy significantes en muchos de los conjuntos de datos. Sin embargo, HLNet destaca principalmente en el conjunto de datos de Demand y CIF16o6, mientras que LSTM parece destacar en el conjunto de datos SolarEnergy (SE).

HLNet parece destacar mayormente en cuanto a la eficiencia, obteniendo mejores tiempos de entrenamiento en 10 de los 13 conjuntos de datos. Además, en los conjuntos de datos M4, ExchangeRate y Tourism la reducción del tiempo es de menos de la mitad. La reducción del tiempo de ejecución en tantos conjuntos de datos no se debe a una reducción en el número de parámetros aparentemente tal y como muestra la Tabla 6.3.

6.6. Análisis de escalabilidad

Con el objetivo de evaluar cómo escala la eficiencia de la arquitectura propuesta con respecto LSTM para diferentes volúmenes de datos, se ha escogido uno de los conjuntos de datos aumentando gradualmente el volumen usado para el entrenamiento. Se ha escogido el conjunto de datos M3, dado que el tiempo de entrenamiento no es demasiado grande como para suponer un coste computacional demasiado grande realizar la búsqueda en rejilla. Además, la diferencia de tiempos entre LSTM y HLNet es suficientemente considerable como para poder observar una diferencia significativa entre ambos métodos.

El volumen de datos ha sido tomado desde el 5 % hasta el 100 % del

volumen total, usando siempre los datos desde el más reciente al más antiguo. Se ha muestreado usando pasos del 5 % hasta llegar al 50 % del volumen total, tras ello, se muestrea los datos usando pasos del 25 %.

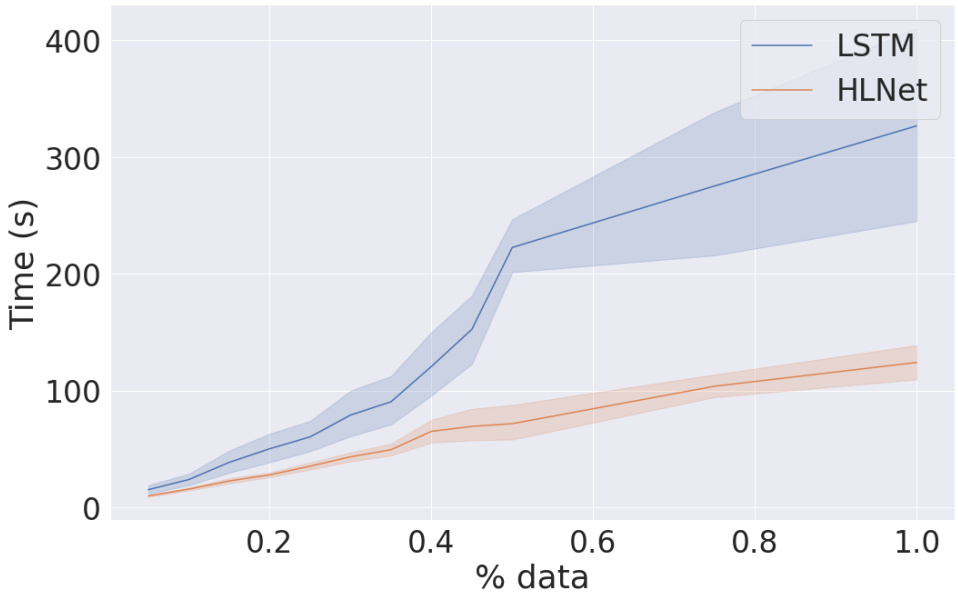


Figura 6.4: Tiempo de entrenamiento por volumen de datos en el conjunto de datos M3.

La Figura 6.4 muestra una gráfica con los resultados de analizar los tiempos medios de ambas arquitecturas para diferentes volumen de datos. Se observa como al principio los tiempos de entrenamiento son bastantes similares hasta que, entre el 40 % y 50 % los tiempos de entrenamiento para LSTM empiezan a aumentar considerablemente.

Estos resultados parecen indicar que HLNet es capaz de converger antes que la arquitectura LSTM. Esto apoya la Hipótesis 1, dado que los primeros niveles aprenden más rápido y, por lo tanto, pueden transferir el conocimiento más rápidamente.

Para demostrar que la convergencia se produce antes gracias a la metodología aplicada en HLNet, se muestra el número de pasos necesarios para converger de cada arquitectura en la Figura 6.5. Las gráficas muestran una forma similar a la obtenida en el caso del tiempo de entrenamiento, dado que se necesita procesar más datos para lograr converger.

6.7. Conclusiones

En general, la arquitectura HLNet presenta una eficacia similar a la red LSTM. Aparentemente, esto se debe principalmente a que, aunque la difi-

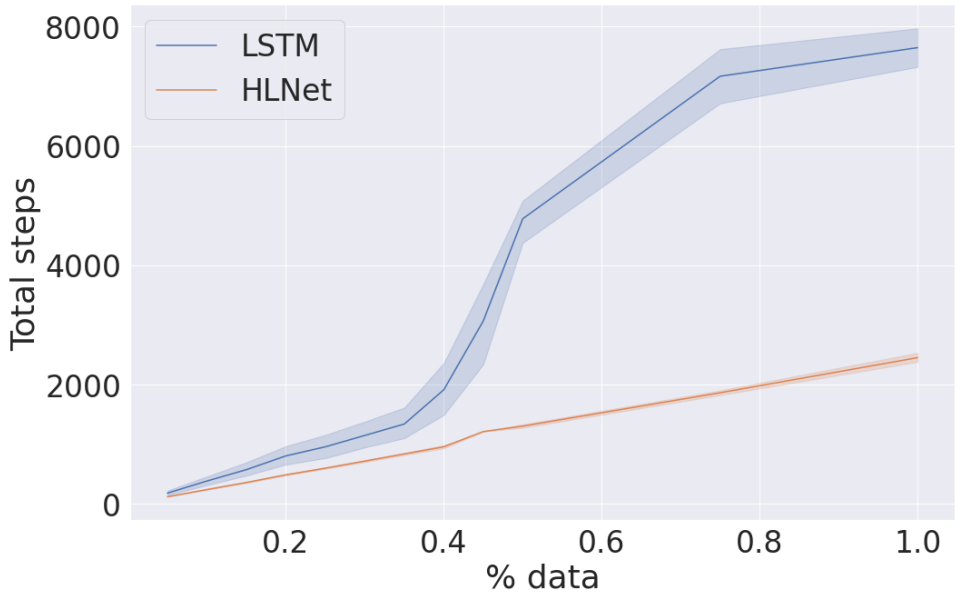


Figura 6.5: Número de pasos totales por volumen de datos en el conjunto de datos M3.

cultad se ha incrementado de forma gradual, el conocimiento final adquirido es similar en la mayoría de los casos. Sin embargo, la arquitectura destaca principalmente por su rápida convergencia. Por lo tanto, la red resulta igual de eficaz que una LSTM, pero más eficiente en la mayoría de los casos.

Capítulo 7

Propagated HIerarchical Learning Network (PHILNet)

Carry out a random act of kindness, with no expectation of reward, safe in the knowledge that one day someone might do the same for you.

Princesa Diana.

7.1. Resumen

La arquitectura desarrollada en la presente propuesta, al igual que HLNet, permite mejorar la convergencia mediante el uso de tareas adicionales a la tarea de predicción principal. Las tareas adicionales transforman la tarea principal mediante un proceso de suavizado que facilita la capacidad de predicción. Cada capa tiene asignada una tarea adicional y su dificultad incrementa hasta llegar a la dificultad de la tarea original. En HLNet cada una de las tareas adicionales estaba aisladas de manera que, durante el proceso de entrenamiento, solo las tareas adicionales podían influir en el proceso de optimización. En PHILNet se ha incluido la propagación de información durante el proceso de entrenamiento. Este cambio mejora considerablemente el tiempo de entrenamiento, manteniendo la eficacia en la mayoría de los conjuntos de datos.

7.2. Introducción

La arquitectura *Propagated HIerarchical Learning Network* o PHILNet, nace como una evolución de la arquitectura propuesta en la Capítulo 6. PHILNet modifica HLNet mejorando principalmente los tiempos de entrenamiento gracias a la introducción de una hipótesis adicional. En este caso,

en lugar de aislar las capas o grupos de capas, se permite que las tareas más complejas tengan una influencia sobre las tareas más simples.

El contexto y conjuntos de datos son los mismos que los propuestos en HLNet. Sin embargo, en este caso la experimentación ha sido ampliada incluyendo test estadísticos que permitan determinar si las diferencias entre los resultados obtenidos por los distintos métodos son estadísticamente significativas.

En la Sección 7.3 se incluye un breve recordatorio de las hipótesis usadas en HLNet, así como la nueva hipótesis. La Sección 7.3 muestra la metodología usada. Las Secciones 7.5.1 y 7.5.2 compara los resultados de PHILNet con HLNet y LSTM respectivamente. Finalmente, en la Sección 7.6 se establecen las principales conclusiones del trabajo obtenidas.

7.3. Hipótesis

Dado que PHILNet se trata de una evolución de HLNet, las tres hipótesis principales se mantienen y se incluye una hipótesis adicional.

Hipótesis 4 (H4): *Propagar información de tareas más complejas a tareas más sencillas durante el aprendizaje puede mejorar la velocidad de convergencia.*

Esta hipótesis será la base de la mejora y la idea intuitiva detrás consiste en que, al igual que las personas, es posible redefinir las bases de nuestro conocimiento adquirido al experimentar otras tareas diferentes. En este caso, las tareas más complejas pueden mejorar la capacidad de generalización de las características aprendidas en los primeros niveles. De esta forma, es posible que estas características resulten más útiles para los niveles posteriores durante la transferencia de conocimiento.

7.4. Metodología

La metodología seguida en PHILNet es igual a la realizada en HLNet. La diferencia radica durante el proceso de optimización en el que se combinan los gradientes de las tareas más simples y complejas. Durante este proceso, las primeras capas adquieren más información dado que el error de su tarea se une a la de las tareas posteriores en la etapa de retropropagación. Como resultado, los primeros niveles pueden tender a permanecer en zonas del espacio de error en las que el modelo se desempeña bien en más tareas, obteniendo características generales para transmitirlas a las capas inferiores. Además, la combinación de numerosas pérdidas en las capas iniciales produce un gradiente general con mayor amplitud, lo que ayuda a minimizar el tiempo de convergencia.

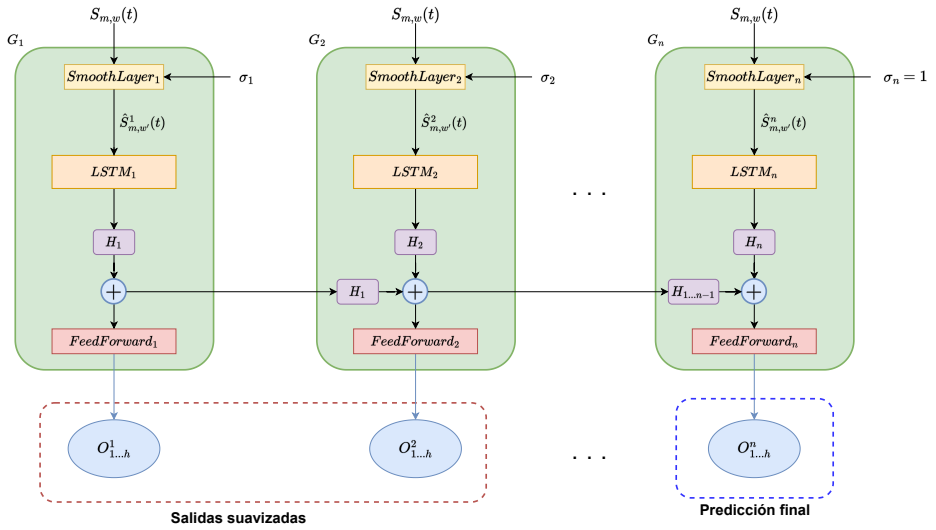


Figura 7.1: Arquitectura PHILNet propuesta.

La Figura 7.1 muestra la arquitectura desarrollada en la propuesta. Nótese que la arquitectura propaga los gradientes entre los distintos niveles a diferencia de HLNet. En este caso, cada nivel G_i contiene una capa $SmoothLayer$ que suaviza la entrada en base al factor de suavizado específico del nivel σ_i. Como resultado, se obtiene $\hat{S}^i_{w'}(t)$ donde w' denota el nuevo tamaño de ventana en la entrada en el momento t. La capa LSTM procesa la salida de la capa $SmoothLayer$ obteniendo el estado oculto (*hidden state*) H_i que codifica la información de $\hat{S}^i_{w'}(t)$. Tras ello, este estado oculto se concatena con todos los estados ocultos anteriores (si existen) con el objetivo de incluir toda la información de capas previas.Por último, la capa $FeedForward$ del nivel i procesa los estados ocultos obteniendo una predicción. Por cada nivel va a existir una salida, siendo la última de todas la predicción final de la capa.

El proceso realizado por cada uno de los niveles pueden definirse formalmente de la siguiente forma:

$$O^i_{1,\dots,h} = f_i(G_i(S^i_w(t)) + \dots + G_1(S^1_w(t))), \tag{7.1}$$

donde G_i representa la función que obtiene el estado oculto del nivel i, f_i la función que realiza la transformación de la capa $FeedForward$ y $O^i_{1,\dots,h}$ la predicción realizada por el nivel.

La mejora en eficiencia puede explicarse debido a la posibilidad de que los gradientes de las tareas más sencillas se desplacen rápidamente a las zonas de bajo error, ya que se espera que la convergencia en estas tareas sea más rápida que en las difíciles. Tras ello, los niveles posteriores pueden utilizar esa información, centrándose en el comportamiento específico de la serie temporal utilizando menos iteraciones.

7.5. Resultados

En esta sección se analizan los resultados obtenidos por PHILNet. Para ello, primero se compara con la propuesta anterior (HLNet) en la Sección 7.5.1. Posteriormente, se realiza la comparación con una red neuronal que usa capas LSTM en la Sección 7.5.2.

Todos los resultados se han analizado bajo las mismas condiciones en cuanto al hardware y software utilizado, con el objetivo de asegurar una comparación justa entre las distintas arquitecturas. Los componentes principales del hardware son: una tarjeta gráfica NVIDIA RTX 3070, un procesador AMD Ryzen 7 5800X 3.8GHz y 32 GB de RAM.

7.5.1. Comparación con HLNet

7.5.1.1. Análisis

En esta sección, se compara el modelo original (HLNet) y su extensión (PHILNet). Los resultados se analizarán del mismo modo que se realizó en HLNet y sobre los mismos conjuntos de datos (Sección 6.5). Además, se añade un test Bayesiano (Sección 2.3.2) para demostrar la significancia estadística de los resultados obtenidos.

Los resultados presentados sirven para conocer los beneficios de incluir la influencia de las tareas más complejas en las tareas más simples. De esta manera, al igual que en HLNet, las tareas simples pueden transferir su conocimiento hacia las capas posteriores. Sin embargo, en PHILNet se ha añadido la propagación de los gradientes correspondientes a las tareas más complejas.

	Modelo							
	PHILNet				HLNet			
	MAE	MSE	WAPE (%)	TIME	MAE	MSE	WAPE (%)	TIME
Demand	899	1.84e06	3.31	**3'08"**	**893**	**1.77e06**	**3.27**	3'40"
CIF16o12	**1.22e04**	**8.66e09**	16.48	**0'05"**	1.40e04	9.06e09	**15.32**	0'17"
CIF16o6	**1.89e06**	**1.90e13**	**19.54**	0'05"	2.14e06	2.37e13	19.93	**0'04"**
ER	**1.80e-4**	**6.25e-6**	**0.28**	**1'05"**	**1.80e-4**	8.00e-6	**0.28**	2'02"
M3	**674**	1.52e06	15.43	**1'34"**	683	**1.50e06**	**15.28**	2'49"
M4	604	2.00e06	**14.71**	**6'07"**	**598**	**1.94e06**	14.78	12'18"
NN5	**3.48**	30.20	18.40	**5'27"**	3.49	**29.49**	**18.34**	23'23"
SE	**1.77**	**8.75**	**183.00**	16'14"	2.16	12.4	255.00	**7'21"**
Tourism	2224	**8.28e07**	19.49	**1'28"**	**2217**	9.61e07	**18.95**	1'40"
Traffic	**1.2e-3**	6.77e-4	34.44	**6'27"**	**1.2e-3**	**5.77e-4**	**34.40**	21'12"
TML	**1.99**	**8.60**	**3.33**	**11'26"**	2.00	8.82	3.34	12'16"
TPB	**0.92**	2.13	**1.38**	**14'17"**	**0.92**	**2.12**	**1.38**	33'41"
WWT	12.00	9171	**47.17**	**9'28"**	**11.99**	**9009**	47.38	45'24"

Tabla 7.1: MAE, MSE, WAPE y tiempo de entrenamiento obtenido por cada conjunto de datos para HLNet y PHILNet.

La Tabla 7.1 muestra los resultados en cuanto a eficacia y eficiencia. Si se

define al modelo más eficaz como el modelo con menor error en más métricas, se puede observar que HLNet supera a PHILNet en ocho conjuntos de datos mientras que PHILNet solo obtiene mejor eficacia en 5 de ellos. Sin embargo, las diferencias no son muy significativas en general y las reducciones de error más grandes se dan cuando PHILNet supera a HLNet en tres de los cinco casos.

En cuanto a la eficiencia, PHILNet supera a HLNet en 11 de los 13 datasets. La reducción más considerable se ha producido en los conjuntos de datos NN5, Traffic, Traffic-perms-bay (TPB), and WikiWebTraffic (WWT) donde los tiempos de entrenamiento se han reducido a más de la mitad. La única excepción ha sido el conjunto de datos SolarEnergy (SE) que duplicado el tiempo de entrenamiento comparado con HLNet.

7.5.1.2. Tests estadísticos

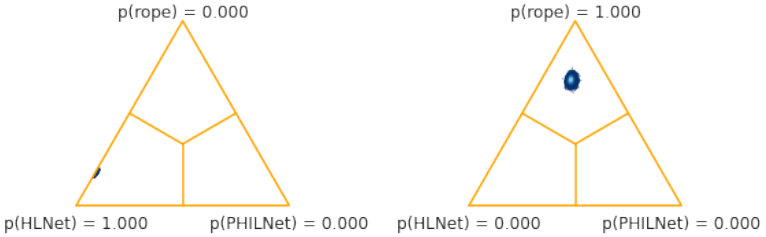
Con el objetivo de obtener conclusiones sobre si las diferencias entre PHILNet y HLNet son estadísticamente significativas, se ha optado por usar un test estadístico que garantice que los resultados obtenidos en la Sección 7.5.1.1 corresponden a la realidad. Se han realizado dos tests diferentes, uno para la eficacia y otro para la eficiencia basados en la propuesta de Alessio Benavoli del test de rangos con signo de Wilcoxon [15].

La prueba bayesiana de rangos con signo de Wilcoxon recibe cada par de métricas para los modelos A y B obtenidos tras evaluar cada una de las combinaciones de hiperparámetros, obteniendo como salida tres valores. El primer valor representa la probabilidad de que el modelo A sea mejor que el modelo B (p(A>B)), el segundo valor representa el escenario inverso (p(B>A)) y el tercer valor representa la probabilidad de que el modelo A y el B sean similares (p(Rope)). Para estimar cuándo dos resultados se consideran similares, necesitamos establecer un umbral que indique el rango de valores en el que se consideran dos soluciones similares. Los umbrales se establecerán posteriormente durante el análisis de los resultados por cada métrica analizada.

Por un lado, la métrica usada para realizar el test sobre la eficacia ha sido el WAPE debido a que es una medida relativa que permite interpretarlo fácilmente para establecer conclusiones. Por otro lado, para la eficiencia se ha usado el tiempo de entrenamiento/convergencia.

Dado que el test Bayesiano (Sección 2.3.2) necesita establecer un umbral para definir un rango de equivalencia, se han seleccionado dos umbrales para cada uno de los tests. En el caso de la eficacia se ha considerado un umbral del 1 %. Esto quiere decir que experimentos con una diferencia menor al 1 % en WAPE son considerados similares. Para la eficiencia, se ha considerado el 1 % de la diferencia media entre los tiempos de entrenamiento, esto quiere decir que diferencias en el tiempo de entrenamiento por debajo del 1 % de la media se consideran similares.

El test estadístico usado es no paramétrico y pareado. Los individuos son los resultados obtenidos por cada una de las combinaciones de hiperparámetros usados durante la búsqueda en rejilla en cada conjunto de datos. Dado que los hiperparámetros son compartidos, todos los individuos están pareados por lo que es posible realizar el test Bayesiano sin problema.



(a) Tiempo de entrenamiento. (b) WAPE.

Figura 7.2: Probabilidad de obtener un mayor WAPE entre PHILNet y HLNet.

La Figura 7.2 muestra los resultados obtenidos tras aplicar el test Bayesiano a través de un triángulo equilátero donde cada una de las esquinas representa una probabilidad. La probabilidad de que HLNet obtenga un mayor valor de la métrica usada se representa como p(HLNet), el escenario contrario se representa como p(PHILNet) y en caso de no mostrar diferencias significativas se representa como p(Rope). Tal y como se observa, la probabilidad de que HLNet obtenga un peor tiempo de entrenamiento es 100 % en general. Del mismo modo, la probabilidad de que el WAPE entre HLNet y PHILNet sea similar es del 100 % en general.

Con estos resultados, se concluye que la influencia de las tareas más complejas sobre las más simples tiene un efecto positivo en los tiempos de entrenamiento manteniendo la eficacia. Una hipótesis para explicar este fenómeno puede explicarse por la combinación de los gradientes. Al obtener una combinación de los gradientes provenientes de las distintas tareas relacionadas, los primeros niveles llegan a obtener características más generales que son más útiles a la hora de transferirlos. Otra posible explicación sea debido a que la inclusión de diferentes gradientes disminuya el efecto conocido como *negative transfer* [159].

7.5.2. Comparación con LSTM

7.5.2.1. Análisis

Dado que la arquitectura se ha implementado usando capas LSTM, se ha realizado la comparación con el objetivo de observar si se ha obtenido una mejora significativa. Al igual que la comparación con HLNet, se presentan los resultados y se han aplicado unos test Bayesianos para garantizar la significancia estadística de los resultados.

La Tabla 7.2 muestra los hiperparámetros obtenidos tras la búsqueda en rejilla por cada uno de los conjuntos de datos. Tenga en cuenta que el parámetro σ representa el factor de suavizado.

	LSTM		PHILNet		
	Capas	Neuronas	Capas	Neuronas	σ
Demand	2	32	4	32	2
CIF16o12	4	64	2	128	4
CIF16o6	4	32	4	128	2
ExchangeRate	2	64	2	128	4
M3	2	128	4	32	4
M4	1	32	2	64	8
NN5	1	128	4	32	8
SolarEnergy	2	32	4	32	4
Tourism	1	128	2	128	8
Traffic	2	128	2	128	4
Traffic-metr-la	2	64	2	64	6
Traffic-perms-bay	4	64	4	128	4
WikiWebTraffic	4	32	2	32	2

Tabla 7.2: Número de capas, neuronas (se asume mismo número de neuronas por cada capa) y factor de suavizado σ (solo en el caso de PHILNet) de los mejores resultados obtenidos tras la búsqueda en rejilla.

Los parámetros no presentan ningún patrón específico que indique que PHILNet necesite más capas o neuronas en cada capa que LSTM en general.

La Tabla 7.3 muestra los resultados con respecto la eficacia y eficiencia obtenidos entre LSTM y PHILNet.

La eficacia obtenida es bastante similar en ambas arquitecturas considerando el mejor modelo como aquel que obtiene menor error en más métricas. En cinco de los trece conjuntos de datos, LSTM mejora a PHILNet. Mientras que en ocho de los trece, PHILNet mejora a LSTM. Sin embargo, las mejoras no superan ni el 10 % por lo que no son muy significativas en la mayoría de

	Modelo							
	LSTM				PHILNet			
	MAE	MSE	WAPE (%)	TIME	MAE	MSE	WAPE (%)	TIME
Demand	1055	2.21e06	3.83	**0'49"**	**899**	**1.84e06**	**3.31**	3'08"
CIF16o12	1.26e04	**7.52e09**	17.10	0'30"	**1.22e04**	8.66e09	**16.48**	**0'05"**
CIF16o6	2.34e06	2.56e13	32.23	**0'05"**	**1.89e06**	**1.90e13**	**19.54**	**0'05"**
ER	1.9e-4	6.72e-6	0.30	2'05"	**1.8e-4**	**6.25e-6**	**0.28**	**0'52"**
M3	691	1.55e06	**15.30**	7'22"	**674**	**1.52e06**	15.43	**1'34"**
M4	**592**	**1.94e06**	**14.57**	9'15"	604	2.00e06	14.71	**6'07"**
NN5	3.50	30.50	18.56	20'09"	**3.48**	**30.20**	**18.40**	**5'27"**
SE	**1.73**	**7.60**	222.00	**10'14"**	1.77	8.75	**183.00**	16'14"
Tourism	2349	1.21e08	19.69	3'41"	**2224**	**8.28e07**	**19.49**	**1'28"**
Traffic	**1.2e-3**	**6.53e-4**	**33.92**	41'02"	**1.2e-3**	6.77e-4	34.44	**6'27"**
TML	**1.99**	8.77	**3.33**	14'14"	**1.99**	**8.60**	**3.33**	**11'26"**
TPB	**0.91**	**2.10**	**1.38**	48'27"	0.92	2.13	**1.38**	**14'17"**
WWT	**11.97**	**8833**	**46.89**	46'49"	12.00	9171	47.17	**9'28"**

Tabla 7.3: MAE, MSE, WAPE y tiempo de entrenamiento por cada conjunto de datos y arquitectura.

los conjuntos de datos. Solo en los conjuntos de datos de Demand, CIF16o6 y Tourism hay una mejora significativa donde PHILNet supera a la LSTM.

PHILNet obtiene una clara mejora de eficiencia con respecto a LSTM en la mayoría de los conjuntos de datos. En 10 de los 13 conjuntos de datos PHILNet mejora el tiempo de entrenamiento, de los cuales 8 reducen a más de la mitad el tiempo de entrenamiento. Las únicas excepciones son los conjuntos de datos Demand y SolarEnergy (SE), donde PHILNet obtiene peores tiempos de ejecución.

	LSTM	PHILNet
MAE	1.40 %	**8.41 %**
MSE	6.32 %	**16.21 %**
WAPE	0.98 %	**16.39 %**
Time	55.45 %	**267.71 %**

Tabla 7.4: Porcentaje de mejora de cada métrica en conjuntos de datos donde PHILNet mejora a LSTM (columna PHILNet) y viceversa (columna LSTM).

La Tabla 7.4 muestra el porcentaje de mejora cuando LSTM mejora a PHILNet y viceversa. En general PHILNet obtiene una mejora considerablemente mayor que LSTM cuando mejora tanto en eficacia como en eficiencia.

La Figura 7.3 muestra un gráfico de cajas y bigotes (*boxplot*) del WAPE obtenido sobre el conjuntos de arquitecturas entrenadas para cada uno de los hiperparámetros. Además, el gráfico se ha dividido por factor de suavizado para obtener una visualización del impacto de este hiperparámetro sobre la eficacia. Dado que LSTM no tiene este parámetro, se le ha asignado un valor de cero para representar su inexistencia.

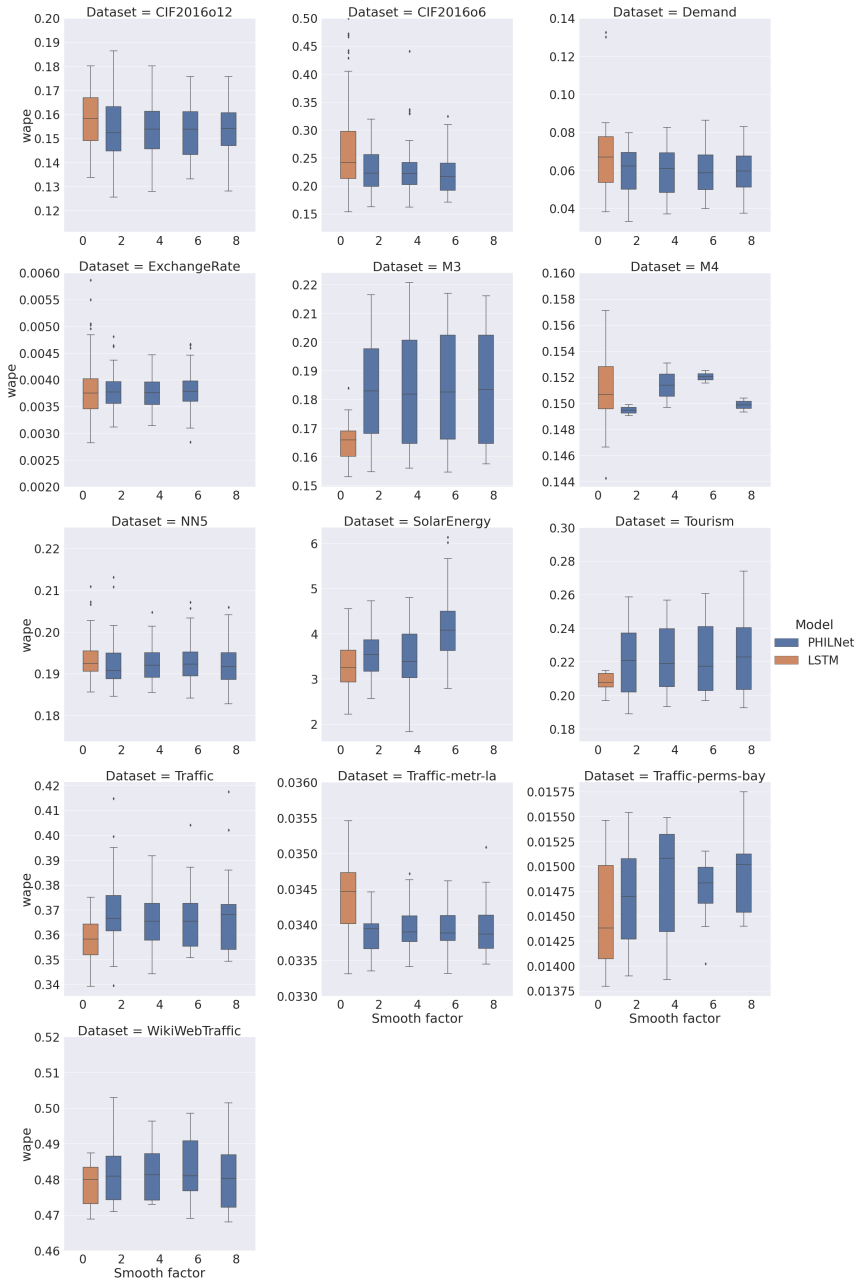


Figura 7.3: *Boxplot* del WAPE por número de neuronas de todos los experimentos.

En la mayoría de los casos no se observa una influencia clara del factor de suavizado en la eficacia de las arquitecturas. La mediana se mantiene similar generalmente con cambios menores rango de las cajas. En los conjuntos de datos M4, Traffic Perms Bay (TPB) y SolarEnergy (SE) los cambios son más

significativos. La arquitectura LSTM presenta más valores atípicos, algunos de los cuales han sido eliminados para obtener una mejor visualización de los datos. Esto nos lleva a pensar que los resultados suelen ser más estables y resistentes a configuraciones no óptimas de los hiperparámetros.

No se ha observado una mejora clara de LSTM sobre PHILNet observando la Figura 7.3. En seis de los conjuntos de datos, PHILNet obtiene una mejor mediana y rango intercuartílicos para al menos un factor de suavizado. LSTM obtiene una mejor mediana o rango intercuartílicos en siete de los trece conjuntos de datos.

La Figura 7.4 muestra un gráfico de cajas y bigotes (*boxplot*) para la eficiencia obtenida por cada una de las arquitecturas entrenadas con una combinación de hiperparámetros. En 11 de los 13 conjuntos de datos PHILNet obtiene una mediana y rango intercuartílico independientemente del factor de suavizado. PHILNet obtiene una mediana mayor para todos los factores de suavizado en el conjunto de datos CIF16o6, aunque la diferencia es menor a un segundo mientras que la eficiencia entre PHILNet y LSTM es similar en el conjunto de datos Demand. Aunque la eficiencia obtenida en la Tabla 7.3 muestra que PHILNet obtiene mayor tiempo que LSTM en el conjunto de datos SolarEnergy (SE), en la gráfica se puede observar que se suelen obtener menores tiempos.

7.5.2.2. Tests estadísticos

En esta sección, al igual que en la Sección 7.5.1.2, se realiza un test Bayesiano para obtener conclusiones que sean estadísticamente significativas. Hay que tener en cuenta que a diferencia de HLNet, LSTM no tiene hiperparámetro para asignar el factor de suavizado. Por lo tanto, es necesario realizar el test por cada uno de los factores para que los tests sean pareados.

	Smooth factor											
	2			4			6			8		
Dataset	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet
Demand	0.18	0.82	0.00	0.22	0.78	0.00	0.28	0.72	0.00	0.21	0.79	0.00
CIF16o6	0.97	0.03	0.00	0.96	0.03	0.00	0.98	0.02	0.00	-	-	-
CIF16o12	0.56	0.43	0.00	0.54	0.45	0.00	0.60	0.39	0.00	0.54	0.46	0.00
M3	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
M4	0.00	1.00	0.00	0.10	0.77	0.14	0.00	0.99	0.00	0.05	0.89	0.05
NN5	0.14	0.86	0.00	0.15	0.85	0.00	0.13	0.87	0.00	0.16	0.84	0.00
SE	0.01	0.01	0.99	0.14	0.04	0.82	0.00	0.00	1.00	-	-	-
ER	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	-	-	-
Tourism	0.79	0.15	0.06	0.76	0.17	0.07	0.77	0.16	0.07	0.76	0.17	0.07
Traffic	0.00	0.41	0.59	0.00	0.82	0.18	0.00	0.74	0.26	0.00	0.55	0.45
TML	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
TPB	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
WWT	0.70	0.07	0.23	0.70	0.07	0.23	0.70	0.07	0.23	0.69	0.07	0.24

Tabla 7.5: Probabilidad de que LSTM obtenga un WAPE mayor a PHILNet (columna LSTM), que PHILNet obtenga un mayor WAPE que LSTM (columna PHILNet) y que se obtenga un error similar (columna rope), por cada factor de suavizado (2, 4, 6 and 8).

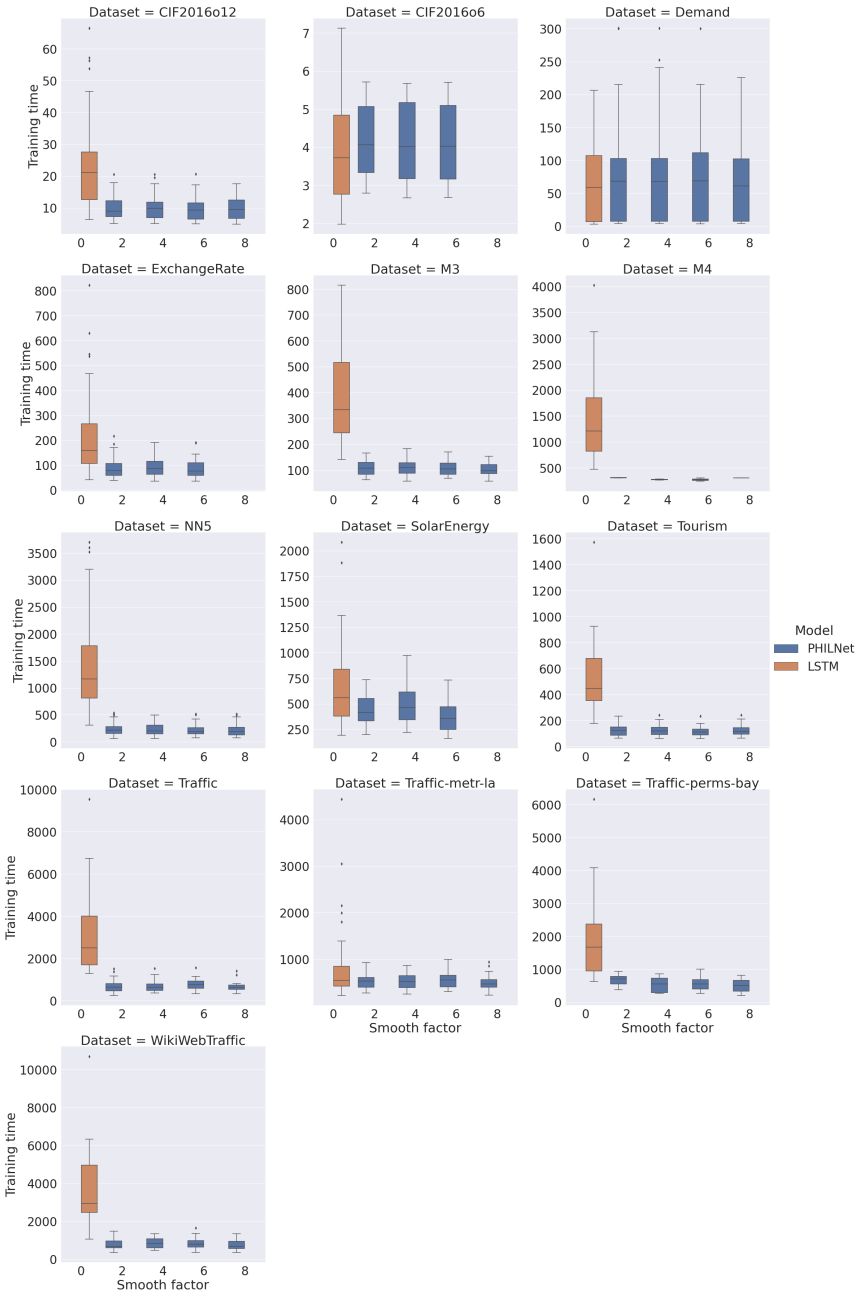
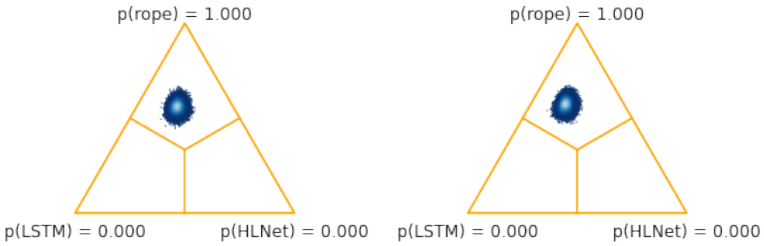


Figura 7.4: *Boxplot* de los tiempos de entrenamiento de cada experimento por cada factor de suavizado.

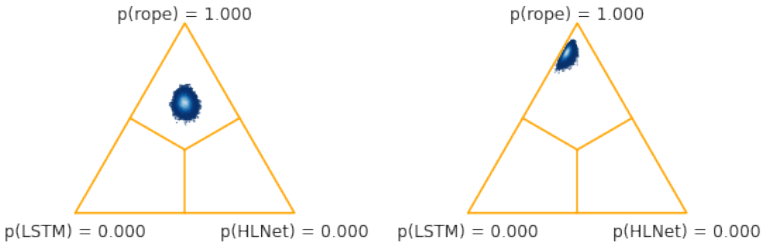
La Tabla 7.5 muestra la probabilidad de que LSTM tenga mayor WAPE que PHILNet (LSTM), que PHILNet tenga mayor WAPE que LSTM (PHILNet) y que no existan diferencias significativas (rope) por cada factor de suavizado y conjunto de datos. Al igual que en HLNet, se ha establecido

un rango de un 1 % para determinar que no hay diferencias significativas. Además, tal y como se propuso en el artículo original [14] para los tests bayesianos, es recomendable establecer un segundo umbral para determinar cuando la probabilidad no se considera suficientemente grande como para considerarse significativa. Este umbral ha sido establecido al 75 %, por lo tanto, cualquier probabilidad por debajo del 75 % no se considera lo suficiente significativa como para obtener unas conclusiones fiables. Se ha establecido el color verde para aquellas probabilidades que superan el 75 %, mientras que el color naranja se ha establecido para aquellas por debajo.



(a) Smooth factor 2.

(b) Smooth factor 4.



(c) Smooth factor 6.

(d) Smooth factor 8.

Figura 7.5: Probabilidad de que LSTM tenga mayor WAPE que PHILNet y viceversa.

La mayoría de las probabilidades mayores al 75 % determinan que no hay diferencias significativas, lo que indica que los resultados entre LSTM y PHILNet son similares en la mayoría de los conjuntos de datos. En el caso del conjunto de datos CIF16o12, aunque la probabilidad mayoritaria establece que LSTM tiene mayor error, los resultados no permiten establecer conclusiones fiables para ningún factor de suavizado. En el conjunto de datos Traffic, solo se consideran suficientemente fiables para un factor de suavizado igual a 4 donde se establece que no hay diferencias significativas. Solo en el conjunto de datos SolarEnergy, se ha concluido de forma significativa que

PHILNet obtiene mayor error que LSTM.

La Figura 7.5 muestra el test Bayesiano aplicado a la eficacia para todos los conjuntos de datos. La interpretación de las probabilidades mostradas en la figura es la misma que la descrita en la tabla anterior. Se han obtenido cuatro triángulos equiláteros, uno por cada factor de suavizado. En general, todas las probabilidades parecen indicar que PHILNet y LSTM obtienen unos resultados similares.

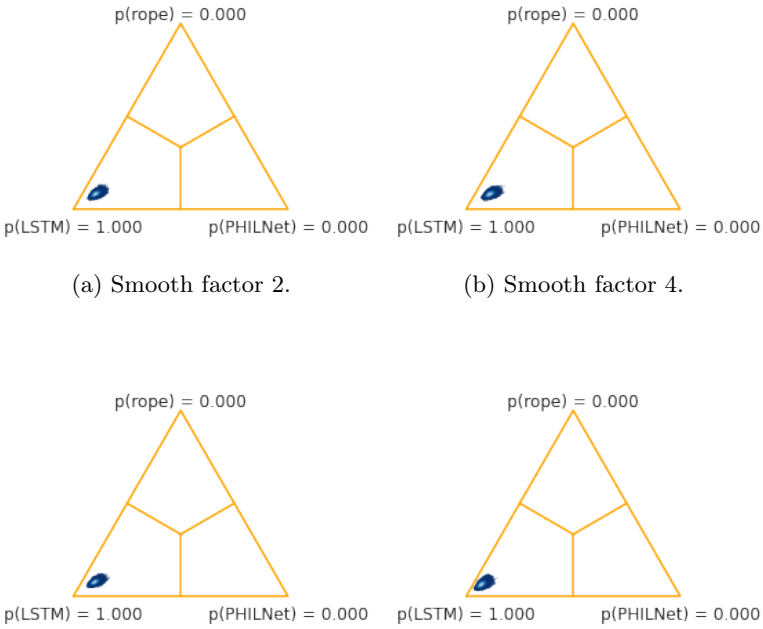
	Smooth factor											
	2			4			6			8		
Dataset	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet	LSTM	rope	PHILNet
Demand	0.15	0.05	0.80	0.22	0.04	0.73	0.12	0.05	0.83	0.12	0.05	0.83
CIF16o6	0.00	0.00	1.00	0.00	0.01	0.99	0.00	0.01	0.99	-	-	-
CIF16o12	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
M3	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
M4	0.91	0.02	0.08	0.90	0.02	0.08	0.95	0.01	0.04	0.91	0.02	0.07
NN5	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
SE	1.00	0.00	0.00	0.98	0.01	0.01	1.00	0.00	0.00	-	-	-
ER	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	-	-	-
Tourism	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
Traffic	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
TML	1.00	0.00	0.00	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.01	0.00
TPB	0.96	0.02	0.02	0.98	0.01	0.01	0.96	0.02	0.02	0.97	0.01	0.01
WWT	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.01	0.00	0.99	0.00	0.00

Tabla 7.6: Probabilidad de que LSTM obtenga un tiempo de entrenamiento mayor a PHILNet (columna LSTM), que PHILNet obtenga un mayor tiempo de entrenamiento que LSTM (columna PHILNet) y que se obtenga unos tiempos similares (columna rope), por cada factor de suavizado.

Con respecto a la eficiencia, se ha repetido el mismo proceso que con la eficacia. Al igual que el caso de HLNet, se ha establecido un rango en el que se consideran tiempos de ejecución similares. Por lo tanto, la diferencia de los tiempos por debajo del 1 % de la media de las diferencias son considerados similares. Así mismo, se ha establecido el umbral del 75 % para establecer conclusiones fiables.

La Tabla 7.6 muestra las probabilidades de que LSTM obtenga un tiempo de entrenamiento mayor a PHILNet (LSTM), que PHILNet obtenga un mayor tiempo de entrenamiento que LSTM (PHILNet) y que se obtenga unos tiempos similares por cada factor de suavizado (*rope*). Todas las probabilidades se han considerado fiables en este caso, mostrando que LSTM obtiene mayores tiempos de ejecución en general con una probabilidad superior al 99 %. En el conjunto de datos Demand y CIF16o6 se concluye que PHILNet obtiene unos tiempos superiores a LSTM.

La Figura 7.6 muestra un triángulo equilátero por cada factor de suavizado mostrando las probabilidades LSTM, PHILNet y *rope*. Para todos los factores de suavizado, se concluye que LSTM tiene mayores tiempos de entrenamiento que PHILNet en todos los conjuntos de datos.



(a) Smooth factor 2.

(b) Smooth factor 4.

(c) Smooth factor 6.

(d) Smooth factor 8.

Figura 7.6: Probabilidad de obtener un mayor tiempo de entrenamiento entre LSTM y PHILNet por cada factor de suavizado.

7.5.3. Análisis de escalabilidad

En esta sección se realizará un análisis del efecto que tiene el número de capas, neuronas en cada capa y el volumen de datos en la eficiencia. El número de capas y neuronas en cada capa es un análisis interesante para poder observar si hay una mayor tendencia a empeorar la eficiencia en las diferentes arquitecturas. Además, analizar la eficiencia con respecto el volumen de datos aporta información sobre el volumen de datos necesario para que la arquitectura propuesta obtenga una ventaja con respecto las otras.

Se ha escogido el conjunto de datos M3 para realizar el análisis. El conjunto de datos es lo suficientemente manejable como para no consumir demasiados recursos computacionales, mientras que se obtiene una mejora considerable como para poder ver la diferencia con facilidad. Los datos han sido submuestreados desde el 5 % al 100 % del volumen total tomando pasos del 5 % hasta el 50 % de los datos y pasos del 25 % a partir de ahí. A la hora de submuestrear, se toman siempre los datos más recientes eliminando los datos más antiguos.

La Figura 7.7 muestra la evolución del tiempo de entrenamiento para

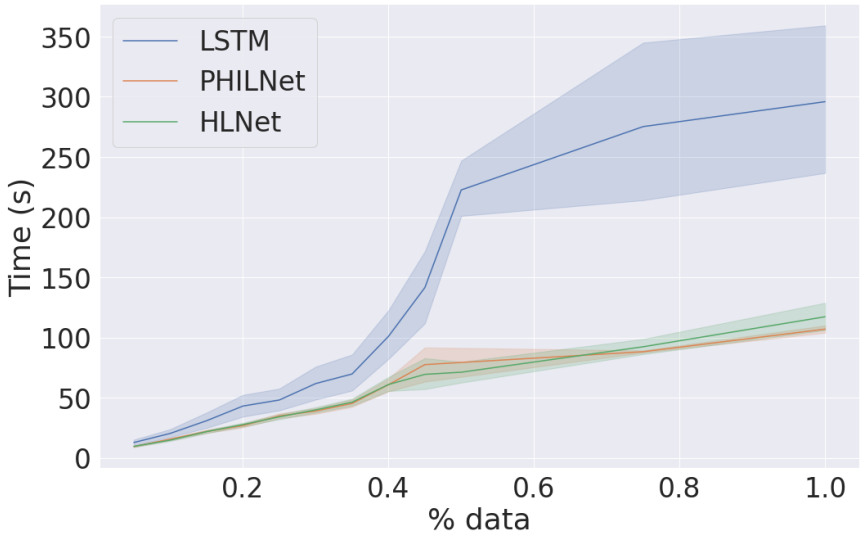


Figura 7.7: Tiempo de entrenamiento analizado volumen de datos en el conjunto de datos M3.

cada una de las arquitecturas analizadas por cada volumen de datos. Se observa que al principio todas las arquitecturas comienzan con un tiempo similar con un leve incremento en la red LSTM. A partir del rango entre el 40 % y 50 % se observa un incremento exponencial en LSTM mientras que PHILNet y HLNet se mantienen con un crecimiento lineal. Finalmente, se observa una clara mejora por parte de PHILNet y HLNet comparado con LSTM.

La Figura 7.8 muestra la evolución del tiempo de entrenamiento con respecto el número de capas y neuronas por cada capa. Aparentemente, en casi todos los conjuntos de datos hay un incremento exponencial. La arquitectura LSTM presenta una mayor mediana de tiempo de entrenamiento y rango intercuartílico. Solo en CIF201606 y Demand tiene unos tiempos de entrenamiento más parecidos entre las arquitecturas.

7.6. Conclusiones

Los resultados obtenidos, junto al análisis estadístico muestran que PHILNet propone una mejora en tiempos de ejecución con respecto HLNet manteniendo una eficacia similar. También se ha demostrado que LSTM presenta unos tiempos mayores que PHILNet con un error similar sin necesidad de aumentar el número de capas o neuronas. Esta mejora de la arquitectura HLNet parecer dar soporte a las hipótesis formuladas con una metodología sencilla, intuitiva y de propósito general.

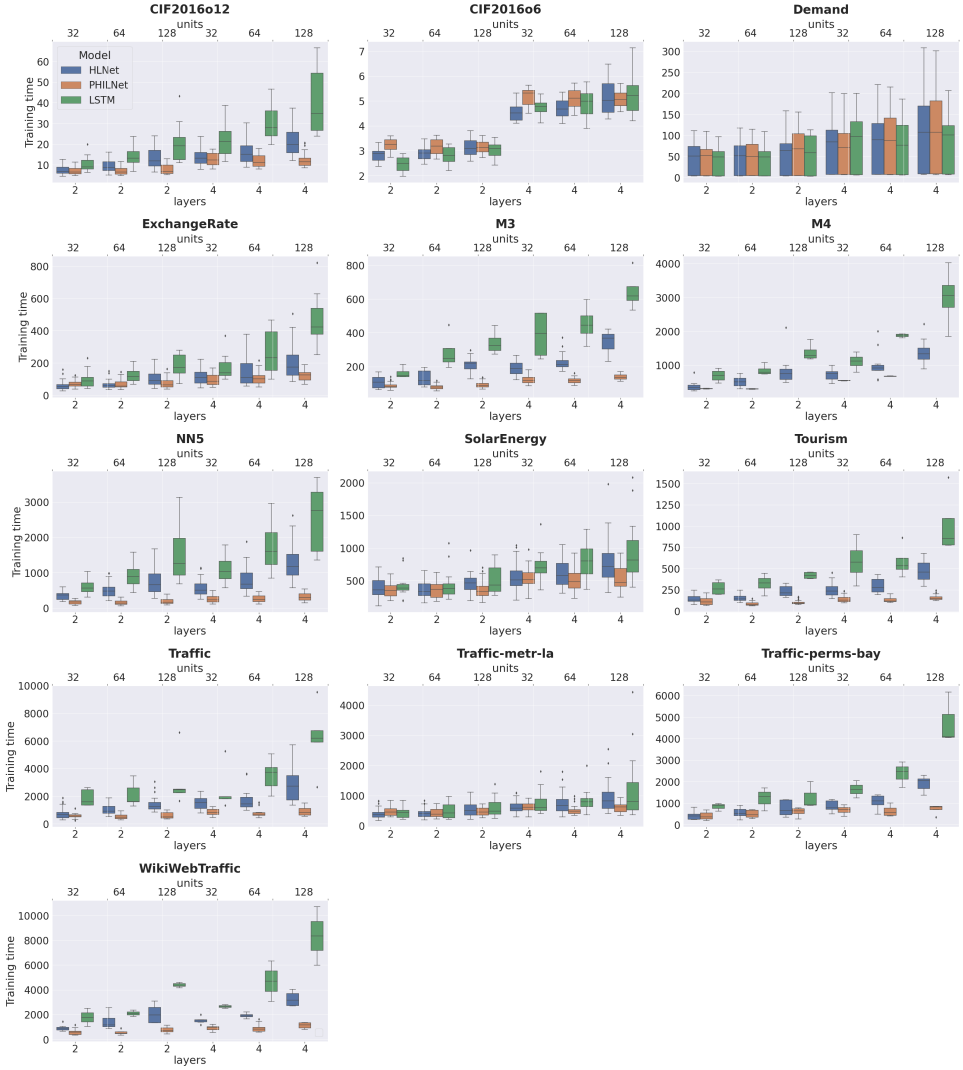


Figura 7.8: Tiempo de entrenamiento obtenido por número de capas y unidades.

Capítulo 8

Smooth Residual Connection Network (SRCNet)

Pero la vida es corta: viviendo, todo falta; muriendo, todo sobra.

Felix Lope de Vega y Carpio.

8.1. Resumen

En la presente propuesta se ha mostrado un nuevo mecanismo para redes neuronales aplicado a la predicción de la temperatura del aceite aislante en un transformador eléctrico. El funcionamiento principal consiste en realizar una descomposición de la serie temporal de forma secuencial en una serie de componentes con distintas propiedades. La red neuronal se encarga de obtener cada componente y aprender una transformación que obtenga una predicción parcial. Las principales ventajas de este mecanismo es la capacidad de dividir la tarea de predicción en varias predicciones parciales enfocadas en aspectos que colaboran para obtener la predicción final. Cada predicción parcial puede enfocarse en cualquier aspecto sin introducir un sesgo inductivo demasiado restrictivo.

8.2. Introducción

La predicción de la demanda eléctrica ha sido ampliamente estudiada en la literatura desde hace varios años y sigue siendo un gran reto actualmente [103][117]. Una predicción a largo plazo puede traer grandes beneficios en cuanto a la optimización de recursos, sin embargo, actualmente no existe una solución perfecta que cumpla con todas las necesidades. Una mala predicción no solo puede conllevar a un excesivo gasto, puede causar un daño irreversible a transformadores eléctricos. Este es uno de los motivos por

el que el estudio del estado de los transformadores ha ganado importancia. Uno de los componentes esenciales de los transformadores es el aceite aislante (*insulation oil*), cuyo rol es ayudar a regular la temperatura y evitar daños que puedan afectar la vida útil de los transformadores [37][150]. Por lo tanto, una predicción de la temperatura del aceite aislante puede ayudar al proceso de mantenimiento al estimar aquellas temperaturas que puedan llegar a ser peligrosas. Sin embargo, la predicción de la temperatura del aceite aislante presenta problemas de eficiencia. Dado que diferentes transformadores eléctricos pueden presentar comportamientos distintos, una aproximación común es entrenar una arquitectura distinta por cada uno de los transformadores usando la aproximación *bottom-up* [108][140]. Esta aproximación puede presentar problemas de escalabilidad cuando el número de transformadores eléctricos es grande y la arquitectura consume bastantes recursos. Por esta razón, es necesario buscar arquitecturas que sean capaces de presentar buenos resultados reduciendo la cantidad de recursos posibles.

Las redes neuronales han demostrado ser una solución competitiva a otras arquitecturas para la predicción del aceite aislante. Varios trabajos usan arquitecturas como los Transformers [156], obteniendo una de las mejores eficacias de la literatura en problemas de predicción de series temporales. Sin embargo, estas arquitecturas presentan varios problemas de eficiencia que pueden llegar a impedir su aplicación [171]. Además, en trabajos aplicados a la predicción de la temperatura del aceite aislante se analiza si el uso de los Transformers es necesario en ese problema, obteniendo resultados competitivos sin ellos [167].

Teniendo en cuenta los factores anteriormente expuestos, se presenta la red neuronal llamada *Smooth Residual Connection Network*, de aquí en adelante SRCNet. El objetivo de esta arquitectura es reducir el coste computacional en comparación con otras arquitecturas del estado del arte manteniendo una eficacia similar. La arquitectura propuesta, se basa en el uso de redes convolucionales usando el nuevo mecanismo llamado: *smooth residual stacking*. Los resultados mostrados por Minhao Liu et al. [97] demuestran que las redes convolucionales, normalmente más eficientes que los Transformers, pueden lograr mejor eficacia que la mayoría de arquitecturas de la literatura combinando características a múltiples niveles aprendidas por las capas convolucionales y conexiones residuales [75]. El mecanismo propuesto en este trabajo, se basa también en el presentado por Boris N. Oreshkin et. al. [107]. De esta manera, se reduce la complejidad de la ventana de entrada en cada capa centrándose en aspectos diferentes de la entrada. Esta idea, aunque similar, presenta una metodología distinta con diferentes hipótesis a las propuestas HLNet (Capítulo 6) y PHILNet (Capítulo 7) con una eficiencia superior al basarse en redes convolucionales en lugar de recurrentes y presentando un mecanismo con menor sesgo inductivo en la arquitectura.

En la Sección 8.3 se presenta la hipótesis principal de la propuesta. La

Sección 8.4 describe la metodología desarrollada en esta propuesta y sus componentes principales. La Sección 8.5 muestra los resultados obtenidos tras la experimentación. La Sección 8.6 comenta las conclusiones obtenidas de los resultados. Por último, la Sección 8.1 resume la propuesta presentada.

8.3. Hipótesis

Muchas arquitecturas dentro del aprendizaje automático integran sesgos inductivos que modifican el funcionamiento interno para ajustarse al conocimiento experto del problema [53]. Los sesgos inductivos introducen algunas restricciones o regularizaciones que priorizan ciertas soluciones frente a otras. En el caso de las redes neuronales, los sesgos inductivos modifican las transformaciones realizadas por cada una de las capas y sus interacciones para ajustarse al problema.

La primera de las hipótesis fundamentales se basa en el efecto que tiene el sesgo inductivo en la eficacia obtenida.

Hipótesis 5 (H5): *Restricciones/Penalizaciones introducidas por el sesgo inductivo pueden mejorar la eficiencia al reducir la probabilidad de explorar una solución fuera del espacio de búsqueda priorizado.*

La Hipótesis 5 establece que el sesgo inductivo añadido a la arquitectura basado en el conocimiento experto puede ayudar a mejorar la eficiencia durante el proceso de entrenamiento. Esto es consecuencia de la priorización introducida por el sesgo inductivo, reduciendo el espacio de soluciones, penalizando o eliminando aquellas soluciones que no se ajustan al diseño de la arquitectura. Gracias a ello, la probabilidad de converger al espacio de soluciones aumenta reduciendo el tiempo de convergencia. Sin embargo, una arquitectura con menos sesgo inductivo penalizaría menos soluciones, incrementando el espacio de soluciones lo que puede llevar a incrementar el tiempo de entrenamiento.

Un problema del sesgo inductivo es que aquellas soluciones fuera del espacio priorizado pueden obtener una mejor eficacia. Por lo tanto, el sesgo inductivo puede dar lugar a soluciones subóptimas debido a la penalización o restricción. Suavizar la penalización aumenta el espacio de soluciones y la probabilidad de encontrar una solución eficaz. Un balance adecuado del sesgo inductivo puede llevar a soluciones óptimas tomando menos tiempo para converger.

Otras arquitecturas descomponen la serie temporal en sus distintos componentes y restringen el diseño para que la arquitectura aprenda dicha descomposición. En la arquitectura propuesta en esta tesis, se establece una descomposición con las siguientes propiedades:

A. La descomposición debe eliminar complejidad a la ventana de entrada eliminando la versión suavizada de esta.

B. La descomposición debe realizarse de forma secuencial, por lo que la descomposición siguiente depende de la anterior.

C. Cada *Smooth Residual Block* se centra en un componente sin compartir información.

D. Cada *Smooth Residual Block* debe colaborar usando una descomposición diferente para producir la predicción final.

8.4. Metodología

En esta sección se describe en detalle los principales componentes que forman a SRCNet. La Sección 8.4.1 presenta la nomenclatura de los distintos términos usados durante toda la metodología. La Sección 8.4.2 muestra el primer paso de la metodología y la primera capa de la arquitectura. Por último, el bloque principal de la arquitectura se detalla en la Sección 8.4.3.

8.4.1. Nomenclatura

A continuación se describen los términos utilizados para definir la metodología llevada a cabo::

- W denota el número de eventos pasados usados para alimentar la arquitectura y generar la predicción, también conocido como tamaño de ventana.
- D denota el número de características en cada evento. Ten en cuenta que en una serie temporal univariante $D = 1$.
- F denota el número de características de salida obtenida como resultado de aplicar la capa Embedding.
- $S^{W,D}$ denota la ventana de entrada compuesta por W eventos compuesto, a su vez, por D características.
- $S^{W,F}$ denota la ventana de entrada codificada tras la transformación aplicada por la capa Embedding con F características.
- $S^{W,F}_{Smooth}$ denota la versión suavizada de la ventana de entrada codificada.
- $\hat{S}^{W,F}$ denota el resultados de restar la versión suavizada de la ventana de entrada codificada a la ventana codificada, también llamada residuo.
- H denota el número de eventos futuros a predecir, también conocido como el tamaño del horizonte. Ten en cuenta que en una predicción de un paso $H = 1$.

- P denota el número de características futuras predichas. Ten en cuenta que puede, o no, ser igual a D.
- N denota el número de *Smooth Residual Blocks* en la arquitectura.
- $O_i^{H,P}$ denota la predicción parcial realizada por el *Smooth Residual Blocks* i.

8.4.2. Embedding

La capa Embedding es la primera transformación aplicada a la ventana de entrada. Esta capa usa capas convolucionales causales (ver Sección 5.2) para transformar las D características en F características, donde F viene determinado por el número de filtros existentes en la capa convolucional. De forma adicional, una codificación posicional es añadida al igual que la usada en la arquitectura original de los Transformers [156] para incluir información sobre el orden temporal de los eventos de entrada codificados.

8.4.3. Smooth Residual Block

El *Smooth Residual Block* es el componente principal de la arquitectura. La idea principal es descomponer $S^{W,F}$ en N componentes, cuya suma conforma la salida $S^{W,H}$. Las redes convolucionales causales son las capas usadas internamente para transformar la entrada en la salida esperada.

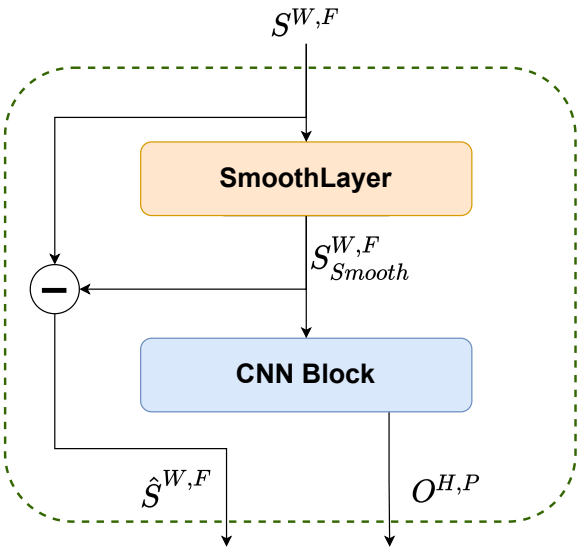


Figura 8.1: *Smooth Residual Block*.

En la primera capa *Smooth Residual Block* general el residuo $S^{W,F}_{Smooth}$ para la entrada codificada y obtiene una predicción parcial. Obsérvese que

la descomposición se aplica a la versión codificada de la secuencia de entrada $S^{W,F}$, lo que nos permite trabajar con series temporales univariantes y multivariantes, ya que cualquier tipo se proyecta sobre un mismo espacio latente. Además, la codificación puede ser beneficiosa para encontrar un espacio latente que representa la serie temporal en un espacio con características más informativas.

La componente $S^{W,F}_{Smooth}$ es generada por la capa $SmoothLayer$, cuya entrada es la secuencia anterior $S^{W,F}$ en el primer bloque, y la salida es la componente $S^{W,F}_{Smooth}$. La descomposición se realiza mediante medias móviles. Las medias móviles nos permiten obtener una representación suave de la secuencia original. Si elegimos el tamaño de ventana adecuado o repetimos la media móvil varias veces, se obtiene la tendencia. Sin embargo, si elegimos un tamaño de ventana más corto aplicado una vez, podemos obtener una función intermedia con una combinación de tendencia y estacionalidad en la que se reduce el ruido. Nos interesa esta representación, ya que no intentamos restringir la descomposición de la ventana de entrada en tendencia y estacionalidad.

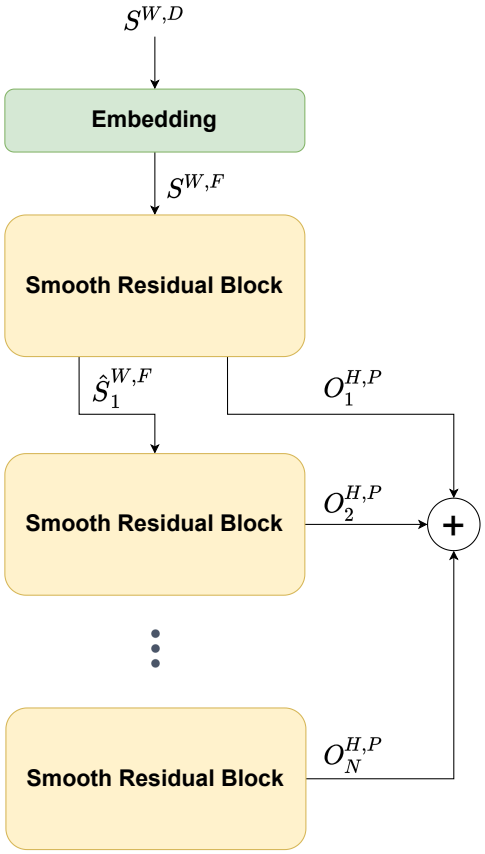


Figura 8.2: *Arquitectura SRCNet completa.*.

Como utilizamos medias móviles, necesitamos definir un hiperparámetro de tamaño de ventana para $SmoothLayers$ en todos los bloques residuales suaves. El hiperparámetro es el mismo para todas las capas excepto la última cuyo tamaño de ventana es uno, permaneciendo la secuencia sin cambios. Esto significa que la última capa se centra sólo en los residuos de todas las capas anteriores obteniendo principalmente el ruido de la secuencia original.

Después de generar el componente $S^{W,F}_{Smooth}$ en el primer *Smooth Residual Block*, el componente se resta a la secuencia de entrada $S^{W,F}$. A continuación, el siguiente bloque aplica el mismo proceso utilizando el residuo $\hat{S}^{W,F}$ como ventana de entrada. Este paso es esencial, porque este mecanismo permite que los bloques se centren en un aspecto diferente de la secuencia. Además, es importante normalizar la entrada para mantener cada componente dentro de la distribución de los datos cuando restamos el componente de la secuencia original.

El componente $S^{W,F}_{Smooth}$ se introduce en dos capas de convolución causal que emiten la predicción $O^{H,P}_i$ para la capa i. Finalmente, la salida de cada bloque se suma para obtener la predicción final. Como la suma total se utiliza para la predicción final, los bloques colaboran utilizando su respectiva representación de la entrada. Esta colaboración, combinada con la descomposición explicada anteriormente, motiva una división colaborativa del problema.

La Figura 8.1 muestra las transformaciones realizadas internamente por cada uno de los Smooth Residual Blocks y la Figura 8.2 muestra la arquitectura completa usando N bloques.

8.5. Resultados

En la presente sección se detalla todo el proceso llevado a cabo hasta la obtención de los resultados de la propuesta descrita. En la Sección 8.5.1 se detallan los conjuntos de datos usados durante la experimentación. POsteriormente, la Sección 8.5.2 presenta el proceso llevado a cabo apra obtener los resultados. Tras ello, la Sección 8.5.3 discute los resultados obtenidos tras la experimentación. La Sección 8.5.4 analiza las salidas parciales obtenidas por la arquitectura con el objetivo de comprender su funcionamiento interno. Por último, se realiza un test estadístico en la Sección 8.5.5 sobre las distintas arquitecturas para asegurar que existen diferencias significativas.

8.5.1. Conjunto de datos

Los datos estudiados en este trabajo fueron los conjuntos de datos Electricity Transformer [97], que contienen datos de 2 transformadores de electricidad de 2 estaciones en China (ETTh1 y ETTh2). Los datos se tomaron con una frecuencia horaria desde 2016/07 hasta 2018/07. El objetivo de ambos

conjuntos de datos es predecir la temperatura del aceite aislante que refleja el estado del transformador eléctrico.

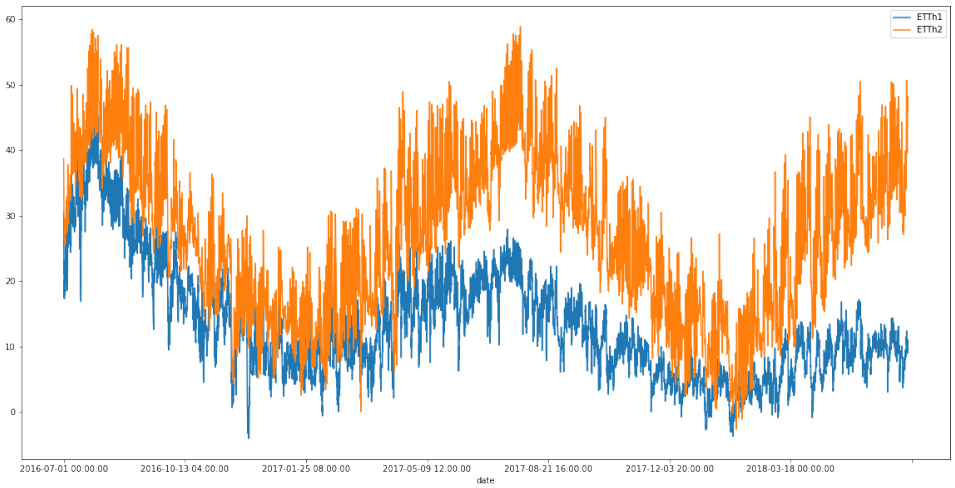


Figura 8.3: Temperatura del aceite aislante para los conjuntos de datos ETTh1 y ETTh2.

Los datos contienen varias características relacionadas con el nivel de carga del transformador. Sin embargo, estudios recientes muestran que el uso de estas características aumenta el error en comparación con la versión univariante considerando solo la temperatura del aceite aislante. Por ello, nos centramos en utilizar la temperatura del aceite como entrada sin ninguna característica adicional.

De forma visual, la Figura 8.3 muestra un comportamiento diferente para ambos conjuntos de datos, lo que anima a utilizar diferentes arquitecturas que modelen el comportamiento de cada conjunto de datos de forma independiente. Esta es una de las razones para encontrar un método que no sólo sea eficaz, sino también eficiente.

8.5.2. Marco experimental

El procesamiento aplicado a los conjuntos de datos es el mismo que el propuesto en el artículo original [171], utilizando la misma división para entrenamiento, validación y prueba para realizar comparaciones justas con otras arquitecturas de la literatura. Los datos se estandarizaron utilizando la división del entrenamiento para obtener la media y la desviación típica.

Como método de optimización, se ha seleccionado una búsqueda en rejilla aplicado a los hiperparámetros. Este tipo de búsqueda se utiliza para asegurar que todas las arquitecturas exploradas utilizan el mismo espacio de hiperparámetros con el objetivo de realizar una comparación justa. Los hiperparámetros optimizados durante la búsqueda en cuadrícula se definen

en la Tabla 8.1.

Hiperparámetros	Valores
Tamaño de ventana	24, 48, 72, 96, 120
Tamaño de embedding	8, 16
# Neuronas en cada capa	8, 16, 32, 64
# Capas	1, 2, 3
Tamaño del kernel por cada filtro	3, 5, 7

Tabla 8.1: Espacio de hiperparámetros usados durante la búsqueda en rejilla.

Algunos hiperparámetros no se incluyen en la búsqueda en rejilla debido a limitaciones computacionales o debido a su poca influencia en la eficacia del modelo. Se seleccionó un tamaño del lote (*batch size*) de 32 ya que suele dar buenos resultados normalmente en múltiples trabajos de la literatura. El número de épocas se fijó en 100, que es más que suficiente para obtener la convergencia en las arquitecturas seleccionadas. Se considera que una arquitectura entrenada ha convergido si el error de validación no disminuye en 10 épocas. Cuando la arquitectura converge, el entrenamiento se detiene en esa época restaurando los pesos con menor error obtenidos. Como horizonte de predicción, se considera suficiente una previsión de 24 épocas futuras. Por último, todos los experimentos para cada combinación de hiperparámetros se repitieron tres veces para reducir la influencia de la aleatoriedad, reduciendo la influencia de los resultados atípicos.

Las arquitecturas seleccionadas para comparar SRCNet son: una red neuronal convolucional (CNN) utilizada como línea de base, NBEATS para comparar el mecanismo propuesto en su trabajo con el presentado en esta tesis, SCINet como una de las arquitecturas con mayor eficacia en problemas de predicción de series temporales de la literatura y HLNet como predecesor de SRCNet.

Todos los resultados se han analizado bajo las mismas condiciones en cuanto al hardware y software utilizado, con el objetivo de asegurar una comparación justa entre las distintas arquitecturas. Los componentes principales del hardware son: una tarjeta gráfica NVIDIA RTX 3070, un procesador AMD Ryzen 7 5800X 3.8GHz y 32 GB de RAM.

8.5.3. Discusión de los resultados

Los resultados se presentan teniendo en cuenta el MAE estandarizado. Todas las tablas muestran las métricas obtenidas para las arquitecturas con la mejor eficacia encontrada en la búsqueda en rejilla. Las métricas representan la media de los tres experimentos dado que para una misma combinación de hiperparámetros se repite tres veces el mismo experimento. Además, se

incluye la desviación estándar de las métricas de eficacia entre los tres experimentos.

Conjunto	Ventana	CNN	HLNet	NBEATS	SCINet	SRCNet
ETTh1	24	0.172 (±0.03)	0.197 (±0.014)	0.201 (±0.012)	0.186 (±0.006)	**0.138 (±0.01)**
	48	0.167 (±0.015)	0.193 (±0.019)	0.21 (±0.043)	0.166 (±0.021)	**0.137 (±0.006)**
	72	0.139 (±0.013)	0.202 (±0.007)	0.152 (±0.009)	0.149 (±0.013)	**0.131 (±0.003)**
	96	0.144 (±0.007)	0.205 (±0.014)	0.149 (±0.013)	0.146 (±0.01)	**0.129 (±0.001)**
	120	0.156 (±0.012)	0.192 (±0.007)	0.15 (±0.018)	0.138 (±0.01)	**0.131 (±0.004)**
ETTh2	24	0.205 (±0.008)	0.333 (±0.016)	**0.197 (±0.003)**	0.198 (±0.006)	0.198 (±0.006)
	48	0.198 (±0.002)	0.339 (±0.011)	0.201 (±0.003)	0.201 (±0.009)	**0.195 (±0.005)**
	72	0.195 (±0.004)	0.334 (±0.01)	0.202 (±0.002)	0.196 (±0.009)	**0.189 (±0.003)**
	96	0.2 (±0.005)	0.338 (±0.003)	0.207 (±0.003)	0.197 (±0.004)	**0.193 (±0.002)**
	120	0.201 (±0.005)	0.339 (±0.004)	0.206 (±0.002)	0.198 (±0.004)	**0.194 (±0.005)**

Tabla 8.2: MAE obtenido por las mejores arquitecturas dividido por tamaño de ventana. El mejor MAE por cada ventana se resalta en negrita.

La Tabla 8.2 muestra la métrica MAE de los mejores modelos por cada tamaño de ventana. Como se observa, SRCNet obtiene los mejores resultados independientemente del tamaño de ventana seleccionado para todos los conjuntos de datos, seguido de SCINet. HLNet tiene un error mayor que todos los modelos seleccionados, presumiblemente porque el uso de capas LSTM en su interior no logra obtener unos buenos resultados.

En el conjunto de datos ETTh1, usar un tamaño de ventana de 24 o 48 eventos pasados, parece ser insuficiente para obtener toda la información necesaria para realizar la predicción, especialmente para SCINet, NBEATS y CNN. Además, SRCNet y HLNet muestran menos variación en la eficacia obtenida para el tamaño de ventana más corto y grande. Parece existir una relación entre el tamaño de la ventana y el MAE en SRCNet y SCINet donde, a mayor tamaño, mejor eficacia se obtiene. Sin embargo, se necesita un tamaño de ventana de 120 para que SCINet obtenga el mismo MAE que SRCNet, que consume cinco veces más recursos en el momento de la inferencia. Además, en términos de desviación típica entre experimentos, SRNet obtiene la menor desviación típica siendo diez veces menor que SCINet en el mejor de los casos.

En el conjunto de datos ETTh2 el MAE es mayor que el obtenido en ETTh1 en todas las prouestas, ya que el conjunto de datos parece ser más difícil de predecir. No hay relación entre el tamaño de la ventana y el error obtenido. Los tamaños de ventana más grandes, como 96 y 120, no parecen mejorar la eficacia de las arquitecturas en general. La mejor eficacia se obtiene con un tamaño de ventana de 72 usando las arquitecturas CNN, SRCNet y SCINet, mientras que HLNet y NBEATS obtienen la mejor para un tamaño de ventana de 24. La diferencia observada entre el mejor y el peor MAE es menor que en ETTh1 en la mayoría de los casos. Además, la desviación estándar entre experimentos parece ser menor en todas las arquitecturas. NBEATS y SRCNet parecen tener la menor desviación estándar, seguidas de SCINet. La desviación estándar de NBEATS y SRCNet es tres veces mejor

que la de SCINet comparando la configuración con la mejor eficacia.

Conjunto	Tamaño de ventana	CNN	HLNet	NBEATS	SCINet	SRCNet
ETTh1	24	**0.43**	0.91	1.79	1.10	0.50
	48	**0.42**	1.13	1.08	2.14	1.26
	72	**0.94**	0.84	0.98	2.81	1.90
	96	**1.00**	3.40	0.85	3.70	1.01
	120	1.27	1.41	**0.88**	1.75	1.04
ETTh2	24	**0.62**	1.33	0.94	3.97	3.23
	48	**0.51**	2.80	0.98	5.23	1.20
	72	**0.76**	1.26	0.92	5.60	1.57
	96	0.93	2.05	**0.92**	5.05	1.17
	120	1.12	1.36	**0.39**	7.24	1.25

Tabla 8.3: Tiempo de entrenamiento obtenido para la mejor arquitectura (según MAE) por cada tamaño de ventana y conjunto de datos. La arquitectura con menor número de parámetros por cada tamaño de ventana se ha marcado en negrita.

La Tabla 8.3 muestra el tiempo de entrenamiento (en minutos) obtenido para los mejores modelos hasta llegar a la convergencia. En general, CNN obtiene los mejores tiempos de entrenamiento, lo cual es esperable al ser la arquitectura más sencilla.

En ETTh1, CNN tiene los mejores tiempos para todos los tamaños de ventana, excepto para el mayor, donde NBEATS obtiene tiempos mejores. No parece haber ninguna relación entre el tiempo de entrenamiento y el tamaño de la ventana, excepto en el caso de CNN, donde el tiempo aumenta a medida que aumenta la ventana. SCINet tiene la peor eficiencia para casi todos los tamaños de ventana, lo que parece indicar que el mecanismo utilizado en la arquitectura resulta ineficiente. Sin embargo, SRCNet obtiene mejores resultados que SCINet con más de un 50 % de disminución de tiempo.

En ETTh2, CNN y NBEATS obtienen los mejores tiempos de entrenamiento. CNN parece tener la misma relación entre el tamaño de la ventana y el tiempo de entrenamiento. NBEATS parece mantener un tiempo de entrenamiento similar, excepto para el tamaño de ventana más grande, mientras que los otros modelos no muestran ningún patrón respecto al tamaño de la ventana. SCINet tiene los tiempos de entrenamiento más altos para cada tamaño de ventana y ha aumentado considerablemente con respecto a los tiempos obtenidos en ETTh1. Sin embargo, SRCNet no muestra ningún incremento considerable excepto para un tamaño de ventana de 24 horas.

La Tabla 8.4 muestra el número de parámetros de los mejores modelos. En este caso, ningún modelo requiere de más o menos parámetros aparentemente.

En ETTh1, el modelo CNN parece necesitar más parámetros a medida que aumenta el tamaño de la ventana, excepto para la ventana mayor. SCINet necesita menos parámetros que cualquier otro método para obtener la mejor

Conjunto	Tamaño de ventana	CNN	HLNet	NBEATS	SCINET	SRCNet
ETTh1	24	**2497**	11081	5760	1260	7538
	48	2825	11081	**1440**	3196	5555
	72	8057	**1265**	108288	4220	8676
	96	8633	105065	**4416**	12188	12004
	120	6569	11279	29952	**4620**	34738
ETTh2	24	**1041**	38635	89856	7500	7307
	48	15673	38700	31104	11036	**6019**
	72	**3609**	38700	35712	8652	6515
	96	12729	38700	**6624**	18460	14875
	120	**4249**	38700	42240	12764	36644

Tabla 8.4: Número de parámetros obtenido para la mejor arquitectura (según MAE) por cada tamaño de ventana y conjunto de datos. La arquitectura con menor número de parámetros por cada tamaño de ventana se ha marcado en negrita.

eficacia con un tamaño de ventana de 120. Además, SRCNet necesita más parámetros para obtener mejores resultados que SCINet pero con menos tamaños de ventana y tiempo de entrenamiento. En general, no parece haber relación entre el número de parámetros y el tiempo necesario para converger.

En ETTh2, SRCNet parece tener menos parámetros que SCINet obteniendo mejores resultados con menos tiempo de entrenamiento. En general, CNN y SRCNet son los métodos que necesitan menos parámetros. Además, obtener la mejor eficacia parece no implicar necesariamente tener más parámetros en general.

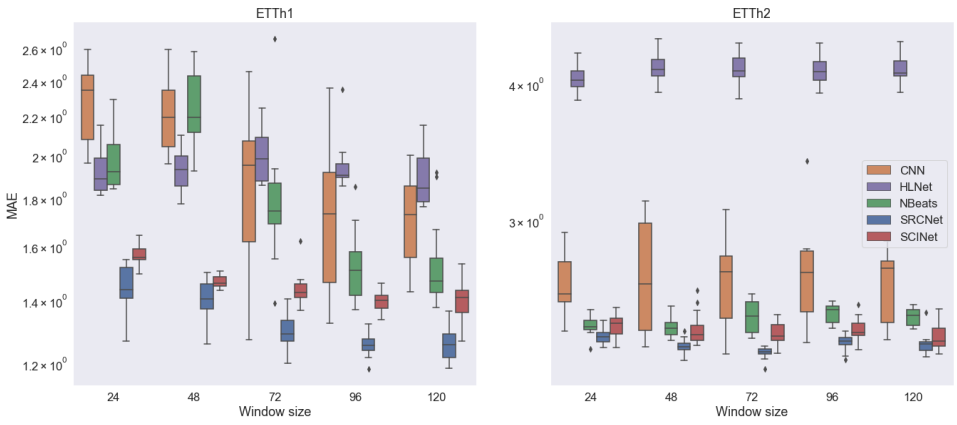


Figura 8.4: MAE por cada modelo y tamaño de ventana.

La Figura 8.4 representa el MAE de todos las arquitecturas entrenadas para los conjuntos de datos ETTh1 y ETTh2. Los gráficos de caja y bigotes (*boxplot*) representan la distribución del MAE obtenido para cada combinación de hiperparámetros durante la búsqueda en rejilla.

En el conjunto de datos ETTh1, el modelo CNN obtiene el mayor rango

intercuartílico con una mediana decreciente a medida que aumenta el tamaño de la ventana. HLNet tiene un rango intercuartílico menor, pero parece que el modelo no mejora con un aumento en el tamaño de la ventana mientras que la mediana se mantiene casi en el mismo rango independientemente del tamaño de la ventana. NBeats parece mejorar la eficacia a medida que aumenta el tamaño de la ventana, pero los resultados siguen siendo peores que los obtenidos con SRCNet que tiene mejores resultados en general, especialmente para tamaños de ventana mayores. SCINet parece tener la mejor mediana y rango intercuartílico para tamaños de ventana pequeños, pero SRCNet mejora la mediana usando tamaños mayores.

En el conjunto de datos ETTh2, los errores son mayores que en ETTh1, tal y como se menciona en la Tabla 8.2. HLNet obtiene la peor eficacia por un amplio margen, independientemente del tamaño de la ventana. Los mejores modelos son NBEATS, SCINet y SRCNet por un margen bajo, pero la mediana de NBEATS parece disminuir a medida que aumenta el tamaño de la ventana. Los tamaños de ventana más grandes parecen introducir ruido que perjudica la eficacia. En general, SRCNet obtiene el mínimo global, seguido de SCINet, sin que exista una fuerte relación entre el tamaño de la ventana y el error.

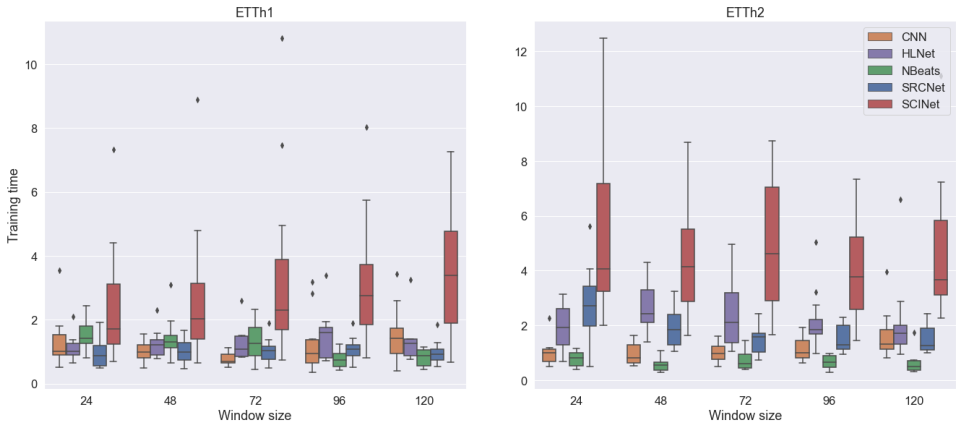


Figura 8.5: Tiempos de entrenamiento (en minutos) por cada arquitectura y tamaño de ventana.

La Figura 8.5 muestra el tiempo de entrenamiento (en minutos) obtenido para todos los modelos entrenados.

En los conjuntos de datos ETTh1, la arquitectura CNN obtiene los tiempos de entrenamiento más bajos para tamaños de ventana de 24, 48 y 72. Sin embargo, para tamaños de ventana mayores el tiempo de entrenamiento aumenta considerablemente. HLNet presenta el mismo comportamiento que CNN, aumentando el tiempo de entrenamiento para tamaños de ventana mayores. NBEATS obtiene un comportamiento diferente al de CNN y

HLNet ya que el tiempo de entrenamiento es menor para tamaños de ventana mayores. SRCNet no obtiene mejores tiempos de entrenamiento que los modelos más rápidos como CNN o NBEATS sino que se mantiene casi constante independientemente del tamaño de la ventana.

En el conjunto de datos ETTh2, NBEATS obtiene la mediana de tiempo de entrenamiento más baja para todos los tamaños de ventana, seguido de CNN. HLNet y SRCNet parecen tener mayores tiempos de entrenamiento para los tamaños de ventana de 24 y 48 y luego se reducen ligeramente.

En general, el tamaño de la ventana de entrenamiento no tiene una gran influencia en el tiempo de modelado hasta la convergencia. SCINet tiene los tiempos de entrenamiento más elevados en comparación con todos los demás modelos independientemente del tamaño de ventana.

8.5.4. Análisis de la salida

Con el objetivo de comprender mejor el comportamiento aprendido por SRCNet, analizamos las predicciones parciales obtenidas por la arquitectura. El análisis tiene en cuenta la mejor combinación de hiperparámetros obtenido en el conjunto de datos ETTh1 donde las predicciones parciales se han obtenido del conjunto de prueba.

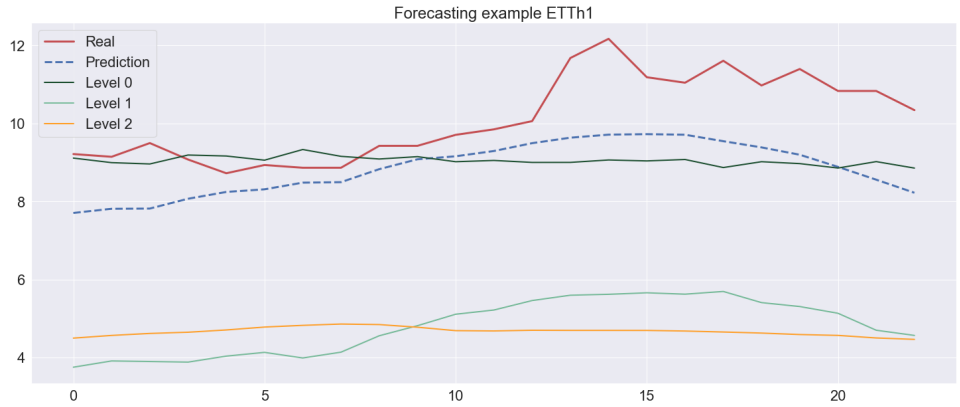


Figura 8.6: Salidas parciales producidas por SRCNet para una ventana dentro del conjunto de prueba en ETTh1. Se incluye la predicción final calculada a partir de la suma de todas las predicciones parciales.

La Figura 8.6 muestra el comportamiento real de la serie temporal (rojo), la predicción final de SRCNet (azul punteado), y las predicciones parciales obtenidas (verde oscuro, verde claro y amarillo) representados como nivel 0, 1 y 2.

Aparentemente, el primer bloque parece predecir una línea de base para la predicción con una media de los eventos futuros. El segundo bloque se centra en obtener la forma general de la predicción, centrándose en los picos

o zonas con mayor valor. Por último, la última predicción parcial parece centrarse en los valles.

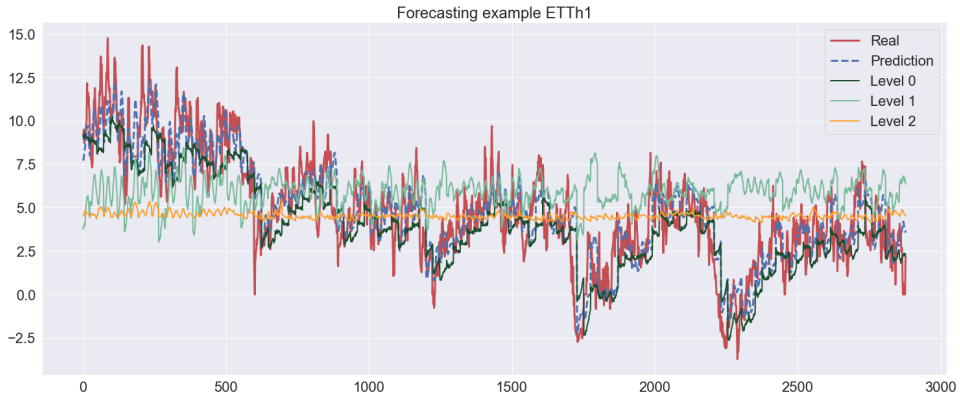


Figura 8.7: Salidas parciales producidas por SRCNet para todo el conjunto de prueba en ETTh1. La predicción incluye la predicción final calculado a partir de la suma de todas las predicciones parciales.

En la predicción global, tal y como se observa en la Figura 8.7 el primer bloque obtiene una estimación inicial de las predicciones sin centrarse demasiado en los picos. El segundo y tercer bloque parecen aprovechar esta predicción de base, centrándose en los picos de la predicción y zonas con mayor varianza. Por último, el segundo bloque parece obtener una representación global de los picos mientras que el último bloque parece centrarse en pequeños cambios.

8.5.5. Tests estadísticos

En esta sección, se aplica un test estadístico a todos los resultados obtenidos durante la búsqueda en rejilla para respaldar las conclusiones obtenidas. El test aplicado es la versión bayesiana del test de rango de signos de Wilcoxon [160] propuesta por Alessio Benavoli et al. [14].

Como el método seleccionado es una prueba pareada no paramétrica, necesitamos obtener cada par de resultados para cada combinación de hiperparámetros en ambos conjuntos de datos. Sin embargo, cada modelo tiene un número diferente de hiperparámetros, donde algunos de ellos son compartidos y otros no. Para obtener los pares, sólo se han seleccionado los hiperparámetros compartidos y se ha obtenido los mejores hiperparámetros no compartidos de entre todas las combinaciones.

La Tabla 8.5 muestra los resultados de la prueba bayesiana aplicada al WAPE. La probabilidad de que el modelo de referencia obtenga mayor WAPE se representa como by p(reference>SRCNet), mientras que el caso contrario se representa como p(SRCNet>reference). Por último, la probabi-

Conjunto de datos	Referencia	p(referencia>SRCNet)	p(Rope)	p(SRCNet>referencia)
ETTh1	CNN	1	0	0
	NBEATS	1	0	0
	HLNet	1	0	0
	SCINet	1	0	0
ETTh2	CNN	1	0	0
	NBEATS	1	0	0
	HLNet	1	0	0
	SCINet	1	0	0

Tabla 8.5: Resultados del test Bayesiano comparando el MAE de todos los modelos como referencia con SRCNet.

lidad de no encontrar diferencias significativas se representa como p(Rope). El umbral utilizado en la prueba estadística fue un valor del 0,1 % del error relativo, lo que significa que los resultados cuya diferencia de error es inferior al 0,1 % se consideran similares. La prueba muestra una probabilidad del 100 % en todos los casos, lo que indica que SRCNet es mejor que los modelos de referencia, obteniendo un menor error en general.

Dataset	Reference	p(reference>SRCNet)	p(Rope)	p(SRCNet>reference)
ETTh1	CNN	0	1	0
	NBEATS	0.31	0.69	0
	HLNet	0.25	0.75	0
	SCINet	1	0	0
ETTh2	CNN	0	1	0
	NBEATS	0	1	0
	HLNet	0.02	0.98	0
	SCINet	1	0	0

Tabla 8.6: Resultados del test Bayesiano comparando la eficiencia de todos los modelos como referencia con SRCNet.

La Tabla 8.6 muestra la prueba bayesiana aplicada a la métrica de eficiencia. Al igual que el caso anterior, la probabilidad de que el modelo de referencia obtenga mayor tiempo de entrenamiento se representa como p(reference>SRCNet), mientras que el caso contrario se representa como p(SRCNet>reference). Por último, la probabilidad de no encontrar diferencias significativas se representa como p(Rope). En este caso, el umbral seleccionado fue de 30 segundos, lo que significa que los tiempos de entrenamiento con diferencias inferiores a 60 segundos se consideran similares. Los resultados de CNN, NBEATS y HLNet muestran que son similares a SRCNet en general para todos los conjuntos de datos. A pesar de que SCINet tiene los resultados de eficacia más cercanos a SRCNet, la eficiencia es claramente peor en ambos conjuntos de datos.

8.6. Conclusiones

SRCNet ha demostrado obtener una eficacia comparable con las mejores arquitecturas de la literatura con una gran mejora de la eficiencia. La eficiencia se obtuvo mediante el diseño utilizando un mecanismo más rápido basado en la descomposición de la ventana de entrada. La descomposición obtenida fue diferente a la habitual de tendencia y estacionalidad, reforzando la hipótesis de que una reducción del sesgo en la arquitectura puede mejorar la eficacia ya que la libertad permite obtener una mejor descomposición de los datos. Además, la introducción del *Smooth Residual Block* parece haber introducido la cantidad necesaria de sesgo para mejorar la eficacia sin empeorar la eficiencia.

Capítulo 9

Feature-Aware Drop Layer (FADL)

Se dice que el tiempo es un gran maestro; lo malo es que va matando a sus discípulos.

Hector Berlioz.

9.1. Resumen

En la presente propuesta se describe un método de selección de atributos no paramétrico integrado dentro de una red neuronal llamada Feature-Aware Drop Layer o FADL. La selección de atributos se realiza mediante una capa adicional que contiene una serie de puertas capaces de eliminar la influencia de ciertos atributos. Esta capa se conecta a cada uno de los atributos de entrada, obteniendo tantas puertas como atributos de entrada contiene la red neuronal. Durante el proceso de entrenamiento, la red neuronal ajusta los pesos de la capa propuesta de la misma forma que cualquier otra capa. Finalmente, la capa obtiene un subconjunto de atributos que son relevantes para la red. La regularización e inicialización cumplen un factor vital al establecer tanto una penalización al número de puertas activas como una hipótesis inicial sin grandes sesgos.

9.2. Introducción

Las redes neuronales han demostrado obtener buenos resultados en varias áreas, sin embargo, aún existen varios problemas bien conocidos que impiden su aplicación en distintos casos. En este documento se ha analizado principalmente el problema de la eficiencia, pero existen otros problemas adicionales. Por ejemplo, la maldición de la dimensionalidad [13] es uno de

estos problemas donde, si los datos contienen una gran cantidad de atributos, son necesarios mucho más datos para obtener buenos resultados. Esto conlleva a un incremento de los recursos necesarios para almacenar y procesar los datos durante el entrenamiento de una red neuronal. Otro ejemplo es la interpretabilidad [170] que impide "entender" los motivos que han llevado a la arquitectura a obtener una salida, reduciendo la confianza en las predicciones obtenidas.

La selección de atributos es una técnica ampliamente usada hoy en día que resuelve muchos de los problemas bien conocidos dentro y fuera del contexto de las redes neuronales, mejorando de alguna manera la eficacia y eficiencia. Tal y como se describió en la Sección 2.4, esta técnica consiste en reducir el conjunto de atributos manteniendo aquellos con relevancia para el problema. Esta técnica es capaz de mejorar tanto la eficacia como la eficiencia de cualquier método y su aplicación puede resultar esencial en diversos ámbitos donde medir cada uno de los atributos requiera de una vasta cantidad de recursos.

Un problema de la selección de atributos es que no es fácilmente aplicable a redes neuronales debido a que los métodos basadas en *Wrapper* son muy ineficientes y los métodos de selección de atributos basados en filtros pueden no representar el comportamiento de la arquitectura. Además, muchas de estos métodos establecen una puntuación que representa el grado de relevancia de los atributos, lo que implica buscar el umbral adecuado para conocer qué atributos mantener y cuáles no. Los métodos que son capaces de integrar la selección de atributos de forma inherente (Integrada) suelen ser las mejores opciones manteniendo todas las ventajas de la selección de atributos sin requerir recursos adicionales. Sin embargo, no todos los algoritmos de aprendizaje automático poseen un proceso de selección de atributos integrado.

Por lo tanto, el objetivo de esta propuesta es integrar la selección de atributos dentro de una red neuronal de una forma sencilla y eficiente, obteniendo sus ventajas aun a costa de perder un poco la eficacia. Esta propuesta se encuentra en un contexto donde la eficiencia es más importante que la eficacia. Este caso es común en problemas donde la arquitectura debe ser integrada en dispositivos con una capacidad computacional limitada, donde la obtención/administración de los datos sea costosa, donde no exista un conocimiento profundo del problema, entre otros problemas.

En este capítulo se describe la propuesta *Feature-Aware Drop Layer* o, de aquí en adelante, FADL. Esta propuesta se aplica tanto a tareas de clasificación, como predicción de series temporales. Esta propuesta realiza una selección de atributos a través de una capa adicional inspirada en la capa *Dropout* [141], que asocia un peso a cada atributo de entrada y se integra dentro del proceso de aprendizaje de la arquitectura como una capa adicional. Gracias a esto, la capa es capaz de aprender los atributos relevantes

considerados por la arquitectura de forma automática en forma de una máscara que hace de filtro. Además, esta capa obtiene todas las ventajas de la selección sin necesidad de reentrenar la arquitectura varias veces.

En la Sección 9.3 se discute la hipótesis principal. En la Sección 9.4 se describe la metodología usada para integrar la capa en la arquitectura. La Sección 9.5 muestra los resultados obtenidos a través de la experimentación realizada sobre los distintos conjuntos de datos. La Sección 9.6 comenta las principales conclusiones obtenidas.

9.3. Hipótesis

En esta sección se define la hipótesis principal en la que se basa la implementación de la propuesta y sus implicaciones.

En teoría, las redes neuronales son capaces de reducir el impacto de los atributos irrelevantes modificando sus pesos. Sin embargo, esta selección puede dificultarse a causa de la cantidad de datos, complejidad del problema, sobreajuste, entre otras causas. Además, varios estudios apoyan la idea de que la selección de atributos es beneficiosa no solo para la eficiencia, también para la eficacia y que las arquitecturas fallan al realizar una selección de atributos [85][119].

Hipótesis 6 (H6): *Es posible facilitar el proceso de selección de atributos estableciendo unos pesos que cumplan el rol de puertas que permitan o no la propagación de información.*

En definitiva, esta hipótesis abre la posibilidad de que la red neuronal realice la selección de atributos de una forma más sencilla y directa. De esta manera, durante el entrenamiento, la selección se centrará en modificar tantos pesos como atributos de entrada, en lugar de los múltiples pesos de toda la arquitectura.

Las puertas deben funcionar de forma binaria, de forma que solo permitan propagar la información de un atributo o eliminen su influencia. De esta manera, la eficacia obtenida de la arquitectura entrenada con la capa FADL representará fielmente la eficacia una vez realizada la selección de atributos.

9.4. Metodología

En esta sección se detalla el funcionamiento interno de la presente propuesta. Para ello, en la Sección 9.4.1 se detalla la nomenclatura usada durante las figuras y la descripción de las operaciones realizadas. En la Sección 9.4.2 se realiza la descripción paso a paso del funcionamiento de la propuesta. Finalmente, la Sección 9.4.3 presenta la inicialización y regularización requerida por la propuesta para asegurar el correcto funcionamiento.

9.4.1. Nomenclatura

- Sea D el número de atributos de la entrada de la red.
- Sea M el tamaño de la ventana de entrada. Hay que tener en cuenta que $M > 1$ solo si la entrada se trata de una serie temporal.
- Sea N el número de salidas obtenida por la red.
- Sea X^{MxD} la entrada de la red con tamaño de ventana M y D características.
- Sea H la función escalón de Heaviside.
- Sea W_L^D los pesos de la propuesta que cumplen el rol de puertas. Tenga en cuenta que cada peso se relaciona únicamente con una característica.
- Sea O^N la predicción obtenida mediante la red neuronal con N salidas.

9.4.2. Descripción

Formalmente, la transformación realizada en la capa FADL para llevar a cabo la selección puede ser definida como:

$$FADL(X^{MxD}) = H(W_L^{1xD}) \circ X^{MxD} \,. \tag{9.1}$$

Además, la función escalón de Heaviside (H) se aplica a todos los pesos $j \in 1..D$ de forma independiente de la siguiente forma:

$$H(W_L^j) = \begin{cases} 1, \ if \ W_L^j >= 0{,}5, & (9.2) \\ 0, \ if \ W_L^j < 0{,}5. & (9.3) \end{cases}$$

Es importante tener en cuenta que los pesos de la capa FADL se aplican independientemente a todos los eventos de la ventana de tamaño M. Por lo tanto, cuando una característica sea eliminada en una serie temporal, se eliminará para todos los instantes dentro de la ventana.

La Figura 9.1 muestra un diagrama con una red neuronal totalmente conectada a la que se le ha añadido la capa FADL. Tenga en cuenta que las líneas discontinuas representan la multiplicación elemento a elemento (producto Hadamard) y las líneas sólidas representan la multiplicación matricial entre las salidas y los pesos. Por último, O_i, $i \in [1, H]$, representa la salida de la red neuronal.

La entrada X^{MxD} se conecta con la capa FADL mediante un producto de Hadamard, conectando cada entrada con cada una de las puertas. Estas puertas, son el resultado de aplicar la función escalón de Heaviside a los pesos $H(W_L^D)$. De esta manera, la red establecerá un valor de cero para la puerta correspondiente al atributo que se desea eliminar. Las capas posteriores de

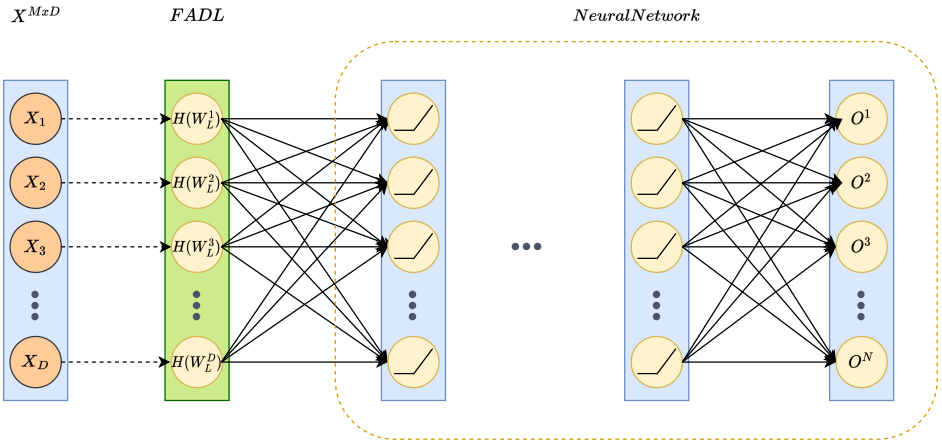


Figura 9.1: La capa FADL propuesta aplicada a una red neuronal totalmente conectada (se supone $M = 1$).

la red neuronal recibirán un valor de cero para ese atributo, eliminando su influencia. La red neuronal no sufre ningún cambio adicional y realiza las transformaciones habituales hasta producir la predicción.

Durante la fase de entrenamiento, el algoritmo de descenso de gradiente modificará los pesos incluyendo a la capa FADL. Sin embargo, al eliminar un atributo, el algoritmo de retropropagación establecerá que la contribución de los pesos relacionados con el atributo eliminado es nula. Por lo tanto, las puertas representan de una forma más directa la selección de los atributos aprendida por la red neuronal.

Las ventajas en cuanto a la eficacia son considerables considerando que, a diferencia del método *wrappper*, no es necesario repetir el proceso de entrenamiento completamente al integrar la selección de forma inherente dentro de la red neuronal. Sin embargo, es posible que el proceso de entrenamiento se vea afectado de forma considerable. Esto se debe a que, al eliminar un atributo, todos los pesos de la red neuronal deben ser reajustados por la eliminación de un atributo que posteriormente estaba presente. Este impacto es analizado en la Sección 9.5.

Para analizar el comportamiento de la capa FADL, consideraremos dos escenarios extremos:

- El primer escenario considera que todas las puertas permiten propagar la información de todos los atributos. En este caso, FADL no tendrá ningún impacto en el comportamiento de la red neuronal más allá de añadir varios pesos adicionales que pueden afectar el proceso de entrenamiento. Además, en este caso FADL no aportará demasiada información sobre el comportamiento aprendido por la red neuronal más allá de que todos los atributos son relevantes. La eficiencia tampoco

se verá mejorada, ya que la capa FADL no considera la eliminación de ningún atributo.

- El segundo escenario considera que las puertas no permiten a ningún atributo propagar la información hacia la red neuronal. Este escenario considera que ninguno de los atributos usados en la entrada es relevante para el problema que se desea solucionar. Este escenario es improbable en la práctica. Sin embargo, es posible que una mala configuración de la red neuronal, una cantidad de datos insuficiente o unos atributos con poca correlación con el objetivo puedan afectar a este hecho.

El segundo escenario puede controlarse a priori dado que se tratan de problemas fuera del contexto de nuestra propuesta. Sin embargo, el primer problema puede ser resuelto usando una regularización sobre la capa FADL. La siguiente sección describe el proceso de regularización e inicialización realizado.

9.4.3. Inicialización y regularización

La inicialización y regularización de los pesos es un aspecto fundamental en FADL. En esta sección se analizarán los motivos y las consecuencias de la inicialización y regularización realizada.

Los pesos iniciales pueden verse como la hipótesis inicial de la relevancia que tienen los atributos en FADL. Unos pesos muy grandes introducen un sesgo indicando que los atributos deben estar presentes. Como consecuencia, durante el proceso de entrenamiento, se necesitará más iteraciones para ajustar los pesos hasta reducir sus valores. Lo mismo ocurre con valores muy bajos, sesgando una selección donde ningún atributo es relevante. En FADL, no se impone ningún sesgo demasiado fuerte sobre los pesos iniciales. Los pesos son inicializados para que inicialmente todos los atributos sean relevantes. Sin embargo, los pesos se mantienen cercanos a cero para que FADL pueda eliminar la influencia de un atributo con facilidad. Por estas razones, todos los pesos son inicializados a un valor de 0,001.

La regularización de los pesos puede verse como la tolerancia a error permitida por la red a la hora de añadir una nueva característica. Lasso [151] es la regularización usada en cada una de las puertas en FADL. Hay que tener en cuenta que la regularización se aplica a las puertas, y no sobre los pesos de FADL. La regularización penaliza con un valor constante por cada uno de los atributos seleccionados, independientemente del valor que contenga el peso. De esta manera, a menos atributos se seleccionen, menor será la penalización recibida por la regularización.

La regularización es el mecanismo por el que evitamos uno de los escenarios extremos mencionados en la sección anterior. En caso de seleccionar todos los atributos la penalización será máxima, por lo que se evitará este

escenario a no ser que el error introducido sea demasiado bajo. Otro posible escenario que se desea evitar consiste en que una solución con F características tiene el mismo error que una solución con P características. Si consideramos $F < P$, gracias a la regularización, la primera solución será escogida sobre la segunda. De esta manera, se obtendrá un error similar con un número menor de características.

9.5. Resultados

En esta sección se analizarán los resultados obtenidos tras evaluar una red neuronal con FADL sobre un conjunto de datos en un marco de experimentación establecido. En primer lugar, se describirán los datos escogidos y el motivo. Tras ello, se detallará el marco experimental usado para evaluar la propuesta. Por último, se analizarán y discutirán los resultados obtenidos.

Todos los resultados se han analizado bajo las mismas condiciones en cuanto al hardware y software utilizado, con el objetivo de asegurar una comparación justa entre las distintas arquitecturas. Los componentes principales del hardware son: un procesador AMD Ryzen 7 5800X 3.8GHz y 32 GB de RAM.

9.5.1. Conjuntos de datos

La Tabla 9.1 muestra un resumen de los conjuntos de datos usados durante la experimentación. En total, se tratan de tres conjuntos de datos relacionados con el sector de la salud, uno relacionado con desastres naturales, uno con detección de fallo y cuatro relacionados con predicción de series temporales.

	#Características	#Instancias	Tamaño ventana	#Salidas	Tipo
BreastCancer [161]	10	570	1	2	Clasificación
Heart [113]	19	319,796	1	2	Clasificación
TCGA [43]	20,531	802	1	5	Clasificación
Earthquake [26]	38	260,602	1	3	Clasificación
Waterpump [148]	39	59,401	1	3	Clasificación
TorneoCO [55]	7	4,017	24	1	Regresión
TorneoNO_2 [55]	7	4,017	24	1	Regresión
TorneoO_3 [55]	7	4,017	24	1	Regresión
TorneoPM_{10} [55]	7	4,017	24	1	Regresión

Tabla 9.1: Número de características, número de instancias, número de clases/tamaño del horizonte, tamaño de la ventana y tipo de problema por cada conjunto de datos.

El conjunto de datos BreastCancer [161] contiene 10 características obtenidas de una imagen digital mediante la aspiración de aguja fina de la masa mamaria. El objetivo es clasificar entre tumores benignos y malignos. En

este caso, la selección de atributos resulta de interés para comprender las características que determinan la predicción de una clase y ahorrar el proceso de recopilación características menos relevantes.

El conjunto de datos Heart [113] contiene 19 características obtenidas del *Behavioral Risk Factor Surveillance System* (BRFSS) realizada en Estados Unidos a través de entrevistas telefónicas sobre diferentes factores de riesgo para la salud. El objetivo es clasificar la posibilidad de alguna enfermedad coronaria (*coronary heart disease*) o infarto de miocardio (*myocardial infarction*). Además de comprender el impacto de ciertas preguntas, la selección de atributos tiene especial interés en los conjuntos de datos basados en entrevistas. Una entrevista con menos preguntas aumenta la probabilidad de que el usuario entrevistado no se canse y que el entrevistador tenga la posibilidad de entrevistar a más personas en el mismo tiempo.

El conjunto de datos extraído de la base de datos TCGA [43] contiene 20531 características correspondientes a niveles de expresión genética tomados usando la plataforma Illumina HiSeq. Las expresiones genéticas son tomadas de forma aleatoria sobre una serie de pacientes con diferentes tipos de tumores. El objetivo es clasificar la existencia de cinco tipos diferentes de tumores: *Breast invasive carcinoma* (BRCA), *Kidney renal clear cell carcinoma* (KIRC), *Colon adenocarcinoma* (COAD), *Lung adenocarcinoma* (LUAD) and *Prostate adenocarcinoma* (PRAD). La selección de atributos resulta de interés, dado que conocer los genes que pueden relacionarse con la aparición con ciertos tipos de cáncer puede ser beneficioso para comprender y estudiar tratamientos relacionados.

El conjunto de datos Earthquake [26] contiene 38 características tomadas mediante una serie de encuestas realizadas por *Kathmandu Living Labs* y el *Central Bureau of Statistics* en Nepal. Las entrevistas estaban relacionadas con el impacto del terremoto Gorkha en 2015 a partir de las características de diferentes edificios. El objetivo es clasificar el nivel de daños realizado sobre los edificios en tres niveles: bajo, medio y alto. Al igual que en otros conjuntos de datos donde se realizaba encuestas, los beneficios de la selección son la reducción de preguntas y la comprensión de las preguntas más relevantes.

El conjunto de datos WaterPump [148] contiene 39 características de fuentes de agua en Taarifa. Las características fueron obtenidas del Tanzania Ministry of Water. El objetivo es clasificar el estado de la fuente de agua entre: funcional, funcional con reparación y no funcional. La selección de atributos puede ayudar a establecer una relación entre las características de la fuente, su localización y la clase objetivo.

Los conjuntos de datos TorneoCO, TorneoNO_2, TorneoO_3 y TorneoPM_{10} pertenecen al mismo conjunto de datos, pero con distinto objetivo. Los datos son una serie temporal tomada en la zona de Torneo en Sevilla a través de un sensor, componiendo siete características climatológicas diferentes. Los sensores realizaban mediciones sobre cuatro contaminantes distintos: Monó-

xido de carbono (CO), Dióxido de nitrógeno (NO_2), Ozono (O_3) y Partículas sólidas de una micra (PM_{10}). Cada contaminante ha sido establecido como objetivo de la predicción usando como entrada las otras siete características.

9.5.2. Marco experimental

Los datos han sido preprocesados de la misma forma siguiendo un proceso estándar. Algunas características de algunos conjuntos de datos representaban identificadores únicos, por lo que han sido eliminados. Las características categóricas ha sido transformadas usando la técnica conocida como *one-hot encoding* [57]. En caso de contener datos perdidos, se han completado usando el valor medio para características reales y la moda para características categóricas. Se ha estandarizado todas las características excepto las categóricas, para ayudar a la convergencia de la red neuronal.

En cuanto a la división de los datos, se ha usado el 70 % de los datos para el entrenamiento, el 10 % para la validación y el 20 % restante para el testeo. La división ha sido escogida de forma aleatoria para los conjuntos de datos de tipo clasificación. Para los conjuntos de datos de tipo regresión, el conjunto de entrenamiento y validación corresponde a datos más antiguos que los datos usados para el testeo. Además, se ha llevado a cabo un proceso de ventaneo se ha llevado a cabo usando 24 eventos para la ventana con el objetivo de predecir un evento en el futuro.

Todos los conjuntos de datos han sido entrenados y evaluados usando la misma arquitectura. Esta arquitectura ha sido una red totalmente conectada con una capa de entrada, dos capas intermedias y dos capas de salida. La primera capa intermedia tiene tantas neuronas como el 50 % de las características de entrada, mientras que la segunda capa contiene un 25 % de las características de entrada. La función de activación usada en las capas intermedias es la función $ReLU$, mientras que en la capa de salida se usa la función *Softmax* o identidad para clasificación y regresión respectivamente. El objetivo de esta arquitectura es servir como una referencia sin necesidad de usar muchos recursos gracias a su simplicidad. Los hiperparámetros se mantendrán fijos durante toda la experimentación para centrarse en el análisis de la selección de atributos.

El proceso de entrenamiento se ha realizado durante 100 épocas usando un factor de aprendizaje de 0.001. El optimizador escogido ha sido *Adam* [80] y se ha usado la parada temprana (*early stopping*) para terminar el proceso de entrenamiento si no se logra mejorar el error sobre el conjunto de validación en 10 épocas al menos. La Tabla 9.2 contiene un resumen de los hiperparámetros usados durante el proceso de experimentación de la arquitectura.

FADL ha sido comparada con cuatro estrategias, tres de ellas correspondientes a técnicas de selección de atributos basados en Filtro. Al igual que

Hiperparámetro	**Valor**
Épocas máximas	100
Tamaño del lote (Batch size)	32
Optimizador	Adam
Factor de aprendizaje	0.001
Parada temprana (early stopping)	10
Número de capas ocultas	2
Número de neuronas en cada capa en función al número de atributos de entrada	50 %, 25 %
Función de activación capas intermedias	$ReLU$

Tabla 9.2: Resumen de los hiperparámetros usados durante la experimentación realizada en la propuesta.

nuestra propuesta, estas técnicas no incrementan el coste computacional conforme aumenta el número de características. Las estrategias escogidas para realizar la comparación son:

- Sin selección: Esta estrategia no usa ningún tipo de selección, su utilidad es comparar si la selección de atributos es beneficiosa o no.
- Selección por correlación: Este método usa la correlación con el objetivo para eliminar las características por debajo de cierto umbral.
- Selección lineal: Este proceso usa un modelo lineal para establecer una ponderación a cada característica, posteriormente, se establece un umbral para filtrar aquellas características por debajo de cierto umbral.
- Selección por información mutua: Este algoritmo usa la información mutua para establecer una valoración sobre cada característica y filtrar aquellas que estén por debajo de cierto umbral.

Todas las estrategias seleccionadas necesitan establecer un umbral para filtrar los atributos considerados irrelevantes. Dado que este hiperparámetro puede llegar a influir enormemente en el proceso de selección, se ha llevado a cabo una búsqueda en rejilla por cada estrategia de selección de atributos usada en la comparación. Se ha usado el rango [60 %, 95 %] para los umbrales con pasos del 5 %. De esta manera, el mínimo número de características cuya suma relativa al total esté por encima del umbral serán las seleccionadas.

Las métricas usadas para evaluar cada una de las estrategias sobre todos los conjuntos de datos usando la arquitectura descrita son las mismas para la eficiencia y diferentes para la eficacia.

Las métricas seleccionadas para medir la eficiencia son el tiempo de entrenamiento y el número de características obtenidas por la estrategia . Hay que

tener en cuenta que, en FADL y la estrategia sin selección, no es necesario repetir el proceso de entrenamiento para evaluar de nuevo la arquitectura. Por lo tanto, los tiempos de entrenamiento para las estrategias de selección con umbral serán sumados.

En cuanto a la eficacia, el error cuadrático medio se ha usado para los conjuntos de datos de tipo regresión, mientras que el F1, *Recall*, *Precission* y *Accuracy* [65] se ha usado para los conjuntos de datos de clasificación.

9.5.3. Discusión de los resultados

En esta sección se analiza la eficacia y eficiencia de la propuesta con cada una de las estrategias establecidas en el marco experimental.

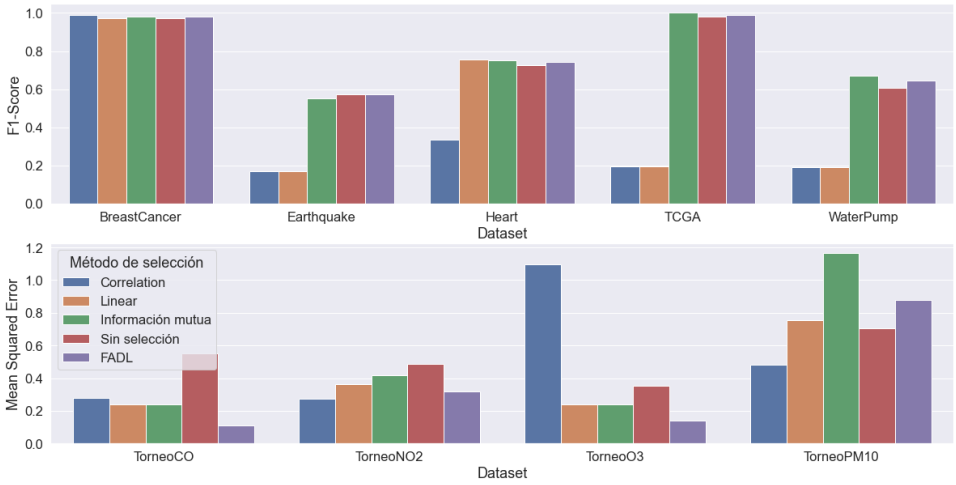


Figura 9.2: Resultados de eficacia para los conjuntos de clasificación (F1-Score) y regresión (MSE).

La Figura 9.2 muestra la eficacia obtenida por cada una de las estrategias seleccionadas. El análisis se realizará por separado para los conjuntos de datos de clasificación y regresión. Para las estrategias que requieren optimizar el umbral, se ha mostrado la mejor métrica de eficacia que se ha obtenido de entre todos los experimentos.

En los conjuntos de datos de clasificación, la estrategia de selección usando la información mutua parece obtener los mejores resultados en los conjuntos: Heart, TCGA y Waterpump. La estrategia de selección mediante correlación obtiene los mejores resultados en el conjunto de datos BreastCancer. FADL obtiene los mejores resultados junto a la estrategia sin selección en el conjunto de datos EarthQuake. Se observa que la estrategia de selección mediante información mutua y FADL son las más estables, ya que obtienen buenos resultados en todos los conjuntos. Sin embargo, las estrategias de selección basadas en un modelo lineal y correlación muestran unos resultados

muy deficientes en los conjuntos Earthquake, TCGA y Waterpump. Aparentemente, FADL es la única técnica que supera o iguala a la estrategia sin selección de características en todos los conjuntos de datos.

En los conjuntos de datos de regresión, los mejores resultados son obtenidos por la estrategia de selección mediante correlación y FADL en general. En los conjuntos de datos TorneoCO y TorneoO_3, FADL muestra una reducción del error considerable. En el conjunto de datos TorneoNO_2, la estrategia mediante correlación obtiene los mejores resultados seguido por FADL. El conjunto de datos TorneoPM_{10} parece más complejo, dado que todas las estrategias de selección empeoran a la estrategia sin selección menos usando correlación. En otros conjuntos de datos, todas las estrategias de selección mejoraban a la estrategia sin selección menos la correlación en TorneoO_3.

Figura 9.3: Número de características y tiempo de entrenamiento para las mejores arquitecturas. Tenga en cuenta que la escala del eje vertical es exponencial.

La Figura 9.3 contiene los resultados de todos los conjuntos de datos representando las dos métricas usadas para medir la eficiencia. Para las estrategias que requieren la optimización de un umbral, se ha usado la suma de los tiempos (en segundos) y el número de características que mejor eficacia obtuvo correspondiente a un umbral.

Si se analizan el número de características junto a la Tabla 9.2, se puede observar que aquellas estrategias de selección que peor funcionaban obtienen una sola característica. Esto quiere decir que una sola característica obtiene la mayoría (más del 95 %) de importancia, lo cual lleva a resultados por debajo de la eficacia que obtendría si se seleccionaran más características. Estos resultados, refuerzan la idea de la importancia que tiene una correcta selección de umbral.

En general, FADL obtiene el menor número de características en los

conjuntos de datos a la vez que mantiene unos resultados competitivos. En los conjuntos de datos de regresión, la estrategia de selección basada en la correlación obtiene menos características en TorneoNO_2 y TorneoPM_{10}, mientras que FADL obtiene menos características en TorneoO_3 y TorneoCO.

En términos de tiempo de entrenamiento, todas las estrategias incrementan el tiempo de entrenamiento comparado con la estrategia sin selección menos en TorneoO_3 donde FADL es más eficiente. Los casos en los que se seleccionaban una sola característica y se obtenía una mala eficacia obtienen unos tiempos bajos en general. En los casos donde la selección dio lugar a unos resultados competitivos, FADL obtiene los mejores tiempos de ejecución. En general, el incremento de tiempo en FADL no es muy grande en los conjuntos de datos de clasificación. Sin embargo, el tiempo de entrenamiento en los conjuntos de tipo regresión parece incrementar en mayor medida. Aparentemente, FADL puede llegar a afectar al proceso de entrenamiento hasta cierta medida. Aun así, los tiempos son bastante menores comparados con otras propuestas al no realizar una búsqueda donde sea necesario entrenar y evaluar el modelo varias veces.

9.6. Conclusiones

FADL ha sido evaluada sobre diversos conjuntos de datos para la predicción de series temporales y clasificación mostrando una eficiencia y eficacia competitivas con otras estrategias de selección. Los resultados parecen indicar que la propuesta obtiene una selección donde la eficacia no se ve afectada en gran medida en la mayoría de los conjuntos de datos. La propuesta muestra los resultados más estables comparado con otras estrategias de selección. Por último, se ha demostrado los beneficios de la selección de atributos en base a la eficacia aplicado a redes neuronales en problemas de clasificación y regresión.

Parte IV

Conclusiones y Trabajos Futuros

Capítulo 10

Conclusiones

Mathematical science shows what is. It is the language of unseen relations between things. But to use and apply that language, we must be able to fully to appreciate, to feel, to seize the unseen, the unconscious.

Ada Lovelace.

Las series temporales presentan uno de los tipos de datos más comunes y complejos de tratar, con varios beneficios para la sociedad en el ámbito médico, social, económico, entre otros. La predicción es uno de los retos que aportan un mayor beneficio y, a su vez, presenta mayor complejidad a nivel de eficacia y eficiencia. A nivel de eficacia, no es posible determinar un sólo método que domine a los demás en cuanto a calidad de resultados. Además, la gran diversidad de necesidades dificulta la selección de una solución única genérica que se adapte a cualquier tipo de predicción o serie temporal como: predicción/serie multivariante, predicción multipaso, predicción probabilística, entre otros. A nivel de eficiencia, es necesario desarrollar métodos que sean capaces de escalar a grandes volúmenes de datos. De esta manera, es posible adaptar el método a entornos con recursos computacionales limitados, facilitar la experimentación de diferentes configuraciones, reajustar el método sin requerir muchos recursos, entre otros.

Una de las posibles soluciones para enfrentar los diferentes retos que presenta la predicción de series temporales pasa por desarrollar nuevas arquitecturas que sean capaces de mejorar u obtener una eficacia similar a las actuales, mejorando la eficiencia. De entre todas las técnicas dentro del aprendizaje automático, las redes neuronales han sido las seleccionadas para explorar nuevas arquitecturas que permitan afrontar los retos. Esto se debe principalmente a que, actualmente, han demostrado una buena eficacia en gran cantidad de escenarios. Además, normalmente, requieren grandes recursos computacionales para funcionar correctamente por lo que resulta indispensable reducir el consumo de estos lo máximo posible.

En esta tesis, se han presentado cuatro arquitecturas enmarcadas dentro del contexto del *deep learning* que mejoran o igualan la eficacia a métodos dentro del estado del arte obteniendo una mejora de eficiencia. Tres de estas arquitecturas se relacionan con el proceso de aprendizaje, escalado y división de responsabilidades dentro del diseño de la arquitectura, mientras que la última integra un proceso de selección de características en la arquitectura.

HLNet es la primera arquitectura propuesta. El objetivo es igualar la eficacia obtenida por LSTM mejorando la eficiencia. Los resultados fueron probados con 13 conjuntos de datos con distintas características y ámbitos de aplicación. Los resultados mostraron que no existía una mejora clara en cuanto a eficacia en la mayoría de los conjuntos de datos. Sin embargo, la eficiencia fue mejorada claramente de forma significativa en la mayoría de los conjuntos de datos probados. Este hecho parece indicar que el sesgo inductivo introducido en el diseño de la arquitectura no suele ayuda a la arquitectura a obtener mejores resultados en la mayoría de conjuntos de datos, en su lugar parece ayudar a converger de forma más rápida.

PHILNet nació como una mejora de HLNet introduciendo una hipótesis adicional. El objetivo seguía siendo el mismo: mejorar la eficiencia obtenida por LSTM igualando su eficacia. Además, se requería analizar el impacto de la nueva hipótesis comparando los resultados obtenidos con HLNet. Los resultados mostraron una mejora de eficiencia mayor que la obtenida con HLNet con unos resultados similares de eficacia. Estos resultados determinan que una optimización conjunta de las distintas jerarquías mejora el tiempo de convergencia aún más que aislando cada nivel.

SRCNet es otra propuesta para algunas de las hipótesis presentadas en HLNet y PHILNet. El objetivo es aplicar este método al área del mantenimiento de transformadores eléctricos, separando la responsabilidad de forma que se descomponga la serie temporal en distintos componentes con el objetivo de mejorar la eficacia y eficiencia. Los resultados mostraron que SRCNet obtenía el mejor balance entre eficiencia y eficacia, obteniendo una eficiencia similar a los métodos más sencillos y rápidos y superando los resultados obtenidos por arquitecturas del estado del arte. Estos resultados parecen indicar que la descomposición propuesta, a la vez que sencilla y fácil de aplicar, parece ayudar a obtener una mejor representación de los datos mejorando la eficacia.

FADL fue la última propuesta y está basada en la selección de atributos. El objetivo es reducir la dimensionalidad del problema durante el proceso de aprendizaje de la red neuronal, reduciendo la cantidad de recursos consumidos e introduciendo cierto grado de interpretabilidad. Fue aplicado a 9 conjuntos de datos dentro del ámbito médico, social y medio ambiental. Se obtuvieron unos mejores resultados que otros métodos de selección de atributos, con un comportamiento bastante estable y un tiempo de convergencia menor en todos los conjuntos de datos. Estos resultados parecen indicar que

es posible realizar una selección de atributos integrando la optimización dentro del proceso de aprendizaje de la red neuronal, sobre todo, en problemas de regresión.

Capítulo 11

Trabajos futuros

So much universe, and so little time.
Terry Pratchett

En general, parece que la exploración de nuevas arquitecturas es una buena forma de mejorar la eficacia y eficiencia. En esta tesis se han aportado cuatro propuestas. Sin embargo, existen una gran variedad de implementaciones y distintas soluciones posibles, tantas como ideas puedan ser formalizadas dentro del diseño de la arquitectura. Existen muchas otras ideas que pueden ser exploradas que ataquen uno o más de los retos presentados por la predicción de series temporales:

- Se implementará una arquitectura que explote las diferentes peculiaridades presentadas por la serie temporal. En ocasiones, una misma serie temporal presenta comportamientos muy dispares tales como fines de semana, festivos, incidentes, entre otros. Usar una arquitectura común para extraer el conocimiento de toda la serie temporal corre el peligro de ajustarse demasiado al caso mayoritario, discriminando las ocasiones menos representadas. Usar una arquitectura que se centre en cada una de las ocasiones puede presentar problemas para detectar cuando se dan esas ocasiones. Además, es probable que haya un conocimiento común entre el comportamiento habitual y los comportamientos dispares que no se está aprovechando. Es por ello, que es necesario implementar una arquitectura que aproveche el conocimiento común, a la vez que las peculiaridades específicas de la serie temporal.
- La predicción univariante de series temporales multivariantes es una aplicación muy común que se ha explorado en esta tesis. Diversos estudios apoyan la idea de que una estimación del futuro con cierto grado de error es capaz de incrementar la eficacia. Esto ocurre, por ejemplo, en series temporales que incluyen variables meteorológicas, ya que

existen diversos servicios que son capaces de obtener unas predicciones con cierto grado de confianza. Sin embargo, en otros entornos no existe un servicio y crearlos implicaría una gran cantidad de conocimiento experto. Por este motivo, se propone una serie temporal que realice la predicción en dos fases, en la primera fase realiza una estimación de todas las variables mientras que en la segunda se usa esa estimación como entrada para la predicción efectiva. De esta manera, se comprobará si la acumulación de error es más perjudicial que poseer una estimación de las variables a futuro.

- Estudios recientes proponen que las redes neuronales habituales aplican realmente una interpolación spline [9]. Esto quiere decir que la red neuronal aplica una división de los datos de forma jerárquica modelando de forma independiente cada subconjunto de datos separado. Esto quiere decir, de alguna manera, que el comportamiento de una red neuronal es parecida al de un árbol de decisión. La idea principal es ser capaz de traducir una red neuronal a un árbol de decisión, de manera que su comportamiento sea más interpretable y eficiente manteniendo la eficacia de una red neuronal.

Capítulo 12

Conclusions

Mathematical science shows what is. It is the language of unseen relations between things. But to use and apply that language, we must be able to fully to appreciate, to feel, to seize the unseen, the unconscious.

Ada Lovelace.

Time series is one of the most common and complex types of data to deal with, with several benefits for society in the medical, social and economic fields, among others. Prediction is one of the challenges that brings the greatest benefit and, at the same time, presents the greatest complexity in terms of effectiveness and efficiency. In terms of efficacy, it is not possible to determine a single method that dominates the others in terms of quality of results. In addition, the great diversity of needs makes it difficult to select a single generic solution that adapts to any type of prediction or time series, such as multivariate prediction/series, multi-pass prediction, probabilistic prediction, among others. In terms of efficiency, it is necessary to develop methods that are capable of scaling to large volumes of data. In this way, it is possible to adapt the method to environments with limited computational resources, facilitate the experimentation of different configurations, readjust the method without requiring many resources, among others.

One of the possible solutions to face the different challenges presented by time series prediction is to develop new architectures that are capable of improving or obtaining a similar effectiveness to the current ones, improving efficiency. Among all the techniques within machine learning, neural networks have been selected to explore new architectures to meet the challenges. This is mainly because, currently, they have demonstrated good efficiency in a large number of scenarios. In addition, they usually require large computational resources to work properly, so it is essential to reduce their consumption as much as possible.

In this thesis, we have presented four architectures framed within the con-

text of e-learning that improve or equal the effectiveness of state-of-the-art methods, obtaining an improvement in efficiency. Three of these architectures relate to the learning process, scaling and division of responsibilities within the architecture design, while the last one integrates a feature selection process into the architecture.

HLNet is the first proposed architecture. The goal is to match the effectiveness obtained by LSTM by improving efficiency. The results were tested with 13 datasets with different characteristics and application domains. The results showed that there was no clear improvement in efficiency for most of the datasets. However, efficiency was clearly significantly improved in most of the datasets tested. This fact seems to indicate that the inductive bias introduced in the architecture design does not usually help the architecture to perform better on most datasets, instead it seems to help to converge faster.

PHILNet was born as an enhancement to HLNet by introducing an additional hypothesis. The goal remained the same: to improve the efficiency obtained by LSTM by matching its effectiveness. In addition, it was required to analyze the impact of the new hypothesis by comparing the results obtained with HLNet. The results showed a higher efficiency improvement than that obtained with HLNet with similar effectiveness results. These results determine that a joint optimization of the different hierarchies improves the convergence time even more than isolating each level.

SRCNet is another approach to some of the hypotheses presented in HLNet and PHILNet. The objective is to apply this method to the area of electrical transformer maintenance, separating responsibility in a way that decomposes the time series into different components with the objective of improving effectiveness and efficiency. The results showed that SRCNet obtained the best balance between efficiency and effectiveness, obtaining an efficiency similar to the simplest and fastest methods and surpassing the results obtained by state-of-the-art architectures. These results seem to indicate that the proposed decomposition, while simple and easy to implement, seems to help to obtain a better representation of the data improving efficiency.

FADL was the last proposal and is based on attribute selection. The objective is to reduce the dimensionality of the problem during the learning process of the neural network, reducing the amount of resources consumed and introducing some degree of interpretability. It was applied to 9 datasets within the medical, social and environmental domains. Better results were obtained than other attribute selection methods, with fairly stable behavior and shorter convergence time on all data sets. These results seem to indicate that it is possible to perform attribute selection by integrating optimization within the neural network learning process, especially in regression problems.

Capítulo 13

Future works

So much universe, and so little time.

Terry Pratchett

In general, it seems that the exploration of new architectures is a good way to improve effectiveness and efficiency. Four proposals have been showed in this thesis. However, there are a wide variety of implementations and different possible solutions, as many as there are ideas that can be formalized within the architecture design. There are many other ideas that can be explored that face one or more of the challenges presented by time series forecasting:

- An architecture will be implemented that exploits the different peculiarities presented by the time series. Sometimes, the same time series presents very disparate behaviors such as weekends, holidays, incidents, etc. Using a common architecture to extract knowledge from the entire time series runs the risk of fitting too closely to the majority case, discriminating against the less represented occasions. Using an architecture that focuses on individual occasions may present problems in detecting when such occasions occur. In addition, it is likely that there is common knowledge between the usual behavior and the disparate behavior that is not being exploited. Therefore, it is necessary to implement an architecture that takes advantage of the common knowledge as well as the specific peculiarities of the time series.

- Univariate prediction of multivariate time series is a very common application that has been explored in this thesis. Several studies support the idea that an estimation of the future with a certain degree of error is able to increase efficiency. This occurs, for example, in time series that include meteorological variables, since there are several services that are able to obtain predictions with a certain degree of confidence.

However, in other environments there are no services and creating them would involve a large amount of expert knowledge. For this reason, a time series is proposed that performs the prediction in two phases, in the first phase it performs an estimation of all the variables while in the second phase this estimation is used as input for the actual prediction. In this way, it will be tested whether the accumulation of error is more detrimental than having an estimate of the variables in the future.

- Recent studies propose that the usual neural networks actually apply spline interpolation. This means that the neural network applies a hierarchical partitioning of the data by independently modeling each separate subset of data. This means that, in a way, the behavior of a neural network is similar to that of a decision tree. The main idea is to be able to translate a neural network into a decision tree, so that its behavior is more interpretable and efficient while maintaining the effectiveness of a neural network.

Parte V

Apéndices

Apéndice A

Glosario de términos

- **Deep learning**: Término definido por primera vez en 1986. El término viene de la idea de que se obtiene un conocimiento más profundo de los datos al aumentar el número de capas ocultas en una red neuronal. En el contexto de esta tesis, deep learning y red neuronal representan el mismo concepto.
- **Arquitectura**:Representación del conocimiento extraído de unos datos a través de un proceso de aprendizaje. La arquitectura es capaz de obtener una salida a partir de una serie de entradas.
- **Modelo**: En el contexto de esta tesis, el término modelo y arquitectura representan lo mismo.
- **Matriz**: Colección de números estructurado en dos dimensiones llamadas filas y columnas. En el contexto de la tesis, las matrices son la representación matemática de los distintos pesos de la red neuronal.
- **Eficacia**: Capacidad que tiene una arquitectura para ajustar las salidas obtenidas a las esperadas. A más parecida sean, mayor será la eficacia. En el contexto de la tesis, distintas fórmulas de error son usadas. De esta manera, a menor sea el error, más parecidas será la salida obtenida a la esperada.
- **Eficiencia**: Cantidad de recursos requeridos por una arquitectura. A menor cantidad de recursos requeridos, mejor será la eficiencia. Los recursos pueden incluir: tiempo de inferencia, tiempo de entrenamiento, almacenamiento requerido, etc. En el contexto de la tesis, se considera principalmente el tiempo de convergencia como medida de eficiencia.
- **Tiempo de convergencia**: Tiempo requerido para que una arquitectura extraiga el conocimiento necesario durante el proceso de entrenamiento. Este tiempo requiere de una condición de parada que determine

que la solución a convergido. En el contexto de la tesis, esta condición la establecerá la técnica llamada "early stopping".

- **Covariable**: Característica que sirve de información adicional para la predicción de una característica en específico.
- **Atributo**: Cualquier información medible dentro de un dominio.
- **Instancia**: Información proveniente de uno o más atributos para un caso específico. En el contexto de las redes neuronales, se trata de la información de todos los atributos en un instante de tiempo.
- **Ventana**: Subconjunto de instancias secuenciales ordenadas con respecto al tiempo.
- **Horizonte**: Representa el conjunto de valores que sirve como estimación de uno o varios atributos en el futuro.
- **Suavizado**: Proceso por el cual se modifica una secuencia de datos ordenados reduciendo las fluctuaciones.
- **Embedding**: Representación de información de entrada codificada en un vector.
- **Espacio latente**: En el contexto de la tesis, espacio latente y embedding representan el mismo concepto.
- **Test estadístico**: Proceso por el cual se contrasta una propiedad sobre una población estableciendo un valor cuantitativo sobre la veracidad de esa propiedad.
- **Experimento**: En el contexto de la tesis, un experimento hace referencia al proceso de entrenamiento de una arquitectura y su posterior evaluación.
- **Hipótesis**: Suposición realizada sobre una idea que puede ser formalizada en el diseño de una arquitectura estableciendo una serie de sesgos inductivos.
- **Sesgo inductivo**: Se trata del conjunto de restricciones o penalizaciones integrados en una arquitectura para fomentar ciertas soluciones sobre otras.

Apéndice B

Características seleccionadas por FADL

En esta sección se detallan las características seleccionadas por cada uno de los métodos de selección detallados en la Sección 9. Cada fila de la imagen representa un método de selección y las columnas representan las características. El valor amarillo de un píxel representa que se ha seleccionado esa característica.

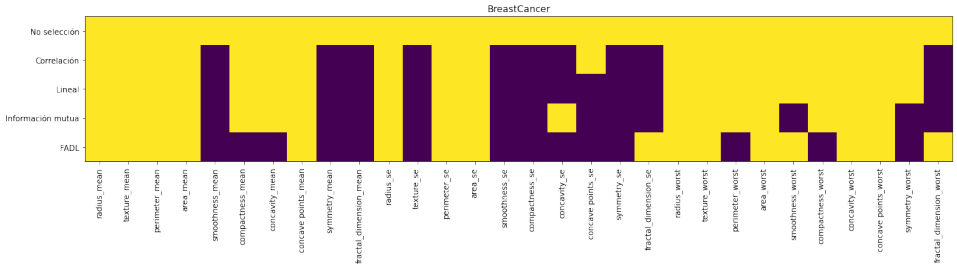


Figura B.1: Selección obtenida sobre el conjunto de datos Breast.

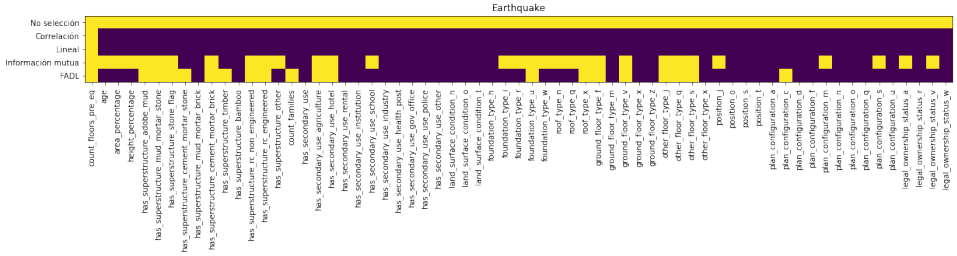


Figura B.2: Selección obtenida sobre el conjunto de datos Earthquake.

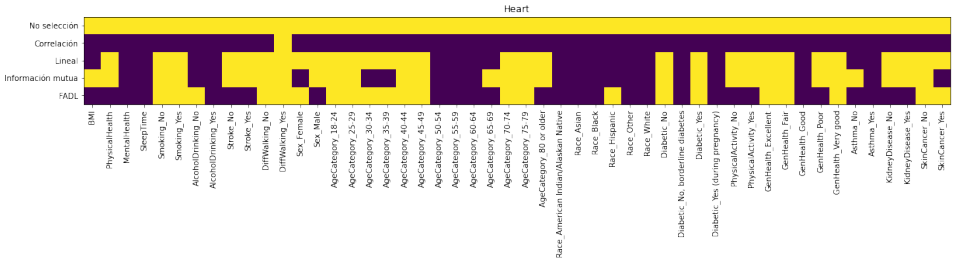


Figura B.3: Selección obtenida sobre el conjunto de datos Heart.

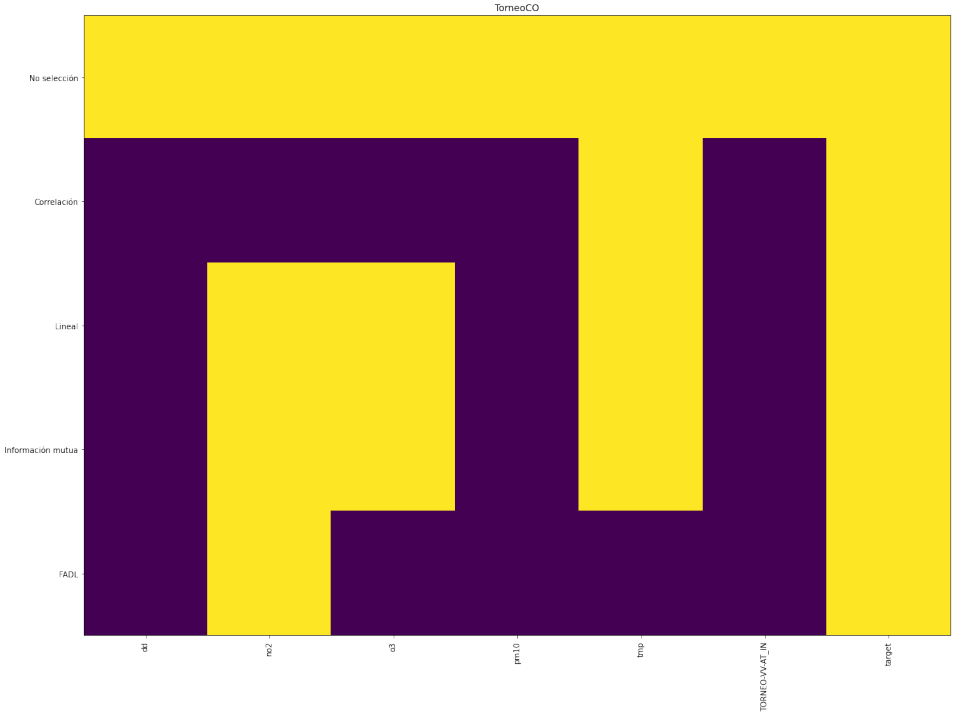


Figura B.4: Selección obtenida sobre el conjunto de datos Torneo *CO*.

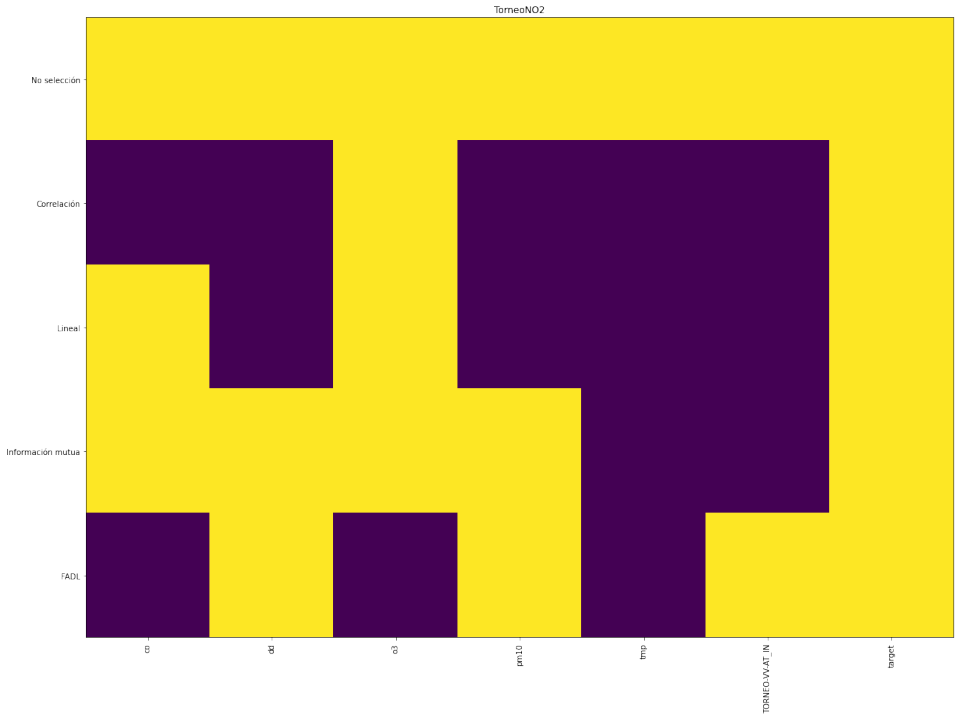


Figura B.5: Selección obtenida sobre el conjunto de datos Torneo NO_2.

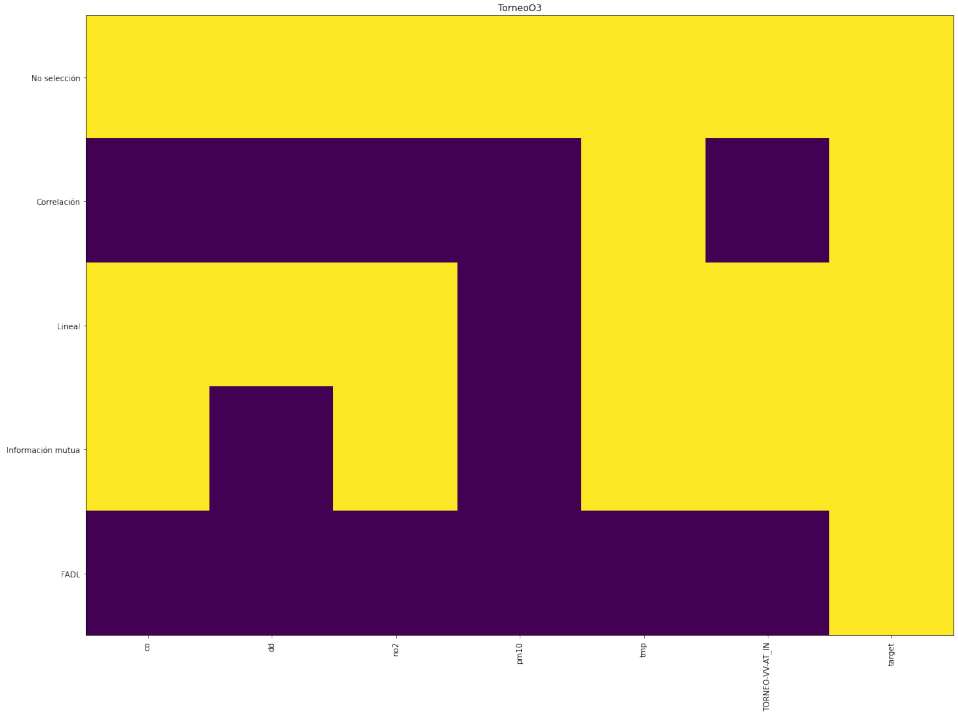


Figura B.6: Selección obtenida sobre el conjunto de datos Torneo O_3.

Bibliografía

[1] Q. Abdulqader. Time Series Forecasting Using Arima Methodology with Application on Census Data in Iraq. *Science Journal of University of Zakho*, 4:258–268, 12 2016.

[2] California State Transportation Agency. `https://pems.dot.ca.gov/`. *California department of transportation*, 2015.

[3] S. Agrahari and A. K. Singh. Concept drift detection in data stream mining : A literature review. *Journal of King Saud University - Computer and Information Sciences*, 2021.

[4] M. M. Ahsan, M. A. P. Mahmud, P. K. Saha, K. D. Gupta, and Z. Siddique. Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance. *Technologies*, 9, 2021.

[5] Jasbir S. Arora. 11 - more on numerical methods for constrained optimum design. In J. S. Arora, editor, *Introduction to Optimum Design (Second Edition)*, pages 379–412. Academic Press, second edition edition, 2004.

[6] N. Arunraj, D. Ahrens, and M. Fernandes. Application of SARIMAX Model to Forecast Daily Sales in Food Retail Industry. *International Journal of Operations Research and Information Systems*, 7:1–21, 04 2016.

[7] G. Athanasopoulos, R. J. Hyndman, H. Song, and D. C. Wu. The tourism forecasting competition. *International Journal of Forecasting*, 27:822–844, 2011.

[8] S. Bai, J. Z. Kolter, and V. Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, 2018.

[9] R. Balestriero and R. baraniuk. A spline theory of deep learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 374–383. PMLR, 2018.

[10] T. Ban, R. Zhang, S. Pang, A. Sarrafzadeh, and D. Inoue. Referential knn regression for financial time series forecasting. In *20th International Conference on Neural Information Processing*, pages 601–608, 11 2013.

[11] D. Barrow, N. Kourentzes, R. Sandberg, and J. Niklewski. Automatic robust estimation for exponential smoothing: Perspectives from statistics and machine learning. *Expert Systems with Applications*, 160:113637, 2020.

[12] R. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.

[13] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[14] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon. Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis. *Journal of Machine Learning Research*, 18:2653–2688, 2017.

[15] A. Benavoli, F. Mangili, G. Corani, M. Zaffalon, and F. Ruggeri. A Bayesian Wilcoxon signed-rank test based on the dirichlet process. In *Proceedings of the International Conference on Machine Learning*, volume 3, pages 2703–2713, 01 2014.

[16] R. N. Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[17] D. Marcu C and C. Grava. The impact of activation functions on training and performance of a deep neural network. In *Proceedings of 2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–4. IEEE, 2021.

[18] J. Chai, H. Zeng, A. Li, and E. W.T. Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021.

[19] 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(1):321–357, 2002.

[20] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM, 2016.

[21] K. Cho, Ç. Gülçehre B. v. Merrienboer, H. Schwenk D. Bahdanau, F. Bougares, and Y. Bengio. Learning phrase representations using

RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP*, pages 1724–1734. ACL, 2014.

[22] B. Choubin, G. Zehtabian, A. Azareh, E. R. Sardooi, F. S. Hosseini, and O. Kisi. Precipitation forecasting using classification and regression trees (cart) model: a comparative study of different approaches. *Environmental Earth Sciences*, 77:314, 04 2018.

[23] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In Yoshua Bengio and Yann LeCun, editors, *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

[24] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] S. F. Crone. NN5 time series forecasting competition for neural networks, `http://www.neural-forecasting-competition.com/`, 2008.

[26] Nepal Earthquake Open Data. `http://eq2015.npc.gov.np/#/`, 2015.

[27] Red Eléctrica de España. `www.ree.es.`, 2015.

[28] D.Q.F. de Menezes, D.M. Prata, A.R. Secchi, and J.C. Pinto. A review on robust m-estimators for regression analysis. *Computers & Chemical Engineering*, 147:107254, 2021.

[29] Rina Dechter. Learning while searching in constraint-satisfaction-problems. In *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence*, AAAI'86, page 178–183. AAAI Press, 1986.

[30] B. Derrick, D. Toher, and P. White. How to compare the means of two samples that include paired observations and independent observations: A companion to derrick, russ, toher and white (2017). *The Quantitative Methods for Psychology*, 13(2):120–126, 2017.

[31] A. Dghais and M. T. Ismail. A comparative study between discrete wavelet transform and maximal overlap discrete wavelet transform for testing stationarity. *International Journal of Mathematical Science and Engineering*, 7:471–475, 12 2013.

[32] P. Dhal and C. Azad. A comprehensive survey on feature selection in the various fields of machine learning. *Applied Intelligence*, 52:4543–4581, 03 2022.

[33] M. Ding, H. Zhou, H. Xie, M. Wu, K. Liu, Y. Nakanishi, and R. Yokoyama. A time series model based on hybrid-kernel least-squares

support vector machine for short-term wind power forecasting. *ISA Transactions*, 108:58–68, 2021.

[34] Y. Dodge. *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, 2008.

[35] K. Dragomiretskiy and D. Zosso. Variational mode decomposition. *IEEE Transactions on Signal Processing*, 62(3):531–544, 2014.

[36] C. W. Dunnett. A multiple comparison procedure for comparing several treatments with a control. *Journal of the American Statistical Association*, 50(272):1096–1121, 1955.

[37] K. Dursun. Oil and winding temperature control in power transformers. In *Proceedings of Power Engineering, Energy and Electrical Drives (POWERENG)*, pages 1631–1639, 05 2013.

[38] C. E-Shannon. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, 5(1):3–55, 2001.

[39] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona. A survey on missing data in machine learning. *Journal of Big Data*, 8, 05 2021.

[40] A. Permanasari Erna, I. Hidayah, and I. A. Bustoni. SARIMA (Seasonal ARIMA) implementation on time series to forecast the number of Malaria incidence. In *proceedings of 2013 International Conference on Information Technology and Electrical Engineering (ICITEE)*, pages 203–207, 2013.

[41] T. Fang and R. Lahdelma. Evaluation of a multiple linear regression model and sarima model in forecasting heat demand for district heating system. *Applied Energy*, 179:544–552, 2016.

[42] J. Fattah, L. Ezzine, Z. Aman, H. Moussami, and A. Lachhab. Forecasting of demand using ARIMA model. *International Journal of Engineering Business Management*, 10:184797901880867, 10 2018.

[43] S. Fiorini. UCI Gene Expression Cancer RNA-Seq, 2016.

[44] D. Freedman, R. Pisani, and R. Purves. Statistics (international student edition). *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.

[45] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[46] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

[47] S. Geer. *Encyclopedia of Statistics in Behavioral Science*, volume 2, chapter Least Squares Estimation, pages 1041–1045. Wiley, 10 2005.

[48] E. R. Girden. *ANOVA: Repeated measures*, volume 1 of *Quantitative Applications in the Social Sciences*. SAGE Publications, 1992.

[49] 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*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256. PMLR, 2010.

[50] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pages 315–323. PMLR, 11-13 Apr 2011.

[51] S. Gocheva-Ilieva, D. Voynikova, M. Stoimenova, A. Ivanov, and I. Iliev. Regression trees modeling of time series for air pollution analysis and forecasting. *Neural Computing and Applications*, 31:9023–9039, 12 2019.

[52] Google. Web traffic time series forecasting competition, 2017.

[53] A. Goyal and Y. Bengio. Inductive biases for deep learning of higher-level cognition. *ArXiv*, 2020.

[54] D. Gupta, M. Pratama, Z. Ma, J. Li, and M. Prasad. Financial time series forecasting using twin support vector regression. *PloS one*, 14(3):e0211402, 2019.

[55] A. Gómez-Losada, G. Asencio–Cortés, F. Martínez–Álvarez, and J.C. Riquelme. A novel approach to forecast urban surface-level ozone considering heterogeneous locations and limited information. *Environmental Modelling & Software*, 110:52–61, 2018. Special Issue on Environmental Data Science and Decision Support: Applications in Climate Change and the Ecological Footprint.

[56] C. Bergmeir H. Hewamalage and K. Bandara. Recurrent Neural Networks for Time Series Forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.

[57] J. Hancock and T. Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7:1–41, 04 2020.

[58] W. Haynes. *Bonferroni Correction*, pages 154–154. Springer New York, 2013.

[59] W. Haynes. *Tukey's Test*, pages 2303–2304. Springer New York, 2013.

[60] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of 2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[61] S. Hochreiter. The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[62] S. Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[63] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[64] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[65] M. Hossin and S. M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5:01–11, 03 2015.

[66] G. Huang, D. W. Hui, and Y. Lan. Extreme learning machines: a survey. *International journal of machine learning and cybernetics*, 2(2):107–122, 2011.

[67] G. Huang, Z. Liu, L. V. D. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[68] I. Ilic, B. Görgülü, M. Cevik, and M. Baydoğan Gökçe. Explainable boosted linear regression for time series forecasting. *Pattern Recognition*, 120:108144, 2021.

[69] T. Iqbal and S. Qureshi. The survey: Text generation models in deep learning. *Journal of King Saud University - Computer and Information Sciences*, 34(6, Part A):2515–2528, 2022.

[70] N. Jaitly, Q. Le V, O. Vinyals, I. Sutskever, D. Sussillo, and S. Bengio. An online sequence-to-sequence model using partial conditioning. In *Proceedings of advances in Neural Information Processing Systems*, volume 29, pages 5067—-5075. Curran Associates, Inc., 2016.

[71] J.-S.R. Jang. Anfis: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(3):665–685, 1993.

[72] P. Jiang, R. Li, N. Liu, and Y. Gao. A novel composite electricity demand forecasting framework by data processing and optimized support vector machine. *Applied Energy*, 260:114243, 2020.

[73] J. Joyce. Bayes' Theorem. In *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.

[74] M. Tiwari K, R. C. Deo, and Jan F. Adamowski. Short-term flood forecasting using artificial neural networks, extreme learning machines, and m5 model tree. In *Advances in streamflow forecasting*, pages 263–279. Elsevier, 2021.

[75] S. Ren K. He, X. Zhang and J. Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[76] N. Kalchbrenner, I. Danihelka, and A. Graves. Grid long short-term memory, 2015.

[77] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[78] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[79] D. Kim and J. Baek. Bagging ensemble-based novel data generation method for univariate time series forecasting. *Expert Systems with Applications*, 203:117366, 2022.

[80] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[81] S. Kiranyaz, T. Ince, H. Ridha, and M. Gabbouj. Convolutional Neural Networks for Patient-Specific ECG Classification. In *Proceedings of Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, volume 37, pages 2608–2611, 08 2015.

[82] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, volume 30, page 972–981. Curran Associates, Inc., 2017.

[83] M. Kück and M. Freitag. Forecasting of customer demands for production planning by local k-nearest neighbor models. *International Journal of Production Economics*, 231:107837, 2021.

[84] S. Kumar, P. Kaur, and A. Gosain. A Comprehensive Survey on Ensemble Methods. In *IEEE 7th International conference for Convergence in Technology*, pages 1–7, 2022.

[85] S. Kwon. Optimal feature selection based speech emotion recognition using two-stream deep convolutional neural network. *International Journal of Intelligent Systems*, 36(9):5116–5135, 2021.

[86] G. Lai, W. Chang, Y. Yang, and H. Liu. Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 95–104, 2018.

[87] M. Larrea, A. Porto, E. Irigoyen, A. J. Barragán, and J. M. Andújar. Extreme learning machine ensemble model for time series forecasting boosted by pso: Application to an electric consumption problem. *Neurocomputing*, 452:465–472, 2021.

[88] Y. A. LeCun, L. Bottou, G. B. Orr, and K. Müller. *Efficient BackProp*, pages 9–48. Springer Berlin Heidelberg, 2012.

[89] J. Li and W. Chen. Forecasting macroeconomic time series: Lasso-based approaches and their forecast combinations with dynamic factor models. *International Journal of Forecasting*, 30(4):996–1015, 2014.

[90] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, 2019.

[91] X. Li, X. Ma, F. Xiao, C. Xiao, F. Wang, and S. Zhang. Time-series production forecasting method based on the integration of bidirectional gated recurrent unit (bi-gru) network and sparrow search algorithm (ssa). *Journal of Petroleum Science and Engineering*, 208:109309, 2022.

[92] B. Lim, S. Ö. Arık, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[93] B. Lim and S. Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, 2021.

[94] G. Lin, A. Lin, and J. Cao. Multidimensional knn algorithm based on eemd and complexity measures in financial time series forecasting. *Expert Systems with Applications*, 168:114443, 2021.

[95] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia CIRP*, 99:650–655, 2021. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

[96] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. International Series in Engineering and Computer Science. Springer, 1998.

[97] M. Liu, A. Zeng, Z. Xu, Q. Lai, and Q. Xu. Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction, 2021.

[98] I. E. Livieris, E. Pintelas, and P. Pintelas. A CNN–LSTM model for gold price time-series forecasting. *Neural computing and applications*, 32(23):17351–17360, 2020.

[99] R. R. Maaliw, Z. P. Mabunga, and F. T. Villa. Time-series forecasting of covid-19 cases using stacked long short-term memory networks. In *Proceedings of 2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 435–441. IEEE, 2021.

[100] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proceedings of ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[101] S. Makridakis and M. Hibon. The M3-Competition: results, conclusions and implications. *International Journal of Forecasting*, 16(4):451–476, 2000.

[102] S. Maldonado, A. González, and S. Crone. Automatic time series analysis for electric load forecasting via support vector regression. *Applied Soft Computing*, 83:105616, 2019.

[103] F. Martínez-Álvarez, A. Troncoso, G. Asencio-Cortés, and J. C. Riquelme. A survey on data mining techniques applied to electricity-related time series forecasting. *Energies*, 8(11):13162–13193, 2015.

[104] Q. McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, June 1947.

[105] R. Näätänen. *Attention and brain function*. Psychology Press, 1992.

[106] K. I. Nikolopoulos, M. Z. Babai, and K. Bozos. Forecasting supply chain sporadic demand with nearest neighbor approaches. *International Journal of Production Economics*, 177:139–148, 2016.

[107] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. N-beats: Neural basis expansion analysis for interpretable time series forecasting. *arXiv*, 2019.

[108] O. O. Owolabi and D. A. Sunter. Bayesian optimization and hierarchical forecasting of non-weather-related electric power outages. *Energies*, 15(6), 2022.

[109] X. Peng. Tsvr: an efficient twin support vector machine for regression. *Neural Networks*, 23(3):365–372, 2010.

[110] D. B. Percival and A. T. Walden. *The Maximal Overlap Discrete Wavelet Transform*, page 159–205. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2000.

[111] P. Phyo, Y. Byun, and N. Park. Short-term energy forecasting using machine-learning-based ensemble voting regression. *Symmetry*, 14(1):160, 2022.

[112] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. CatBoost: Unbiased Boosting with Categorical Features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649. Curran Associates Inc., 2018.

[113] K. Pytlak. Personal Key Indicators of Heart Disease `https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease`, 2020.

[114] R. J. Quinlan. Learning with continuous classes. In *Proceedings of 5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

[115] G. Ristanoski, W. Liu, and J. Bailey. Time Series Forecasting using Distribution Enhanced Linear Regression. In *Proceedings of Advances in Knowledge Discovery and Data Mining*, volume 7818, page 484–495, 04 2013.

[116] H. E. Robbins. A Stochastic Approximation Method. *Annals of Mathematical Statistics*, 22:400–407, 2007.

[117] A. Román-Portabales, M. López-Nores, and J. J. Pazos-Arias. Systematic review of electricity demand forecast using ann-based machine learning algorithms. *Sensors*, 21(13):4544, 2021.

[118] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[119] D. Roy, S. R. M. Kodukula, and K. M. Chalavadi. Feature selection using deep neural networks. In *International Joint Conference on Neural Networks*, pages 1–6, 07 2015.

[120] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[121] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature Publishing Group*, 323(6088):533–536, 1986.

[122] E. Spiliotis S. Makridakis and V. Assimakopoulos. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, 2020.

[123] A. Rafael S. Parmezan and G. E.A.P.A. Batista. A study of the use of complexity measures in the similarity search process adopted by knn algorithm for time series prediction. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pages 45–51, 2015.

[124] S. Safi and O. Sanusi. A hybrid of artificial neural network, exponential smoothing, and arima models for covid-19 time series forecasting. *Model Assisted Statistics and Applications*, 16:25–35, 03 2021.

[125] A. B. Said, A. Erradi, H. A. Aly, and A. Mohamed. Predicting covid-19 cases using bidirectional lstm on multivariate time series. *Environmental Science and Pollution Research*, 28(40):56043–56052, 2021.

[126] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[127] I. H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):1–21, 2021.

[128] A. Sbrana, A. L. Debiaso Rossi, and M. Coelho Naldi. N-beats-rnn: deep learning for time series forecasting. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 765–768, 2020.

[129] S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52(3-4):591–611, dec 1965.

[130] R. R. Sharma, M. Kumar, S. Maheshwari, and K. P. Ray. Evdhm-arima-based time series forecasting model and its application for covid-19 cases. *IEEE Transactions on Instrumentation and Measurement*, 70:1–10, 2021.

[131] S. Shastri, A. Sharma, V. Mansotra, A. Sharma, A. Bhadwal, and M. Kumari. A study on exponential smoothing method for forecasting. *International Journal of Computer Sciences and Engineering*, 6:482–485, 04 2018.

[132] S. Siami-Namini, N. Tavakoli, and A. Siami Namin. A comparison of arima and lstm in forecasting time series. In *Poceedings of IEEE International Conference on Machine Learning and Applications (ICMLA)*, volume 17, pages 1394–1401, 2018.

[133] F. Sidqi and I. Sumitra. Forecasting Product Selling Using Single Exponential Smoothing and Double Exponential Smoothing Methods. *IOP Conference Series: Materials Science and Engineering*, 662:032031, 11 2019.

[134] V. Singh, R. C. Poonia, S. Kumar, P. Dass, P. Agarwal, V. Bhatnagar, and L. Raja. Prediction of covid-19 corona virus pandemic based on time series data using support vector machine. *Journal of Discrete Mathematical Sciences and Cryptography*, 23(8):1583–1597, 2020.

[135] B. Siregar, I. Butar-Butar, R. Rahmat, U. Andayani, and F. Fahmi. Comparison of Exponential Smoothing Methods in Forecasting Palm Oil Real Production. *Journal of Physics Conference Series*, 801:012004, 12 2016.

[136] S. Smyl. Ensemble of specialized neural networks for time series forecasting. In *Proceedings of International Symposium on Forecasting*, volume 37, pages 25–28, 06 2017.

[137] S. Smyl. A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, 36(1):75–85, 2020.

[138] M. E. Sobel. *The Analysis of Contingency Tables*, pages 251–310. Springer US, 1995.

[139] Qiang Song and Brad S. Chissom. Fuzzy time series and its models. *Fuzzy Sets and Systems*, 54(3):269–277, 1993.

[140] M. Soto-Ferrari, O. Chams-Anturi, J. P. Escorcia Caballero, N. Hussain, and M. Khan. *Evaluation of Bottom-Up and Top-Down Strategies for Aggregated Forecasts: State Space Models and ARIMA Applications*, pages 413–427. Computational Logistics, 09 2019.

[141] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[142] M. Stepnicka and M. Burda. Computational Intelligence in Forecasting (CIF), `https://irafm.osu.cz/cif/main.php`, 2016.

[143] E. Stevenson, V. Rodriguez-Fernandez, E. Minisci, and D. Camacho. A deep learning approach to solar radio flux forecasting. *Acta Astronautica*, 193:595–606, 2022.

[144] Z. Su and J. Jiang. Hierarchical gated recurrent unit with semantic attention for event prediction. *Future Internet*, 12(2), 2020.

[145] S. Sun, W. Chen, L. Wang, X. Liu, and T. Liu. On the Depth of Deep Neural Networks: A Theoretical View. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2066–2072. AAAI Press, 2016.

[146] J. Surowiecki. The wisdom of crowds: Why the many are smarter than the few. london. *Abacus*, page 295, 01 2004.

[147] J. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9:293–300, 06 1999.

[148] Taarifa. Water pump `https://taarifa.org/`, 2022.

[149] N. Tang, S. Mao, Y. Wang, and R. M. Nelms. Solar power generation forecasting with a lasso-based approach. *IEEE Internet of Things Journal*, 5(2):1090–1099, 2018.

[150] V. A. Thiviyanathan, P. J. Ker, Y. S. Leong, F. Abdullah, A. Ismail, and Md. Z. Jamaludin. Power transformer insulation system: A review on the reactions, fault detection, challenges and future prospects. *Alexandria Engineering Journal*, 61(10):7697–7713, 2022.

[151] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288, 1996.

[152] A. N. Tikhonov. Solution of Incorrectly Formulated Problems and the Regularization Method. *Soviet Mathematics Doklady*, 4:1624–1627, 1963.

[153] A. Troncoso, S. Salcedo-Sanz, C. Casanova-Mateo, J.C. Riquelme, and L. Prieto. Local models-based regression trees for very short-term wind speed prediction. *Renewable Energy*, 81:589–598, 2015.

[154] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *NeurIPS 2020 Competition and Demonstration Track*, pages 3–26. PMLR, 2021.

[155] J. V. Belle, R. Crevits, and W. Verbeke. Improving forecast stability using deep learning. *International Journal of Forecasting*, 2022.

[156] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 6000–6010, 2017.

[157] R. Villegas, J. Yang, Y. Zou, S. Sohn, X. Lin, and H. Lee. Learning to generate long-term future via hierarchical prediction. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, page 3560–3569. JMLR.org, 2017.

[158] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Proceedings of Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.

[159] Z. Wang, Z. Dai, B. Poczos, and J. Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11285–11294, 06 2019.

[160] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[161] W. H. Wolberg, W. Nick Street, and O. L. Mangasarian. UCI Breast Cancer Wisconsin (Diagnostic), 1995.

[162] Z. Wu and N. Huang. Ensemble empirical mode decomposition: a noise-assisted data analysis method. *Advances in Adaptive Data Analysis*, 1:1–41, 01 2009.

[163] J. Xue and B. Shen. A novel swarm intelligence optimization approach: sparrow search algorithm. *Systems Science & Control Engineering*, 8(1):22–34, 2020.

[164] C. Shahabi Y. Li, R. Yu and Y. Liu. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting, 2018.

[165] P. T. Yamak, L. Yujian, and P. K. Gadosey. A comparison between arima, lstm, and gru for time series forecasting. In *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, page 49–55. Association for Computing Machinery, 2019.

[166] M. D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *Proceeddings of the European Conference on Computer Vision*, volume 8689, pages 818–833, 2014.

[167] A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are Transformers Effective for Time Series Forecasting?, 2022.

[168] Y. Zhang. Solar power data for integration studies (NREL), `https://www.nrel.gov/grid/solar-integration-data.html`, 2007.

[169] Y. Zhang, H. Qu, W. Wang, and J. Zhao. A novel fuzzy time series forecasting model based on multiple linear regression and time series clustering. *Mathematical Problems in Engineering*, 2020:9546792, 2020.

[170] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 5(5):726–742, 2021.

[171] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

[172] L. Zhu, G. Wang, and X. Zou. Improved information gain feature selection method for chinese text classification based on word embedding. In *Proceedings of the 6th International Conference on Software and Computer Applications*, page 72–76, New York, NY, USA, 2017. Association for Computing Machinery.

[173] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019.

[174] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society, Series B*, 67:301–320, 2005.