COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Modelos de series temporales basados en aprendizaje profundo para la predicción de la generación eléctrica en parques eólicos

Madrid
2025

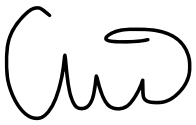
Autor: Claudia Coduras Gracia
Director: Fernando San Segundo Barahona
Subdirector: Antonio Muñoz San Roque

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Modelos de series temporales basados en aprendizaje profundo para la predicción de

la generación eléctrica en parques eólicos marinos

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico **2024/25** es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

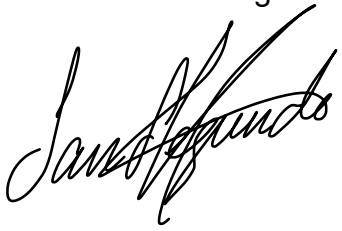


Fdo.: Claudia Coduras Gracia Fecha: 19/ 06/ 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Fernando San Segundo Barahona Fecha: 19/ 06/ 2025



COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Modelos de series temporales basados en aprendizaje profundo para la predicción de la generación eléctrica en parques eólicos

Madrid
2025

Autor: Claudia Coduras Gracia
Director: Fernando San Segundo Barahona
Subdirector: Antonio Muñoz San Roque

Modelos de series temporales basados en aprendizaje profundo para la predicción de la generación eléctrica en parques eólicos marinos

Autor: Claudia Coduras Gracia
Director: Fernando San Segundo Barahona
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Palabras clave: energía eólica marina, aprendizaje profundo, predicción, Transformer, KAN, series temporales, mercado intradiario.

RESUMEN DEL PROYECTO

1. Introducción

La integración de fuentes renovables en los sistemas eléctricos requiere herramientas predictivas precisas y robustas. En este contexto, la energía eólica marina destaca por su gran potencial, aunque presenta desafíos importantes en la predicción de su generación eléctrica debido a la alta variabilidad del viento [1]. Este trabajo de fin de máster (TFM) se enmarca dentro de dicha problemática, proponiendo una comparación entre diferentes modelos de predicción de series temporales basados en aprendizaje profundo, con el fin de mejorar la precisión en los pronósticos a corto plazo para parques eólicos marinos. El trabajo amplía y actualiza el análisis realizado previamente por Carlos de los Santos [2], con el objetivo de realizar una comparativa sólida del desempeño de los modelos para la predicción de series temporales, incorporando arquitecturas más avanzadas y recientes como los Transformers y las Kolmogorov–Arnold Networks (KAN).

2. Definición del proyecto

El objetivo principal es evaluar y comparar la capacidad predictiva de distintos modelos de Deep Learning en la estimación de la potencia eólica a un horizonte temporal de 3 horas, intervalo especialmente relevante para la operativa en el mercado intradiario. Esta ventana temporal se ha seleccionado no con el propósito de analizar el impacto del horizonte en el rendimiento predictivo, sino para centrar la comparativa en la eficacia de los distintos modelos. Como referencia, se retoman los modelos utilizados en el TFM previo (ARIMA, XGBoost, LSTM y GRU), y se amplía el análisis incorporando nuevas propuestas de modelos basados en Transformer y redes Kolmogorov–Arnold (KAN). El estudio se lleva a cabo utilizando datos reales de producción eólica y predicciones meteorológicas (NWP) generadas por el modelo ICON-D2.

3. Descripción del trabajo realizado

El flujo de trabajo seguido abarca la utilización de dos conjuntos de datos:

- **Conjunto de datos Alpha Ventus**: la herencia de los datos provenientes del TFM de Carlos [2] y preprocesamiento de datos, incluyendo la selección de las variables relevantes.
- **Conjunto de datos públicos disponibles en Kaggle** [3]: El presente trabajo utiliza un conjunto de datos obtenido de Kaggle, una plataforma ampliamente reconocida por albergar datos abiertos. Su incorporación permite evaluar la capacidad de generalización y escalabilidad de los modelos desarrollados en un entorno realista y replicable.

Para ambos conjuntos de datos, se aplicó un preprocesamiento riguroso que incluyó la imputación de valores ausentes, la normalización de las variables y la generación de variables temporales (lags, medias móviles y estacionales) para enriquecer las entradas de los modelos. A continuación, se dividieron los datos en conjuntos de entrenamiento, validación y test empleando una estrategia de corte temporal (hold-out), manteniendo la secuencia cronológica para evitar fugas de información.

Posteriormente, se implementaron y entrenaron diversos modelos de predicción, incluyendo arquitecturas base como ARIMA, XGBoost, LSTM y GRU, así como modelos más avanzados como Transformers y Kolmogorov–Arnold Networks (KAN). Para mejorar el rendimiento de los modelos, se aplicaron técnicas de ajuste de hiperparámetros mediante la librería Optuna. Como una de las contribuciones clave del trabajo, se construyó un repositorio accesible y debidamente documentado que permite la reproducción completa de los experimentos, fomentando la transparencia y la reutilización del código en futuros estudios.

La evaluación del desempeño se realizó a través de métricas estándar de regresión (MAE)

Además, se llevó a cabo una comparativa entre los conjuntos de datos para valorar la robustez de los modelos en distintos contextos

4. Resultados

Resultados conjunto de datos AV

AV	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	320.90	530.91	236.98	263.77	394.64	236.97	239.57
2h	384.08	634.94	375.87	411,96	716.48	377.24	378.78
3h	442.75	736.11	487.08	518.84	901.91	486.45	484.76

Figura 1: Comparativa y análisis de aplicabilidad

Resultados conjunto de datos Kaggle

Kaggle	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	0.0280	0.0170	0.0035	0.0034	0.0067	0.0053	0.0041
2h	0.0478	0.0236	0.0117	0.0113	0.0229	0.0157	0.0129
3h	0.0664	0.0266	0.0242	0.0241	0.0467	0.0290	0.0262

Figura 2: Comparativa y análisis de aplicabilidad

5. Conclusiones

Conclusiones clave del análisis en AV

- Los modelos basados en Transformer (PatchTST) y KAN (RMoK) superan con claridad a las arquitecturas clásicas y recurrentes.
- PatchTST se posiciona como el modelo más eficaz para capturar dependencias temporales de corto plazo (por ejemplo, predicciones a 1 hora), mientras que TiDE y RMoK muestran un rendimiento superior a medida que se amplía el horizonte de predicción, resultando más adecuados para modelar dinámicas temporales de largo plazo.
- RMoK destaca por su equilibrio entre precisión y estabilidad, lo que lo convierte en una opción muy adecuada para entornos de producción donde se requieren predicciones robustas a múltiples horizontes.

- LSTM y NBEATS presentan limitaciones evidentes, y su rendimiento decrece notablemente con el tiempo.

Conclusiones clave del análisis en Kaggle

- Los transformers (TimeXer y PatchTST) dominan el benchmark en este dataset, alcanzando los mejores resultados en todos los horizontes. Su rendimiento extremadamente bajo en MAE confirma su idoneidad para datos limpios y complejos.
- RMoK y TiDE se consolidan como alternativas muy sólidas, especialmente para tareas donde el compromiso entre precisión y explicabilidad sea relevante.
- XGBoost y LSTM quedan claramente superados, lo que refuerza la idea de que la modelización de dependencias temporales profundas requiere arquitecturas modernas.
- La coherencia entre los resultados de AV y Kaggle aporta solidez a las conclusiones del trabajo, y sugiere que los modelos avanzados (Transformers y KAN) ofrecen mejor capacidad de generalización.

6. Referencias

[1] H. Hao Wang Jingzhen Ye, «A multivariable hybrid prediction model of offshore windpower based on multi-stage optimization and reconstruction prediction». 2023. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0360544222023106

[2] C. de los Santos Jiménez, «Análisis comparativo de modelos de aprendizaje automático parala predicción de la generación eléctrica de un parque eólico marino». Universidad PontificiaComillas, 2023. Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/83380

[3] M. Rahim, «Wind Power Generation Data – Forecasting». Kaggle dataset, 2024. Dispo-nible en: https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

Deep Learning-Based Time Series Models for Predicting Power Generation in Offshore Wind Farms

Autor: Claudia Coduras Gracia
Director: Fernando San Segundo Barahona
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

Keywords: offshore wind energy, Deep Learning, forecasting, Transformer, KAN, time series, intraday electricity market.

ABSTRACT

1. Introduction

The integration of renewable energy sources into power systems requires accurate and robust predictive tools. In this context, offshore wind energy stands out for its great potential, although it presents significant challenges in forecasting its power generation due to the high variability of wind [1]. This master's thesis (TFM) addresses this issue by proposing a comparison between different Deep Learning-based time series prediction models, with the aim of improving short-term forecasts for offshore wind farms. The work extends and updates the analysis previously conducted by Carlos de los Santos [2], with the goal of providing a solid performance comparison of time series forecasting models, incorporating more advanced and recent architectures such as Transformers and Kolmogorov–Arnold Networks (KAN).

2. Project Definition

The main objective is to evaluate and compare the predictive capabilities of various Deep Learning models in forecasting wind power generation over a 3-hour horizon—a time frame that is particularly relevant for operations in the intraday electricity market. This time window was selected not to analyze the impact of forecast horizon on performance, but rather to focus the comparison on model effectiveness. As a baseline, the models used in the previous TFM (ARIMA, XGBoost, LSTM, and GRU) are revisited, and the analysis is extended by incorporating new model proposals based on Transformer architectures and Kolmogorov–Arnold Networks (KAN). The study is conducted using real wind power production data and weather forecasts (NWP) generated by the ICON-D2 model.

3. Description of the Work Performed

The workflow followed involves the use of two datasets:

- Alpha Ventus Dataset: Inherited from Carlos's Master's Thesis [2], including data preprocessing and the selection of relevant variables.
- Public Dataset Available on Kaggle [3]: This work uses a dataset obtained from Kaggle, a widely recognized platform for open data. Its inclusion allows for evaluating the generalization and scalability capabilities of the developed models in a realistic and replicable environment.

For both datasets, rigorous preprocessing was applied, including missing value imputation, variable normalization, and the generation of temporal features (lags, moving averages, and seasonal indicators) to enrich the model inputs. The data was then split into training, validation, and test sets using a temporal hold-out strategy, maintaining chronological order to avoid data leakage.

Subsequently, various predictive models were implemented and trained, including baseline architectures such as ARIMA, XGBoost, LSTM, and GRU, as well as more advanced models such as

Transformers and Kolmogorov–Arnold Networks (KAN). To enhance model performance, hyperparameter tuning techniques were applied using the Optuna library. As one of the key contributions of this work, an accessible and well-documented repository was built to enable full reproducibility of the experiments, promoting transparency and code reuse in future studies.

Model performance was evaluated using standard regression metrics (MAE).

Additionally, a comparison between the datasets was conducted to assess the robustness of the models across different contexts.

4. Results

Results for the AV Dataset

AV	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	320.90	530.91	236.98	263.77	394.64	236.97	239.57
2h	384.08	634.94	375.87	411,96	716.48	377.24	378.78
3h	442.75	736.11	487.08	518.84	901.91	486.45	484.76

Figura 3: Comparativa y análisis de aplicabilidad

Results for the Kaggle Dataset

Kaggle	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	0.0280	0.0170	0.0035	0.0034	0.0067	0.0053	0.0041
2h	0.0478	0.0236	0.0117	0.0113	0.0229	0.0157	0.0129
3h	0.0664	0.0266	0.0242	0.0241	0.0467	0.0290	0.0262

Figura 4: Comparativa y análisis de aplicabilidad

5. Conclusions

Key Findings from the AV Dataset Analysis

- Transformer (PatchTST) and KAN-based models (RMoK) clearly outperform classical and recurrent architectures.
- PatchTST proves to be the most effective model for capturing short-term temporal dependencies (e.g., 1-hour forecasts), while TiDE and RMoK demonstrate superior performance as the forecasting horizon increases, making them more suitable for modeling long-term temporal dynamics.
- RMoK stands out for its balance between accuracy and stability, making it a strong candidate for production environments where robust multi-horizon forecasting is required.
- LSTM and N-BEATS exhibit clear limitations, with performance deteriorating significantly over time.

.

Key Findings from the Kaggle Dataset Analysis

- Transformers (TimeXer and PatchTST) dominate the benchmark on this dataset, achieving the best results across all forecast horizons. Their extremely low MAE confirms their suitability for clean and complex data.
- RMoK and TiDE prove to be very strong alternatives, especially for tasks where a trade-off between accuracy and interpretability is important.
- XGBoost and LSTM are clearly outperformed, reinforcing the notion that modeling deep temporal dependencies requires modern architectures.
- The consistency between the AV and Kaggle results strengthens the conclusions of this work and suggests that advanced models (Transformers and KAN) offer superior generalization capabilities.

6. References

[1] H. Hao Wang Jingzhen Ye, «A multivariable hybrid prediction model of offshore windpower based on multi-stage optimization and reconstruction prediction». 2023. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0360544222023106

[2] C. de los Santos Jiménez, «Análisis comparativo de modelos de aprendizaje automático parala predicción de la generación eléctrica de un parque eólico marino». Universidad PontificiaComillas, 2023. Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/83380

[3] M. Rahim, «Wind Power Generation Data – Forecasting». Kaggle dataset, 2024. Dispo-nible en: https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

Tabla de contenidos

Capítulo 1

Introducción

1.1 Presentación del problema y precedentes

El avance hacia una transición energética sostenible ha situado a las energías renovables en el centro de los sistemas eléctricos modernos. En este contexto, la energía eólica —y en particular la generación en parques marinos— ha ganado un protagonismo creciente gracias a su alto potencial y menor impacto visual y territorial frente a instalaciones terrestres. No obstante, su integración masiva en la red eléctrica plantea importantes desafíos técnicos y económicos, derivados principalmente de su variabilidad e incertidumbre.

Uno de los retos más relevantes es el desarrollo de modelos de predicción fiables de la generación eólica [4], que permitan anticipar la producción con suficiente antelación y precisión. Estas predicciones son esenciales para formular ofertas en los distintos mercados eléctricos, planificar el despacho de energía, reducir los costes por desvíos y garantizar la estabilidad del sistema.

En los últimos años, se ha producido una evolución significativa en las técnicas utilizadas para abordar este problema, desde modelos estadísticos tradicionales hasta métodos basados en aprendizaje automático y redes neuronales profundas. Sin embargo, el dominio específico de la predicción en parques eólicos marinos presenta particularidades que requieren enfoques más sofisticados y adaptados a su complejidad.

Este Trabajo Fin de Máster se sitúa en la línea de investigación abierta por trabajos previos, en particular el desarrollado por Carlos de los Santos Jiménez [2], y busca profundizar en el estudio comparativo de modelos de predicción aplicados a la generación eólica offshore, incorporando arquitecturas emergentes que podrían suponer una mejora sustancial en el rendimiento predictivo.

1.1.1 Predicción de la generación eólica

La predicción de la potencia eólica generada constituye una tarea esencial en la operación de sistemas eléctricos con alta penetración de renovables. Su complejidad radica en la naturaleza caótica, no lineal y no estacionaria del recurso eólico, cuya disponibilidad depende de múltiples factores atmosféricos —como el viento, la presión y la temperatura— que interactúan entre sí a distintas escalas temporales y espaciales.

En particular, el caso de los parques eólicos marinos presenta desafíos adicionales. Aunque el viento en alta mar suele mostrar una mayor estabilidad y menor rugosidad superficial que en entornos

terrestres, lo que a priori facilita su modelado, también implica una mayor exposición a condiciones meteorológicas extremas, una dependencia crítica de predicciones meteorológicas externas (NWP), y unas infraestructuras más costosas y vulnerables. Todo ello incrementa la necesidad de contar con herramientas de forecasting altamente precisas, especialmente en horizontes intradiarios que condicionan las decisiones operativas y económicas de los operadores.

Tradicionalmente, se han empleado modelos como la persistencia o el clásico ARIMA para realizar predicciones a corto plazo (entre 1–9 horas). Si bien estos enfoques proporcionan una base comparativa robusta, presentan limitaciones claras frente a la complejidad del problema, ya que suponen relaciones lineales y requieren un ajuste manual intensivo de los parámetros. Su rendimiento decrece especialmente ante comportamientos no estacionarios o multivariantes, como ocurre en el entorno eólico offshore [1].

Con la expansión del volumen y la calidad de los datos, los métodos de aprendizaje automático han ganado terreno. Algoritmos como Random Forests, Support Vector Machines y, especialmente, los modelos de Gradient Boosting (XGBoost, LightGBM) han demostrado mejoras sustanciales en precisión, gracias a su capacidad para incorporar múltiples variables, detectar interacciones no lineales y adaptarse mejor a ruidos y anomalías.

Más recientemente, el aprendizaje profundo (Deep Learning) ha supuesto un avance significativo en la modelización de series temporales complejas. Las redes neuronales recurrentes (RNN), en particular sus variantes LSTM (Long Short-Term Memory) y GRU (Gated Recurrent Unit), han mostrado una capacidad notable para modelar dependencias temporales prolongadas, capturar patrones no triviales y manejar entradas multivariadas. Estas arquitecturas han sido empleadas con éxito en diversos trabajos, incluido el TFM de Carlos de los Santos [2], con mejoras claras respecto a los modelos clásicos.

No obstante, estas redes también presentan limitaciones relevantes: requieren un diseño manual de entradas (como la selección del número de retardos y variables), tienen dificultades para capturar relaciones de largo alcance o efectos cruzados entre ubicaciones espaciales, y son sensibles a errores en el preprocesado de datos.

Frente a estas limitaciones, se han desarrollado arquitecturas más avanzadas, como los Transformers, que sustituyen la recurrencia por mecanismos de atención (self-attention), permitiendo capturar dependencias globales en la secuencia de forma más eficiente. Modelos como Informer o PatchTST han sido adaptados con éxito al dominio del forecasting energético.

De forma aún más reciente, se han introducido las Kolmogorov-Arnold Networks (KAN) [5], una clase emergente de redes neuronales basadas en el teorema de representación de Kolmogorov-Arnold, que permite construir modelos altamente expresivos con menor número de parámetros. Gracias a sus funciones de activación parametrizables y entrenables, estas redes ofrecen una forma novedosa y prometedora de abordar problemas complejos de predicción multivariante.

En este contexto, el presente trabajo busca evaluar de forma comparativa el rendimiento de estas arquitecturas avanzadas —Transformers y KANs— frente a los modelos tradicionales y recurrentes ya explorados en trabajos anteriores. Esta evaluación se realiza sobre datos reales de un parque eólico marino, con el objetivo de contribuir a la mejora de las herramientas de predicción aplicadas a la operación de energías renovables offshore.

1.1.2 Resumen de las contribuciones del TFM de Carlos de los Santos

El Trabajo Fin de Máster (TFM) de Carlos de los Santos Jiménez [2], titulado *"Análisis comparativo de modelos de aprendizaje automático para la predicción de la generación eléctrica de un parque eólico marino"*, ofrece un estudio riguroso y aplicado sobre la capacidad predictiva de diversos modelos de Machine Learning aplicados al contexto energético. Las principales contribuciones del trabajo se estructuran en torno a los siguientes bloques:

1.1.2.1 Contextualización y motivación del problema

Se enmarca su trabajo dentro del reto que supone la integración de energías renovables, en particular la eólica marina, en los mercados eléctricos intradiarios. Destaca la importancia de contar con herramientas predictivas fiables en horizontes de 1 a 9 horas para corregir las ofertas realizadas en el mercado diario, y justifica la necesidad de métodos avanzados que mejoren la precisión frente a modelos tradicionales.

1.1.2.2 Selección del caso de estudio y recopilación de datos

Se selecciona como caso de estudio el parque eólico marino *Alpha Ventus*, situado en el mar del Norte. Se integran datos históricos proporcionados por los sensores del parque (vía RAVE) junto con predicciones meteorológicas del modelo ICON-D2. Esta combinación de fuentes de datos constituye una aportación metodológica relevante que enriquece la base sobre la que se entrenan los modelos.

1.1.2.3 Metodología de preprocesado y selección de variables

El trabajo presenta un procedimiento sistemático para la preparación de los datos:

- Selección de variables con alta correlación con la potencia eólica (>0.2).
- Imputación de valores perdidos mediante XGBoost.
- Eliminación de valores atípicos usando Isolation Forest. Estas técnicas permiten construir una base de datos robusta para el posterior entrenamiento.

1.1.2.4 Implementación y entrenamiento de modelos

Se implementan cinco modelos de predicción:

- Modelos de referencia: Persistencia y ARIMA.
- Modelos avanzados: MLP, LSTM y GRU.

El entrenamiento se realiza para diferentes configuraciones y horizontes temporales, evaluando los resultados con métricas como MAE y RMSE. Se presta especial atención a la elección de hiperparámetros, número de retardos y variables explicativas.

1.1.2.5 Evaluación comparativa y resultados

Una de las principales contribuciones del TFM es la comparación cuantitativa de los modelos, donde se concluye que:

- Todos los modelos basados en redes neuronales (MLP, LSTM y GRU) superan claramente a los modelos de referencia.
- El modelo GRU ofrece el mejor desempeño para horizontes de predicción cercanos.

Sin embargo, a pesar de los buenos resultados obtenidos, estos modelos presentan limitaciones que afectan su aplicabilidad en entornos reales: son altamente sensibles a la calidad del preprocesado, requieren un diseño manual de la entrada (por ejemplo, la selección del número de retardos y las variables), y tienen dificultades para capturar dependencias a largo plazo o relaciones espaciales complejas presentes en los datos meteorológicos. Estas limitaciones justificaron el planteamiento de un nuevo enfoque para avanzar en la línea de investigación: explorar arquitecturas más modernas y expresivas dentro del Deep Learning, como Transformers, que permitan superar estas barreras y capturar mejor la complejidad de la predicción eólica marina.

1.1.3 Conjunto de datos AV

El estudio se centra en la predicción de la generación eléctrica en un parque eólico marino, utilizando como caso de estudio el parque Alpha Ventus, ubicado en el mar del Norte. Para ello, se emplea un conjunto de datos híbrido compuesto por:

- Serie histórica RAVE, que contiene las mediciones registradas por los sensores instalados en los aerogeneradores, obtenidas a través del portal de la Agencia Federal Marítima e Hidrográfica de Alemania (BSH), una vez autorizado su uso para fines educativos.
- Predicciones meteorológicas (NWP) obtenidas del modelo ICON-D2, con una resolución espacial de 2,2 km, que cubre Alemania, Benelux, Suiza, Austria y áreas colindantes.

Estas variables están disponibles con una frecuencia entre 0,2 y 50 Hz, aunque en el estudio se usan agregados de 10 minutos (media, máximo, mínimo y desviación típica). Además, cada medición incorpora dos indicadores de calidad: flag y detailed_flag, que evalúan la fiabilidad del dato.

Se seleccionan como variables predictoras aquellas con una correlación superior a 0,2 respecto a la potencia eólica. Entre ellas destacan: la velocidad del viento, la presión atmosférica, la velocidad del generador, y valores anteriores de la potencia.

Además, se hace un uso explícito del viento como variable exógena en los modelos, ya que su comportamiento influye directamente sobre la generación eléctrica. Este enfoque está respaldado por estudios recientes, como en el artículo por Hanifi [6], donde se demuestra que la velocidad del viento —tanto medida como pronosticada— es la variable exógena más determinante en la predicción de la potencia generada en parques eólicos marinos. En dicho trabajo, se obtienen mejoras significativas al centrarse únicamente en variables relacionadas con el viento, eliminando ruido introducido por otras entradas menos relevantes y reduciendo la complejidad del modelo sin sacrificar precisión.

1.1.3.1 Preprocesado de los datos

El pipeline de tratamiento de datos incluye varios pasos fundamentales:

- **Paso 1: Selección de variables.** Se escogen aquellas más relevantes con base en su correlación con la variable objetivo.
- **Paso 2: Imputación de valores perdidos.** Se utiliza el algoritmo XGBoost, entrenado sobre las propias variables del dataset, junto con la ecuación teórica de la potencia eólica, para estimar valores ausentes
$$P = \frac{1}{2} A \rho v^3 C_p$$
- **Paso 3: Eliminación de valores atípicos.** Se aplica el algoritmo Isolation Forest para detectar y eliminar atípicos que podrían distorsionar la predicción.

- **Paso 4: Preparación para modelos temporales.** Se generan retardos (lags) de las variables explicativas como entradas para redes neuronales recurrentes.

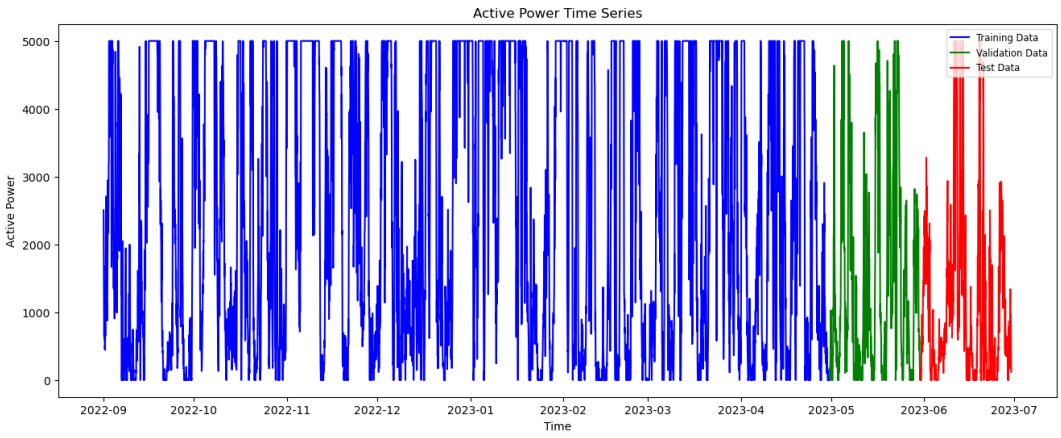


Figura 1.1: Active Power Time series AV

1.1.4 Conjunto de datos KaggleWPGD

El análisis se basa en un conjunto de datos público titulado Wind Power Generation Data – Forecasting, disponible en la plataforma Kaggle[1], que proporciona información meteorológica y de producción eléctrica para una localización concreta.

Este conjunto incluye observaciones horarias desde el 2 de enero de 2017 hasta el 30 de octubre de 2017, con un total de 7248 registros. Cada entrada contiene variables meteorológicas tales como:

- Velocidad del viento a 10 metros,
- Temperatura a 2 metros,
- Humedad relativa,
- Punto de rocío,
- Dirección del viento a diferentes alturas,
- Ráfagas de viento,

junto con la variable objetivo: la potencia activa generada (Active_Power), expresada en pu per unit.

De todas las variables disponibles, se ha seleccionado explícitamente la velocidad del viento a 10 metros como variable exógena, dada su alta correlación con la producción eléctrica, y su relevancia probada en estudios recientes como el de Hanifi [6]. Esta decisión permite centrar el modelado en el factor físico más influyente, reduciendo complejidad y evitando introducir ruido con variables de menor relevancia.

1.1.4.1 Preprocesado de los datos

El tratamiento de datos se ha estructurado en los siguientes pasos:

[1] https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

- Paso 1: Selección de variables. Se conservan únicamente las columnas de Active_Power y Wind_speed_tower}, que representan respectivamente la variable objetivo y la variable exógena de entrada al modelo.
- Paso 2: Comprobación de calidad. Se verifica la ausencia de valores perdidos y de fechas duplicadas, así como la cobertura completa de la serie temporal con pasos horarios regulares.
- Paso 3: División temporal. Se divide el conjunto en subconjuntos de entrenamiento, validación y prueba con proporciones 80%-10%-10%, manteniendo la integridad temporal. Específicamente, se utilizan 5800 registros para entrenamiento, 724 para validación y otros 724 para prueba.
- Paso 4: Escalado. Se aplica una normalización MinMax a todas las variables, ajustando los parámetros sobre el conjunto de entrenamiento y replicándolos en los conjuntos de validación y prueba para evitar fuga de información.
- Paso 5: Formateo del dataset. Finalmente, se adapta el conjunto a la estructura requerida por la librería NeuralForecast [unique_id, ds, y, x].

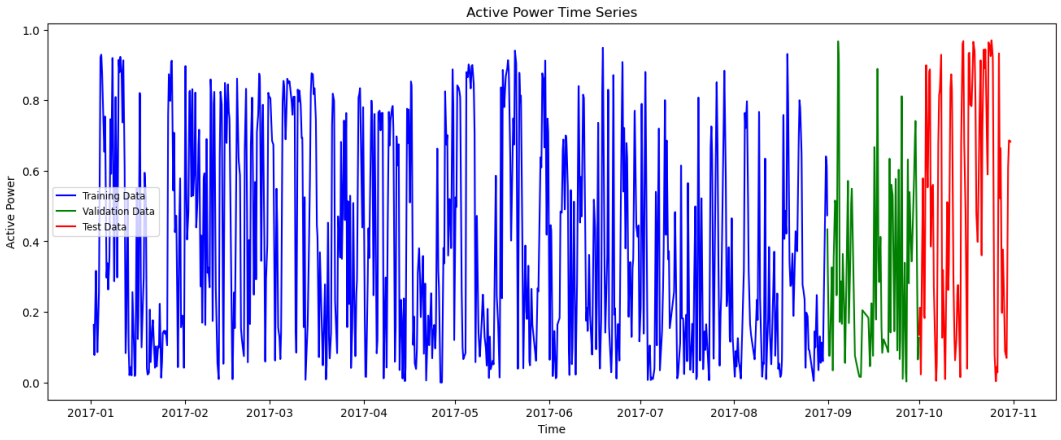


Figura 1.2: Active Power Time series Kaggle

1.2 Estado del arte

La predicción de series temporales en contextos energéticos, especialmente en generación eólica marina, ha evolucionado significativamente en los últimos años. Este avance ha estado impulsado por el aumento en la disponibilidad de datos, mejoras en la capacidad computacional y la proliferación de nuevas arquitecturas de aprendizaje profundo. En esta sección se presenta una revisión de los principales enfoques metodológicos utilizados en la predicción de la generación eléctrica en parques eólicos, desde modelos estadísticos clásicos hasta los desarrollos más recientes basados en redes neuronales avanzadas.

1.2.1 Modelos estadísticos tradicionales

Los modelos estadísticos, como ARIMA (AutoRegressive Integrated Moving Average), han constituido durante décadas el pilar de la predicción en series temporales. Su capacidad para capturar patrones lineales y estacionalidades ha sido ampliamente validada en una gran variedad de dominios.

No obstante, su rendimiento se ve comprometido cuando se enfrentan a datos con dinámicas altamente no lineales o patrones complejos, como los que caracterizan la generación eólica en entornos marinos.

Además, estos modelos requieren un ajuste manual considerable, tanto en la identificación de parámetros como en la preparación de los datos, lo que limita su escalabilidad frente a conjuntos de datos masivos o multivariantes. A pesar de ello, modelos como ARIMA siguen siendo utilizados como referencia (baseline) en estudios comparativos debido a su interpretabilidad y solidez teórica.

1.2.2 Enfoques de Machine Learning y Deep Learning

El uso de técnicas de aprendizaje automático ha incrementado significativamente en el ámbito de la predicción energética. Algoritmos como Random Forests, Support Vector Machines y, especialmente, los modelos de **Gradient Boosting** (como XGBoost o LightGBM), han demostrado ser altamente competitivos en tareas de forecasting, desplazando gradualmente a los modelos estadísticos tradicionales en múltiples benchmarks.

En paralelo, el aprendizaje profundo ha ofrecido herramientas especialmente prometedoras para capturar relaciones temporales complejas. Las **Redes Neuronales Recurrentes** (RNN) y sus variantes mejoradas, **Long Short-Term Memory** (LSTM) y **Gated Recurrent Units** (GRU), han sido ampliamente adoptadas por su capacidad para manejar dependencias a largo plazo y su relativa robustez al ruido. Estas arquitecturas fueron el núcleo del trabajo de Carlos de los Santos [2], y se utilizan en este TFM como referencia para evaluar mejoras introducidas por modelos más recientes.

1.2.3 Modelos basados en Transformers

Desde la introducción del modelo **Transformer** por Vaswani [7], se ha producido una transformación profunda en la forma de abordar el procesamiento secuencial. Su mecanismo de *self-attention*, que permite capturar relaciones a distintas escalas temporales sin necesidad de recurrencia, ha dado lugar a múltiples variantes adaptadas al forecasting, como **PatchTST** [8] y el resto de modelos que veremos en el Capítulo 4 de este TFM.

1.2.4 Modelos con arquitecturas MLP y Modelos KAN

En 2024, se introducen las **Kolmogorov-Arnold Networks (KAN)** [5] , una nueva clase de redes neuronales donde las funciones de activación son parametrizables y entrenables. Inspiradas en el teorema de representación de Kolmogorov-Arnold, estas redes presentan una alternativa radical a las arquitecturas neuronales tradicionales, permitiendo una mayor expresividad con menor número de parámetros.

Aunque todavía en una fase exploratoria, los modelos KAN han comenzado a demostrar su aplicabilidad a problemas de *forecasting*, en particular mediante variantes como **Mixture of KANs (MoK)**. Su capacidad para aproximar funciones multivariantes complejas las posiciona como una dirección de investigación emergente, y este TFM incluye su evaluación experimental como parte de la comparativa.

Además de los avances representados por las KAN, recientemente se ha observado un resurgimiento del interés por las arquitecturas basadas en MLP en el ámbito de las series temporales. Modelos como **TiDE (Time-series Dense Encoder)** y **N-HiTS (Neural Hierarchical Interpolation**

for Time Series) han demostrado un rendimiento competitivo frente a arquitecturas más complejas como Transformers, combinando eficiencia computacional y precisión. TiDE introduce una separación explícita entre la codificación del pasado y la predicción futura, utilizando capas densas para capturar relaciones no lineales sin recurrencia ni mecanismos de atención. Por su parte, N-HiTS propone una arquitectura jerárquica y modular que propone un muestreo a diferentes frecuencias y una interpolación jerárquica, lo que permite una mayor capacidad de generalización y mejor manejo de horizontes de predicción largos. Estos modelos destacan por su escalabilidad, facilidad de entrenamiento y simplicidad estructural.

1.2.5 Comparativas existentes en el ámbito eólico

Diversos estudios han abordado la predicción de generación eólica mediante técnicas de Machine Learning y Deep Learning [6], con resultados alentadores. Sin embargo, gran parte de la literatura se centra en parques terrestres o modelos estadísticos convencionales, y existe una carencia de estudios sistemáticos que comparen arquitecturas avanzadas (Transformer, KAN) específicamente en entornos marinos.

Este trabajo busca llenar ese vacío mediante una evaluación sistemática de modelos tradicionales, recurrentes y no recurrentes de última generación, aplicada al mismo conjunto de datos utilizado en trabajos previos. El objetivo es proporcionar una panorámica actualizada y orientada a la práctica sobre el rendimiento relativo de estas arquitecturas en tareas de predicción de generación eólica marina.

1.2.6 Las competiciones M

Las competiciones M fueron creadas por Spyros Makridakis en los años 80 como una forma de evaluar de forma objetiva el comportamiento de los modelos de predicción de series temporales en un contexto de problemas del mundo real, de forma similar a cómo se estaba haciendo en otros problemas de Machine Learning o Deep Learning. Las primeras competiciones, de la M1 en 1982 a la M3 en el 2000, espaciadas aproximadamente cada década, estuvieron dominadas por los modelos que ahora consideramos clásicos, como la familia ARIMA o los métodos de Holt-Winters. Estas primeras competiciones usaban un conjunto de pocos miles de series temporales como banco de pruebas, con distintas estacionalidades y procedentes de ámbitos diversos.

La competición M4 de 2018 supuso un cambio importante, tanto por la cantidad de series temporales (cien mil, más variadas que en anteriores ediciones) como por su resultado. Por primera vez la competición fue dominada por los métodos híbridos, que combinaban los clásicos con Métodos de Machine Learning en estructuras tipo ensemble. Ese cambio de tendencia se consolidó en la competición M5 de 2020, en la que los métodos de Machine Learning, especialmente los de tipo boosting (ver Sección 3.3) dominaron claramente las posiciones ganadoras. Aunque la selección de series para esta edición ha sido criticada por su sesgo, al ser menos variada que las anteriores, la atención de la comunidad de investigadores y usuarios de estos métodos empezó a desplazarse desde entonces claramente hacia los métodos basados en Machine Learning, por más que las técnicas clásicas sigan jugando un papel esencial. Eso ha quedado de manifiesto también en la última edición, la M6 en 2023 [9], en la que los métodos ganadores fueron combinaciones de métodos clásicos y métodos de Machine Learning.

En cuanto a los métodos basados en redes neuronales, su participación en estas competiciones ha sido creciente, pero hasta la fecha no han conseguido desplazar a esas combinaciones ganadoras de las últimas ediciones. Esto puede deberse a la selección de series temporales empleadas en las

últimas ediciones, que han sido criticadas por sus sesgos hacia tipos específicos de series. En el resto de este trabajo discutiremos cuál es la situación actual, qué tipo de modelos se están planteando y cuáles son las perspectivas que cabe esperar en el futuro inmediato. Lo que es innegable es que el interés y el desarrollo de este tipo de modelos ha aumentado extraordinariamente en los últimos cinco años.

Capítulo 2

Metodología

2.1 Introducción

Aunque las aplicaciones prácticas han estado siempre presentes, el análisis de series temporales fue en su origen un campo dominado muy mayoritariamente por investigadores especializados en Estadística y Econometría. Los modelos clásicos desarrollados durante ese periodo, como loo métodos de suavizado exponencial (*exponential smoothing*) o la familia de modelos Arima entre otros, se denominan a menudo *modelos estadísticos* precisamente por ese sesgo original. Por esa misma razón las primeras herramientas computacionales que se usaron para estos problemas provienen de esa misma comunidad de investigadores de formación matemática y estadística. El lenguaje de programación S y su sucesor R han dominado durante muchos años en este terreno y siguen siendo, a fecha de hoy, el ecosistema más consolidado de herramientas *para el uso de esos modelos clásicos.* Uno de los exponentes más claros de la madurez de esas herramientas puede encontrarse en el ya clásico libro *Forecasting: Principles and Practice* de R. Hyndman y G. Athanosopoulos [10]. Ese texto, ya en su tercera edición, está disponible online íntegramente y va acompañado inseparablemente de la librería `fpp3` de R.

Lo mismo sucede con muchos otros textos clásicos del análisis de series temporales que, cuando no incluyen código R en el propio texto, a menudo se acompañan de librerías o repositorios de código R para que los lectores puedan reproducir los resultados del análisis por si mismos.

Sin embargo, como hemos comentado en la Sección 1.2, los últimos años han visto aumentar enormemente el interés por los métodos basados en Machine Learning, donde el lenguaje Python está firmemente asentado y tiene una base muy amplia de usuarios y, más recientemente, por los métodos basados en redes neuronales y Deep Learning, donde el dominio de Python frente a otros lenguajes es, a fecha de hoy, esencialmente incontestable.

2.2 El ecosistema actual de series temporales en Python

Como acabamos de ver en la introducción, Python está posicionándose en estos últimos como un lenguaje de referencia en el campo del análisis de series temporales, desplazando a menudo a R que había sido tradicionalmente el lenguaje preferido para estos problemas por los usuarios e investigadores. Es pertinente por tanto para este trabajo analizar las herramientas que el lenguaje Python nos ofrece.

2.2.1 La fragmentación del ecosistema

Si nos aproximamos a esta discusión desde una perspectiva más amplia, podemos destacar varias librerías presentes en Python que son relevantes en lo que sigue:

- Por un lado tenemos la libraría `scikit-learn`[1] [11], un referente establecido para los modelos de Machine Learning. Esta librería está muy bien conectada con muchas otras librerías de análisis de datos, destacando `pandas`[2] [12] para el análisis y manipulación de datos, `numpy`[3] [13] para el análisis numérico y funciones matemáticas o `matplotlib`[4] [14] y `seaborn`[5] [15] para la visualización de datos. Todas estas librerías están muy bien conectadas entre ellas y constituyen el núcleo más cohesionado de herramientas para este tipo de problemas en Python.

- La librería `statsmodels`[6] [16] ocupa un lugar especial y relevante para este trabajo. Esta librería tiene una orientación más puramente estadística y proporciona herramientas para el análisis de modelos clásicos desde una perspectiva más teórica y formal. Podría suponerse que la conexión con las implementaciones de esos modelos en librerías como scikit-learn debería ser profunda y sencilla de utilizar, pero lo cierto es que las dos comunidades de desarrollo parecen no haber explorado esas conexiones con suficiente interés. Ese es uno de los primeros aspectos de la fragmentación a la que nos referimos en este apartado.

Por otro lado, desde la edición de 2012 del concurso de clasificación de imágenes ILSVRC (ImageNet Large Scale Visual Recognition Challenge), en el que una red neuronal convolucional profunda derrotó a todos los modelos de Machine Learning clásicos, las redes de **Aprendizaje Profundo** (Deep Learning) se han convertido en el ingrediente esencial de muchos de los avances en Computación e Inteligencia Artificial. Puede verse una descripción de este periodo en el primer capítulo de [17]. Su estudio teórico ha acompañado a los avances técnicos en el desarrollo de procesadores tipo GPU o TPU cada vez más potentes y accesibles. Desde el principio de esta revolución del Aprendizaje Profundo, Python se posicionó como el lenguaje más utilizado en este campo, gracias a herramientas como:

- El binomio Tensorflow/Keras[7] [19]. Estas librerías proporcionan respectivamente accesos de bajo y alto nivel a la arquitectura de redes neuronales y a las operaciones tensoriales necesarias para su entrenamiento, y facilitan mucho el aprovechamiento de los recursos de paralelización que proporcionan las GPUs y TPUS. Tensorflow es un desarrollo de Google y su uso se ha popularizado enormemente gracias a Keras [20].

- La competencia de ese binomio es la librería Pytorch[8] [21], desarrollada en Facebook en 2016. Aunque inicialmente más relevante en el ámbito académico, mientras Tensorflow dominaba las aplicaciones, en los últimos años Pytorch ha ganado mucha popularidad gracias a su integración con el desarrollo de modelos de tipo transformer y plataformas como *Hugging Face*[22]. El uso de Pytorch se describe en detalle en referencias como [23] o, para el análisis de series temporales [24]

La disyuntiva entre Pytorch y Tensorflow/Keras contribuye también a la fragmentación y se traslada

[1] Web oficial https://scikit-learn.org/
[2] Web oficial https://pandas.pydata.org
[3] Web oficial https://numpy.org
[4] Web oficial http://matplotlib.org/
[5] Web oficial https://seaborn.pydata.org
[6] Web oficial https://www.statsmodels.org
[7] Webs oficiales https://tensorflow.org/ y https://keras.io
[8] Web oficial: https://pytorch.org

a cualquier desarrollo que utilice redes neuronales en Python, como veremos a continuación.

2.2.2 Librerías de análisis de series temporales en Python

Podemos resumir la discusión anterior diciendo que en los últimos años Python ha consolidado su dominio en el ámbito del Machine Learning, donde su ecosistema de librerías está más cohesionado, y ha conquistado casi por completo el nuevo campo del Deep Learning, pese a la fragmentación existente. Gracias a ello ha ido desplazando a otros lenguajes como R, mientras que alternativas más recientes como Julia[9] [25] no disponen todavía de un ecosistema competitivo con el de Python. En el terreno del análisis de series temporales, donde Python no era hasta poco un competidor a la altura de R, se da actualmente una situación de *alta fragmentación* en la que tenemos:

- Librerías como `statsmodels`, que ya hemos mencionado, o `pmdarima`[26] ofrecen implementaciones de modelos clásicos como la familia Arima, modelos Holt-Winters, etc. En general desde un punto de vista más formal y estadístico, similar a lo que ofrecen varias librerías de R pero con menos popularidad en la comunidad académica de orientación estadística.
- La librería `scikit-learn` ofrece un conjunto pequeño y poco estructurado de herramientas para el trabajo con series temporales, desconectado del desarrollo de `statsmodels`.

En ese contexto muchas otras librerías han ido surgiendo para llenar el evidente vacío existente en Python. Destacamos:

- `sktime`[10] [27], que pretende dotar a Python de un marco completo para el trabajo con series temporales, buscando una interfaz muy similar a la de `scikit-learn` con la que la gran mayoría de usuarios ya están familiarizados. Ofrece muchos modelos clásicos, dispone de muy buena documentación y permite utilizar esencialmente cualquier modelo de regresión implementado en `scikit-learn` para predicción de series temporales. Pero el número de modelos recientes ya implementados es todavía pequeño, y no incluye a muchos de los modelos de interés en este TFM.

- `darts`[11] [herzen2022darts] es en muchos sentidos un proyecto similar a `sktime`, con la misma intención de usar el lenguaje familiar de `scikit-learn` como vía de acceso a una amplia base de usuarios. Ofrece una colección muy similar de modelos, y de nuevo se queda corta a la hora de implementar los modelos más recientes.

Junto a estas dos existe un amplio abánico de librerías menos populares o menos desarrolladas como `GluonTS`[12], especializada en predicciones probabilísticas, o como `skforecast`[13], etc.

2.2.3 El Nixtlaverse

Durante el desarrollo de este TFM, apareció *Forecasting: Principles and Practice, the Pythonic Way* [28]. Se trata de la versión en Python del clásico de Hyndman que ya hemos mencionado y para el que se ha optado por utilizar como herramienta las librerías del llamado *Nixtlaverse*[14] [29], un conjunto de librerías de análisis de series temporales en Python. De hecho, parte del equipo de desarrollo de Nixtla figuran como coautores de este nueva versión del clásico. Dada la enorme influencia de las

[9] Web oficial: https://julialang.org
[10] Web ofocial https://www.sktime.net
[11] Web oficial https://unit8co.github.io/darts/
[12] Web oficial https://ts.gluon.ai/stable/
[13] Web oficial https://skforecast.org
[14] Web oficial https://nixtlaverse.nixtla.io

librerías de R creadas por Hyndman y su equipo hasta la fecha, junto con el evidente esfuerzo que se está haciendo por incorporar al Nixtlaverse los últimos resultados de la investigación muy poco tiempo después de su publicación, creemos que las librerías del Nixtlaverse pueden ser una parte importante del futuro a corto y medio plazo del análisis de series temporales mediante Python. Hemos querido destacar este hecho para justificar parte de las decisiones de implementación que se han tomado en este trabajo. En particular, nuestra elección de las librerías de Nixtla para los modelos que se han usado en los siguientes capítulos.

Como hemos dicho, el Nixtlaverse es un conjunto formado por varias librerías, entre las que destacamos:

- `StatsForecast`, que contiene implementaciones de modelos clásicos como la familia Arima, modelos Holt-Winters de suavizado exponencial, modelos GARCH, etc.
- `MLForecast` que, más que ofrecer una amplia colección de modelos, se especializa en tender un puente hacia librerías como `scikit-learn` o `XGBoost` para permitir el uso de cualquiera de sus modelos de regresión en la predicción de series temporales.
- `NeuralForecast`, especializada en modelos de series temporales basados en redes neuronales y que incluye desde los modelos más clásicos, como MLP o LSTM hasta los modelos más recientes, algunos publicados durante la realización de este TFM.

Esta última componente del Nixtlaverse es la que nos ha llevado finalmente a elegir este conjunto de librerías para el trabajo de los siguientes capítulos.

2.3 Consideraciones generales sobre la implementación de los modelos

Vamos a describir a continuación de forma genérica la forma en la que aplicaremos los modelos que vamos a analizar utilizando las herramientas que hemos descrito.

2.3.1 Estrategia de validación y esquema general del flujo de trabajo

Una vez preprocesado el conjunto de datos (usando librerías como `pandas` o `matplotlib`) los siguientes pasos generales de un flujo de trabajo para el ajuste y evaluación de un modelo de predicción son los siguientes:

2.3.1.1 Conjuntos de training, validación y test

En la predicción de series temporales, debemos tener cuidado al dividir los datos en conjuntos de entrenamiento, validación y prueba. No podemos usar divisiones aleatorias, porque necesitamos preservar el orden temporal de los datos y debemos evitar usar datos futuros desconocidos para predecir datos del pasado.

Por lo tanto, debemos utilizar divisiones que respeten esa ordenación temporal, en las que el conjunto de entrenamiento contiene los datos hasta cierto punto en el tiempo, y el conjunto de prueba contiene los datos a partir de ese punto. La proporción de la división suele ser 80/20 o 70/30 del intervalo temporal de la serie, como en otros problemas de aprendizaje automático. A menudo, la división se realiza directamente separando los datos en dos partes, pero algunas bibliotecas (como `sktime` o `scikit-learn`) tienen funciones específicas para realizar esta división.

2.3.1.2 Selección de hiperparámetros del modelo. La librería Optuna.

La mayoría de los modelos de Machine Learning y todos los modelos de Deep Learning incluyen *hiperparámetros*. Se trata de valores que determinan la arquitectura variable del modelo y que no son entrenables como los llamados *coeficientes* o *pesos* del modelo. Ejemplos típicos de hiperparámetros son el número de capas de una red neuronal, el número de neuronas en esa capa o la profundidad de un árbol binario. Sus valores deben decidirse de antemano antes de poder evaluar la función de pérdida que rige el entrenamiento del modelo. Pero esa elección tiene un impacto a menudo determinante en el comportamiento del modelo y, en particular, juega un papel esencial en el control del posible sobreajuste del modelo y en el dilema sesgo-varianza propio de estos modelos Sección 2.2 de [30].

Para elegir los valores óptimos de los hiperparámetros se utilizan habitualmente estrategias de **validación cruzada** (cross validation). En el caso de las series temporales, esas estrategias deben tener en cuenta nuevamente la ordenación temporal de los datos. En particular se utiliza a menudo la estrategia de *expansión de la ventana*, que se ilustra en la Figura 2.1, procedente de la Sección 5.10 de [10]

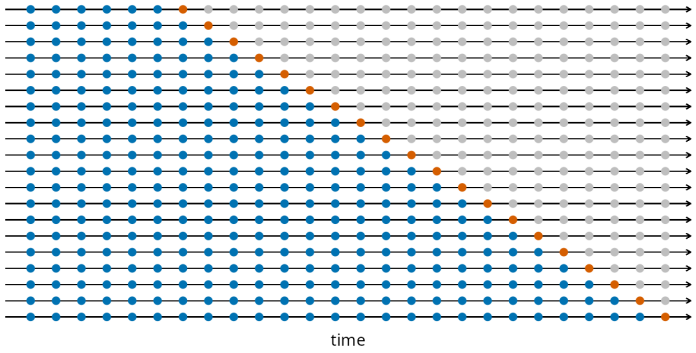


Figura 2.1: Validación cruzada en series temporales, con horizonte de predicción $h = 1$.

En esta figura los puntos azules representan el conjunto de entrenamiento que, como se ve, va aumentando en cada iteración del proceso de validación. Ese aumento puede ser en pasos (*strides*) de varias unidades, pudiendo a menudo ajustarse para definir el conjunto de *folds* que usamos para la validación. El punto rojo representa el horizonte de predicción, siendo $h = 1$ en este caso. La Figura 2.2, procedente asimismo de la Sección 5.10 de [10], ilustra el caso de horizonte de predicción $h = 4$.

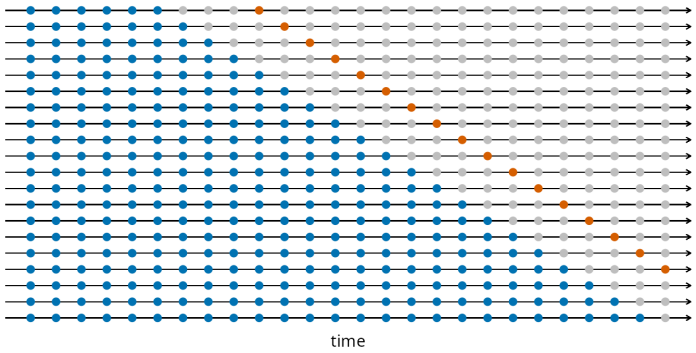


Figura 2.2: Validación cruzada en series temporales, con horizonte de predicción 4.

Como hemos dicho, esta estrategia de validación, repetida en varias iteraciones, permite elegir los mejores hiperparámetros del modelo, modificándolos para obtener valores mínimos de la función de pérdida. Como se ve, en cada fold de este proceso se usa un conjunto de validación disjunto del de entrenamiento para poder evaluar el comportamiento del modelo sin utilizar el conjunto de test que utilizaremos para medir finalmente la calidad y capacidad de generalización de las predicciones del modelo.

La búsqueda de los hiperparámetros óptimos se puede realizar mediante búsqueda exhaustiva (*grid search*) en un espacio determinado, mediante búsqueda al azar (*random search*) o mediante métodos más avanzados de optimización bayesiana. Hay diversas librerías de Python que proporcionan herramientas para llevar a cabo esta búsqueda (ver [31]). Las librerías del Nixtlaverse permiten utilizar esas librerías de forma sencilla, de manera que el usuario esencialmente debe limitarse a definir el espacio de hiperprámetros a explorar y el método de búsqueda elegido. En muchos de los modelos de este TFM se ha optado por utilizar, a través del Nixtlaverse, las herramientas de selección bayesiana de hiperparámetros que proporciona la librería `Optuna`[15] [32]

Una vez elegidos los valores de los hiperparámetros el modelo resultante se reentrena de nuevo utilizando ahora todos los datos que se han usado para entrenamiento y validación, buscando con ello ajustar los mejores valores posibles de los coeficientes, una vez que la arquitectura del modelo ha quedado fijada.

2.3.1.3 Obtención de las predicciones del modelo en el conjunto de test y evaluación del modelo

El paso final consiste en comparar las predicciones del modelo con los valores reales de la serie temporal en el conjunto de test y obtener las métricas correspondientes. Existen muchas métricas disponibles para este paso. En este TFM, siguiendo las recomendaciones de la Sección 5.8 de [10], hemos optado por comparar los modelos usando la métrica MAE (*mean absolute error*, error medio absoluto), definida como

$$\text{MAE} = \text{mean}(|e_t|)$$

siendo e_t los **residuos** definidos para cada instante de tiempo t mediante:

$$e_t = y_t - \hat{y}_t$$

donde y_t es el valor real u observado de la serie temporal e $\hat{y}_t$ es la predicción del modelo.

Los detalles de la forma en la que se obtienen esas predicciones dependen del tipo de modelo y del horizonte de predicción utilizado, por lo que a continuación precisaremos un poco más esta cuestión.

2.3.2 El caso de los modelos clásicos y los modelos de Machine Learning

Este tipo de modelos producen sus predicciones secuencialmente, un instante de tiempo cada vez. Por esa razón, cuando se utilizan para horizontes temporales mayores que 1 se presentan varias opciones:

- **Predicción recursiva:** en este caso, suponemos conocido el pasado de la serie hasta el instante t y para predecir los valores del horizonte de predicción $y_{t+1}, \dots, y_{t+h}$ procedemos recursivamente:

  1. Predecimos el primer valor, sea $\hat{y}_{t+1}$ esa predicción.

[15] Web oficial https://optuna.readthedocs.io

2. Añadimos esa predicción al conjunto de entrenamiento, entrenamos un nuevo modelo usándola y obtenemos $\hat{y}_{t+2}$ con este modelo.
3. Continuamos así hasta obtener $\hat{y}_{t+h}$.

Este procedimiento recursivo tiene varias limitaciones:

- Tiende a acumular un error mayor que otros enfoques alternativos.
- No se puede aplicar de forma sencilla si el modelo usa variables exógenas dinámicas que no son fácilmente predecibles (como el viento en nuestro caso), es decir que no son *variables de calendario*.

- **Predicción multietapa:** modificar el modelo (y la función de pérdida y el proceso de entrenamiento) para hacer que el modelo prediga una salida vectorial en lugar de un escalar. Este enfoque no es habitual y muchas librerías no le dan soporte directo.

- **Predicción directa:** en este caso entrenamos un modelo independiente para cada instante del horizonte de predicción. Es decir que tenemos modelos $M_1, \dots, M_h$ en los que M_i predice $\hat{y}_{t+i}$ usando los valores conocidos (el pasado hasta t) $y_1 \dots, y_t$ de la variable junto con el pasado $\mathbf{x}_1, \dots, \mathbf{x}_t$ de las variables exógenas (no-calendario, las de calendario no son una dificultad en cualquier caso). El modelo M_i es de hecho un modelo a horizonte 1 de la variable salida desplazada $y_{t-(-i)}$ (lo escribimos así para subrayar que en el código se usa un *retardo negativo*).

  Este es el método que usaremos en este trabajo para obtener las prediccciones de un modelo de Machine Learning como es el modelo XGBoost en la sección 3.3 (pág. 37). Al hacerlo es *esencial* asegurar que el modelo M_i sólo tiene acceso a los valores del pasado de la serie y de las variables endógenas que no podemos predecir.

2.3.3 El caso de los modelos de redes neuronales

Los modelos de redes neuronales pueden ser, por el contrario, modelos secuencia-a-secuencia (en inglés *seq2seq models*) de forma nativa. Es decir, que su propia arquitectura y el algoritmo de retropropagación temporal (*backpropagation through time BPTT*) que usamos para entrenarlos se prestan a que la variable de salida sea multidimensional. En particular, se puede entrenar a estos modelos para que hagan **predicción directa en un paso** (*one-shot prediction*) de todos los valores de un horizonte de predicción. Naturalmente, se puede también usar la red para predicciones unidimensionales y aplicar entonces una de las estrategias que hemos descrito. Pero a menudo eso significaría desaprovechar las arquitecturas de este tipo de modelos que, como veremos en los próximos capítulos, están precisamente diseñadas para aprovechar esta propiedad.

2.3.4 Subconjuntos disponibles del conjunto de test para la evaluación

Supongamos por ejemplo que vamos hacer una predicción a horizonte $h = 3$ usando un modelo de red neuronal. Eso significa que desde un instante t del conjunto de training el modelo hará predicciones para los instantes $t+1$, $t+2$ y $t+3$. Si queremos obtener medidas del comportamiento del modelo en el conjunto de test aprovechando al máximo la información disponible de ese conjunto, eso implica que, como se muestra en la Figura 2.3 (pág. 28), debemos prestar atención al instante inicial y al número máximo de predicciones que podemos obtener. Así, para las predicciones a horizonte $h = 3$ que hemos usado en los siguientes capítulos y en el código y suponiendo que y_t es el último valor de entrenamiento (en realidad validación + entrenamiento, en color naranja en la columna de la izquierda de la figura):

- Para obtener la primera predicción a 3h (se muestran en color morado, en la columna de la derecha) para el primer instante y_{t+1} del conjunto de test, sea (en color azul) debemos usar la predicción hecha desde $t-2$. Por esa misma razón como se muestra, los tres últimos instantes del conjunto de test no se pueden usar para obtener esas predicciones a 3 horas: al ir ampliando el conjunto que se usa, debemos tener en cuenta el último valor desde el que podemos hacer una predicción a 3h *que se pueda comparar con un valor real observado del conjunto de test.*
- De la misma forma para obtener la primera predicción a 2h (en color verde) o a 1h (en color gris), debemos debemos usar las predicciones hechas desde $t-1$ y t. Y quedarán sin usar para la predicción, respectivamente, los dos últimos o el último instante del conjunto de test.

Estas observaciones se han tenido al diseñar los bucles de evaluación de los modelos, utilizados para calcular la métrica MAE a 1h, 2h y 3h que usaremos para compararlos.

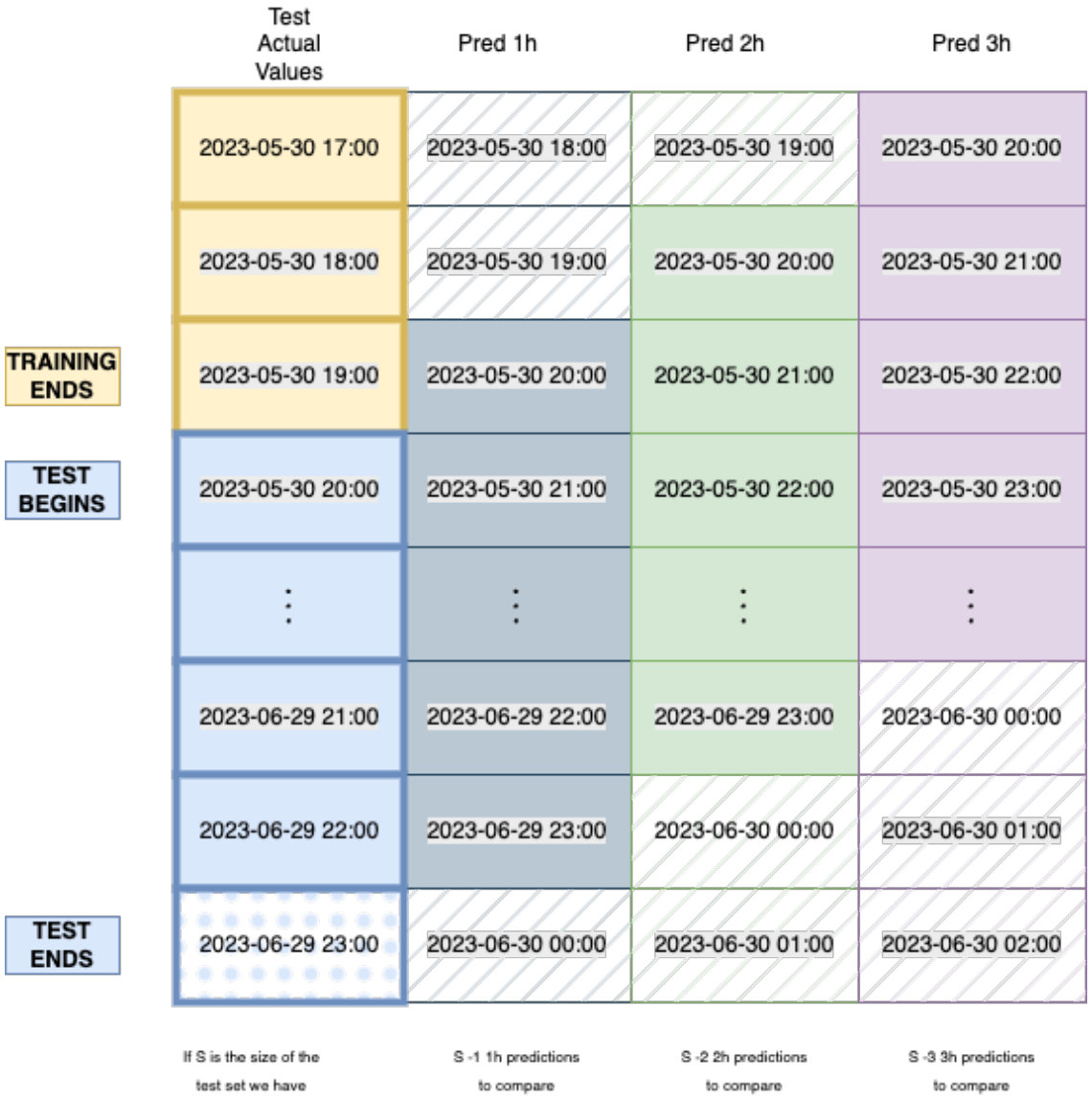


Figura 2.3: Subconjuntos disponibles del conjunto de test para la evaluación

2.4 El repositorio de código

Como se ha descrito en secciones previas, los modelos que se han abordado en este trabajo son muy recientes (algunos de aparición durante su desarrollo). Y las librerías de Python que se han utilizado están todavía en una fase de desarrollo temprano, sin haber alcanzado el nivel de madurez de otras herramientas como `scikit-learn`. Por esa razón uno de los objetivos de este trabajo ha sido el de proporcionar ejemplos accesibles, homogéneos y bien organizados del uso de estos modelos para facilitar la tarea de los usuarios que en el futuro quieran emplearlos.

Para ello se ha creado un repositorio público almacenado en la plataforma GitHub[16], en el que están disponibles dichos ejemplos en formato de notebooks de Jupyter y también en formato HTML para facilitar la consulta de código y resultados sin necesidad de un entorno de desarrollo Jupyter completo.

Además en esta memoria se han incluido enlaces a dichos documentos HTML para facilitar su consulta asociada a las correspondientes secciones de este documento.

[16] en la URL https://github.com/fsansegundo/tfm_cc_2025

Capítulo 3

Modelos de referencia

3.1 Introducción

3.2 ARIMA

3.2.1 Modelo ARIMA

Partimos de un modelo clásico de tipo ARIMA como modelo de referencia. De esa forma, tras implementar modelos más complejos, podremos analizar el comportamiento de los distintos modelos y obtener una comparativa de sus rendimientos. Dado que nuestras series de generación eólica no presentan comportamiento estacional, nos limitamos a modelos ARIMA no estacionales. Para la descripción de esta familia de modelos y algunos aspectos de la implementación que usaremos nos remitimos al Capítulo 9 del libro *Forecasting: Principles and Practice, the Pythonic Way* [10] del que ya hemos hablado.

La estructura de un modelo ARIMA no estacional viene dada por la ecuación

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$$

donde

- y'_t representa la serie temporal diferenciada un número d de veces hasta conseguir la estacionaridad (y transformada si es preciso para conseguir estabilizar la varianza).
- ε_t representa una serie temporal de ruido blanco gaussiano.
- c es el término independiente
- la parte **autoregresiva** AR de orden p del modelo tiene coeficientes $\phi_1, \ldots, \phi_p$.
- la parte **media móvil** (*moving average*) MA de orden q del modelo tiene coeficientes $\theta_1, \ldots, \theta_q$.

Usando el operador de retardo $B = 1 - I$ podemos escribir esta ecuación en forma de polinomios operadores:

$$\phi(B)(1 - B)^d y_t = c + \theta(B)\varepsilon_t$$

Los parámetros (p, d, q) caracterizan la estructura de este tipo de modelos que se denominan modelos ARIMA(p, d, q). Esos parámetros constituyen los hiperparámetros del modelo ARIMA. En el enfoque estadístico tradicional un experto realiza un análisis exploratorio de los gráficos de las funciones de autocorrelación y autocorrelación parcial de la serie temporal y propone una estructura del modelo para su ajuste.

3.2.2 El algoritmo AutoARIMA de Hyndman-Khandakar

Ese procedimiento manual de ajuste es laborioso, requiere conocimiento experto, y se presta mal a su automatización para el análisis de conjuntos grandes de series temporales. El énfasis en este procedimiento manual está en la verificación de las hipótesis estadísticas del modelo ARIMA. Frente a esa perspectiva más formal, el enfoque aplicado busca un modelo ARIMA que muestre el mejor comportamiento predictivo, incluso a costa de sacrificar algunos de esos requisitos teóricos. El algoritmo AutoARIMA de Hyndman-Khandakar [33] sirve precisamente para encontrar el *mejor* modelo ARIMA, en ese sentido predictivo.

En versión resumida el algoritmo realiza una búsqueda guiada en el espacio de valores (p, d, q) de acuerdo con estos pasos:

- Determinar el número de diferencias necesarias d, usando contrastes de hipótesis y aplicarlas a la serie.
- Seleccionar los valores de p y q para el modelo ARIMA
  - Para cada combinación de p y q dentro de un rango definido:
    - Ajustar un modelo ARIMA(p, d, q)(P, D, Q)[m] a los datos ya diferenciados.
    - Estimar los parámetros del modelo usando máxima verosimilitud.
    - Evaluar el modelo utilizando un criterio de información (por ejemplo, AICc, AIC o BIC).
- Seleccionar el modelo que tenga el menor valor del criterio de información seleccionado.
- Verificar los residuos del modelo ajustado. Si los residuos parecen ruido blanco aceptar el modelo, en caso contrario continuar la búsqueda.

La Figura 3.1 (pág. 32, Figura 9.11 de [10]) ilustra cómo procede este *algoritmo voraz* realizando una búsqueda en el espacio de parámetros.

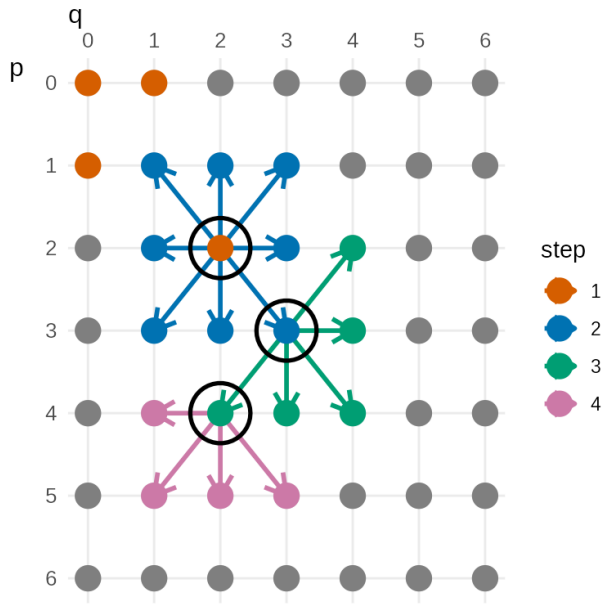


Figura 3.1: Ilustración del algoritmo AutoARIMA de Hyndman & Khandakar

Este algoritmo AutoARIMA representado en el lado derecho de la Figura 3.2 (pág. 33, Figura 9.12 de [10]) se plantea así como una alternativa a la metodología tradicional de Box-Jenkins (que aparecen en el lado izquierdo), descrito en detalle en [34].

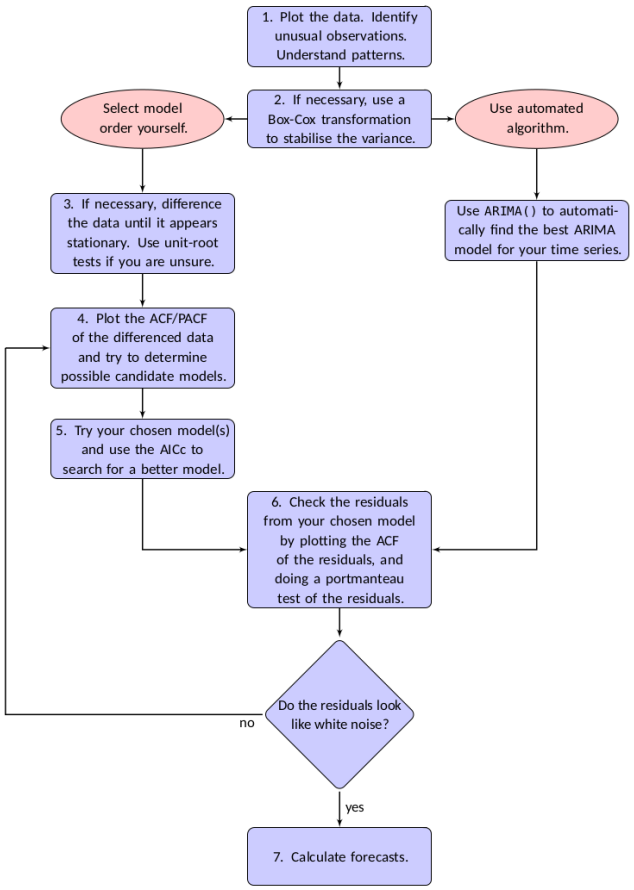


Figura 3.2: Metodología tradicional vs AutoARIMA

Como puede verse, el algoritmo tradicional requiere una intervención más experta del usuario y, aunque en ocasiones el algoritmo AutoARIMA tiende a producir modelos más complejos de los que un experto es capaz de seleccionar, su introducción supuso la posibilidad de incorporar la metodología ARIMA a flujos automatizados o semiautomatizados de predicción de series temporales.

3.2.2.1 Uso de variables exógenas. Modelos de regresión dinámica.

Como se describe en el Capítulo 10 de [10], se puede extender el modelo ARIMA para obtener un modelo de regresión lineal con errores ARIMA de la forma

$$\begin{cases} y_t = \beta_0 + \beta_1 x_{1,t} + \cdots + \beta_k x_{k,t} + n_t \\ \phi(B)(1-B)^d n_t = \theta(B)\varepsilon_t \end{cases}$$

siendo $x_{i,t}$ las variables exógenas del modelo. Es decir, que el *error* del modelo de regresión se modeliza mediante un modelo ARIMA. El algoritmo AutoARIMA se extiende de forma natural a este tipo de **modelos de regresión dinámica.**

3.2.3 Implementación y resultados

La implementación que usaremos es la que está disponible en la librería `StatsForecast` de Nixtla. Veremos a continuación cuál es el tipo de modelo seleccionado- para cada uno de los conjuntos de datos que utilizamos.

Al tratarse de un tipo de modelo clásico, se aplican las consideraciones que hemos hecho en la Sección 2.3.2. En este caso la implementación de Nixtla permite obtener simultáneamente todos los valores del horizonte de predicción (internamente se utiliza la estrategia recursiva que hemos descrito en 2.3.2).

Por otra parte, dado que nuestra variable exógena, el viento, es una variable dinámica que no es predecible, esa estrategia recursiva exige que proporcionemos al modelo una predicción de sus valores. En este caso hemos optado por utilizar la media de valores de la variable en la ventana de entrada del modelo como valor común para el horizonte de predicción.

3.2.3.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `AutoArima_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento AutoArima_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.3 (34). El tiempo de entrenamiento del modelo (con selección de hiperparámetros y predicción de resultados) es de aproximadamente 75 minutos.

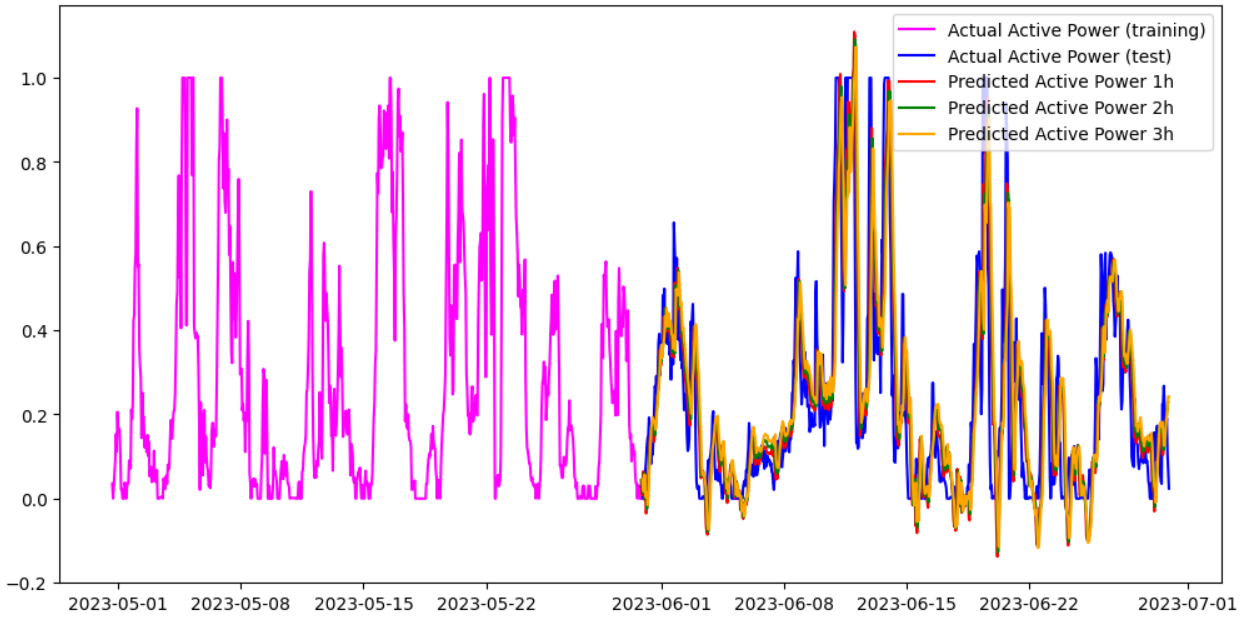


Figura 3.3: Predicciones del modelo AutoArima para el dataset AV

El modelo de regresión dinámica ajustada es un modelo con errores ARIMA(2,0,1).

Los residuos del modelo aparecen en la Figura 3.4 que permite comprobar que se comportan aproximadamente como ruido blanco gaussiano.

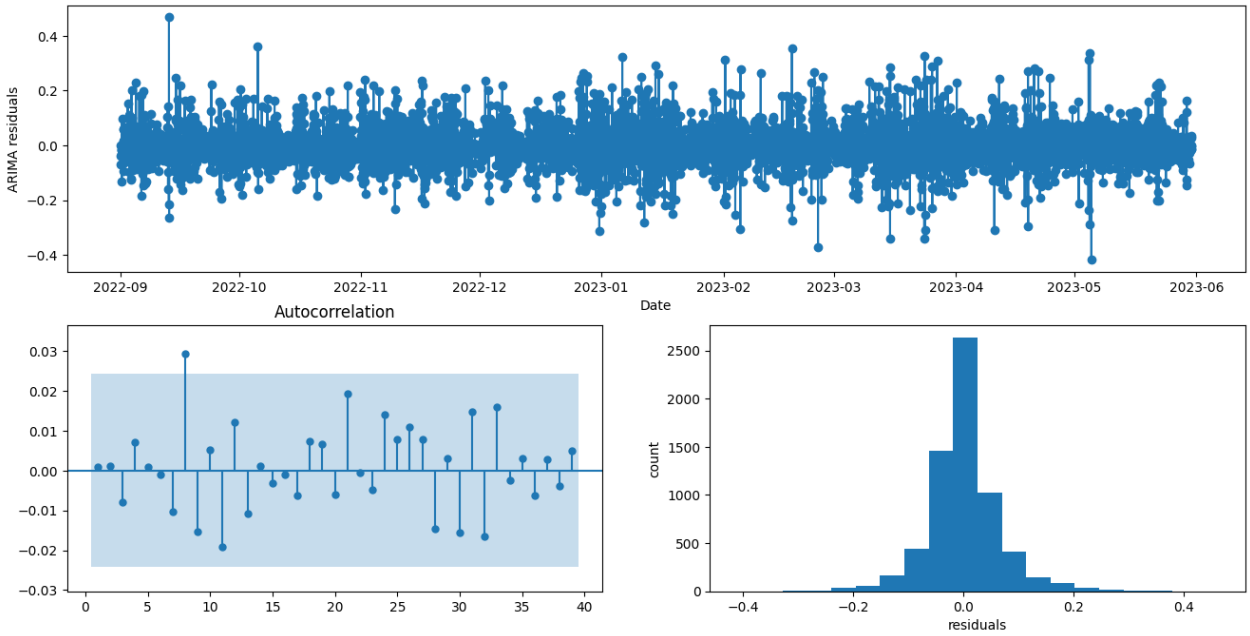


Figura 3.4: Residuos del modelo AutoArima para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 464.44
- MAE a 2h: 471.50
- MAE a 3h: 482.92

3.2.3.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `AutoArima_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento AutoArima_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.5 (36). El tiempo de entrenamiento del modelo (con selección de hiperparámetros y predicción de resultados) es de aproximadamente 94 minutos.

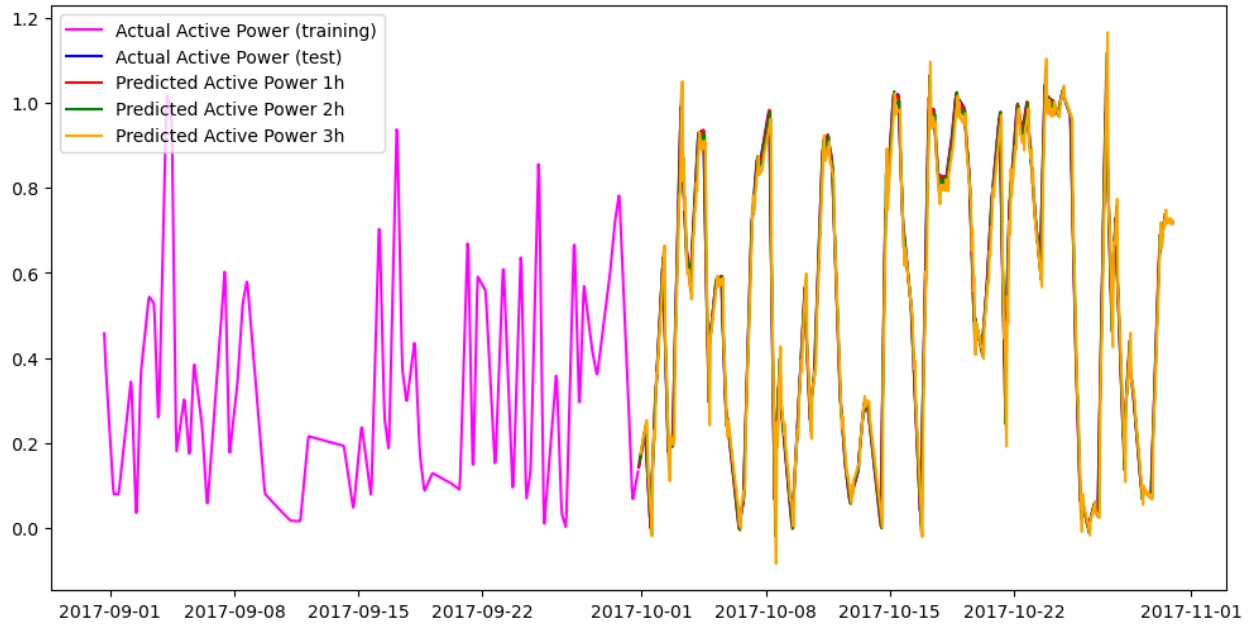


Figura 3.5: Predicciones del modelo AutoArima para el dataset KaggleWPGD.

El modelo de regresión dinámica ajustada es un modelo con errores ARIMA(2,0,2).

Los residuos del modelo aparecen en la Figura 3.6 que permite comprobar que en este caso los residuos no se comportan como ruido blanco gaussiano, un resultado común en algunas aplicaciones del algoritmo AutoARIMA.

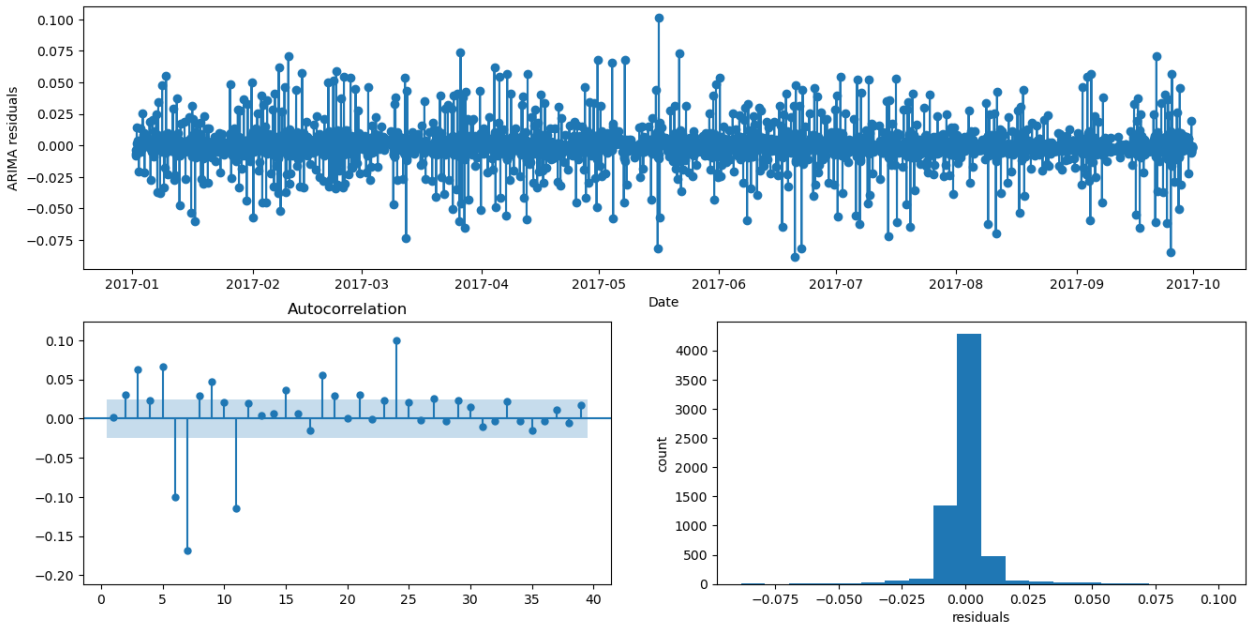


Figura 3.6: Residuos del modelo AutoArima para el dataset KaggleWPGD

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.00551
- MAE a 2h: 0.02076
- MAE a 3h: 0.03736

3.3 XGBoost

3.3.1 Introducción

Como hemos señalado en el Capítulo 2, en los últimos años y en el contexto del uso creciente de modelos de Machine Learning y Deep Learning, los modelos de boosting han formado parte de muchos de los modelos que ocupan los puestos más altos en las competiciones de predicción de series temporales. El análisis de este tipo de modelos sin embargo no forma parte del TFM de Carlos de los Santos [2] que nos sirve de precedente. Parece por tanto necesario completar en este trabajo la panorámica de métodos disponibles con el análisis de un modelo de boosting aplicado a estos conjuntos de datos.

3.3.2 Descripción de los modelos de boosting

Los métodos de boosting son sin duda una de las herramientas más potentes de las que se disponen en Machine Learning, porque alcanzan un gran equilibrio entre complejidad computacional aceptable, alto rendimiento predictivo y facilidad de uso. Además permiten de manera muy natural incorporar herramientas de selección de variables (*feature selection*), lo cual los hace útiles en problemas multivariables.

La primera familiar de métodos de boosting apareció con el método *AdaBoost* (de *adaptative boosting*), que fue creado por Freund y Schapire en 1995 ([35]). La idea general del boosting adaptativo es la siguiente: se entrena una sucesión de modelos, de manera que cada modelo trata de mejorar los resultados del anterior. Para ello, al acabar el entrenamiento de un modelo modificamos la forma en la que el siguiente modelo *percibe* el conjunto de entrenamiento. Eso se consigue haciendo que los puntos del conjunto de entrenamiento asociados con errores grandes de predicción reciban un peso mayor, como si de alguna forma fueran más grandes. Esto dirige al siguiente modelo a realizar un esfuerzo mayor en la predicción de esos valores.

Otra característica habitual de los modelos de boosting es que cada uno de los modelos de la sucesión es un modelo comparativamente sencillo o débil (*weak learner*). A menudo, y en particular en las implementaciones más conocidas, esos modelos son árboles binarios de decisión de profundidad uno (*stumps*). Si los stumps del modelo son $h_1, \dots, h_k$, la predicción final del modelo es una suma ponderada

$$\sum_{k=1}^{K} a_k h_k(x)$$

donde los pesos se calculan como parte del proceso de entrenamiento del modelo, para atribuir más peso a los stumps más fiables. Además, para combatir el posible sobreajuste (*overfitting*) del modelo a los datos de entrenamiento se puede introducir como hiperparámetro una tasa de aprendizaje (*learning rate*) λ que modera la influencia de los submodelos más tardíos en las predicciones globales. Esta estrategia de agregación de predicciones de submodelos sitúa a los modelos de boosting dentro de la familia de técnicas de *ensemble*. Pueden verse más detalles sobre estos modelos en el contexto de los métodos de Machine Learning en [36].

En 1999, en el trabajo de Friedman [37] se presentó una segunda familia de métodos de boosting, los denominados métodos de **gradient boosting**. Tienen en común con los anteriores que las predicciones del modelo se obtienen combinando en una suma ponderada las de una sucesión de modelos débiles y que cada modelo tiene como tarea corregir los mayores errores del que lo precede. Pero la forma en que se lleva a cabo este proceso de corrección de errores es distinta en los modelos de gradient boosting. Estos modelos se entrenan para minimizar la función de pérdida mediante un proceso parecido a la técnica de optimización del *descenso del gradiente*. Para ello se utiliza una expresión del gradiente de la función de pérdida en términos de los residuos del modelo, de manera que aquellos puntos con mayores errores tienen residuos mayores y por tanto influyen más en la corrección del gradiente que realiza el siguiente modelo. La Figura 3.7 (pág. 38), procedente de [Figura 5-8 de@Kunapuli2023], ilustra la diferencia entre ambos tipos de modelos de boosting (en este caso en el contexto de un problema de clasificación binaria). La recta representa la frontera entre las clases de un problema de clasificación binaria. A la derecha se ilustra como los métodos de boosting basados en gradiente periten que los puntos peor clasificados ejerzan una influencia mayor en el reposicionamiento de esa frontera para cada iteración del modelo.

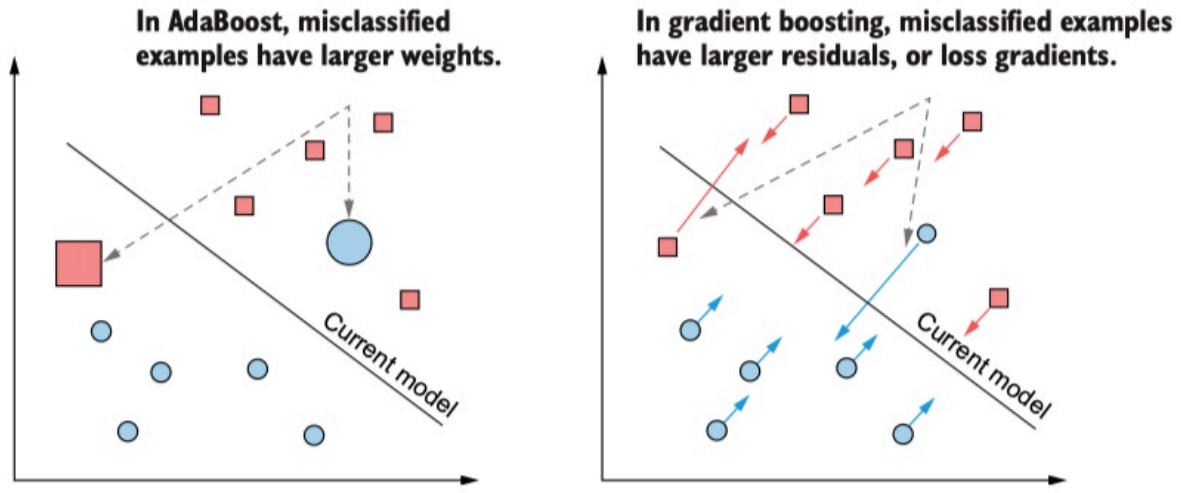


Figura 3.7: Boosting adaptativo (izda) y boosting basado en gradientes (dcha) en clasificación binaria.

De lo anterior se deduce que los hiperparámetros más importantes en la definición de un modelo de boosting (basado en stumps binarios) son:

- el número de stumps,
- las características de cada stump: profundidad máxima, si se permite mayor que uno, pero en general pequeña, umbral de de pureza y tipo de medida de pureza para hacer un split, etc.
- y la tasa de aprendizaje.

3.3.3 Implementación y resultados

En el contexto de las aplicaciones a la predicción de series temporales, los métodos de boosting se utilizan como cualquier otro método de regresión mediante las adaptaciones que hemos visto en la Sección 2.3.2.

Dentro de las implementaciones de modelos de Machine Learning, los modelos de boosting ocupan un lugar especial por la existencia de XGBoost (acrónimo de Extreme Gradient Boosting). Se trata de una implementación muy optimizada del algoritmo de gradient boosting, introducida por Tianqi Chen y Carlos Guestrin en 2016 y caracterizada por su eficiencia, gran escalabilidad y adaptabilidad a muchas tareas de Machine Learning [39]. Esta implementación está disponible de forma nativa como librería en lenguajes como Python, R, Julia, Java, etc. Y en el caso de Python se dispone también de una interfaz que permite utilizar esta implementación fácilmente desde librerías como scikit-learn o, en el caso que nos ocupa, MLForecast de Nixtla.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo que se han considerado en este trabajo son los sugeridos por la propia librería MLForecast:

- `n_estimators`: el número de stumps del modelo,
- `max_depth`: su profundidad máxima,
- `learning_rate`: tasa de aprendizaje,
- `subsample`: tasa de muestreo de los datos de entrenamiento,
- `colsample_bytree`: tasa de muestreo de variables al construir cada stump,
- `reg_lambda`, término L_2 de regularización
- `reg_alpha`, término L_2 de regularización
- `min_child_weight`: peso mínimo de los nodos descendientes para decidir si se hace un split del árbol o no.

Dado que en este caso hemos ajustado un modelo distinto para cada valor del horizonte, los notebooks que aparecen referenciados aquí debajo muestran, en cada caso la combinación de hiperparámetros seleccionada para cada conjunto de datos y cada horizonte de predicción.

3.3.3.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `XGBoost_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento XGBoost_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.8 (40). El tiempo de entrenamiento del modelo (con selección de hiperparámetros y predicción de resultados) es de aproximadamente 5 minutos.

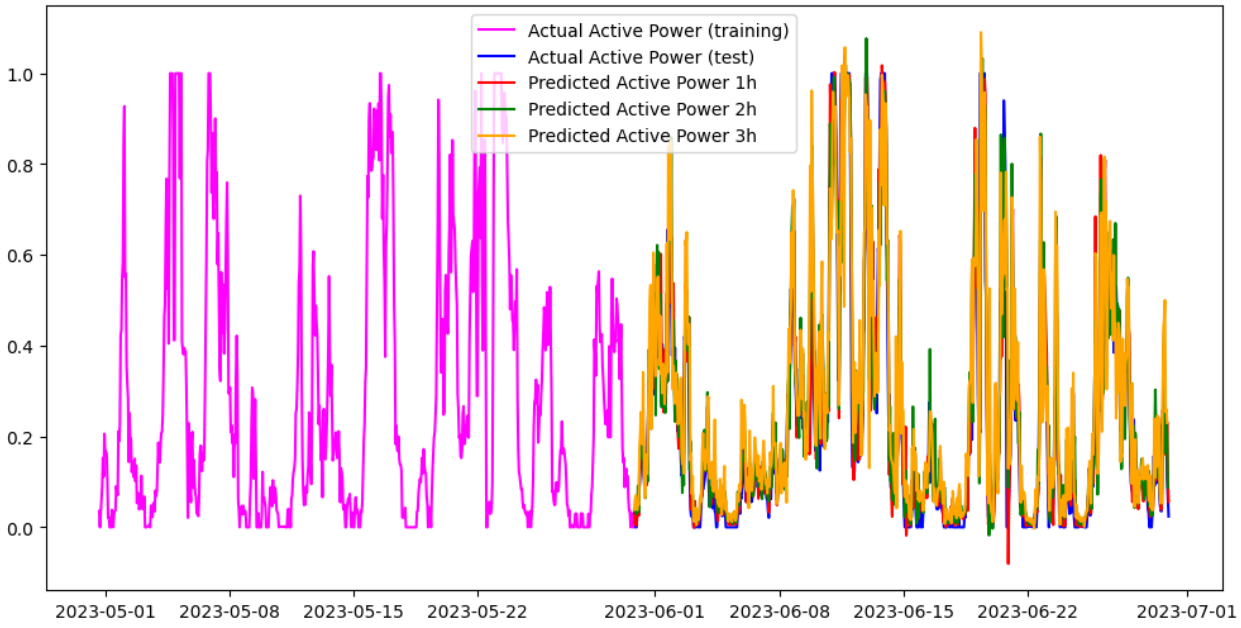


Figura 3.8: Predicciones del modelo XGBoost para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 320.90
- MAE a 2h: 384.08
- MAE a 3h: 442.75

3.3.3.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `XGBoost_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento XGBoost_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.9 (41). El tiempo de entrenamiento del modelo (con selección de hiperparámetros y predicción de resultados) es de aproximadamente 5 minutos.

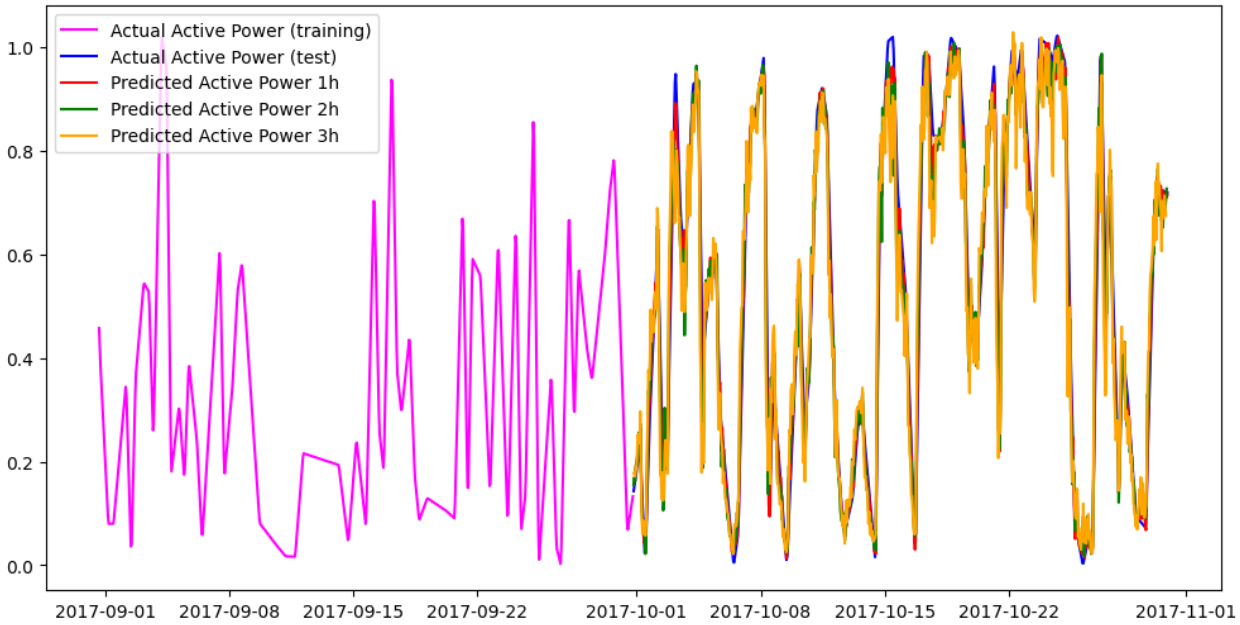


Figura 3.9: Predicciones del modelo XGBoost para el dataset KaggleWPGD.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.02796
- MAE a 2h: 0.04782
- MAE a 3h: 0.06637

3.3.4 Comentarios sobre impementación y resultados del modelo XGBoost

Desde el punto de vista computacional, el tiempo de entrenamiento ha sido razonable (aproximadamente 5 minutos por dataset y horizonte), teniendo en cuenta la búsqueda de hiperparámetros y la posterior predicción para tres horizontes distintos. Esta eficiencia confirma una de las principales ventajas de XGBoost frente a otras alternativas: su alta optimización para el entrenamiento en CPU y su buena escalabilidad incluso en contextos con múltiples modelos por horizonte.

Cabe destacar que, aunque XGBoost no está diseñado específicamente para series temporales, su integración con MLForecast permite adaptarlo de manera efectiva a este tipo de problemas, convirtiéndolo en una opción sólida, especialmente cuando se dispone de datos estructurados con múltiples series o cuando se desea evitar la complejidad de modelos más pesados basados en redes neuronales profundas.

En conjunto, XGBoost ha mostrado un buen compromiso entre precisión, velocidad y facilidad de ajuste, lo que lo convierte en una referencia útil y competitiva para tareas de predicción a corto plazo en series temporales con características tabulares.

3.4 Modelos de redes neuronales recurrentes: LSTM y GRU

3.4.1 Introducción

Las redes neuronales recurrentes (RNN) han sido ampliamente utilizadas en tareas de predicción de series temporales debido a su capacidad para modelar dependencias temporales y secuencias de datos. Entre las variantes más exitosas de estas arquitecturas se encuentran las redes LSTM (Long Short-Term Memory, [40]) y GRU (Gated Recurrent Units, [41]), que introducen mecanismos de puertas para mitigar el problema del desvanecimiento del gradiente y mejorar la retención de información a largo plazo.

En el TFM de Carlos, estas dos arquitecturas fueron empleadas como parte del análisis comparativo frente a modelos tradicionales (persistencia y ARIMA) y a una red MLP (Multi-Layer Perceptron). Los resultados obtenidos mostraron que tanto LSTM como GRU superaban en capacidad predictiva a los modelos base, destacando especialmente GRU por su equilibrio entre rendimiento y eficiencia computacional.

Sin embargo, en este trabajo partimos de la constatación de que, a pesar de su eficacia, los modelos LSTM y GRU presentan limitaciones estructurales y operativas que justifican la necesidad de explorar arquitecturas más modernas dentro del Deep Learning.

3.4.2 Arquitectura de las redes LSTM

La red LSTM fue diseñada específicamente para retener información relevante en horizontes temporales largos, introduciendo una celda de memoria controlada por tres puertas: de entrada, de olvido y de salida. La arquitectura de una celda LSTM se esquematiza en la Figura 3.10 (pág. 42).

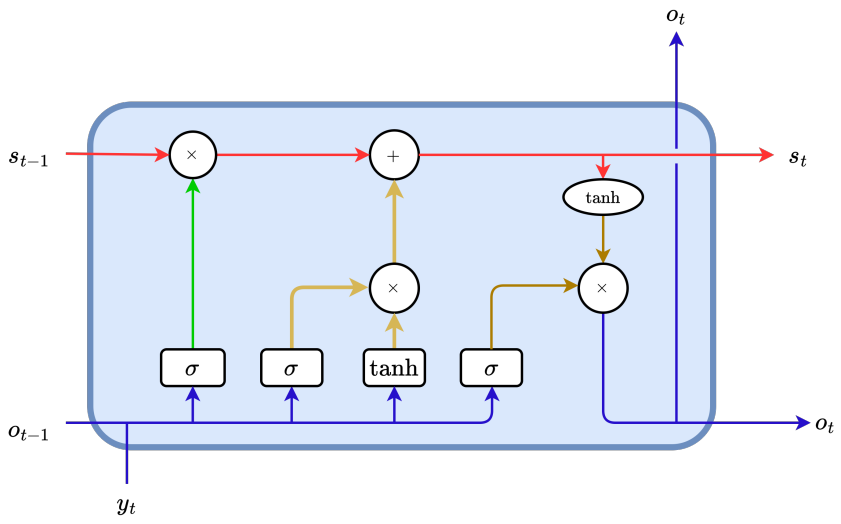


Figura 3.10: Arquitectura del modelo LSTM

Dado un vector de entrada $\mathbf{x}_t$ en el instante t, el estado de la celda se actualiza mediante las siguientes ecuaciones:

$$
\begin{aligned}
f_t &= \sigma(W_f[h_{t-1}, x_t] + b_f) && \text{(puerta de olvido)} \\
i_t &= \sigma(W_i[h_{t-1}, x_t] + b_i) && \text{(puerta de entrada)} \\
\tilde{C}_t &= \tanh(W_C[h_{t-1}, x_t] + b_C) && \text{(información candidata)} \\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t && \text{(estado de celda)} \\
o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) && \text{(puerta de salida)} \\
h_t &= o_t \odot \tanh(C_t) && \text{(salida)}
\end{aligned}
$$

Esta estructura permite a la red conservar y filtrar información a lo largo del tiempo de forma controlada, lo que resulta especialmente útil en tareas como la predicción de series temporales meteorológicas o energéticas.

3.4.3 Arquitectura de las redes GRU

Las GRU simplifican la estructura de las LSTM fusionando las puertas de entrada y olvido en una sola puerta de actualización, y eliminando la celda de memoria explícita. Esta simplificación reduce el número de parámetros, mejorando la eficiencia sin una pérdida significativa de rendimiento.

La Figura 3.11 (pág. 43) muestra el esquema de una celda GRU.

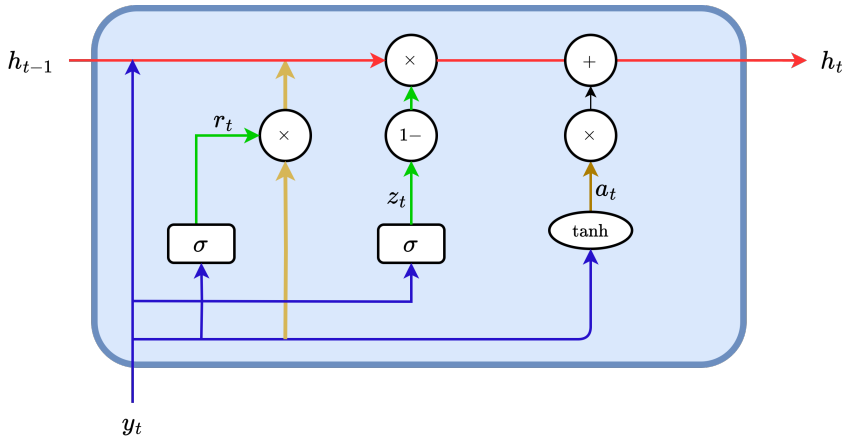


Figura 3.11: Arquitectura del modelo GRU

$$
\begin{aligned}
z_t &= \sigma(W_z[h_{t-1}, x_t] + b_z) && \text{(puerta de actualización)} \\
r_t &= \sigma(W_r[h_{t-1}, x_t] + b_r) && \text{(puerta de reinicio)} \\
\tilde{h}_t &= \tanh(W_h[r_t \odot h_{t-1}, x_t] + b_h) && \text{(estado candidato)} \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{(estado final)}
\end{aligned}
$$

3.4.4 Limitaciones de LSTM y GRU en el contexto de este TFM

A pesar de sus capacidades, tanto LSTM como GRU presentan ciertas limitaciones que se hacen evidentes en problemas complejos como la predicción de generación eólica marina:

- Sensibilidad al preprocesado: El rendimiento es altamente dependiente de la calidad del preprocesamiento de los datos (imputación, eliminación de atípicos, escalado, etc.).

- Escasa escalabilidad a horizontes de predicción largos: A medida que se amplía el horizonte temporal, la capacidad de estas redes para mantener coherencia y precisión decrece, especialmente en escenarios multivariable.

3.4.5 Implementación y resultados

La implementación de LSTM que hemos utilizado es la que está disponible en la librería neuralforecast de Nixtla [29], concretamente mediante las clases `LSTM` y `AutoLSTM`, utilizando para ello el procedimiento descrito anteriormente en la Sección 2.3. El resultado está disponible en la carpeta *models/LSTM* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo que se han considerado en este trabajo es:

- "encoder_n_layers": número de capas de la parte encoder del modelo,
- "encoder_hidden_size": número de neuronas de las redes FF del encoder,
- "encoder_dropout": tasa de descarte para el encoder,
- "decoder_layers": número de capas de la parte decoder del modelo,
- "decoder_hidden_size": número de neuronas de las redes FF del decoder,
- "learning_rate": tasa de aprendizaje.

3.4.5.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `LSTM_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento LSTM_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.12 (45). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 7 minutos. El tiempo del bucle de validación es de aproximadamente 95 minutos.

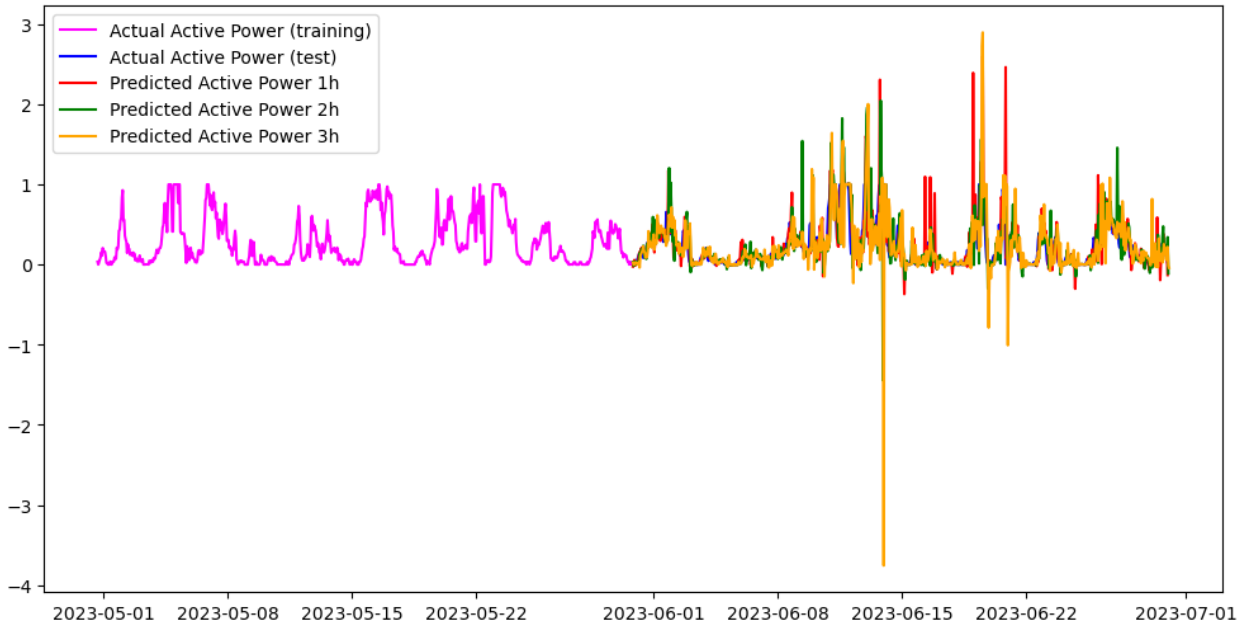


Figura 3.12: Predicciones del modelo LSTM para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 530.91
- MAE a 2h: 634.94
- MAE a 3h: 736.11

3.4.5.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `LSTM_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento LSTM_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 3.13 (46). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 6 minutos. El tiempo del bucle de validación es de aproximadamente 90 minutos.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.017006
- MAE a 2h: 0.023647
- MAE a 3h: 0.026569

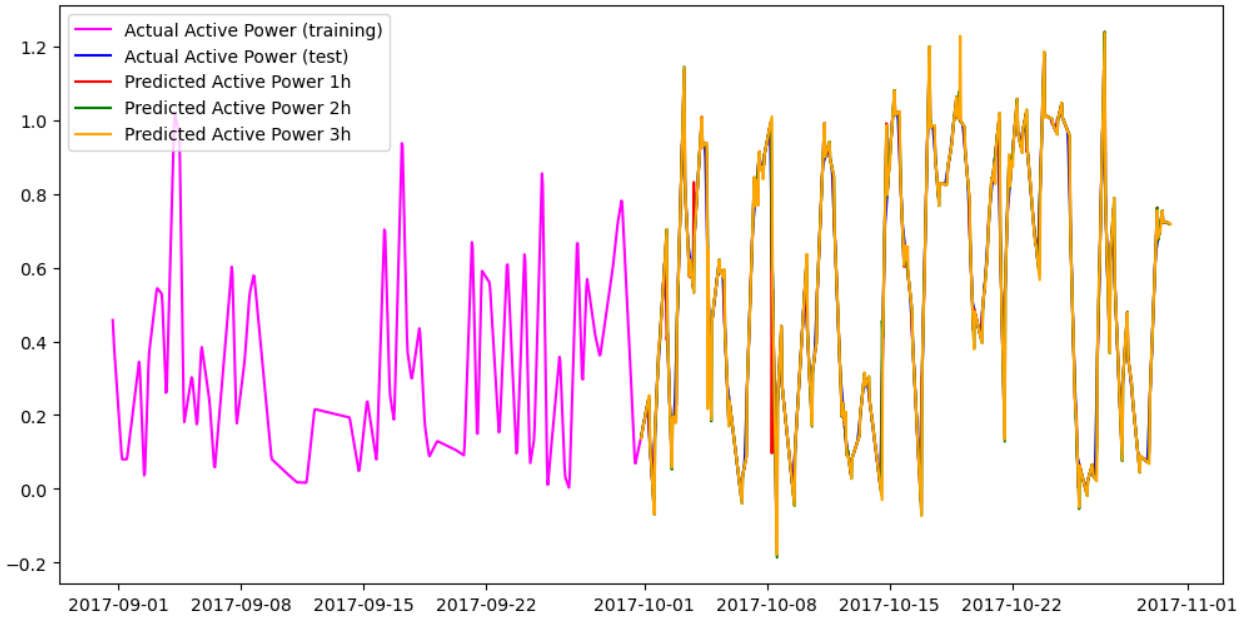


Figura 3.13: Predicciones del modelo LSTM para el dataset KaggleWPGD.

3.4.6 Comentarios sobre implementación y resultados del modelo LSTM

Desde el punto de vista de eficiencia, se observa que, el proceso completo de validación con selección de hiperparámetros puede llegar a durar alrededor de 6 minutos por modelo, lo que representa una carga computacional significativamente mayor que en otros modelos como XGBoost. Esta diferencia es esperable, dado que las redes neuronales requieren múltiples épocas y evaluación en lotes (mini-batches), especialmente en arquitecturas profundas y con mecanismos encoder-decoder como los que se han utilizado.

En conjunto, el modelo LSTM destaca por su capacidad para capturar relaciones temporales complejas y no lineales, lo que le otorga un gran potencial en contextos con datos abundantes y patrones temporales bien definidos. Sin embargo, este potencial viene acompañado de un coste computacional elevado y una mayor sensibilidad al ajuste de hiperparámetros, lo que puede limitar su aplicabilidad en entornos con recursos restringidos o donde se requiera una rápida iteración de modelos.

Capítulo 4

Modelos con arquitecturas Transformer

4.1 Introducción

La arquitectura Transformer fue descrita por primera vez en 2017 en el ya célebre artículo *Attention is all you need* [7]. El problema al que se quería hacer frente con ese modelo era el problema de traducción entre idiomas, un problema básico en *Procesamiento de Lenguaje Natural* (*NLP* por sus siglas en inglés). Su éxito en la solución de ese problema y de otros muchos que siguieron dio comienzo al desarrollo imparable hasta la fecha de los *modelos grandes de lenguaje* (*large language models, LLMs*). Una descripción más completa del periodo de aparición del transformer y sus consecuencias puede verse en los primeros capítulos del libro *Hands-On Large Language Models* [42].

La Figura 4.1 (pág. 48) del artículo original [7], reproducida con permiso de los autores, describe la versión inicial de esta arquitectura para su uso en NLP. Como puede verse en la figura, el transformer se puede ver como una evolución de los modelos de tipo encoder-decoder basados en redes neuronales recurrentes (ver por ejemplo [41] y [43]) que se aplicaban previamente al problema *seq2seq* de traducción automática. Pero a diferencia de estos, el diseño del transformer permite eliminar la recurrencia y facilita el procesamiento en paralelo mediante GPUs.

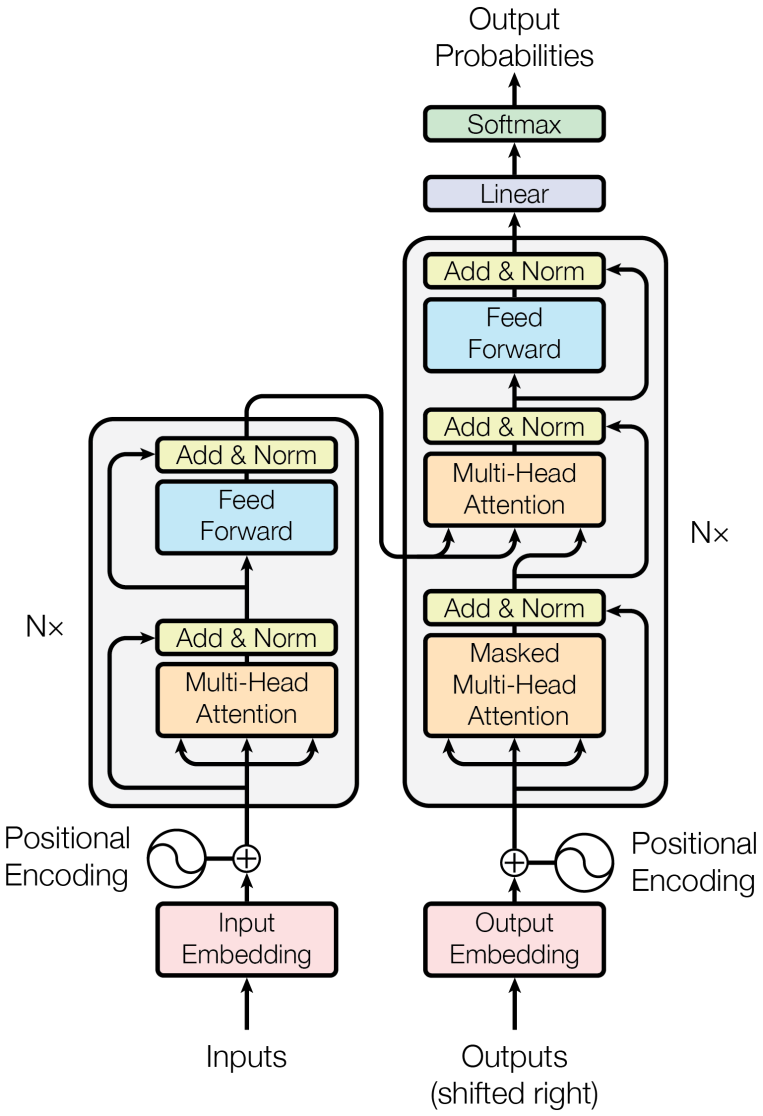


Figura 4.1: Arquitectura del modelo Transformer tal como aparece en el artículo original {*Attention is all you need*} *(ver referencia en el texto)*

Las dos principales contribuciones de las arquitecturas transformer frente a sus predecesoras en el terreno del NLP son:

- el uso de **embeddings**[1] para lograr representaciones vectoriales del lenguaje que capturen su semántica como parte de la propia representación. El embedding aparece en la parte inferior de la Figura 4.1 en las entradas de ambos bloques encoder y decoder. El embedding va acompañado en todos los problemas de secuencias (tanto en NLP como en series temporales) por una **codificación posicional** (*positional encoding*) que permite al modelo incorporar el orden de la secuencia de entrada como parte del propio embedding.
- el mecanismo de **atención**, que permite al decodificador acceder a cualquier parte de su secuencia de entrada y que por tanto permite al modelo usar un *contexto* mucho más amplio y

[1] en español se usa a veces *incrustación* como traducción; nosotros preferimos *encaje*, pero la terminología aún no se ha fijado, por lo que mantendremos el original inglés embedding.

rico para hacer su tarea. Así se incrementa la capacidad del modelo para aprender relaciones entre elementos de la secuencia separados por distancias mucho mayores de lo que era posible previamente. Además el mecanismo de atención es paralelizable y puede beneficiarse del uso de GPUs para su entrenamiento. Esto supone una ventaja computacional enorme frente al procesamiento secuencial que requieren los modelos recurrentes, como los LSTM y GRU.

Además en la arquitectura transformer se usan otras herramientas: algunas provenientes de trabajos previos en procesamiento de imágenes como las **conexiones residuales** (*residual or skip connections*, ver [44]) que permiten mejorar el comportamiento de los gradientes en el entrenamiento del modelo. Y otras, como la **normalización por capas** (*layer normalization*, ver [7]) que forma parte de los métodos de regularización. Ambas técnicas han pasado a ser una herramienta habitual en el diseño de arquitecturas de redes neuronales y volveremos sobre ellas más abajo al analizar los modelos de series temporales que las usan.

4.1.1 Breve introducción a la terminología usada en la arquitectura transformer

No procede aquí entrar en detalle en la descripción formal de la arquitectura transformer original tal como se aplica en los problemas de NLP. Baste decir que su uso se ha generalizado y que partes de ese diseño original se usan ahora por separado, de manera que hay modelos puros de tipo encoder para tareas de representación del lenguaje y modelos puros de tipo decoder que se usan para tareas de generación de lenguaje. Pero para describir más adelante las adaptaciones de esta arquitectura a los problemas de series temporales, es necesario introducir al menos la notación básica correspondiente.

El transformer original utiliza un mecanismo de autoatención multicabeza (*multi-head self-attention*) que, con H cabezas de atención, transforma la entrada $\mathbf{Y}$ en paralelo en

- $Q_h = \mathbf{Y}W_h^Q$: vectores de **consulta** (*queries*).
- $K_h = \mathbf{Y}W_h^K$: vectores de **clave** (*keys*).
- $V_h = \mathbf{Y}W_h^V$: vectores de **valor** (*values*).

con $h = 1, \dots, H$, siendo H el número de cabezas de atención. Por esta razón las capas de atención se denominan también a menudo capas QKV. A continuación el valor de la atención se calcula en cada cabeza mediante

$$\text{Attention}(Q_h, K_h, V_h) = \text{softmax}\left(\frac{Q_h K_h^\top}{\sqrt{d_k}} \cdot M\right) V_h$$

siendo d_k: es la dimensión de los vectores clave. Las matrices $\mathbf{Y}W_h^Q$, $\mathbf{Y}W_h^K$, $\mathbf{Y}W_h^V$ que generan respectivamente consultas, claves y valores son parámetros entrenables en esta arquitectura. El uso de múltiples cabezas en el transformer es análogo al uso de múltiples núcleos (kernels) en cada capa de una red neuronal convolucional, donde cada núcleo se encarga de aprender características o representaciones distintas de los datos de entrada

La matriz M es la identidad en el bloque encoder, mientras que en el bloque decoder es una **matriz de enmascaramiento** (mask matrix) que, durante el entrenamiento del modelo, permite a la vez el procesamiento secuencial y el entrenamiento en paralelo del modelo usando GPUs. Los valores de atención resultantes de cada cabeza se concatenan y proyectan para obtener un valor de la dimensión adecuada. Para una descripción detallada del transformer y el mecanismo de atención ver el artículo original y el Capítulo 20 del libro *Deep Learning: A Visual Approach* de A.S. Glassner [45].

4.2 Arquitectura transformer en modelos de series temporales

4.2.1 Precursores: el modelo LogSparse Transformer.

Desde 2017, tras el éxito de las arquitecturas transformer en procesamiento de lenguaje –que es en muchos casos un problema secuencial– era inevitable que los investigadores buscaran la forma de aplicar estas arquitecturas a los problemas asociados a las series temporales. En 2019 el artículo [46] planteó el uso de estas arquitecturas al problema de predicción en series temporales. Los autores destacan que, tratándose de una arquitectura concebida para abordar problemas de procesamiento de lenguaje, se hace necesario superar varios obstáculos para adaptar la arquitectura a los desafíos específicas de ese tipo de problemas:

- **Aumentar el comportamiento local del transformer:** en su diseño original para NLP, el transformer utiliza todos los elementos de una frase para calcular los valores de atención, aunque puedan aparecer muy distantes en esa frase. Ese comportamiento, que es una virtud en el caso del lenguaje, se convierte a menudo en un obstáculo en las series temporales en las que la atención al entorno local del instante de predicción debe reforzarse para tener en cuenta aspectos como la *forma* de la serie temporal. Para remediarlo, el artículo [46] planteó el uso de un mecanismo de **auto-atención convolucional** (*convolutional self-attention*), usando capas convolucionales con un núcleo de tamaño $k > 1$ ($k = 1$ es el mecanismo de auto-atención original).

  La Figura 4.2 (pág. 51, basada en la Figura 1 de [46]) ilustra en la parte superior como el mecanismo de atención original enfoca a puntos de la serie temporal con un nivel parecido, lo cual no es a menudo útil para la predicción. En cambio, el mecanismo de atención convolucional de la parte inferior enfoca regiones de forma similar a la que rodea al instante de predicción. La atención convolucional es de naturaleza causal a través de un mecanismo de enmascaramiento similar al que hemos mencionado más arriba.

- **Disminuir el uso de memoria del transformer:** la arquitectura original del transformer implica un uso elevado de recursos de memoria, que escalan cuadráticamente con el tamaño N de la entrada. En el contexto del problema de predicción de series temporales esto acarrea dificultades para el entrenamiento de series multivariable con muestras y horizontes de predicción grandes; precisamente el escenario en el que estos modelos buscaban ser competitivos. Para evitarlo, los autores de [46] propusieron un mecanismo de **atención dispersa** (*sparse attention*), en su caso de tipo logarítmico. En este mecanismo cada elemento de la serie en una capa de la red sólo presta atención a un subconjunto de las predecesoras en capas precedentes, pero seleccionadas mediante estrategias que permiten – a través de capas sucesivas de la red– que cada elemento reciba la información de todos los que le preceden.

4.2.1.1 Nota sobre implementación.

Los autores del artículo [46] no han proporcionado una implementación oficial de su modelo públicamente accesible aunque existen repositorios ([47]) con implementaciones no oficiales.

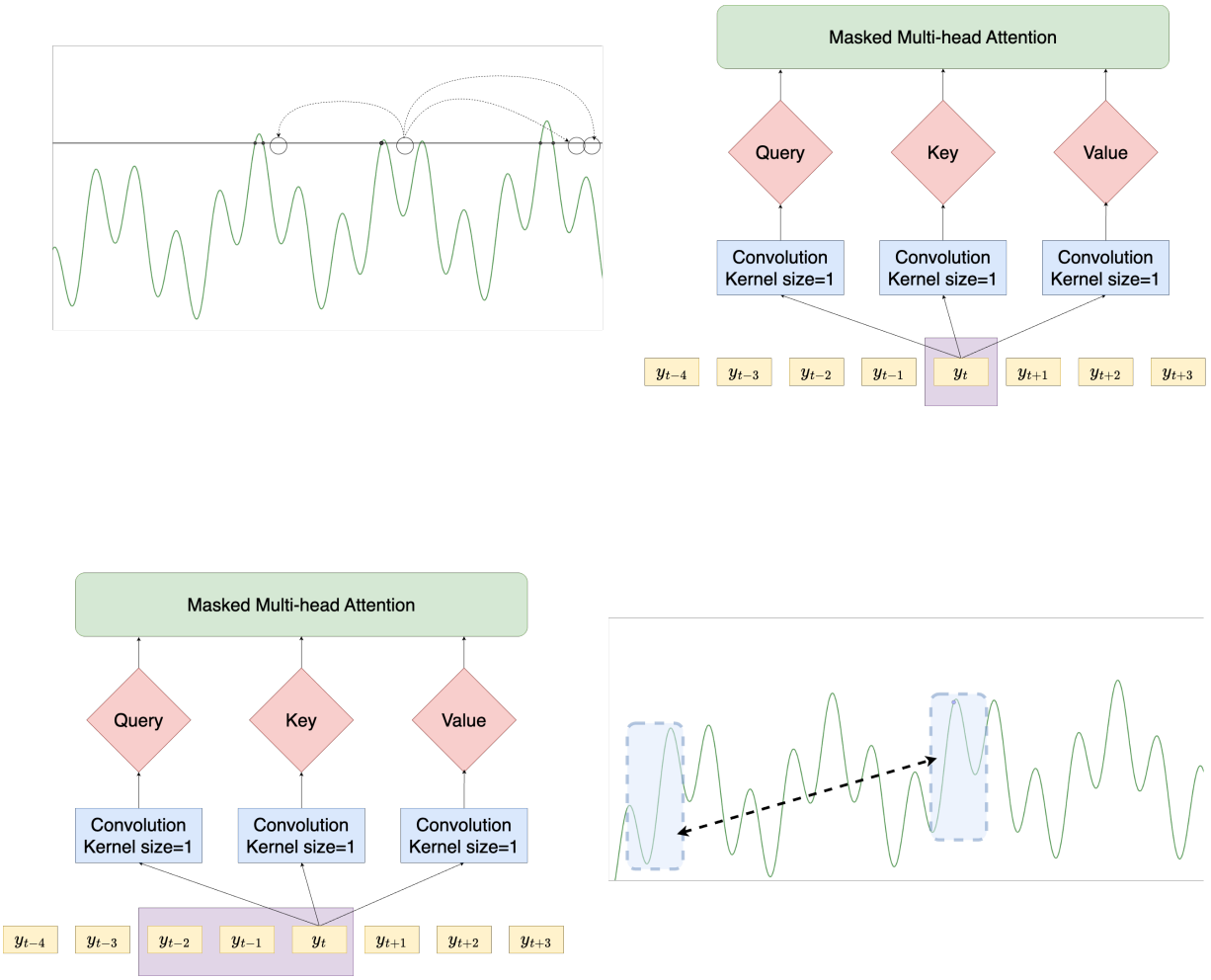


Figura 4.2: Auto-atención convolucional

4.2.2 Modelos de tipo transformer en otros problemas de series temporales

Como ya hemos dicho, el objetivo de este TFM es analizar el comportamiento de distintos modelos específicamente en tareas de predicción de series temporales. Pero antes de seguir adelante profundizando en ese tema, debemos señalar que este tipo de arquitecturas se han usado en un conjunto más amplio de problemas, entre otros:

- **Detección de anomalías:** ver por ejemplo [48], donde se describe el estado del arte para esos problemas en 2021 y en particular el modelo GTA [Chen_2022].
- **Clasificación de series temporales:** ver la Sección 3.4 de [49], en la que se destaca especialmente el modelo Convtran [50].

4.2.3 Evolución reciente de los modelos transformer en predicción de series temporales

Los últimos años han presenciado una auténtica explosión en el número de modelos basados en la arquitectura transformer para la predicción de series temporales, muchos de ellos de aparición posterior al comienzo del desarrollo de este TFM. El artículo de revisión *Deep learning for time*

series forecasting: a survey [51] incluye una amplia lista de referencias a modelos aparecidos durante este periodo (ver en particular la Tabla 1).

En ese desarrollo podemos distinguir un periodo inicial en el que surgieron modelos como Informer [52], Pyraformer [53] o FEDformer [54] entre otros. En torno al 2022 varios autores cuestionaron la utilidad real de las arquitecturas de tipo transformer para la predicción de series temporales (ver [55]), argumentando que incluso modelos de redes neuronales monocapa lineales sencillos podían superar a esas primeras arquitecturas de tipo transformer. Los autores argumentaban que el mecanismo de atención propio del NLP se basa la estructura semántica propia del lenguaje, pero que los primeros transformer no incorporaban un sustituto adecuado de esa información semántica. De hecho según esos mismos autores parte de la capacidad predictiva en horizontes largos de esos modelos parecía poder atribuirse más a su uso de una estrategia de predicción multipaso, más que a la propia arquitectura transformer.

Aunque la metodología de el trabajo [52] ha sido a su vez cuestionada (ver este post [56] de los autores de Informer en HuggingFace), su publicación fue seguida por una nueva ola de modelos de tipo transformer que buscaban confirmar de manera inequívoca la validez de este enfoque. Por ejemplo, el modelo Autoformer [57], de los mismos autores que Informer forma parte de esta hornada de modelos que tratan de combinar aspectos esenciales del transformer, como el embedding, la codificación posicional o el mecanismo de atención, pero combinándolos o modificándolos para incorporar el conocimiento previo acumulado en el estudio de la predicción de series temporales. En Autoformer los autores optan por modificar el propio mecanismo de atención, sustituyéndolo por un mecanismo de *auto-correlación*, acompañado de una *arquitectura de descomposición* que busca aprovechar las ideas de la descomposición clásica en componentes de estacionalidad y tendencia para potenciar el rendimiento predictivo del modelo. Varios modelos han seguido esta vía, modificando la propia arquitectura del transformer, pero la mayoría sufren de un alto coste computacional que los hace difíciles de utilizar en conjuntos grandes con horizontes largos. En la siguiente sección veremos un representante de otro enfoque, que mantiene los ingredientes básicos del transformer, pero los combina con métodos de pre y post procesado de las series temporales que buscan objetivos similares a un menor coste computacional.

4.3 El modelo Patch-TST

4.3.1 Introducción

Este modelo fue descrito a finales de 2022 en el artículo [8], con el objetivo de superar las objeciones planteadas a esta arquitectura. Para ello, los autores del modelo PatchTST propusieron dos ingredientes básicos:

- La *segmentación* (o *parcheado*, en inglés *patching*) de la serie temporal. Este método consiste en dividir la serie temporal en segmentos (*patches*) o intervalos temporales que permitan capturar el comportamiento local de la serie en un entorno de un instante temporal, dotando al modelo de esa *semántica* necesaria. Los segmentos en los que se divide la serie pueden ser disjuntos o tener solapamiento.
  El método recuerda en su finalidad a la auto-atención convolucional que hemos descrito al hablar del modelo LogSparse Transformer. Pero se trata de respuestas distintas a ese problema: mientras que la auto-atención convolucional es una modificación del propio mecanismo de atención, la segmentación es una técnica de preprocesado, pero la arquitectura básica del transformer se mantiene. Además, con esta técnica se reduce el coste computacional del en-

trenamiento del modelo.

- *Independencia del canal.* esta técnica es específica del problema de series temporales multidimensionales y se inspira en métodos similares que se emplean en redes convolucionales. Consiste en considerar cada serie del conjunto multidimensional como un canal independiente, de manera que cada token de entrada contenga información de un único canal. Se espera que esta independencia contribuya también a mejorar la facilidad de entrenamiento del modelo.

Una ventaja adicional del modelo PatchTST es que puede utilizarse como modelo de tipo auto-supervisado (*self-supervised*), para extraer patrones y estructuras temporales útiles para la predicción utilizando conjuntos muy grandes de series temporales para el pre-entrenamiento del modelo que luego se ajusta específicamente al problema de predicción de interés.

4.3.1.1 Arquitectura del modelo

La arquitectura del modelo PatchTST aparece en la Figura 4.3 de la página 54. En esa figura los pequeños rectángulos azules paralelos que parecen en la parte inferior izquierda representan la segmentación de la serie temporal. Todos los segmentos se utilizan en un mismo bloque, que tiene la estructura del encoder de la arquitectura transformer original. El funcionamiento del modelo es el siguiente:

- Si la serie temporal es multidimensional, cada componente de la serie se trata como un canal independiente, una serie temporal univariante. Todos esos canales comparten un único bloque transformer-encoder, pero el resto del procesado es independiente. La predicción global se obtiene concatenando las predicciones individuales de cada componente univariante.
- Cada serie temporal univariante se normaliza (*instance normalization*) y se descompone en N segmentos de longitud P, con lo que se reduce el número de tokens que el modelo debe procesar y se le dota de semántica temporal (los dos bloques de color naranja en la parte inferior izquierda de la Figura 4.3). Sea $x_p^{(i)} \in \mathbb{R}^{P\times N}$ uno de esos segmentos. A continuación el segmento se proyecta linealmente en el espacio de dimensión latente del transformer y se suma una codificación posicional. Ambas operaciones, proyección y codificación posicional se realizan mediante matrices entrenables:

$$x_d^{(i)} = \mathbf{W}_p\, x_p^{(i)} + \mathbf{W}_{pos}\, x_p^{(i)}$$

- A continuación (ver el bloque gris del centro de la Figura 4.3 y su estructura detallada a la derecha) el encoder multi-cabeza con cabezas de 1 a H transforma $x_d^{(i)}$ en H tuplas de matrices query-key-value

$$(Q_h^{(i)}, K_h^{(i)}, V_h^{(i)})$$

  a las que se aplica el mecanismo de atención básico:

$$\text{Attention}(Q_h^{(i)}, K_h^{(i)}, V_h^{(i)}) = \text{softmax}\left(\frac{Q_h^{(i)}(K_h^{(i)})^\top}{\sqrt{d_k}}\right) V_h^{(i)}$$

  El encoder también incluye una campa densa lineal y las conexiones residuales y capas de normalización habituales. Esta estructura del encoder puede repetirse en varias capas.

- Tras el bloque transformer-encoder una capa de aplanamiento lineal (en verde en la parte superior izquierda de la Figura 4.3) prepara las señales para usarlas como salida de predicción del modelo para esa serie univariante.

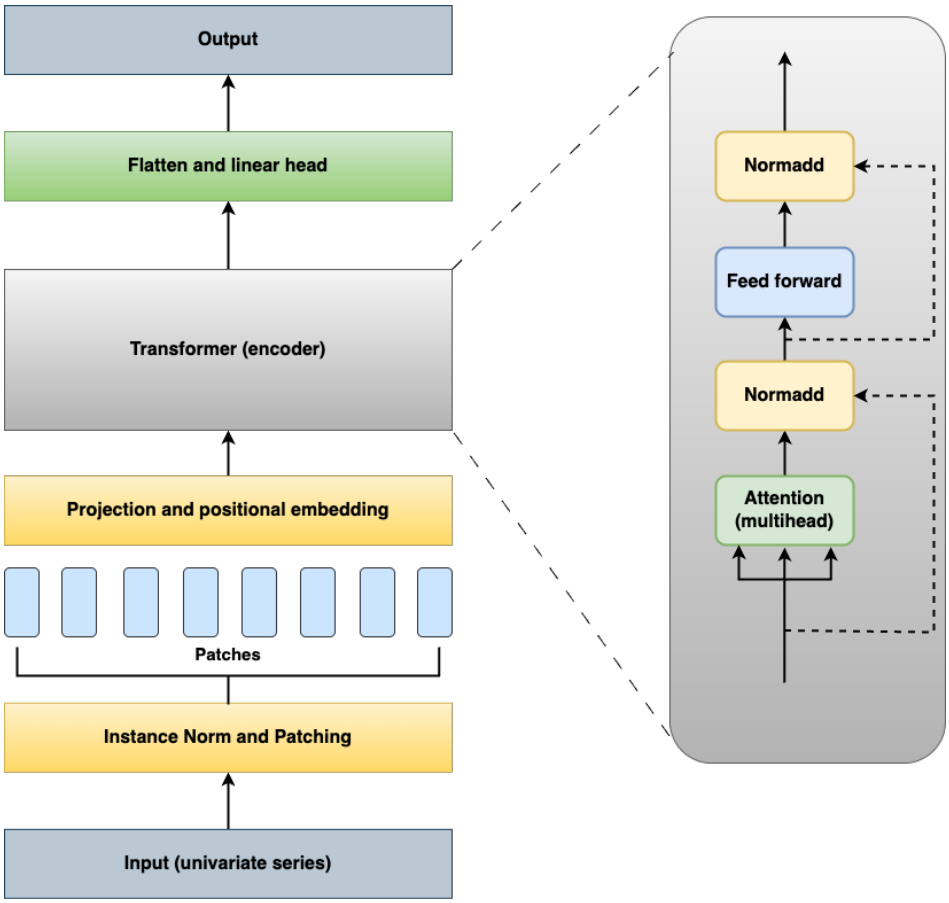


Figura 4.3: Arquitectura del modelo PatchTST

- Los autores del modelo proponen el uso de MSE promediado por canales como función de pérdida para el entrenamiento del modelo.

4.3.2 Implementación y resultados

La implementación de PatchTST que hemos utilizado es la que está disponible en la librería neuralforecast de Nixtla [29], concretamente mediante las clases `PatchTST` y `AutoPatchTST`, utilizando para ello el procedimiento descrito anteriormente en la Sección 2.3. El resultado está disponible en la carpeta *models/PatchTST* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo que se han considerado en este trabajo es:

- "encoder_layers": número de capas de la parte encoder del modelo,
- "n_heads": número de cabezas de atención del bloque de atención,
- "hidden_size": número de neuronas de las redes FF del modelo,
- "dropout": tasa de descarte para las conexiones residuales,
- "head_dropout": tasa de descarte para la capa lineal,
- "attn_dropout": tasa de descarte para la capa de atención,
- "patch_len": longitud del segmento (tach),
- "learning_rate": tasa de aprendizaje.

4.3.2.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `PatchTST_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento PatchTST_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 4.4 (55). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 12 minutos. El tiempo del bucle de validación es de aproximadamente 80 minutos.

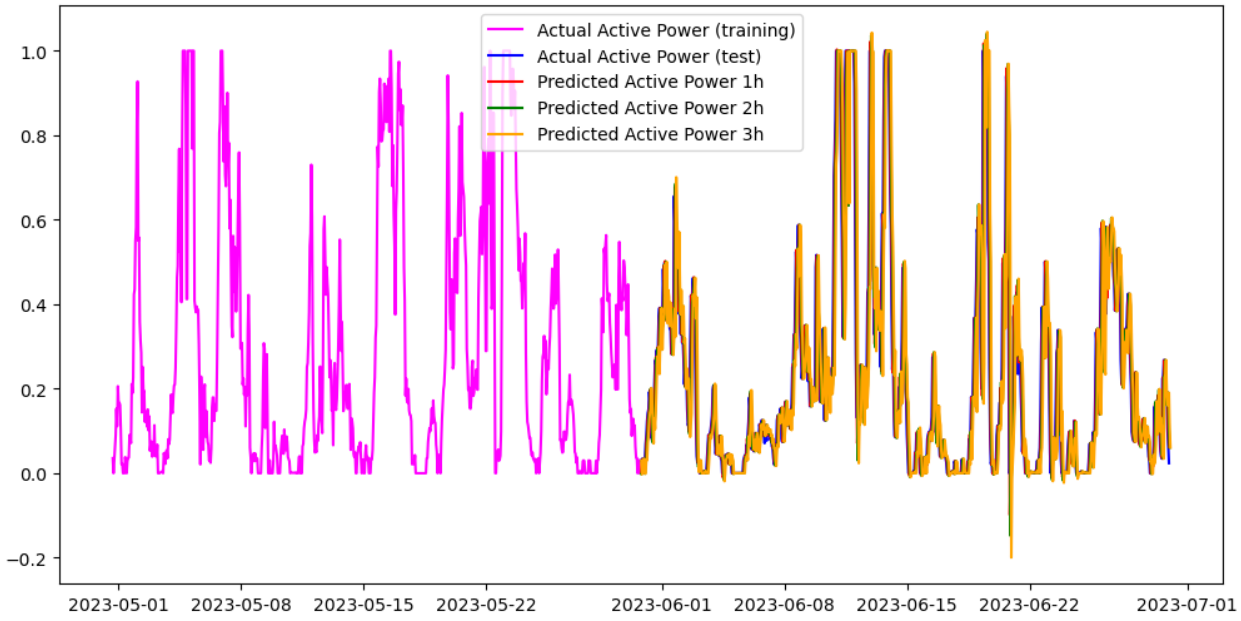


Figura 4.4: Predicciones del modelo PatchTST para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 236.98
- MAE a 2h: 375.87
- MAE a 3h: 487.08

4.3.2.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `PatchTST_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento PatchTST_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 4.5 (56). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 12 minutos.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.003483
- MAE a 2h: 0.011656

- MAE a 3h: 0.024211

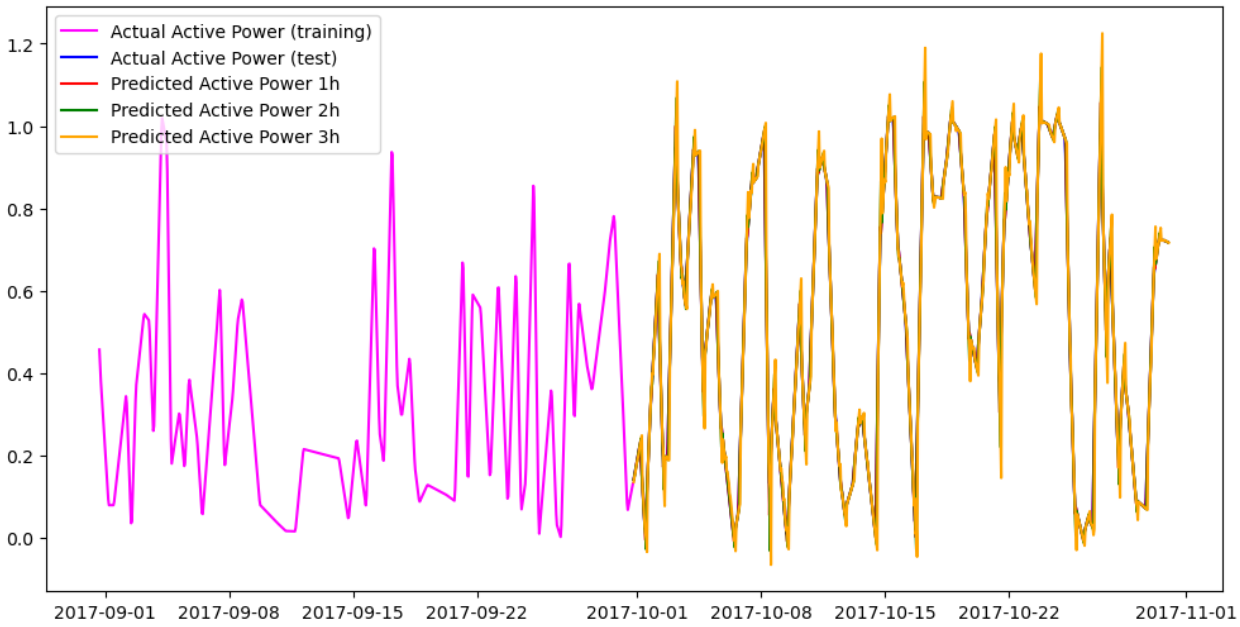


Figura 4.5: Predicciones del modelo PatchTST para el dataset KaggleWPGD.

4.3.2.3 Comentarios sobre implementación y resultados del modelo PatchTST

- Cualitativamente puede verse que el modelo produce predicciones aceptablemente buenas, pero debido a su arquitectura en el caso de dataset AV arroja resultados fuera del intervalo [0, 1] al que pertenecen los valores de la serie temporal original (normalizada). Se podría corregir este comportamiento del modelo de forma sencilla en postprocesado, pero lo mejor sería modificar la arquitectura del modelo para que la capa final de predicción sea una capa sigmoide que produzca de forma natural predicciones en el intervalo deseado.

- Más importante: el modelo PatchTST **no incorpora en su diseño el uso de variables exógenas** como predictores. Por esa razón a continuación veremos un modelo de tipo transformer que sí las utiliza de forma nativa.

4.4 El modelo TimeXer

4.4.1 La arquitectura de TimeXer

Como segundo representante reciente de los modelos de predicción basados en arquitecturas de tipo transformer veremos el modelo TimeXer. Este modelo fue presentado en febrero de 2024 [58]. El principal objetivo de este modelo es incorporar el uso de **variables exógenas** en la predicción de series temporales, a diferencia de muchos de sus predecesores de tipo transformer. Además TimeXer puede usarse en problemas con series temporales multivariable.

La Figura 4.6 (pág. 57) muestra los aspectos fundamentales de la arquitectura del modelo TimeXer. Su diseño busca combinar varios aspectos esenciales de algunos de sus predecesores:

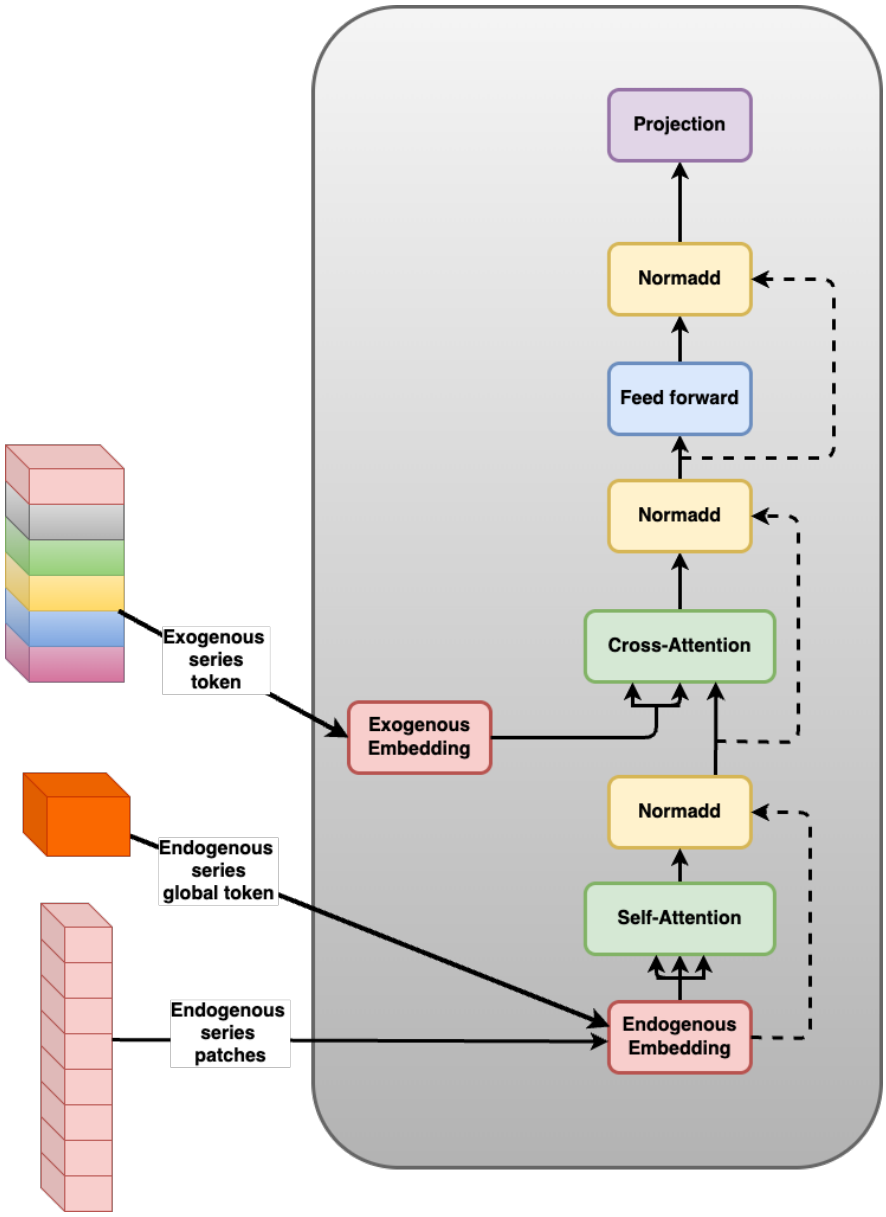


Figura 4.6: Arquitectura del modelo TimeXer

- De PatchTST y modelos similares, se toma la descomposición en segmentos de la serie objetivo, para preservar la información semántica que contiene (por ejemplo, su estructura de autocorrelación).
- De otros modelos de tipo transformer como iTransformer se toma la idea de considerar una serie temporal completa como un token de entrada del mecanismo de atención del modelo, para capturar las interrelaciones y dependencias entre distintas series temporales univariadas.

Ambos enfoques se conjugan en TimeXer mediante una jerarquía de representaciones a distintos niveles, usando tanto el mecanismo de atención básico como un mecanismo de **atención cruzada** (*cross attention*). Los autores de TimeXer buscan asimismo que esa jerarquía de representación permita a TimeXer abordar de forma natural problemas como la alineación temporal o la presencia de valores ausentes al usar variables exógenas.

- En el caso de las variables endógenas, TimeXer utiliza un embedding que produce por una lado una segmentación de la serie temporal similar a la de PatchTST (cubos apilados de color rosa a la izquierda de la Figura 4.6), pero también un **token global** entrenable (el cubo de color naranja a la izquierda de la Figura 4.6) que interviene en el mecanismo de auto-atención de esa variable endógena pero que también, como subrayan los autores, tiende un puente hacia las variables exógenas a través del mecanismo de atención cruzada. En ecuaciones:

$$\begin{cases} \text{Atención de segmento a segmento:} & P_{\text{en}}^{bl,1} = \text{LayerNorm}\left(P_{\text{en}}^{l} + \text{Self-Attention}(P_{\text{en}}^{l})\right), \\ \text{Atención de global a segmento:} & P_{\text{en}}^{bl,2} = \text{LayerNorm}\left(P_{\text{en}}^{l} + \text{Cross-Attention}(P_{\text{en}}^{l}, G_{\text{en}}^{l})\right), \\ \text{Atención de segmento a global:} & G_{l,\text{en}}^{b} = \text{LayerNorm}\left(G_{\text{en}}^{l} + \text{Cross-Attention}(G_{\text{en}}^{l}, P_{\text{en}}^{l})\right) \end{cases}$$

  donde $P_{\text{en}}^{bl,1}$ representa uno de los segmentos de la serie endógena y $G_{\text{en}}^{bl,1}$ es el token global de esa serie.

- Para las variables exógenas se utiliza también un embedding mediante un token global, lo cual dota al modelo de la flexibilidad necesaria para tratar situaciones diversas (por ejemplo, como se dijo antes, valores ausentes).

$$G_{l,\text{en}}^{b} = \text{LayerNorm}\left(G_{l,\text{en}}^{b} + \text{Cross-Attention}(G_{l,\text{en}}^{b}, \mathbf{V}_{\text{ex}})\right)$$

donde $\mathbf{V}_{\text{ex}}$ es el conjunto de representaciones de las variables exógenas (representadas por los cubos de colores apilados en la parte izquierda de la Figura 4.6).

Puede pensarse en esta jerarquía de representaciones como un análogo de la información que en los modelos estadísticos tradicionales se obtenía mediante el estudio de la autocorrelación de las variables endógenas y la correlación cruzada entre variables endógenas y exógenas. Ambas operaciones, las correlaciones clásicas y la atención, se basan en el producto escalar como operación matemática subyacente.

Por lo demás, como se ve en la parte derecha de la Figura 4.6, el modelo TimeXer respeta en lo esencial la arquitectura del transformer encoder original y no incorpora mecanismos de modificación de esa arquitectura como los que hemos comentado en el caso de LogSparse o estrategias similares.

4.4.2 Implementación y resultados

Para el modelo TimeXer hemos usado nuevamente una implementación disponible en la librería neuralforecast de Nixtla [29], mediante las clases `TimeXer` y `AutoTimeXer`, utilizando para ello el procedimiento descrito anteriormente en la Sección 2.3. El resultado está disponible en la carpeta *models/TimeXer* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo TimeXer que se han considerado en este trabajo es:

- `encoder_layers`: número de capas de la parte encoder del modelo,
- `n_heads`: número de cabezas de atención del bloque de atención,
- `hidden_size`: número de neuronas de las redes FF del modelo,
- `dropout`: tasa de descarte para las conexiones residuales,
- `head_dropout`: tasa de descarte para la capa lineal,
- `attn_dropout`: tasa de descarte para la capa de atención,
- `patch_len`: longitud del segmento (tach),
- `learning_rate`: tasa de aprendizaje.

4.4.2.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `TimeXer_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento TimeXer_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 4.7 (59). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 12 minutos. El tiempo del bucle de validación es de aproximadamente 80 minutos.

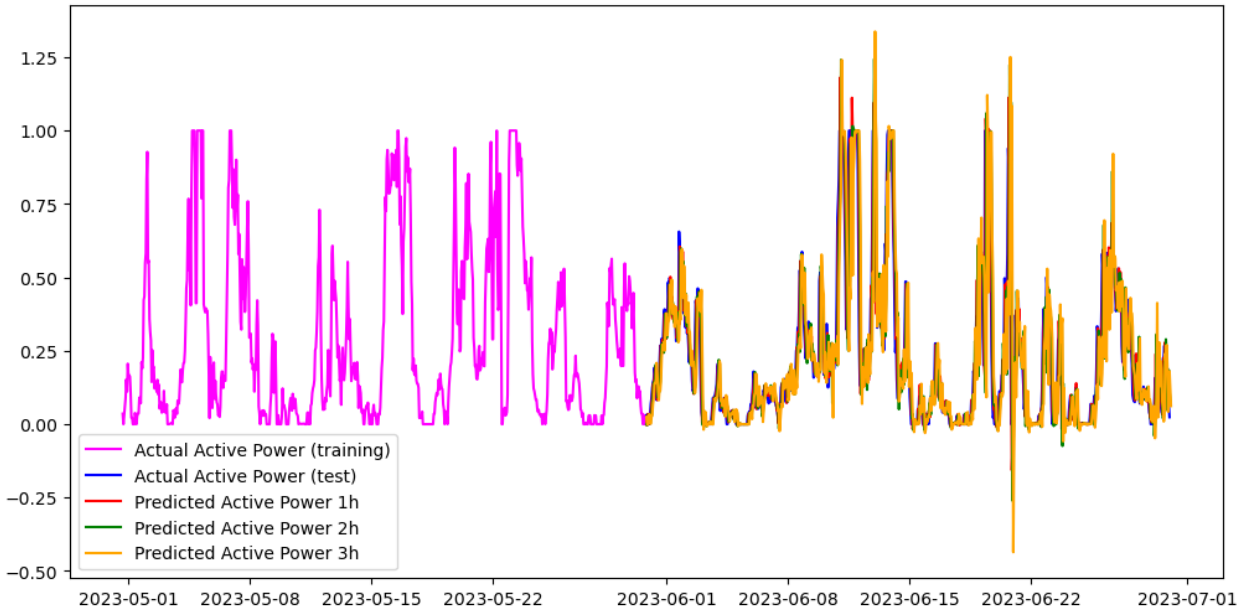


Figura 4.7: Predicciones del modelo TimeXer para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 263.77
- MAE a 2h: 411,96
- MAE a 3h: 518.84

4.4.2.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos de KaggleWPGD se encuentra en el notebook de Jupyter llamado `TimeXer_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento TimeXer_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 4.8 (60). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 30 minutos. El tiempo del bucle de validación es extremadamente largo en el caso de este conjunto de datos, superando ampliamente las 7 horas.

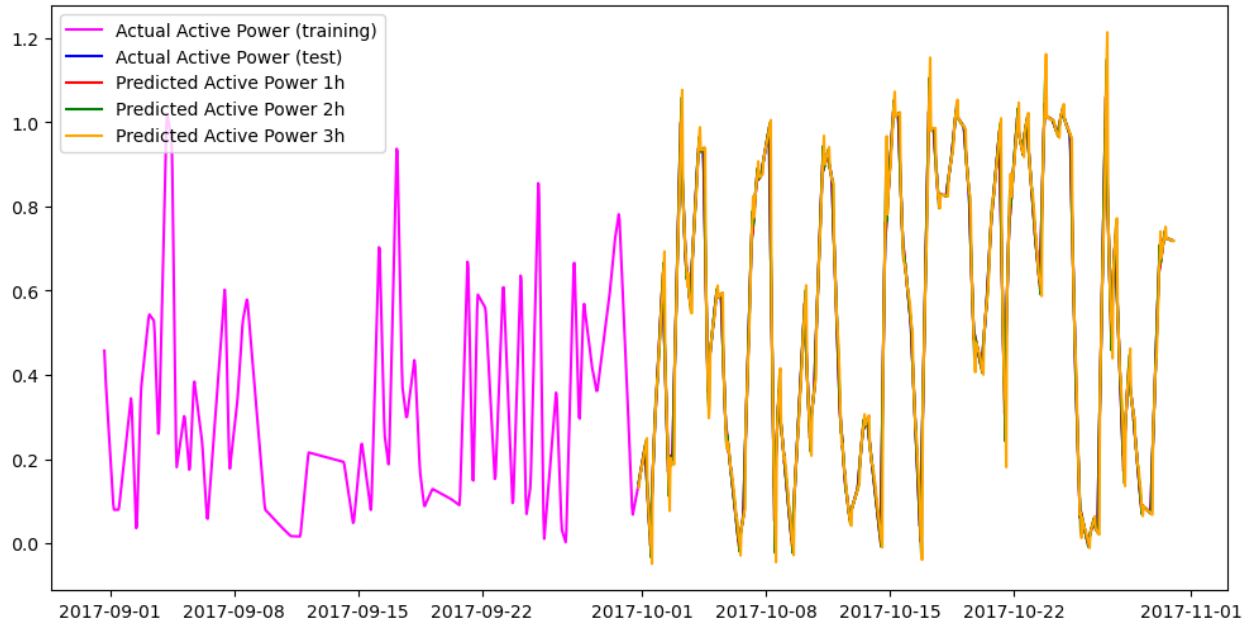


Figura 4.8: Predicciones del modelo TimeXer para el dataset KaggleWPGD

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.00335
- MAE a 2h: 0.01127
- MAE a 3h: 0.02410

4.4.2.3 Comentarios sobre implementación y resultados del modelo TimeXer

- Cualitativamente puede verse que el modelo produce predicciones aceptablemente buenas, pero debido a su arquitectura en el caso de dataset AV arroja resultados fuera del intervalo [0, 1] al que pertenecen los valores de la serie temporal original (normalizada). Se podría corregir este comportamiento del modelo de forma sencilla en postprocesado, pero lo mejor sería modificar la arquitectura del modelo para que la capa final de predicción sea una capa sigmoide que produzca de forma natural predicciones en el intervalo deseado.
- Desde el punto de vista computacional, se observa un incremento notable del coste de validación en comparación con modelos como LSTM o XGBoost. En particular, el conjunto KaggleWPGD ha supuesto un cuello de botella en cuanto a eficiencia, con tiempos de validación superiores a 7 horas. Esta elevada carga puede deberse al uso intensivo de mecanismos de atención en ventanas largas

Capítulo 5

Modelos con arquitecturas MLP y Modelos KAN

5.1 Introducción

Los avances recientes en el modelado de series temporales han puesto de manifiesto el potencial de las arquitecturas de redes neuronales profundas para abordar tareas de predicción en dominios complejos como el energético. Entre estas arquitecturas, los modelos basados en perceptrones multicapa (MLP) han demostrado una notable capacidad para capturar patrones temporales sin necesidad de recurrir a mecanismos recurrentes o convolucionales tradicionales.

En esta sección analizaremos dos líneas de modelos que comparten como elemento central una arquitectura de tipo MLP, pero que abordan la modelización temporal desde enfoques distintos: por un lado, la familia de modelos N-BEATS [59] / N-HiTS [60], centrada en la descomposición jerárquica de la señal a través de bloques funcionales especializados; y por otro, los Kernel Attention Networks (KANs), una propuesta más reciente que introduce funciones base no paramétricas entrenables en lugar de pesos tradicionales, permitiendo una mayor flexibilidad y capacidad de adaptación a patrones complejos.

Ambas familias parten de la premisa de que es posible lograr un rendimiento competitivo en tareas de predicción multihorizonte utilizando únicamente arquitecturas feedforward bien diseñadas, desafiando así la noción extendida de que la recurrencia, la convolución o la atención son componentes imprescindibles en modelos de forecasting. A lo largo de este capítulo se presentarán las motivaciones, estructuras y resultados obtenidos con cada uno de estos modelos en el contexto del problema de predicción de generación eléctrica en parques eólicos.

5.2 El modelo N-HITS

Ambos modelos comparten la misma arquitectura base, de tipo MLP. El primero en aparecer en 2019 fue N-BEATS, seguido en 2022 por N-HiTS [60]. Ambos modelos tienen principios de diseño similares y comparten al menos parte del grupo de autores. Empezaremos por tanto describiendo N-BEATS y veremos en qué sentido N-HiTS extiende y mejora esas ideas.

El modelo N-BEATS apareció en 2019 [59] con el objetivo de demostrar que una arquitectura de tipo MLP sin componentes recurrentes podía alcanzar resultados competitivos, comparables o incluso

mejores que los de los mejores modelos del momento (usando las competiciones-M como referencia). Dos objetivos adicionales de los autores del modelo eran la eficiencia (en términos de facilidad de entrenamiento) y la interpretabilidad.

El nombre del modelo N-BEATS es un acrónimo de *Neural Basis Expansion Analysis for Interpretable Time Series Forecasting*. El concepto de central de este modelo es precisamente el de *desarrollo en base neuronal (neural basis expansion)*. Para explicarlo usaremos el diagrama de la Figura 5.1 (pág. 62), adaptado del que aparece como Figura 1 en [59].

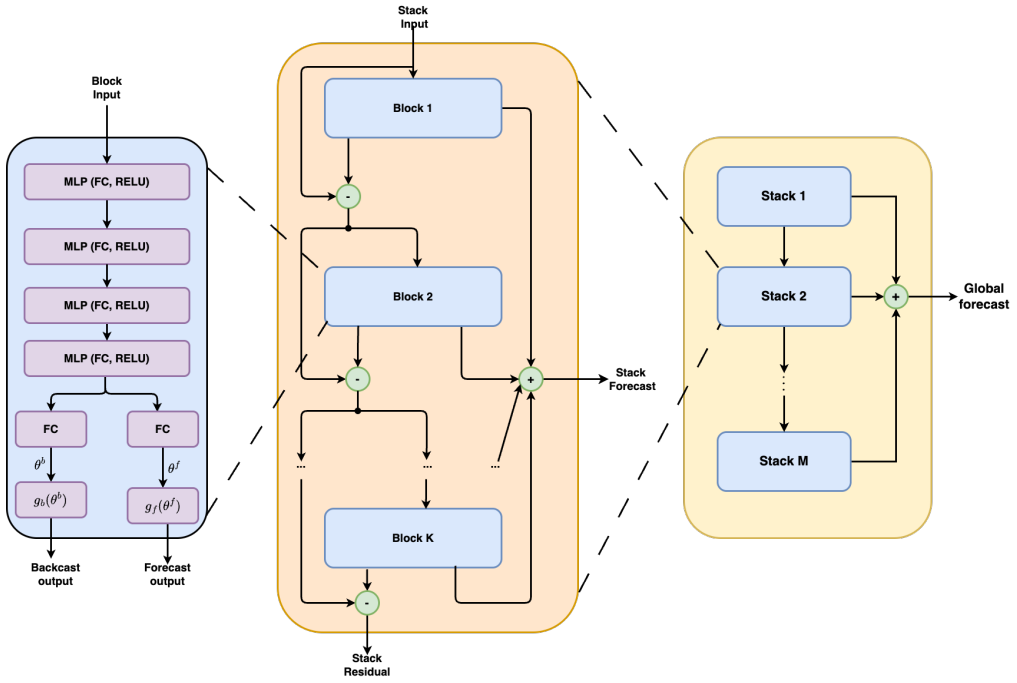


Figura 5.1: Esquema de la arquitectura del modelo N-Beats

Como puede verse en la Figura 5.1, el modelo N-BEATS utiliza una jerarquía de múltiples *pilas* (*stacks*) compuestas cada una de ellas de una serie de *bloques* (*blocks*). La parte izquierda de la figura describe la estructura de uno de esos bloques.

5.2.1 Análisis de los bloques individuales

Sea H el *horizonte de predicción* seleccionado. La entrada $\mathbf{x}_l$ del bloque número l se procesa en primer lugar a través de una red neuronal densa de tipo MLP y con activaciones no lineales de tipo RELU. Después, la salida de esa primera red se bifurca para producir dos salidas: la *predicción (forecast)* $\hat{\mathbf{y}}_l$ de los siguientes H valores de la serie y la *retroproyección (backcast)*, los valores que el modelo ajusta (*fitted values*) correspondientes al mismo intervalo temporal que ocupa $\mathbf{x}_\ell$ para reconstruir la señal de entrada.

Asumiendo que la red densa de entrada del bloque tiene cuatro capas, como en la Figura 5.1, las

ecuaciones que describen el funcionamiento del bloque son estas:

$$\begin{cases} \text{capas densas:} \\ \mathbf{h}_{\ell,1} = \text{RELU}(W_{\ell,1}\mathbf{x}_\ell + \mathbf{b}_{\ell,1}), \\ \mathbf{h}_{\ell,2} = \text{RELU}(W_{\ell,2}\mathbf{h}_{\ell,1} + \mathbf{b}_{\ell,2}), \\ \mathbf{h}_{\ell,3} = \text{RELU}(W_{\ell,3}\mathbf{h}_{\ell,2} + \mathbf{b}_{\ell,3}), \\ \mathbf{h}_{\ell,4} = \text{RELU}(W_{\ell,2}\mathbf{h}_{\ell,3} + \mathbf{b}_{\ell,4}), \\ \text{salida bifurcada:} \\ \theta^b_\ell = W^b_\ell(\mathbf{h}_{\ell,4}), \\ \theta^f_\ell = W^f_\ell(\mathbf{h}_{\ell,4}) \end{cases}$$

Las matrices $W_{\ell,k}$ y los términos independientes $\mathbf{b}_{\ell,k}$) son los parámetros entrenables de la primera parte del bloque. Por su parte las matrices W^f_ℓ y W^b_ℓ son también parámetros entrenables que definen proyecciones lineales para dar como resultado las predicciones y retroproyecciones del bloque, respectivamente.

A continuación las dos salidas del bloque se obtienen mediante sendas **capas de expansión en bases** (las que dan nombre al modelo), dadas mediante:

$$\begin{cases} \hat{\mathbf{x}}_l = g^b_\ell(\theta^b_\ell) = \sum_{i=1}^{\dim(\theta^b_\ell)} \theta^b_{\ell,i}\mathbf{v}^b_i \\ \hat{\mathbf{y}}_l = g^f_\ell(\theta^f_\ell) = \sum_{i=1}^{\dim(\theta^f_\ell)} \theta^f_{\ell,i}\mathbf{v}^f_i \end{cases}$$

Los vectores $\mathbf{v}^f_i$ y $\mathbf{v}^b_i$ forman las bases de predicción y retroproyección respectivamente, cuya misión es proporcionar un conjunto suficientemente expresivo como para representar todos los patrones y componentes presentes en la serie temporal de entrada al bloque. Las *funciones de expansión* g^f_ℓ y g^b_ℓ que corresponden a esas dos bases pueden ser también parámetros entrenables del modelo. Pero también pueden seleccionarse dentro de conjuntos prefijados con la finalidad de inducir al modelo a reflejar determinados patrones en su salida. Esto permite el modelo N-BEATS tenga una interpretabilidad similar a los modelos clásicos de descomposición en componentes (estacionalidad, tendencia).

Las dos salidas de cada bloque se utilizan de la siguiente manera:

- Las predicciones $\hat{\mathbf{y}}_l$ de cada uno de los bloques que componen una pila se suman para dar como resultado la predicción global $\hat{\mathbf{y}}$ de esa pila.
- La salida $\hat{\mathbf{x}}_l$ del bloque se utiliza para crear la conexión residual con la entrada del siguiente bloque que describimos en el siguiente apartado.

5.2.2 Conexiones residuales dobles

La incorporación de conexiones residuales, junto con la normalización, es una de las estrategias comunes en el diseño de arquitecturas de redes neuronales, con el objetivo de combatir el problema del desvanecimiento de los gradientes y mejorar la entrenabilidad de la red. Es un método relacionado con la técnica *norm-add* que se usa en los modelos de tipo transformer [7].

En el modelo N-BEATS esta idea se utiliza por partida doble (ver la parte central de la Figura 5.1):

- Por un lado, la salida de retroproyección de cada bloque $\hat{\mathbf{x}}_l$ se utiliza para generar el *residuo* que servirá como entrada del siguiente bloque en la pila:

  $$\mathbf{x}_l = \mathbf{x}_{l-1} - \hat{\mathbf{x}}_{l-1}$$

  Aparte de los ya mencionados, esta conexión residual permite a cada bloque centrarse en aquellos patrones de la señal que el bloque anterior no ha podido representar adecuadamente, de forma análoga a lo que sucede en los modelos de boosting.

- Por otro lado, como hemos dicho, la predicción de la pila (*stack forecast* en la Figura 5.1) es esta combinación de las predicciones de todos los bloques de la pila

  $$\hat{\mathbf{y}} = \sum_{\ell} \hat{\mathbf{y}}_\ell$$

  constituyendo así una forma de **conexión residual doble** diseñada con el propósito de aumentar la entrenabilidad del modelo.

Hay que aclarar que el caso del primer bloque de cada pila es especial: para la primera pila la entrada del primer bloque es la serie de entrada al modelo y para el primer bloque de las siguientes pilas, a partir de la segunda, la entrada es la salida (el residuo) de la anterior pila. De la misma forma, la salida de retroproyección del último bloque de cada pila (*stack residual* en la Figura 5.1) es la entrada del primer bloque en la siguiente pila.

5.2.3 Interpretabilidad del modelo N-BEATS

Como hemos mencionado antes, el artículo original [59] muestra cómo seleccionar las funciones g_ℓ^f y g_ℓ^b para inducir al modelo a especializar las pilas que lo componen, de forma que sus predicciones puedan asimilarse a una descomposición de la serie temporal análoga a las descomposiciones clásicas. No obstante, en el caso de la generación eólica que nos ocupa en este TFM dichas componentes no están presentes en la serie temporal. Por esa razón en la implementación de estos modelos de tipo N-BEATS/N-HITS se ha optado por usar lo que los autores del modelo denominan **configuración genérica**. En este caso las funciones g_ℓ^f y g_ℓ^b pasan a ser entrenables, dadas por bases de expansión que son proyecciones lineales.

5.2.4 N-HITS como evolución del N-BEATS

El modelo N-HITS (acrónimo de *Neural Hierarchical Interpolation for Time Series Forecasting*) apareció en 2022 en el artículo [61]. Este modelo es una evolución de la arquitectura de N-BEATS, que busca mejorar el desempeño de estos modelos en el problema de la predicción de horizontes temporales largos y su eficiencia computacional. Para ello el modelo incorpora dos ingredientes nuevos que describiremos más abajo:

- El **muestreo a diferentes frecuencias** (*multi-rate data sampling*) de la serie temporal.
- La **interpolación jerárquica** (*hierarchical interpolation*) para la predicción .

La arquitectura del modelo, que se muestra en la Figura 5.2 (pág. 65) está basada en la de N-BEATS, con una estructura de bloques y pilas. Hemos resaltado en color amarillo los elementos novedosos de este modelo en comparación con N-BEATS.

5.2.5 Muestreo de la serie temporal a diferentes frecuencias

Este ingrediente del modelo corresponde a la capa de tipo *MaxPool* que aparece como primera capa en cada bloque del modelo, ver la parte izquierda de la Figura 5.2. Esta capa utiliza un núcleo de tamaño k_ℓ. Con núcleos de tamaño grande el bloque se ve llevado a concentrarse en las componentes de baja frecuencia (y por tanto mayor escala) de la serie de entrada, con lo que se busca beneficiar a la predicción de horizontes temporales largos. Por el contrario, un núcleo de tamaño pequeño lleva al bloque a centrarse en los patrones de frecuencia más alta (y escala menor). Aparte de esa especialización de los bloques, esto contribuye a reducir la carga computacional de muchos de los bloques y el número de parámetros que es necesario para el modelo. Lo que a su vez es una forma de normalización que combate el posible sobreajuste del modelo.

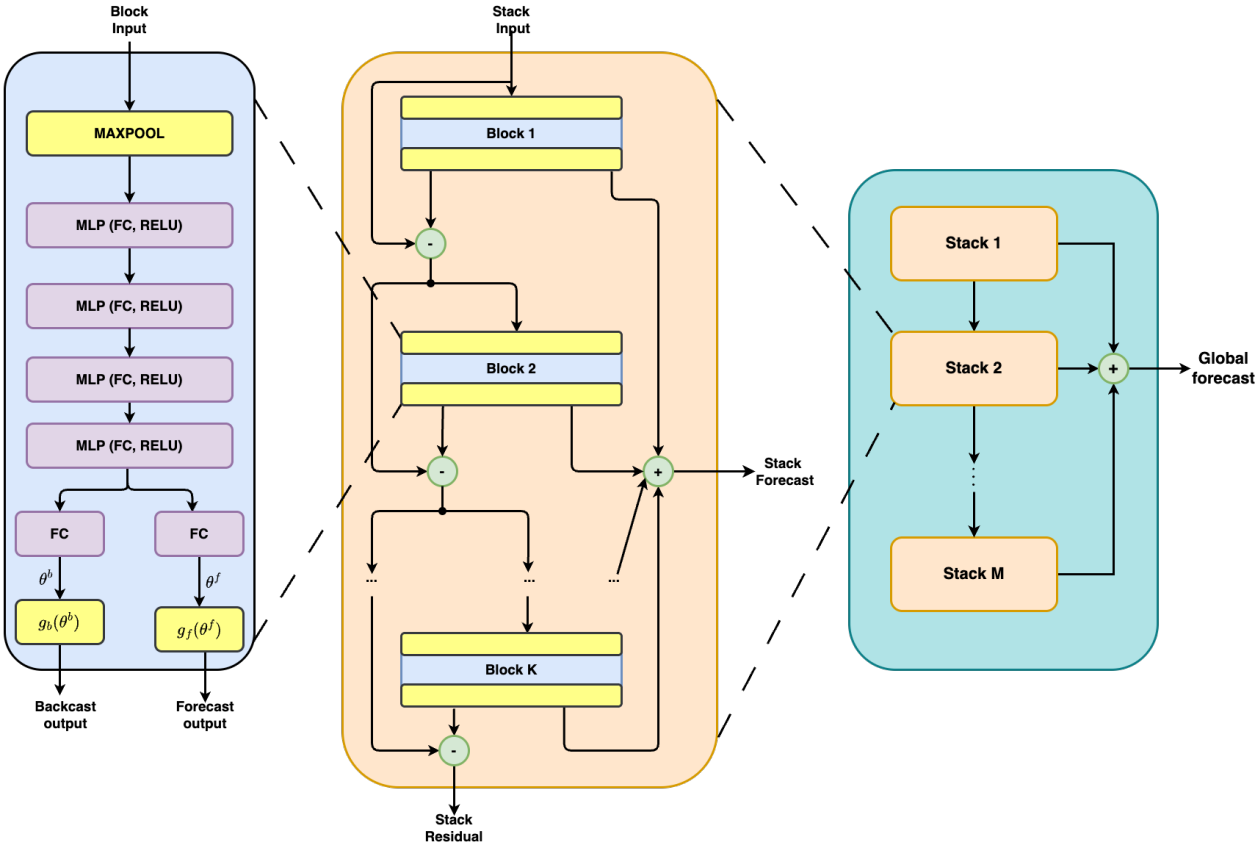


Figura 5.2: Esquema de la arquitectura del modelo N-HiTS

5.2.6 Interpolación temporal jerárquica

En muchos modelos de predicción de series temporales, las predicciones del modelo forman un vector con el mismo número de elementos que el horizonte de predicción H. En cambio, en este método de interpolación jerárquica, cada bloque produce una predicción de dimensión reducida, controlada por una **tasa de expresividad** (*expresiveness ratio*) r_ℓ propia del bloque. Las ecuaciones que gobiernan

este proceso son:

$$\begin{cases} |\theta_\ell^f| = \lceil r_\ell H \rceil\,, \quad |\theta_\ell^b| = \lceil r_\ell L \rceil \\ \\ \hat{\mathbf{y}}_{\tau,\ell} = g(\tau, \theta_\ell^f), \quad \forall \tau \in \{t+1, \dots, t+H\} \\ \tilde{\mathbf{y}}_{\tau,\ell} = g(\tau, \theta_\ell^b), \quad \forall \tau \in \{t-L, \dots, t\} \end{cases}$$

Las dos primeras ecuaciones relacionan el número de coeficientes de salida del bloque (tanto para la predicción como para la retroproyección), siendo H el horizonte de predicción y L el tamaño de la ventana de entrada del modelo. Las dos ecuaciones inferiores describen el uso de una **función de interpolación** g para reconstruir ambas salidas del bloque (predicción $\hat{\mathbf{y}}_{\tau,\ell}$ y retroproyección $\tilde{\mathbf{y}}_{\tau,\ell}$) con las dimensiones adecuadas para el resto del proceso del modelo. La función de interpolación temporal g puede ser una función lineal a trozos, pero también puede usarse interpolación polinómica a trozos de distintos grados, etc.

Nota sobre la notación: el símbolo $\tilde{\mathbf{y}}$ representa en el modelo N-HiTS la salida de retroproyección de cada bloque, lo que en el modelo N-BEATS se indicaba mediante $\hat{\mathbf{x}}_l$. Hemos preferido mantener en ambos casos la notación de los artículos originales, para facilitar la referencia a esos trabajos.

El resultado de este esquema es que un modelo N-HiTS que reciba una señal de entrada horaria puede albergar bloques que hagan una predicción cada hora, otro bloque que haga una predicción diaria, otro con predicciones semanales, etc. La predicción por interpolación jerárquica combina esas predicciones a distintas frecuencias en una predicción final global del bloque en la frecuencia adecuada. Para inducir esta especialización de los bloques los autores del modelo N-HiTS proponen utilizar una sucesión exponencialmente creciente de tasas de expresividad de los bloques o alternativamente, usar tasas de expresividad preseleccionadas para capturar componentes conocidos presentes en la serie. Por lo demás, la arquitectura del modelo N-HiTS incorpora las mismas estructuras que se han comentado en el caso del modelo N-BEATS.

5.2.7 Implementación y resultados

La implementación de N-HITS que hemos usado es la que está disponible en la librería neuralforecast de Nixtla [29], concretamente mediante las clases `NHITS` y `AutoNHITS`, utilizando para ello el procedimiento descrito anteriormente en la Sección 2.3. El resultado está disponible en la carpeta *models/NHITS* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo que se han considerado en este trabajo es:

- `encoder_layers`: número de capas de la parte encoder del modelo,
- `n_heads`: número de cabezas de atención del bloque de atención,
- `hidden_size`: número de neuronas de las redes FF del modelo,
- `dropout`: tasa de descarte para las conexiones residuales,
- `head_dropout`: tasa de descarte para la capa lineal,
- `attn_dropout`: tasa de descarte para la capa de atención,
- `patch_len`: longitud del segmento (tach),
- `learning_rate`: tasa de aprendizaje.

5.2.7.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `NHiTS_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento

NHiTS_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.3 (67). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 12 minutos. El tiempo del bucle de validación es de aproximadamente 80 minutos.

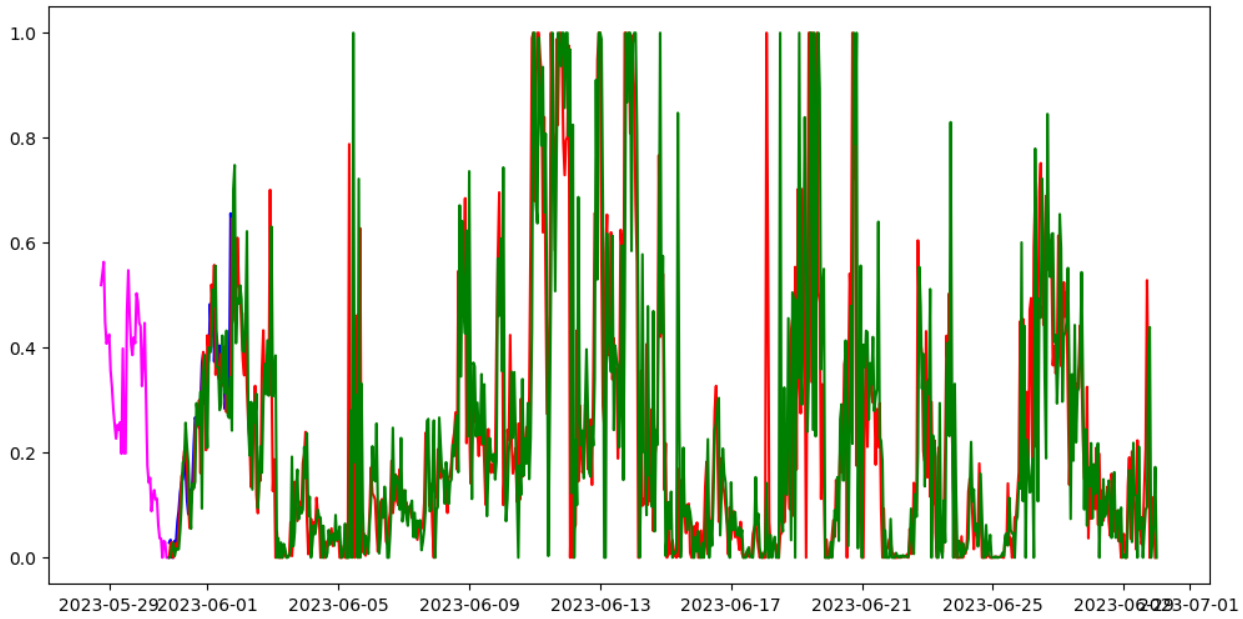


Figura 5.3: Predicciones del modelo N-HiTS para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 394.64
- MAE a 2h: 716.48
- MAE a 3h: 901.91

5.2.7.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `NHits_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento NHits_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.4 (68). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 75 minutos.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.00667
- MAE a 2h: 0.02290
- MAE a 3h: 0.04667

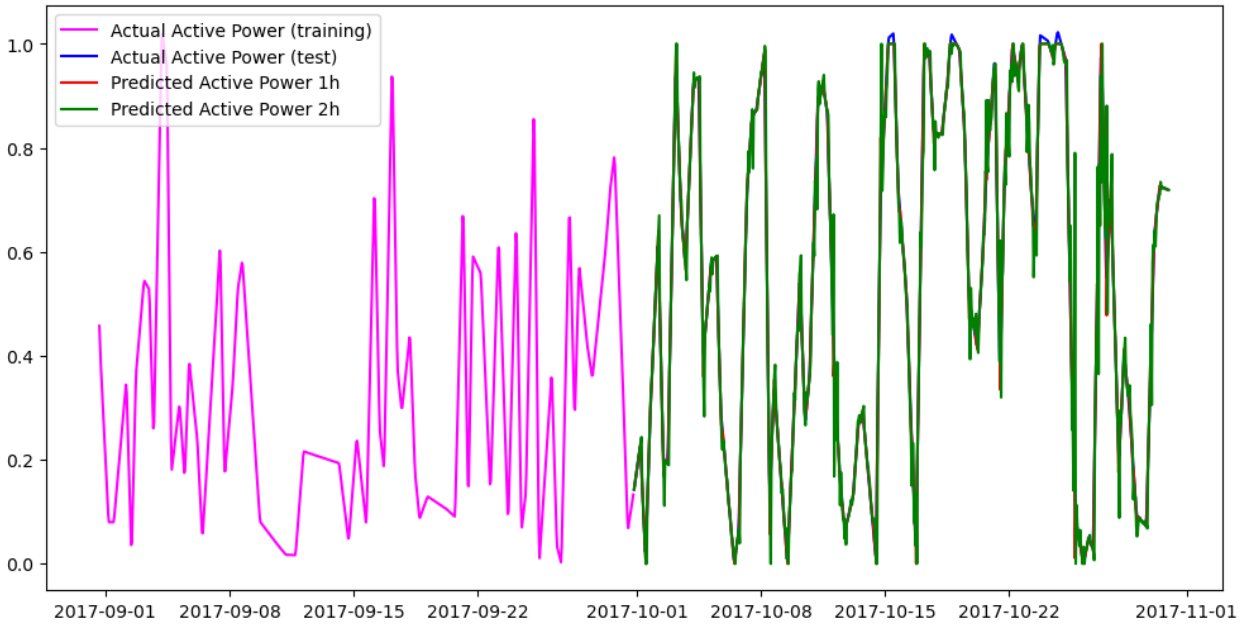


Figura 5.4: Predicciones del modelo N-HiTS para el dataset KaggleWPGD.

5.2.8 Comentarios sobre implementación y resultados del modelo N-HITS

El modelo N-HiTS ha demostrado tener una buena capacidad para modelar dinámicas temporales complejas gracias a su arquitectura jerárquica basada en bloques de backcasting/forecasting.

Sin embargo, esta mejora de la arquitectura viene acompañada de una mayor complejidad computacional y de una sensibilidad mayor a la selección de hiperparámetros, lo que obliga a prestar especial atención al proceso de ajuste mediante validación. En este trabajo, dicha selección se ha realizado utilizando la clase AutoNHITS de la librería neuralforecast, lo que ha permitido automatizar esta tarea y reducir el riesgo de sobreajuste.

5.3 El modelo TiDE

5.3.1 Introducción

Este modelo fue presentado en abril de 2023 por un grupo de investigación vinculado con *Google Research* [62]. La arquitectura del modelo sigue en la tradición de los modelos encoder-decoder previos a la aparición de los de tipo transformer, pero no usa recurrencia ni mecanismos de atención: se basa en redes de tipo MLP puras, para dar como resultado una arquitectura (que los autores describen como *embarrassingly simple*) con la que se busca superar computacionalmente a esas dos alternativas (RNN y transformers) manteniendo capacidad predictiva de máximo nivel en problemas multivariable con horizontes de predicción largos. Una de las motivaciones de los autores para abordar este tipo de arquitectura fue la publicación del artículo de Zeng [55] que ya hemos comentado y que subrayaba las posibles limitaciones de las arquitecturas de tipo transformer en el análisis de problemas de series temporales. Aunque, como se ha visto, esas objeciones parecen haber sido superadas con las nuevas arquitecturas de tipo transformer, su planteamiento ha servido de punto de partida para modelos como el que vamos a describir. Por otro lado, el modelo TiDE permite incorporar variables exógenas de manera sencilla, lo que lo distingue de otros modelos competidores

y lo sitúa en la antesala de los modelos fundacionales para series temporales. La entrada [63] del blog de M. Peixeiro contiene una descripción conceptual de este modelo.

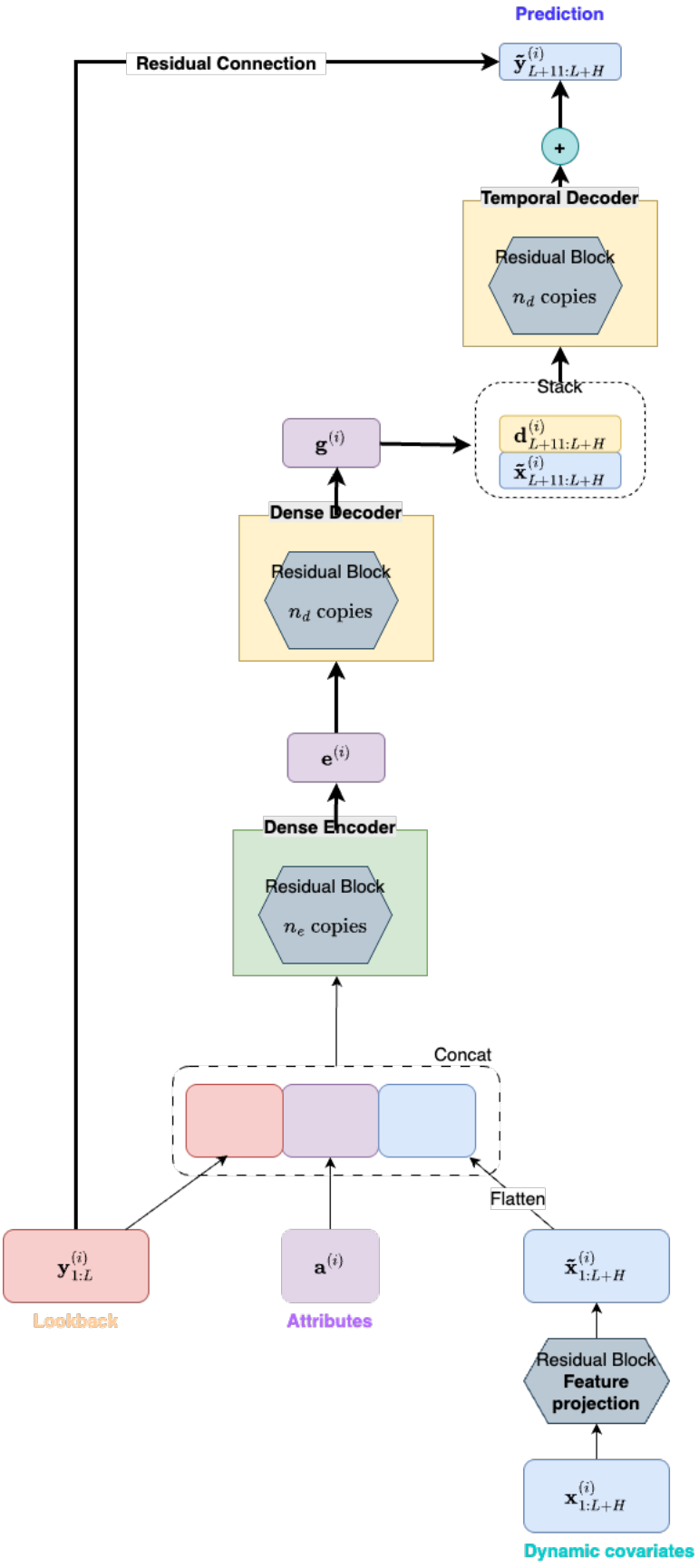


Figura 5.5: Esquema de la arquitectura del modelo TiDE

5.3.2 La arquitectura de TiDE y el bloque residual

La arquitectura del modelo TiDE puede verse en la Figura 5.5 (pág. 70). El ingrediente esencial de esa arquitectura es el **bloque residual**, que se utiliza repetidamente y que en esa figura hemos representado mediante un hexágono de color verde. La estructura del bloque residual aparece descrita en más detalle en la Figura 5.6 (pág. 71).

Como indican los propios autores, los bloques encoder y decoder podrían verse como un único bloque, pero esa estructuración permite visualizar mejor los principios de diseño que guían esta arquitectura. En lo que sigue se considera un problema multidimensional con N series temporales y sean:

- $y_{1:L}^{(i)}$ los valores pasados (L de *lookback*) de la i-ésima serie temporal,
- $\hat{y}_{L+1:L+H}^{(i)}$ los valores de la serie en el horizonte de predicción,
- $x_t^{(i)}$ los valores de las variables exógenas *dinámicas* en el instante t.
- $_t^{(i)}$ los valores de las variables exógenas *estáticas* en el instante t.

5.3.2.1 El bloque residual

El bloque residual, cuya estructura se muestra en la Figura 5.6 (pág. 71) es el auténtico núcleo de esta arquitectura, que se podría decir que es una de las máx clásicas de las que hemos visto, salvo por la incorporación de uno de los ingredientes característicos de muchas propuestas modernas: las conexiones residuales con normalización, tipo normadd. Su operación se puede resumir en la ecuación $\tilde{x}_t^{(i)} = \text{ResidualBlock}\left(x_t^{(i)}\right)$.

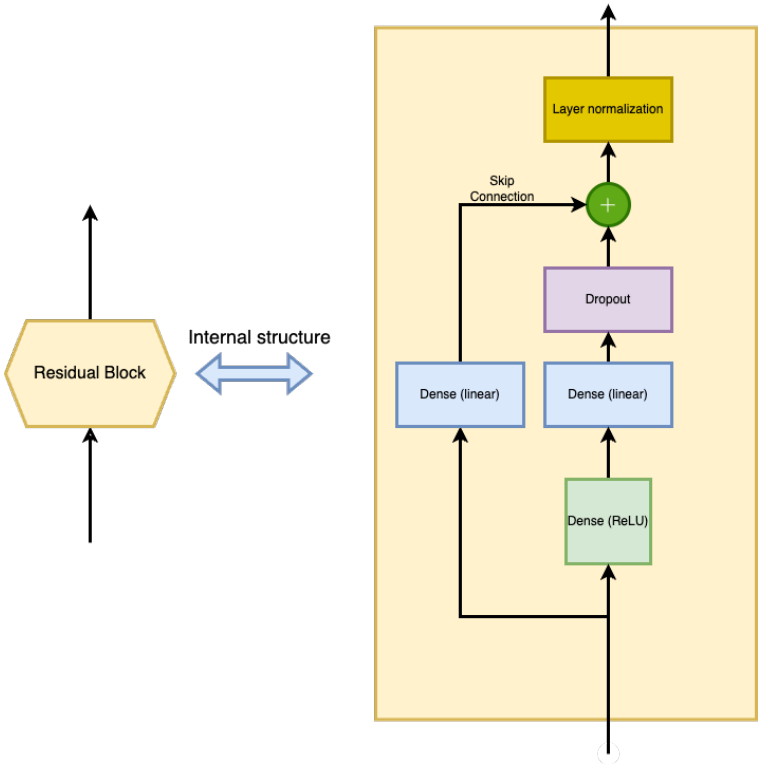


Figura 5.6: Esquema del bloque residual en el modelo TiDE

5.3.2.2 Encoder y decoder

El encoder concatena las variables exógenas, estáticas y dinámicas, junto con el pasado de la serie temporal a predecir y las transforma en un embedding $e^{(i)}$ mediante la aplicación de múltiples bloques residuales. Esta es la parte de la arquitectura que corresponde a la parte inferior de la Figura 5.6 y que da como resultado ese embedding que podemos representar mediante la ecuación:

$$e^{(i)} = \text{Encoder}\left(y^{(i)}_{1:L}; \tilde{x}^{(i)}_{1:L+H}; a^{(i)}\right)$$

Esta operación supone una reducción de la dimensionalidad que busca reducir el coste computacional del modelo.

A continuación el decoder consta de dos subestructuras:

- El *decoder denso*, que mediante varios bloques residuales reduce todavía más la dimensionalidad del embedding. Como se ha dicho antes, esta parte del decoder puede en realidad considerarse como una extensión del encoder. Viene descrito por

$$\begin{cases} g^{(i)} = \text{Decoder}\left(e^{(i)}\right) \in \mathbb{R}^{p \cdot H} \\ D^{(i)} = \text{Reshape}\left(g^{(i)}\right) \in \mathbb{R}^{p \times H} \end{cases}$$

- A continuación el *decoder temporal* recibe la salida del bloque encoder - decoder denso pero también tiene una conexión con la proyección de aquellas variables exógenas dinámicas *de calendario*, cuyos valores futuros se conocen. Ambas entradas se combinan y procesan en un último bloque residual.

EL modelo se complementa además con una última conexión residual hacia los valores pasados de la serie, que junto con la salida del decoder temporal produce las predicciones

$$\hat{y}^{(i)}_{L+t} = \text{TemporalDecoder}\left(d^{(i)}_t; \tilde{x}^{(i)}_{L+t}\right) + \hat{y}^{(i)}_{1:L}$$

para cada instante t en el horizonte de predicción. En resumen, se puede argumentar que la estructura del modelo es una red densa clásica, pero altamente interconectada mediante conexiones residuales que buscan preservar la estructura de correlaciones y autocorrelaciones de las señales de entrada.

5.3.3 Implementación y resultados

La implementación de TiDE que hemos usado es la que está disponible en la librería neuralforecast de Nixtla [29], concretamente mediante las clases `TiDE` y `AutoTiDE`, utilizando para ello el procedimiento descrito anteriormente en la Sección 2.3. El resultado está disponible en la carpeta *models/TiDE* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo TiDE que se han considerado en este trabajo es:

- `decoder_output_dim`: número de unidades para la capa de salida del decoder denso,
- `hidden_size`: número de unidades de las capas densas,
- `num_encoder_layers`: número de capas del encoder,
- `num_decoder_layers`: número de capas del decoder„
- `temporal_decoder_dim`: número de unidades de las capas densas del decoder temporal,
- `dropout`: tasa de descarte,
- `learning_rate`: tasa de aprendizaje.

5.3.3.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `TiDE_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento TiDE_AV.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.7 (73). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 26 minutos. El tiempo del bucle de validación es de *más de seis horas*.

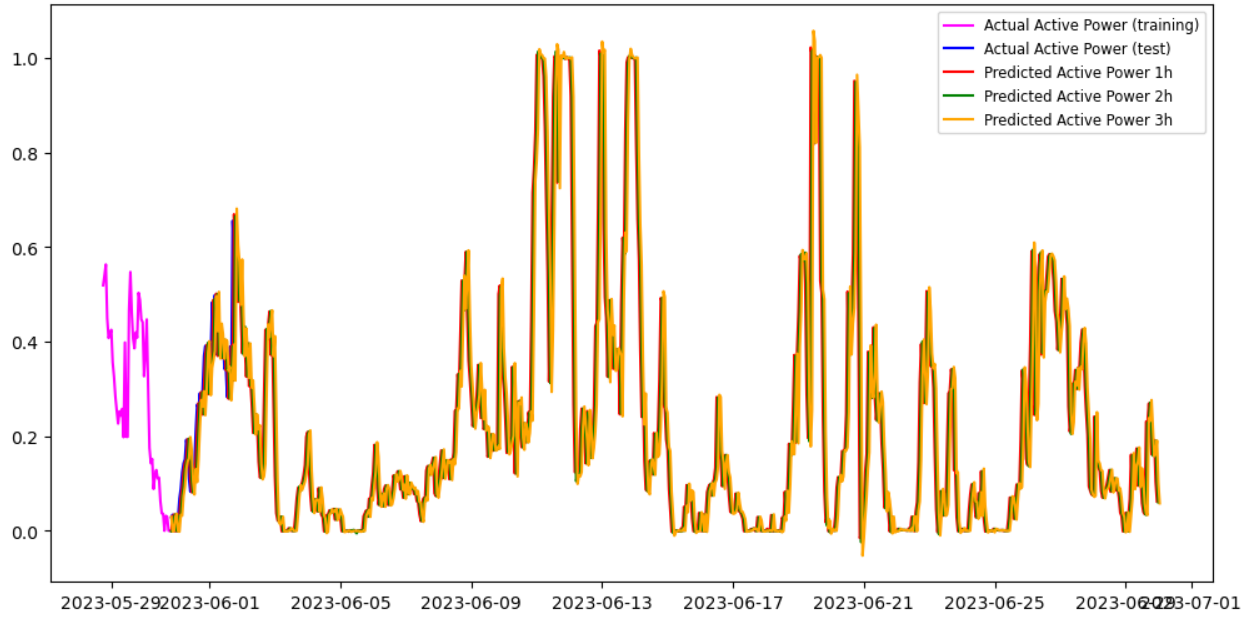


Figura 5.7: Predicciones del modelo TiDE para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 236.97
- MAE a 2h: 377.24
- MAE a 3h: 486.45

5.3.3.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `TiDE_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento TiDE_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.8 (74). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 22 minutos. El tiempo del bucle de validación es sorprendentemente mucho más corto en este caso, por debajo de 30 minutos.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.00534

- MAE a 2h: 0.01569
- MAE a 3h: 0.02901

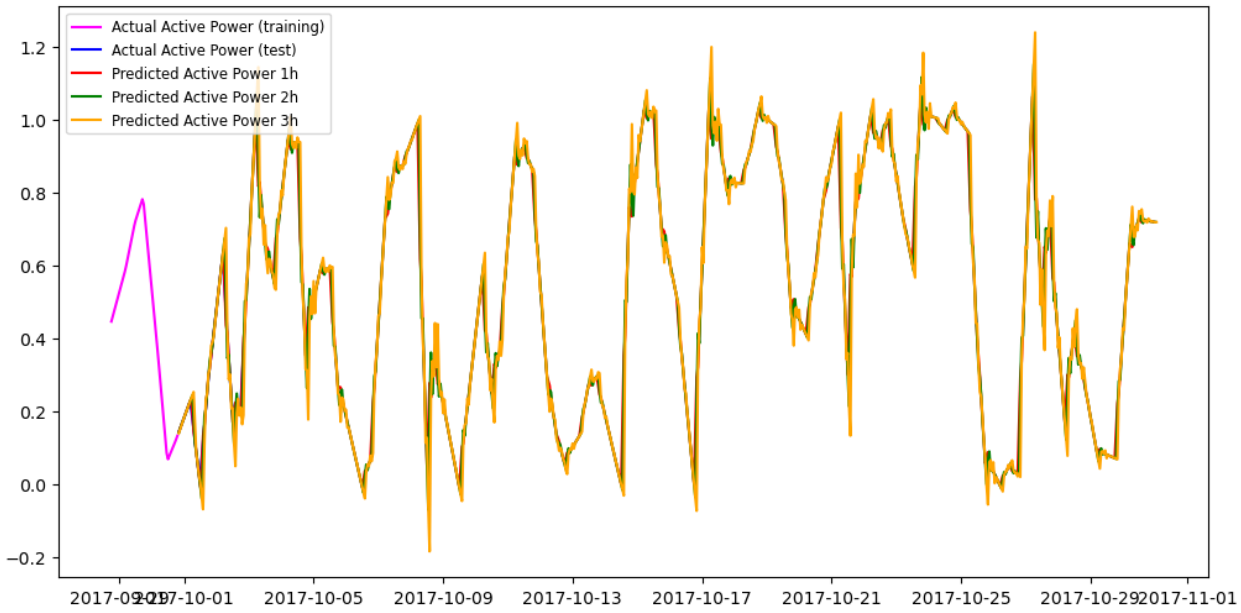


Figura 5.8: Predicciones del modelo TiDE para el dataset KaggleWPGD.

5.3.4 Comentarios sobre el modelo TiDE

- A nivel cualitativo, las predicciones reflejan una buena capacidad de capturar la dinámica de la serie, tanto a corto como a medio plazo.
- A diferencia de modelos como PatchTST, TiDE sí está diseñado para incorporar variables exógenas de forma nativa. Esta característica es especialmente relevante en tareas de predicción energética, donde información adicional (como meteorología o datos operativos) puede ser determinante para mejorar la precisión del modelo.

5.4 Modelos de arquitectura KAN (redes Kolmogorov-Arnold)

5.4.1 Introducción

El modelo RMoK fue propuesto en 2024 como una adaptación de las Kolmogorov-Arnold Networks (KAN) al ámbito de la predicción de series temporales [64]. Su objetivo es explorar las ventajas teóricas y prácticas que ofrecen las arquitecturas KAN frente a los modelos basados en perceptrones multicapa (MLP), transformadores o convoluciones, especialmente en términos de interpretabilidad y eficiencia computacional.

KAN utiliza funciones univariadas (como B-splines, polinomios, wavelets) en los bordes de la red, en lugar de activaciones fijas por neurona, reduciendo el número de parámetros y aumentando la interpretabilidad

RMoK introduce una arquitectura minimalista basada en un único nivel de expertos KAN especializados, asignados mediante un mecanismo de gating. Esta estructura permite representar la dinámica de series temporales con interpretabilidad y competitividad frente a modelos más complejos.

5.4.2 Arquitectura RMoK: Mixture of KAN Experts

El modelo RMoK con la arquitectura de la figura 5.9 (pág. 75), adaptado del que aparece como Figura 2 en [65], utiliza una sola capa de expertos KAN en combinación con un mecanismo de gating y una normalización reversible (RevIN).

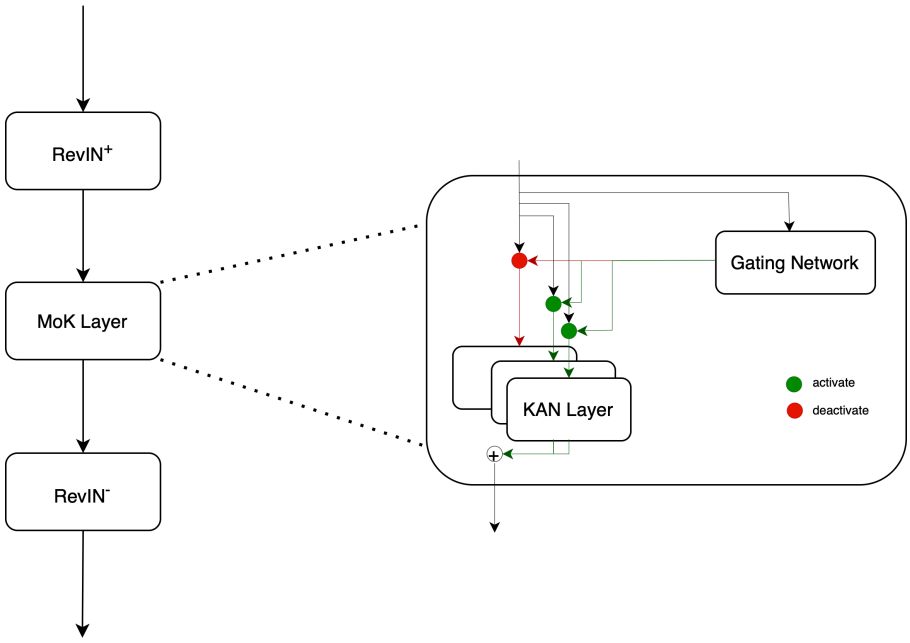


Figura 5.9: Esquema de la arquitectura del modelo RMoK y la capa MoK

El flujo completo es:

RevIN+: normalización de cada variable con una transformación afín aprendible.

Capa MoK: combinación de múltiples expertos KAN, cada uno entrenado sobre una parte de los datos. Un gating network selecciona los expertos más adecuados por variable.

RevIN-: desnormalización de la predicción al espacio original.

$$x_{(\ell+1)} = \sum_{i=1}^{N} G_i\left(x_{(\ell)}\right) \cdot K_i\left(x_{(\ell)}\right)$$

donde:

- $x_{(\ell)}$ representa la entrada a la capa MoK,
- K_i es la salida del i-ésimo experto KAN,
- G_i es el peso asignado por el *gating network* a dicho experto,
- N es el número total de expertos.

5.4.3 Análisis del bloque MoK

La capa MoK es una adaptación específica del paradigma Mixture-of-Experts (MoE), con la particularidad de utilizar como expertos funciones univariadas modeladas mediante arquitecturas KAN. Esto dota al modelo de varias ventajas:

Adaptabilidad local: La capa puede asignar diferentes expertos a distintas regiones del espacio temporal, permitiendo capturar patrones diversos (estacionales, tendencia, abruptos, etc.). Modularidad e interpretabilidad: Cada experto KAN puede analizarse por separado, identificando qué relaciones temporales está modelando. Eficiencia computacional: Al permitir un gating escaso (Top-k gating), se reduce el coste de inferencia al activar solo los expertos más relevantes para cada instancia. Para evitar que el modelo tienda a favorecer siempre a un subconjunto fijo de expertos (lo que limitaría la capacidad expresiva del MoK), se introduce un término de regularización en la función de pérdida que penaliza cargas desequilibradas entre expertos:

$$L_{\text{total}} = \text{MSE}(Y, \hat{Y}) + w_l \cdot (\text{LOADS})^2$$

donde LOADS LOADS cuantifica la varianza en la activación del gating, y w_i es un hiperparámetro que controla la penalización. Así, se incentiva una distribución más equitativa del trabajo entre expertos, mejorando la generalización del modelo.

5.4.4 Conclusión arquitectura RMoK

La arquitectura RMoK se presenta como una solución elegante y potente para la predicción de series temporales multivariantes. Su diseño se sustenta en tres pilares clave:

- **Fundamento matemático sólido**: Se apoya en el teorema de Kolmogorov-Arnold, que garantiza que cualquier función multivariada puede representarse como composición de funciones univariadas, lo cual es precisamente el principio que explotan las redes KAN.
- **Modularidad y simplicidad**: A diferencia de modelos profundos que pueden ser difíciles de interpretar o entrenar, RMoK se basa en una única capa MoK, lo que facilita el análisis, la depuración y la interpretación de sus componentes internos.
- **Interpretabilidad sin sacrificar rendimiento**: Ofrece resultados competitivos en benchmarks estándar sin renunciar a la explicabilidad, lo que lo hace especialmente atractivo para aplicaciones industriales, energéticas o de salud, donde comprender las decisiones del modelo es tan importante como su precisión.
- **Capacidad de visualización y análisis de expertos**: Gracias al mecanismo de gating, es posible examinar qué expertos están activos en cada predicción y qué variables temporales han influido más en la salida, facilitando la extracción de conocimiento explícito.

En resumen, RMoK representa una alternativa sólida, explicable y eficiente a las arquitecturas dominantes basadas en Deep Learning, posicionándose como una opción muy prometedora para tareas de forecasting que requieran tanto precisión como transparencia.

5.4.5 Implementación y resultados

De nuevo en este caso hemos podido utilizar la implementación de RMoK disponible en la librería neuralforecast de Nixtla [29], lo cual es especialmente reseñable en un modelo *publicado en Agosto de 2024*.

La implementación utiliza las clases `RMoK` y `AutoRMoK`, de forma similar a la que hemos visto en otros modelos, mediante el procedimiento descrito en la Sección 2.3. El resultado está disponible en la carpeta *models/RMoK* del repositorio de GitHub.

Junto con los valores fijos descritos en la Sección 2.3, la lista de hiperparámetros específicos del modelo que se han considerado en este trabajo es:

- `taylor_order`: orden del polinomio de Taylor utilizado,
- `wavelet_function`: forma de la función wavelet,

- `dropout`: tasa de descarte,
- `learning_rate`: tasa de aprendizaje

5.4.5.1 Resultados para el conjunto AV

El modelo correspondiente a los datos de AV se encuentra en el notebook de Jupyter llamado `RMoK_AV.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento RMoK_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.10 (77). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 14 minutos. El tiempo del bucle de validación es de aproximadamente 2 horas.

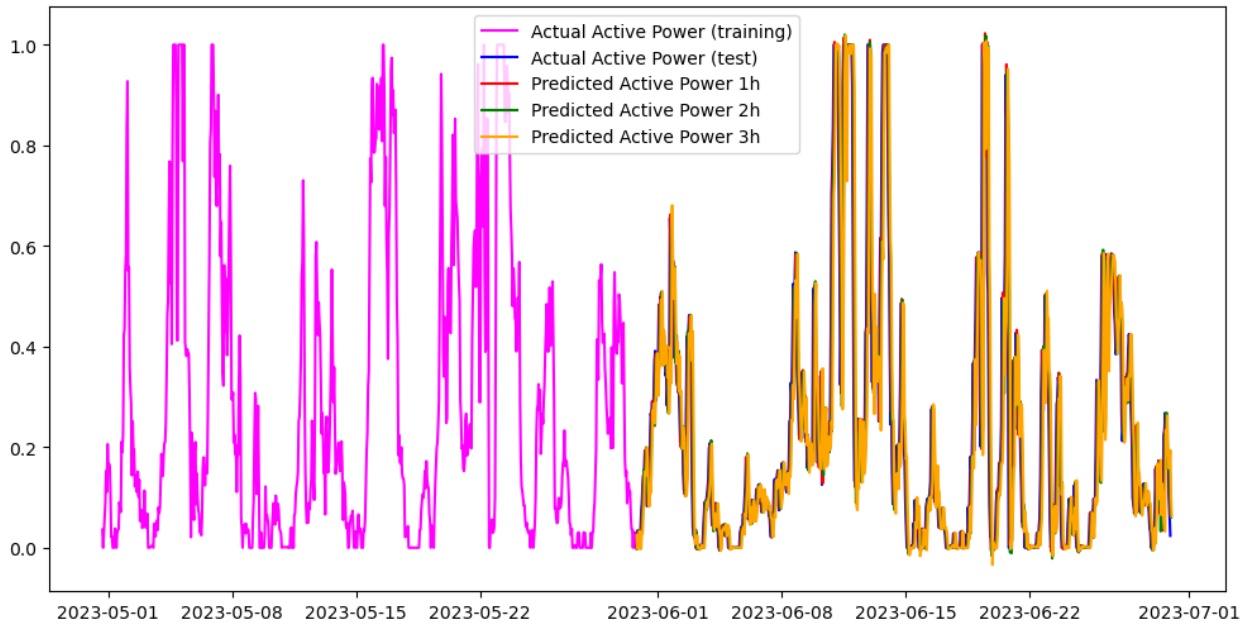


Figura 5.10: Predicciones del modelo RMoK para el dataset AV

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 239.57
- MAE a 2h: 378.78
- MAE a 3h: 484.76

5.4.5.2 Resultados para el conjunto KaggleWPGD

El modelo correspondiente a los datos KaggleWPGD se encuentra en el notebook de Jupyter llamado `RMoK_KaggleWPGD.ipynb`. Pueden verse los resultados de la ejecución de ese modelo en el documento RMoK_KaggleWPGD.html disponible en el repositorio de código.

Con un tamaño de entrada igual a 6h y un horizonte de predicción de 3h se obtienen las predicciones que se ilustran en la Figura 5.11 (78). El tiempo de entrenamiento del modelo (con selección de hiperparámetros) es de aproximadamente 15 minutos. El tiempo del bucle de validación es de aproximadamente 3 horas.

Los valores de MAE obtenidos en las predicciones del conjunto de test son:

- MAE a 1h: 0.00412
- MAE a 2h: 0.01288
- MAE a 3h: 0.02616

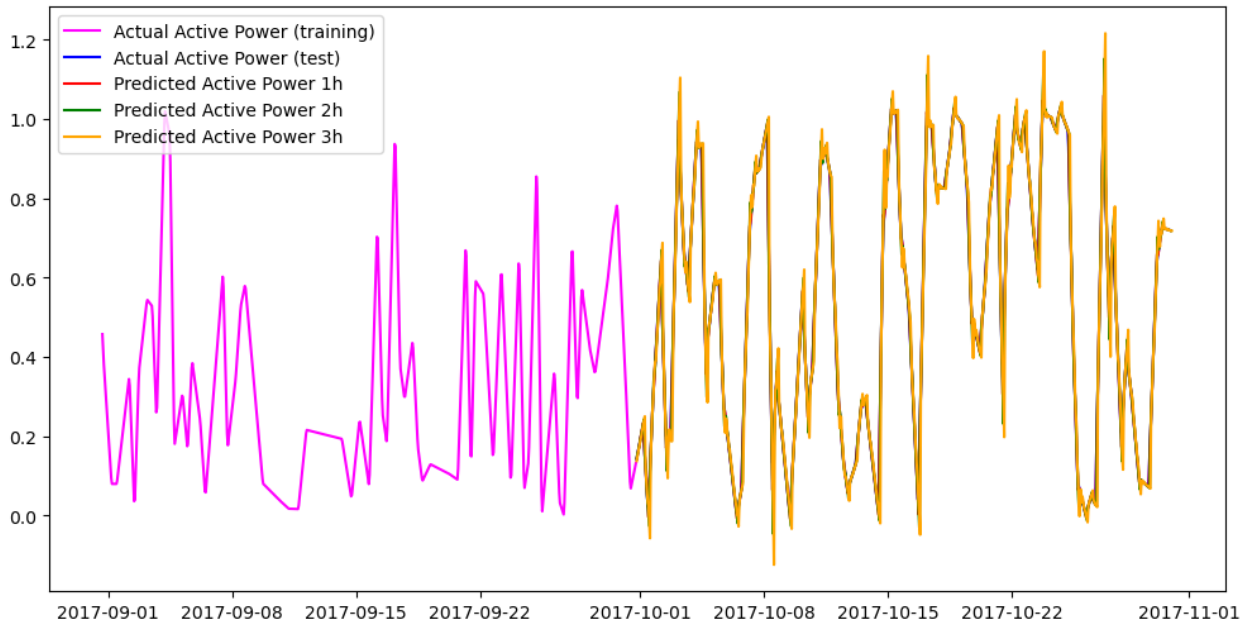


Figura 5.11: Predicciones del modelo RMoK para el dataset KaggleWPGD.

5.4.6 Comentarios sobre impementación y resultados del modelo RMoK

- Para el conjunto AV, los resultados son comparables a los obtenidos con TiDE, ligeramente mejores a largo plazo (3h).
- Al igual que TiDE, RMoK está diseñado para incorporar variables exógenas de forma nativa, lo que lo convierte en un modelo adecuado para contextos complejos donde la variable objetivo depende de múltiples factores externos. Esta característica le proporciona una ventaja significativa frente a arquitecturas como PatchTST, que no permiten esta integración de manera directa.

Capítulo 6

Alineación con los Objetivos de Desarrollo Sostenible (ODS)

6.1 Introducción

Este Trabajo de Fin de Máster (TFM) se enmarca en la estrategia global de desarrollo sostenible promovida por las Naciones Unidas, alineándose con varios Objetivos de Desarrollo Sostenible (ODS) que buscan transformar los sistemas energéticos para enfrentar los retos ambientales y sociales del siglo XXI. En particular, el proyecto tiene una relación directa con los ODS que impulsan la transición hacia energías limpias, la acción contra el cambio climático y la innovación tecnológica, pilares fundamentales para la construcción de un futuro sostenible.

6.1.1 ODS 7: Energía Asequible y No Contaminante

Garantizar el acceso universal a una energía asequible, segura, sostenible y moderna constituye un reto prioritario a nivel mundial. La energía eólica marina se presenta como una fuente energética limpia con un enorme potencial para contribuir a la descarbonización de la matriz eléctrica global. Sin embargo, la integración eficiente de esta energía renovable depende en gran medida de la capacidad para predecir con precisión su generación, dada la naturaleza altamente variable del recurso eólico.

En este contexto, el presente TFM desarrolla y compara modelos avanzados de aprendizaje profundo (Deep Learning), incluyendo arquitecturas recientes como Transformers y Kolmogorov–Arnold Networks (KAN), para mejorar la predicción a corto plazo (3 horas) de la producción eléctrica en parques eólicos marinos. Esta mejora en la capacidad predictiva facilita la gestión óptima de la energía eólica dentro del sistema eléctrico, incrementando su fiabilidad y eficiencia operativa. Como consecuencia, se promueve una mayor penetración de energías renovables en la matriz energética, reduciendo la dependencia de fuentes fósiles contaminantes y ayudando a cumplir con los objetivos de sostenibilidad ambiental y energética.

Además, al centrarse en horizontes temporales relevantes para el mercado intradiario, el proyecto aporta herramientas útiles para la toma de decisiones en tiempo real, apoyando la estabilidad y flexibilidad del sistema eléctrico. Esto contribuye a hacer más accesible y asequible la energía renovable, alineándose con la meta de que todas las personas tengan acceso a fuentes energéticas modernas y sostenibles.

6.1.2 ODS 13: Acción por el Clima

El cambio climático representa una amenaza global urgente, que exige respuestas inmediatas y efectivas para limitar el calentamiento global y mitigar sus impactos. El sector energético, responsable de una gran parte de las emisiones de gases de efecto invernadero, debe liderar la transición hacia un modelo bajo en carbono.

Este TFM contribuye a la acción climática al perfeccionar las técnicas de predicción de generación eólica, facilitando una integración más eficiente de esta energía limpia en las redes eléctricas. La precisión en los pronósticos energéticos permite reducir la necesidad de recurrir a fuentes convencionales fósiles y contaminantes, disminuyendo así las emisiones de gases nocivos para el clima.

Por otro lado, la mejora en la predicción también fortalece la resiliencia del sistema eléctrico frente a eventos meteorológicos extremos y a la alta variabilidad inherente a las energías renovables, aspectos que se vuelven cada vez más críticos en un escenario de cambio climático. Con ello, el proyecto no solo apoya la mitigación sino también la adaptación, promoviendo un sistema energético más robusto y preparado para enfrentar futuros desafíos climáticos.

6.1.3 ODS 9: Industria, Innovación e Infraestructura

La innovación tecnológica y el desarrollo de infraestructuras resilientes son fundamentales para construir economías sostenibles e inclusivas. Este TFM encarna estos principios al aplicar modelos de Deep Learning de última generación para abordar un problema complejo y real: la predicción de la generación eléctrica en parques eólicos marinos.

El trabajo no solo implementa y evalúa modelos tradicionales y modernos, sino que también introduce nuevas arquitecturas basadas en Transformers y Kolmogorov–Arnold Networks, demostrando cómo la innovación en el análisis de series temporales puede superar las limitaciones de los enfoques anteriores. Esto representa un avance relevante en el campo de la energía renovable, con implicaciones directas para la modernización de la infraestructura energética.

Además, el desarrollo de un repositorio de código abierto, documentado y accesible, fomenta la reproducibilidad y extensión de los resultados, potenciando la colaboración y el desarrollo tecnológico en la comunidad científica y profesional. Así, el proyecto contribuye a fortalecer la industria energética mediante soluciones más inteligentes, eficientes y sostenibles, impulsando la industrialización inclusiva y el crecimiento económico verde.

Capítulo 7

Conclusiones y líneas de trabajo futuras

7.1 Conclusiones del benchmarking de los modelos

El presente trabajo ha llevado a cabo un análisis comparativo sistemático de diversos modelos de predicción aplicados a la generación eléctrica en parques eólicos, empleando dos conjuntos de datos reales: AV y KaggleWPGD. La evaluación se ha centrado en métricas de precisión (MAE), capacidad de generalización, eficiencia computacional e interpretabilidad.

En primer lugar, los modelos tradicionales como ARIMA y XGBoost han mostrado un rendimiento competitivo en escenarios con alta estacionalidad y comportamiento lineal, especialmente sobre el conjunto AV. XGBoost, en particular, ha destacado por su eficiencia computacional y su buena adaptación a problemas tabulares multivariantes, a pesar de no estar específicamente diseñado para series temporales

Por otro lado, las arquitecturas basadas en Deep Learning han superado sistemáticamente a los modelos clásicos en tareas que requieren capturar patrones complejos y relaciones no lineales. Entre ellas, los modelos LSTM y GRU han ofrecido una base sólida, si bien han sido superados por arquitecturas más recientes en términos de rendimiento y escalabilidad.

Las arquitecturas tipo Transformer, como PatchTST y TimeXer, han mostrado una gran versatilidad y estabilidad, así como una elevada capacidad para trabajar con secuencias largas y múltiples horizontes temporales, aunque con un mayor coste computacional. Destaca también su capacidad de transferencia entre tareas, lo cual es prometedor para entornos industriales dinámicos

Asimismo, los modelos basados en Kolmogorov–Arnold Networks (KAN) y su variante RMoK han emergido como una alternativa destacada por su balance entre precisión, interpretabilidad y flexibilidad. En particular, KAN ha demostrado un buen ajuste con menor complejidad arquitectónica, y RMoK ha logrado representar relaciones altamente no lineales con un enfoque modular e interpretable

Cabe destacar que, aunque los modelos más avanzados han presentado mejores resultados en términos de precisión (especialmente en el dataset Kaggle, más normalizado y complejo), su uso en entornos productivos requiere considerar también aspectos como la eficiencia en el entrenamiento y la facilidad de despliegue.

En conjunto, los resultados del benchmarking respaldan la adopción de modelos de Deep Learning —y en particular arquitecturas como KAN y Transformer— para la predicción de generación eólica a corto plazo. No obstante, se concluye que la elección óptima depende del contexto específico: la disponibilidad de datos, los requerimientos de explicabilidad, y las limitaciones operativas deben formar parte del proceso de decisión.

7.2 Análisis del rendimiento por modelo (dataset AV)

Modelos tradicionales y baselines

- XGBoost: mantiene un rendimiento aceptable en los tres horizontes (**320.9** → **442.75**), aunque su error crece significativamente con el horizonte. Esto es esperable dado que no captura dependencias temporales de largo plazo de forma nativa.
- LSTM: pese a ser una red recurrente, obtiene resultados notablemente peores en todos los casos. Su deterioro a 3h (**736.11** MAE) muestra problemas para modelar secuencias largas y quizás dificultades en la optimización de hiperparámetros o en la arquitectura usada.

Transformers

- PatchTST: es el modelo más preciso a 1h (**236.98** MAE), superando a todos los demás, incluida TiDE y RMoK. Su rendimiento se degrada al aumentar el horizonte, pero sigue siendo competitivo.
- TimeXer: sigue una tendencia similar a PatchTST, aunque ligeramente peor. Probablemente su diseño sea menos eficiente para la captura de patrones locales relevantes en este dominio.

Modelos MLP recientes

- NBEATS: es el modelo con peor escalabilidad en el horizonte: su MAE se incrementa drásticamente hasta **901.91** a 3h, lo que puede deberse a sobreajuste o limitaciones estructurales para tareas con fuerte estacionalidad y exógenas dinámicas.
- TiDE: obtiene el mejor resultado a 2h (**377.24** MAE) y mantiene su rendimiento incluso a 3h (**486.45** MAE).
- RMoK: es el más equilibrado de todos, con una progresión de error muy suave entre los tres horizontes (**239.57** → **484.76**). Esta estabilidad sugiere una buena capacidad de generalización y robustez.

A continuación la tabla comparativa de los resultados:

AV	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	320.90	530.91	236.98	263.77	394.64	236.97	239.57
2h	384.08	634.94	375.87	411,96	716.48	377.24	378.78
3h	442.75	736.11	487.08	518.84	901.91	486.45	484.76

Figura 7.1: Comparativa y análisis de aplicabilidad

7.3 Análisis del rendimiento por modelo (dataset Kaggle)

Modelos tradicionales y baselines

- XGBoost: muestra el rendimiento más bajo del conjunto, especialmente a medida que el horizonte de predicción se amplía (MAE sube de **0.0280** → **0.0664**). Confirma su limitación en la captura de dinámicas temporales complejas.
- LSTM: aunque mejora significativamente respecto al dataset AV, sigue sin estar a la altura de los modelos más modernos. Aunque empieza con un MAE aceptable a 1h (0.0170), su error crece considerablemente a 3h (**0.0266**), lo que evidencia limitaciones estructurales frente a horizontes largos.

Transformers

- PatchTST: presenta una progresión muy similar, y es el segundo mejor modelo en todos los horizontes, lo que confirma su estabilidad y eficacia general.
- TimeXer: obtiene el mejor MAE absoluto a 1h (**0.0034**) y es también competitivo en 2h y 3h, mostrando una excelente capacidad para capturar relaciones temporales incluso con poco ruido.

Modelos MLP recientes

- NBEATS: al igual que en AV, muestra un patrón de degradación con el horizonte (de **0.0067** → **0.0467**). Esto sugiere que su arquitectura no se adapta bien a predicciones más allá del corto plazo en este conjunto.
- TiDE: mantiene un buen equilibrio en todos los horizontes (p. ej., **0.0157** MAE a 2h), aunque ligeramente inferior a los transformers.
- RMoK: mejora claramente respecto a LSTM y XGBoost, y obtiene resultados similares a TiDE, confirmando su robustez y potencial en problemas multivariantes complejos.

A continuación la tabla comparativa de los resultados:

Kaggle	XGBoost	LSTM	Patch_TST	TimeXer	Nbeats	TiDE	RMoK
1h	0.0280	0.0170	0.0035	0.0034	0.0067	0.0053	0.0041
2h	0.0478	0.0236	0.0117	0.0113	0.0229	0.0157	0.0129
3h	0.0664	0.0266	0.0242	0.0241	0.0467	0.0290	0.0262

Figura 7.2: Comparativa y análisis de aplicabilidad

7.4 Líneas de trabajo futuras

Este trabajo fin de máster ha abordado el problema de la predicción de la generación eléctrica en parques eólicos a partir de datos meteorológicos utilizando modelos de series temporales. En él se ha llevado a cabo un análisis comparativo entre modelos tradicionales y técnicas basadas en Deep Learning, con especial énfasis en la capacidad de generalización, la precisión predictiva y la interpretabilidad de los resultados.

Desde un enfoque metodológico, se ha seguido una estrategia sistemática que ha incluido la preparación y exploración del conjunto de datos, la aplicación de modelos clásicos de Machine Learning como ARIMA, y el desarrollo e implementación de arquitecturas neuronales más avanzadas. Entre estas últimas, se han analizado y comparado modelos recurrentes (LSTM), modelos basados en atención (Transformers), y nuevas propuestas como KAN (Kolmogorov-Arnold Networks) y RMoK (Rank-based Mixture of Kernels), que buscan superar algunas de las limitaciones estructurales de las arquitecturas convencionales.

Los resultados obtenidos han mostrado que, si bien los modelos tradicionales continúan siendo competitivos en contextos con fuerte estacionalidad y comportamiento lineal, los modelos basados en aprendizaje profundo ofrecen ventajas sustanciales cuando se trata de capturar dinámicas no lineales, relaciones de largo plazo y patrones multivariantes. En particular, las arquitecturas como KAN han demostrado una buena capacidad de ajuste con una estructura más interpretable, mientras que los modelos tipo Transformer muestran un potencial elevado en cuanto a escalabilidad y transferencia entre tareas.

Además del rendimiento predictivo, el estudio ha puesto de manifiesto la importancia de considerar la interpretabilidad y la eficiencia computacional como factores clave a la hora de seleccionar una arquitectura para su uso en entornos industriales. Este equilibrio entre precisión, explicabilidad y coste computacional es especialmente relevante en aplicaciones críticas como la planificación energética o la operación de redes eléctricas con alta penetración de renovables.

En conjunto, los hallazgos del trabajo refuerzan la idea de que la combinación de datos meteorológicos con arquitecturas avanzadas de modelado puede constituir una herramienta eficaz para mejorar la gestión y previsión de recursos energéticos variables como la eólica. Sin embargo, aún existen importantes retos abiertos, lo que da lugar a varias líneas de trabajo futuras.

7.4.1 Modificación de aquitecturas

Una línea prometedora para futuras investigaciones consiste en el rediseño y la mejora de las arquitecturas utilizadas, con el objetivo de aumentar su precisión y adaptabilidad. En este sentido, sería interesante explorar variantes híbridas que combinen componentes recurrentes, convolucionales y de atención en una única arquitectura, permitiendo captar tanto dependencias locales como globales en la serie temporal.

Además, se podría avanzar en el ajuste automático de hiperparámetros mediante técnicas de optimización bayesiana o algoritmos genéticos, con el fin de reducir el tiempo de experimentación y mejorar la robustez de los modelos. También se plantea como relevante la aplicación de regularización estructural o dropout adaptativo para evitar el sobreajuste, especialmente en modelos complejos y con datos limitados.

Otra línea de trabajo consiste en profundizar en la interpretabilidad de las predicciones. La incorporación de técnicas explicativas como SHAP (SHapley Additive exPlanations) o Integrated Gradients puede aportar información valiosa sobre la relevancia de cada variable meteorológica en la predicción final, lo cual es esencial en aplicaciones industriales donde se requiere justificar decisiones operativas.

7.4.2 Modelos Fundacionales

En los últimos años, el campo del aprendizaje automático ha sido testigo de un cambio de paradigma propiciado por la aparición de los llamados modelos fundacionales (foundation models). Estos modelos, originalmente concebidos para el procesamiento del lenguaje natural (NLP), están diseñados

para ser entrenados sobre volúmenes masivos de datos heterogéneos, lo que les permite aprender representaciones profundas y generalizables de los datos. Posteriormente, pueden aplicarse a tareas específicas mediante técnicas como el fine-tuning, prompting o zero-shot learning, sin necesidad de construir modelos desde cero para cada caso de uso.

Según el informe fundacional del Stanford Institute for Human-Centered AI (2021), un modelo fundacional se caracteriza por:

- Entrenamiento en conjuntos de datos de escala masiva y diversidad significativa.
- Uso de arquitecturas avanzadas y escalables (usualmente Transformer).
- Aplicabilidad transversal a múltiples tareas sin rediseño completo.
- Capacidad de adaptación a contextos específicos mediante ajustes finos o prompt engineering.

Si bien esta evolución fue liderada inicialmente por modelos como BERT, GPT o PaLM en el ámbito del texto, su lógica ha comenzado a trasladarse al terreno de las series temporales, donde las empresas e instituciones requieren soluciones capaces de escalar entre dominios, reducir los costes de mantenimiento de modelos personalizados y facilitar la adopción de técnicas de inteligencia artificial por equipos no expertos.

Tradicionalmente, el modelado de series temporales se ha basado en arquitecturas específicas y ajustadas caso por caso, como ARIMA, LSTM o modelos estadísticos híbridos. Sin embargo, estas soluciones presentan desafíos en términos de escalabilidad, portabilidad entre dominios, y mantenimiento en entornos industriales con cientos o miles de activos distribuidos (por ejemplo, turbinas eólicas).

Los modelos fundacionales aplicados a series temporales intentan resolver estas limitaciones mediante un enfoque generalista: entrenar un modelo único sobre grandes bases de datos temporales (de múltiples frecuencias, dominios y geografías) que sea capaz de realizar tareas como predicción, clasificación o detección de anomalías de forma generalizada.

En el ámbito energético, y en particular en la predicción de generación eólica, esta aproximación resulta especialmente prometedora por varias razones:

- Variabilidad estacional y geográfica: Los patrones de generación varían entre parques, regiones y estaciones, lo que requiere modelos flexibles.
- Escasez de datos históricos en nuevos emplazamientos: Los modelos fundacionales pueden compensar la falta de datos específicos mediante conocimiento aprendido de otros dominios.
- Necesidad de mantenimiento automatizado: En grandes operadores energéticos, mantener modelos individuales por activo es ineficiente; un enfoque fundacional permite estandarizar el pipeline de predicción.

La irrupción reciente de modelos fundacionales de propósito general para series temporales ofrece una nueva perspectiva en el campo del forecasting. Estos modelos se entrenan sobre conjuntos masivos y heterogéneos de datos temporales, aprendiendo representaciones universales que pueden reutilizarse y afinarse en tareas concretas sin necesidad de reentrenamiento completo. En el contexto energético, su aplicación podría suponer un cambio de paradigma en la forma de abordar problemas predictivos.

A continuación se presentan tres de los modelos fundacionales más relevantes actualmente en desarrollo o producción para el modelado de series temporales, con énfasis en sus características técnicas

y potencial aplicabilidad en generación eólica.

- **Chronos (AWS)**: [66]

Desarrollado por Amazon Web Services, Chronos constituye un marco extensible para la construcción de modelos fundacionales aplicables a tareas de predicción y análisis de series temporales. Su diseño se basa en arquitecturas tipo Transformer adaptadas específicamente para datos secuenciales. Chronos ha sido entrenado sobre una gran variedad de datos procedentes de sectores como retail, salud, manufactura o energía. Características clave:

- Arquitectura modular y compatible con los servicios de cloud de AWS.
- Soporte para tareas múltiples (multi-task learning): forecasting, clasificación, detección de anomalías.
- Optimización para inferencia eficiente en entornos industriales.
- Capacidad de adaptación a diferentes horizontes temporales y granularidades.

Para aplicaciones como la predicción de energía eólica, Chronos facilita la integración de modelos predictivos en pipelines operativos (por ejemplo, en Amazon SageMaker), lo que permite realizar despliegues automáticos y escalar sin fricciones.

Este Trabajo de Fin de Máster propone explorar Chronos (AWS), un framework de modelado de series temporales desarrollado por Amazon, que aborda precisamente estos retos. Chronos permite entrenar y desplegar modelos a gran escala y está diseñado para adaptarse a diferentes tareas, lo que lo convierte en una solución integral para empresas que requieren respuestas ágiles y robustas basadas en datos temporales. Su consolidación en sectores como el retail y las operaciones industriales lo posiciona como una herramienta de alto potencial estratégico.

- **TimeGPT** (Nixtla):

TimeGPT es un modelo autoregresivo de propósito general desarrollado por la startup Nixtla, especializada en forecasting. A diferencia de Chronos, TimeGPT ofrece una interfaz directa en forma de API-as-a-Service, lo que permite a los usuarios realizar predicciones sin necesidad de entrenar un modelo localmente.

Características clave:

- Entrenado sobre más de 100 millones de series temporales de múltiples dominios.
- Capacidad de inferencia zero-shot, sin necesidad de reentrenamiento.
- Exposición mediante prompt en lenguaje natural o input estructurado.
- Resultados competitivos frente a modelos entrenados manualmente.

Su mayor valor reside en la democratización del forecasting avanzado: permite obtener predicciones precisas en contextos donde los recursos técnicos o computacionales son limitados. Para empresas energéticas, esto implica poder realizar pruebas de concepto rápidas, analizar la generación de nuevos parques con pocos datos históricos, o complementar modelos existentes.

- **TimesFM**

TimesFM (Time Series Foundation Model) es la apuesta de Google DeepMind por construir un modelo universal para el aprendizaje de series temporales, siguiendo la misma lógica que los modelos fundacionales en NLP.

Características clave:

- Enfoque preentrenado con auto-supervisión (similar a BERT o PaLM).
- Entrenado sobre datos temporales de distintos dominios, frecuencias y países.
- Capacidad multitarea: forecasting, imputación de datos, clasificación.
- Arquitectura basada en Transformer eficiente para series largas.

A diferencia de TimeGPT, TimesFM está aún en fase de investigación, pero su diseño apunta a convertirse en un modelo universal para tareas temporales. Su principal atractivo es la capacidad de adaptarse a múltiples tareas con un solo modelo base, reduciendo la complejidad del stack técnico en organizaciones que necesitan soluciones heterogéneas.

Modelo	Arquitectura	Acceso	Escalabilidad	Tareas soportadas	Adaptabilidad industrial
Chronos	Transformer modular	SDK / AWS	Alta	Forecasting, clasificación, anomalías	Alta (nativa en cloud)
TimeGPT	Autoregresiva	API pública	Media	Forecasting (*zero-shot*)	Media/Alta (plug-and-play)
TimesFM	Transformer universal	Investigación	Alta	Multitarea	Alta (en futuro cercano)

Figura 7.3: Comparativa y análisis de aplicabilidad

Capítulo 8

Referencias

[1] L. H. Hao Wang Jingzhen Ye, «A multivariable hybrid prediction model of offshore wind power based on multi-stage optimization and reconstruction prediction». 2023. Disponible en: https://www.sciencedirect.com/science/article/abs/pii/S0360544222023106

[2] C. de los Santos Jiménez, «Análisis comparativo de modelos de aprendizaje automático para la predicción de la generación eléctrica de un parque eólico marino». Universidad Pontificia Comillas, 2023. Disponible en: https://repositorio.comillas.edu/xmlui/handle/11531/83380

[3] M. Rahim, «Wind Power Generation Data – Forecasting». Kaggle dataset, 2024. Disponible en: https://www.kaggle.com/datasets/mubashirrahim/wind-power-generation-data-forecasting

[4] Z. Lin y X. Liu, «Wind Power Forecasting of an Offshore Wind Turbine Based on High-Frequency SCADA Data and Deep Learning Neural Network», *Energy*, vol. 201, p. 117693, 2020, doi: 10.1016/j.energy.2020.117693.

[5] Z. Liu *et al.*, «KAN: Kolmogorov-Arnold Networks», *arXiv preprint arXiv:2404.19756*, 2024, Disponible en: https://arxiv.org/abs/2404.19756

[6] S. Hanifi, H. Zare-Behtash, A. Cammarano, y S. Lotfian, «Offshore Wind Power Forecasting Based on WPD and Optimised Deep Learning Methods», *Renewable Energy*, vol. 218, p. 119241, 2023, doi: 10.1016/j.renene.2023.119241.

[7] A. Vaswani *et al.*, «Attention Is All You Need», en *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017, pp. 5998-6008. Disponible en: https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[8] Y. Nie, N. H. Nguyen, P. Sinthong, y J. Kalagnanam, «A Time Series is Worth 64 Words: Long-Term Forecasting with Transformers». 2023. Disponible en: https://arxiv.org/pdf/2211.14730

[9] S. Makridakis, E. Spiliotis, R. Hollyman, F. Petropoulos, N. Swanson, y A. Gaba, «The M6 forecasting competition: Bridging the gap between forecasting and investment decisions», *arXiv preprint arXiv:2310.13357*, 2023, Disponible en: https://arxiv.org/abs/2310.13357

[10] R. J. Hyndman y G. Athanasopoulos, *Forecasting: Principles and Practice*. OTexts, 2021. Disponible en: https://otexts.com/fpp3/

[11] F. Pedregosa *et al.*, «Scikit-learn: Machine Learning in Python», *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011, Disponible en: https://jmlr.org/papers/v12/pedregosa11a.html

[12] W. McKinney, «Data Structures for Statistical Computing in Python», *Proceedings of the 9th Python in Science Conference*, pp. 56-61, 2010, Disponible en: https://doi.org/10.25080/Majora-92bf1922-00a

[13] C. R. Harris *et al.*, «Array programming with NumPy», *Nature*, vol. 585, pp. 357-362, 2020, doi: 10.1038/s41586-020-2649-2.

[14] J. D. Hunter, «Matplotlib: A 2D Graphics Environment», *Computing in Science & Engineering*, vol. 9, n.º 3, pp. 90-95, 2007, doi: 10.1109/MCSE.2007.55.

[15] M. L. Waskom, «Seaborn: Statistical Data Visualization», *Journal of Open Source Software*, vol. 6, n.º 60, p. 3021, 2021, doi: 10.21105/joss.03021.

[16] S. Seabold y J. Perktold, «Statsmodels: Econometric and Statistical Modeling with Python», en *Proceedings of the 9th Python in Science Conference (SciPy)*, 2010, pp. 92-96. Disponible en: https://conference.scipy.org/proceedings/scipy2010/pdfs/seabold.pdf

[17] I. Goodfellow, Y. Bengio, y A. Courville, *Deep Learning*. Cambridge, MA: MIT Press, 2016. Disponible en: https://www.deeplearningbook.org/

[18] F. Chollet, «Keras». 2015. Disponible en: https://keras.io

[19] M. Abadi *et al.*, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems». 2015. Disponible en: https://www.tensorflow.org/

[20] F. Chollet, *Deep Learning with Python*, 3rd ed. Shelter Island, NY: Manning Publications, 2024. Disponible en: https://www.manning.com/books/deep-learning-with-python-third-edition

[21] A. Paszke *et al.*, «PyTorch: An Imperative Style, High-Performance Deep Learning Library», en *Advances in Neural Information Processing Systems*, 2019, pp. 8024-8035. Disponible en: https://papers.nips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[22] T. Wolf *et al.*, «Transformers: State-of-the-Art Natural Language Processing», en *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Association for Computational Linguistics, 2020, pp. 38-45. Disponible en: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[23] S. Raschka, Y. (Hayden) Liu, y V. Mirjalili, *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python*. Packt Publishing, 2022. Disponible en: https://www.packtpub.com/product/machine-learning-with-pytorch-and-scikit-learn/9781801819312

[24] V. Cerqueira y L. Roque, *Deep Learning for Time Series Cookbook: Use PyTorch and Python Recipes for Forecasting, Classification, and Anomaly Detection*. Packt Publishing, 2024. Disponible en: https://www.packtpub.com/product/deep-learning-for-time-series-cookbook/9781805129233

[25] J. Bezanson, S. Karpinski, V. B. Shah, y A. Edelman, «Julia: A Fast Dynamic Language for Technical Computing», en *Proceedings of the IEEE High Performance Extreme Computing Conference (HPEC)*, 2012, pp. 1-6. doi: 10.1109/HPEC.2012.6396530.

[26] J. D. Smith y Contributors, «pmdarima: ARIMA estimators for Python». https://github.com/alkaline-ml/pmdarima, 2023.

[27] M. Löning *et al.*, «sktime: A Unified Interface for Machine Learning with Time Series», *arXiv preprint*, 2019, Disponible en: https://arxiv.org/abs/1909.07872

[28] R. J. Hyndman, G. Athanasopoulos, A. Garza, C. Challu, M. Mergenthaler Canseco, y K. G. Olivares, «Forecasting: Principles and Practice, the Pythonic Way». [En línea]. Disponible en: https://otexts.com/fppy/

[29] N. Developers, «Nixtla: Machine Learning and Statistical Methods for Time Series Forecasting». https://nixtla.github.io/, 2023.

[30] G. James, D. Witten, T. Hastie, R. Tibshirani, y J. Taylor, *An Introduction to Statistical Learning with Applications in Python*. Springer International Publishing, 2023. doi: 10.1007/978-3-031-38747-0.

[31] L. Owen, *Hyperparameter Tuning with Python*. Packt Publishing, 2022. Disponible en: https://www.packtpub.com/product/hyperparameter-tuning-with-python/9781803235875

[32] T. Akiba, S. Sano, T. Yanase, T. Ohta, y M. Koyama, «Optuna: A Next-generation Hyperparameter Optimization Framework», *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623-2631, 2019, doi: 10.1145/3292500.3330701.

[33] R. J. Hyndman y Y. Khandakar, «Automatic Time Series Forecasting: The Forecast Package for R», *Journal of Statistical Software*, vol. 27, n.º 3, pp. 1-22, 2008, doi: 10.18637/jss.v027.i03.

[34] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, y G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ: John Wiley & Sons, 2015. Disponible en: https://www.wiley.com/en-us/Time+Series+Analysis%3A+Forecasting+and+Control%2C+5th+Edition-p-9781118675021

[35] Y. Freund y R. E. Schapire, «A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting», en *Proceedings of the Second European Conference on Computational Learning Theory*, Springer-Verlag, 1995, pp. 23-37. Disponible en: https://link.springer.com/chapter/10.1007/3-540-59119-2_166

[36] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 3rd ed. Sebastopol, CA: O'Reilly Media, 2022. Disponible en: https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/

[37] J. H. Friedman, «Greedy Function Approximation: A Gradient Boosting Machine», *Annals of Statistics*, vol. 29, n.º 5, pp. 1189-1232, 2001, doi: 10.1214/aos/1013203451.

[38] T. Chen y C. Guestrin, «XGBoost: A Scalable Tree Boosting System», en *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, ACM, 2016, pp. 785-794. doi: 10.1145/2939672.2939785.

[39] T. Chen y Contributors, «XGBoost: Scalable and Flexible Gradient Boosting». https://github.com/dmlc/xgboost, 2024.

[40] S. Hochreiter y J. Schmidhuber, «Long short-term memory», *Neural computation*, vol. 9, n.º 8, pp. 1735-1780, 1997.

[41] K. Cho *et al.*, «Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation», en *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1724-1734. doi: 10.3115/v1/D14-1179.

[42] J. Alammar y M. Grootendorst, *Hands-On Large Language Models: Language Understanding and Generation*. O'Reilly Media, 2024. Disponible en: https://www.oreilly.com/library/view/hands-on-large-language/9781098150952/

[43] I. Sutskever, O. Vinyals, y Q. V. Le, «Sequence to Sequence Learning with Neural Networks». 2014. Disponible en: https://arxiv.org/abs/1409.3215

[44] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition», *arXiv preprint arXiv:1512.03385*, 2015, Disponible en: https://arxiv.org/abs/1512.03385

[45] A. S. Glassner, *Deep Learning: A Visual Approach*. San Francisco, CA: No Starch Press, 2021. Disponible en: https://nostarch.com/deep-learning-visual-approach

[46] S. Li *et al.*, «Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting». 2020. Disponible en: https://arxiv.org/abs/1907.00235

[47] I. Godfried, K. Mahajan, M. Wang, K. Li, y P. Tiwari, «FlowDB a large scale precipitation, river, and flash flood dataset». 2020. Disponible en: https://arxiv.org/abs/2012.11154

[48] K. Choi, J. Yi, C. Park, y S. Yoon, «Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines», *IEEE Access*, vol. 9, pp. 120043-120065, 2021, doi: 10.1109/ACCESS.2021.3107512.

[49] N. M. Foumani, L. Miller, C. W. Tan, G. I. Webb, G. Forestier, y M. Salehi, «Deep Learning for Time Series Classification and Extrinsic Regression: A Current Survey», *ACM Comput. Surv.*, 2024, doi: 10.1145/3649448.

[50] N. M. Foumani, C. W. Tan, G. I. Webb, y M. Salehi, «Improving Position Encoding of Transformers for Multivariate Time Series Classification», *Data Mining and Knowledge Discovery*, vol. 38, n.º 1, pp. 22-48, 2024, doi: 10.1007/s10618-023-00948-2.

[51] X. Kong *et al.*, «Deep Learning for Time Series Forecasting: A Survey», *International Journal of Machine Learning and Cybernetics*, 2025, doi: 10.1007/s13042-025-02560-w.

[52] H. Zhou *et al.*, «Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting». 2021. Disponible en: https://arxiv.org/abs/2012.07436

[53] S. Liu *et al.*, «Pyraformer: Low-Complexity Pyramidal Attention for Long-Range Time Series Modeling and Forecasting», en *Proceedings of the 10th International Conference on Learning Representations (ICLR)*, 2022. Disponible en: https://openreview.net/forum?id=0EXmFzUn5I

[54] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, y R. Jin, «FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting». 2022. Disponible en: https://arxiv.org/abs/2201.12740

[55] A. Zeng, M. Chen, L. Zhang, y Q. Xu, «Are Transformers Effective for Time Series Forecasting?» 2022. Disponible en: https://arxiv.org/abs/2205.13504

[56] H. Face, «Autoformer: Decomposition Architecture for Long-term Series Forecasting». https://huggingface.co/blog/autoformer, 2022.

[57] H. Wu, J. Xu, J. Wang, y M. Long, «Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting». 2022. Disponible en: https://arxiv.org/abs/2106.13008

[58] Y. Wang *et al.*, «TimeXer: Empowering Transformers for Time Series Forecasting with Exogenous Variables». 2024. Disponible en: https://arxiv.org/abs/2402.19072

[59] B. N. Oreshkin, D. Carpov, N. Chapados, y Y. Bengio, «N-BEATS: Neural Basis Expansion Analysis for Interpretable Time Series Forecasting», *arXiv preprint arXiv:1905.10437*, 2019, Disponible en: https://arxiv.org/abs/1905.10437

[60] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza, M. Mergenthaler-Canseco, y A. Dubrawski, «N-HiTS: Neural Hierarchical Interpolation for Time Series Forecasting», *arXiv preprint arXiv:2201.12886*, 2022, Disponible en: https://arxiv.org/abs/2201.12886

[61] C. Challu, K. G. Olivares, B. N. Oreshkin, F. Garza Ramirez, M. Mergenthaler Canseco, y A. Dubrawski, «NHITS: Neural Hierarchical Interpolation for Time Series Forecasting», en *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, pp. 6989-6997. doi: 10.1609/aaai.v37i6.25854.

[62] W. Kong, A. Das, A. Leach, R. Sen, y R. Yu, «Long-term Forecasting with TiDE: Time-series Dense Encoder», *arXiv preprint arXiv:2304.08424*, 2023, Disponible en: https://arxiv.org/abs/2304.08424

[63] M. Peixeiro, «Time Series Forecasting with TiDE». enero de 2024. Disponible en: https://www.datasciencewithmarco.com/blog/time-series-forecasting-with-tide

[64] X. Han, X. Zhang, Y. Wu, Z. Zhang, y Z. Wu, «Are KANs Effective for Multivariate Time Series Forecasting?» 2025. Disponible en: https://arxiv.org/abs/2408.11306

[65] X. Han, X. Zhang, Y. Wu, Z. Zhang, y Z. Wu, «KAN4TSF: Are KAN and KAN-based models Effective for Time Series Forecasting?», *arXiv preprint arXiv:2408.11306*. 2024. Disponible en: https://arxiv.org/html/2408.11306v1

[66] A. F. Ansari *et al.*, «Chronos: Learning the Language of Time Series», *arXiv preprint arXiv:2403.07815*, 2024, Disponible en: https://arxiv.org/abs/2403.07815