COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MASTER'S IN INDUSTRIAL ENGINEERING
AND SMART INDUSTRY

MASTER'S THESIS

FORECASTING ELECTRICITY PRICES WITH NEURAL ODE

Author: Sanz Muñoz, Jaime

Director: Muñoz San Roque, Antonio

Co-Director: Malpica Morales, Antonio

Madrid

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

Forecasting electricity prices with Neural ODE

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022/23 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.

[illegible]

Fdo.: Jaime Sanz Muñoz Fecha: 20/08/2023

Autorizada la entrega del proyecto

[illegible]

Fdo.: Antonio Muñoz San Roque Fecha: 23 / 08 / 2023

[illegible]

Fdo.: Antonio Malpica Morales Fecha: 23 / 08 / 2023

MÁSTER EN TECNOLOGÍAS INDUSTRIALES E INDUSTRIA CONECTADA

TRABAJO FIN DE MÁSTER

PREDICCIÓN DE PRECIOS DE LA ELECTRICIDAD CON NEURAL ODE

Autor: Sanz Muñoz, Jaime

Director: Muñoz San Roque, Antonio

Co-Director: Malpica Morales, Antonio

Madrid

FORECASTING ELECTRICITY PRICES WITH NEURAL ODE

Author: Sanz Muñoz, Jaime
Supervisors: Muñoz San Roque, Antonio
Malpica Morales, Antonio
Collaborating Entity: ICAI – Universidad Pontificia Comillas

ABSTRACT

This master's thesis develops and evaluates neural Ordinary Differential Equations (ODEs) for short-term electricity price forecasting (EPF) in the 2019 Spanish day-ahead market. The neural ODEs are implemented in PyTorch and compared against benchmark models including SARIMA, Facebook Prophet, MLP, LSTM and CNN-LSTM. The univariate neural ODE achieved the lowest average error, outperforming statistical and machine learning (ML) methods. However, the multivariate Prophet provided the best performance, making the most of the exogenous variables. While computationally expensive, neural ODEs show promising results.

Keywords: Forecasting; Electricity price; ODE; Day-ahead market

1. Introduction

EPF plays a critical role in the effective functioning and stability of energy markets. Having accurate price forecasts enables utility companies to optimize generation schedules, traders to maximize profits and consumers to plan usage [1]. Nevertheless, electricity prices show substantial volatility due to fluctuations in demand, supply, and other exogenous factors like the weather, making accurate EPF challenging.

Over the years, a variety of statistical and ML techniques have been applied to EPF, including autoregressive integrated moving average (ARIMA) models, support vector machines (SVM) and artificial neural networks [2][3]. As these methods evolved, the hybrid approach was introduced, combining strengths of both worlds to offer a robust way to deal with the increasing price volatility. Recently, a new class of deep learning model called neural ODEs has shown promising results for learning complex continuous-time dynamics [4].

This paper provides the first in-depth implementation of neural ODEs for day-ahead EPF, more specifically in the Spanish electricity market. Both univariate and multivariate approaches are developed. The univariate only uses historical price data, whereas the multivariate approach includes three additional exogenous variables: the P48 power demand and wind production (obtained from the system operator OMIE), and the day of the week.

The primary objectives of this master's thesis start with conducting an extensive review of the current research and state-of-the-art methods in EPF, as new data and algorithms are rapidly emerging in this evolving field. Moreover, a series of benchmark models including the statistical, probabilistic and ML models will be implemented to provide a baseline for the posterior analysis of results. The neural ODE will then be implemented in PyTorch, and its results will be assessed against the benchmark models using the appropriate error metrics. Finally, the results will be summarized in this academic paper, describing the strengths and weaknesses of neural ODE for EPF. Overall, the outcomes will provide valuable insights for all the market agents and EPF community.

2. Benchmark models

This study uses hourly electricity price data from the day-ahead market of Spain in a period spanning from mid-2018 to late-2019. A consistent dataset division has been selected for the benchmark models, using the previous year of observations as training period, three months as validation set and the following three months for the test set, which will span from August 2019 to October 2019. The five benchmark models that have been developed, for both univariate and multivariate approaches, are described subsequently.

The seasonal ARIMA (SARIMA) model has been selected for its ability to capture seasonal electricity price cycles, focusing for this study on the regular, daily and weekly seasonalities. Implemented in R, the Box-Cox methodology is followed to find and test the best combination of hyperparameters. The model is trained on the preceding 13 weeks of data, with the resulting training residuals analyzed.

Characterized by its flexibility, robustness and computational efficiency, the probabilistic Prophet model has been included in this study. This model has been developed by Facebook to mitigate the frequent lack of resilience of forecasting methods [5]. The Prophet model uses the preceding 6 weeks of data to forecast the following 24-hourly electricity prices.

The Multilayer Perceptron (MLP) models are a feedforward artificial neural networks that can capture complex nonlinear relationships in data through traditional backpropagation and stochastic gradient descent. The model takes the prior 168 prior hourly prices (as in the previous week) as input and predicts the following 24 hours.

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN), which have proven to be very effective in modelling sequential data, given their ability to remember long-term data. They have also been trained using the preceding week of data.

Finally, hybrid models such as the Convolutional Neural Network-LSTM (CNN-LSTM) model, combine the strengths of both techniques, bringing together their abilities to capture temporal and spatial characteristics in the data. Before passing the input data to the LSTM model, a 1D convolutional layer extracts features and patterns, using a sequence of learnable filters, This is especially well-suited for the multivariate approach, as there may be hidden interconnections or recurring patterns that may not be immediately apparent.

The evaluation of the models' performance has been carried out by following the standard mean absolute error (MAE) on the test set forecasts.

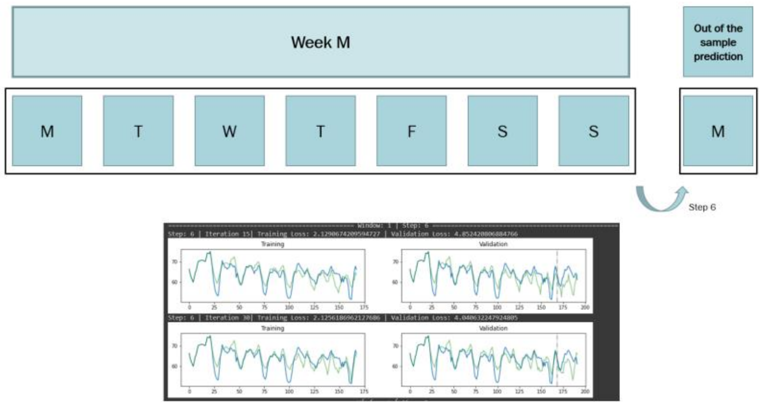
3. Neural ODE

Five years ago, in 2018, Ricky T. Q. Chen and his colleagues published the paper introducing this new family of deep neural network models, neural ODEs. Essentially, they are a continuous version of the RNN models, where the hidden state at a given time is modeled as a function of time, instead of a sequence of discrete time steps [4]. By combining the flexibility of deep neural networks with the mathematical properties of ODEs, Chen and his colleagues demonstrated that this new model could accurately model complex, dynamic systems, outperforming traditional approaches in certain scenarios. Ever since, neural ODEs have been implemented in other fields, achieving state-of-the-art performance in areas like irregular time series and robotics control [6]. However, their application to EPF has been limited and the main goal of this master's thesis is to implement it for the first time on the Spanish day-ahead electricity market.

The innovative idea behind a neural ODE is to avoid the traditional backpropagation. Differentiating through the operations of the forward pass is straightforward, but incurs a high memory cost and introduces additional numerical error [4]. Nevertheless, the ODE solver is represented as a black box and calculates the solution of the initial value problem (IVP), computing the gradients using the so-called *adjoint sensitivity method*. This approach computes the gradients by solving a second, augmented ODE backwards in time, and is applicable to all ODE solvers.

The neural network takes as input the initial condition, i.e., the input at the first time-step, and evolves it over time according to the learned ODE behind the dynamics. The time-steps will correspond to each day of the input week, serving the last one as the out-of-sample prediction. The neural ODE is trained with 5 time-steps, which correspond to the preceding week's price trajectory. Then, the out-of-sample forecast is performed at the sixth time-step.

The ODE solver, and adaptive *dopri5* (Dormand-Prince Runge-Kutta of fifth order), computes an arbitrary number of iterations for each day's data, before advancing to the next time-step for evaluation. The neural ODE validation for the out-of-the-sample forecast is as follows:



```
------------------------------------------ Window: 1 | Step: 6 ------------------------------------------
Step: 6 | Iteration 15| Training Loss: 2.1290674209594727 | Validation Loss: 4.852428806884766
Step: 6 | Iteration 30| Training Loss: 2.1256186962127686 | Validation Loss: 4.040632247924805
```

Figure 1: Evaluation at the end of the last time-step - Out-of-the-sample prediction

Neural ODEs, unlike traditional ML models, require that the dimensionality of input and output match, due to the nature of ODEs. This requirement shapes the model's architecture, including the number of layers and input data preprocessing. For the case of the univariate neural ODE, the initial condition of the IVP matches the desired forecast size, 24. However, the multivariate neural ODE computes and predicts the values for all the three variables (P48 power demand, P48 wind generation and electricity price) flattened values, i.e., 72 values. Nevertheless, to concentrate the computational resources on the primary task of EPF, the loss function is applied only to the last 24 values of the neural network's last layer. The following figure shows the output values of the ODE's solution, having fit a rectangle in the last 24 outputs, which represents the final price forecasts for the following day:

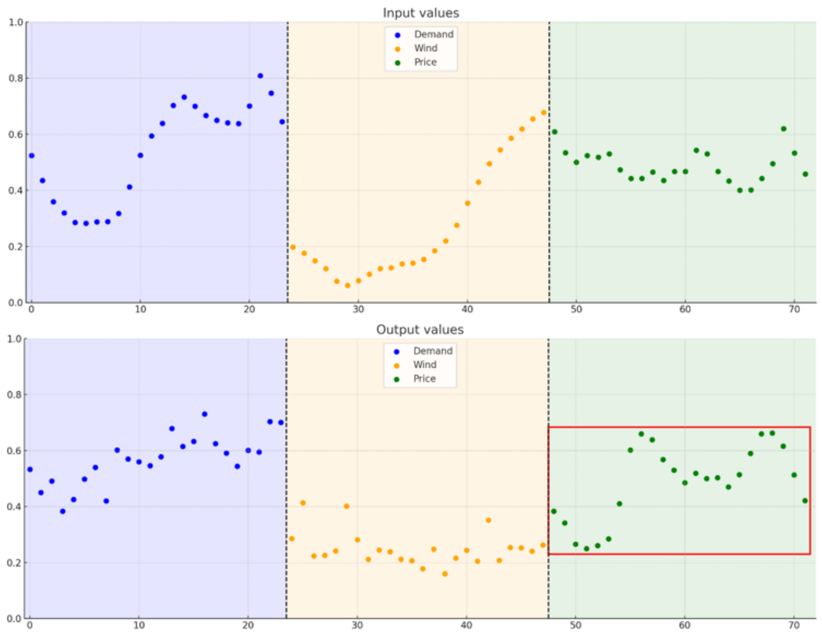


Figure 2: Input and output values of the multivariate neural ODE

4. Final Results

The resulting loss function values of each univariate model for every day of the week are as follows:

Day to forecast	*Naïve*	*SARIMA*	*Prophet*	*MLP*	*LSTM*	*CNN-LSTM*	*Neural ODE*
Monday	8.121	**4.25**	**4.94**	**5.331**	**5.292**	**5.639**	**3.786**
Tuesday	3.935	4.49	4.27	3.941	3.961	4.679	**3.460**
Wednesday	3.352	4.59	3.44	4.322	**3.178**	**3.239**	3.915
Thursday	3.554	3.73	4.09	4.636	4.350	4.503	3.727
Friday	2.768	4.01	3.76	3.418	3.309	3.437	3.613
Saturday	4.665	**4.56**	4.84	**3.909**	**3.908**	**4.057**	**3.381**
Sunday	4.093	6.87	4.68	**3.715**	**3.473**	**3.177**	**3.648**
Average MAE	4.355	4.643	**4.289**	**4.182**	**3.924**	**4.104**	**3.647**

Table 1: Univariate Results

These results highlight the importance of running day-specific models in EPF. This is because different days of the week may have different price dynamics due to external factors like demand patterns, making it suitable to tailor models to specific days. Looking at the average values of the loss function for the different models, the univariate neural ODE outperforms the rest of the benchmark models. Even though the models have run on CPU, they achieve the best results overall. It must be noted that this does not apply for all days but generally, the neural ODE is the model that makes the most out of the electricity price time series. The resulting loss function values of each multivariate model for every day of the week are as follows:

Day to forecast	*Naïve*	*SARIMAX*	*Prophet*	*MLP*	*LSTM*	*CNN-LSTM*	*Neural ODE*
Monday	8.121	**4.51**	**2.45**	**5.213**	**5.756**	**3.804**	**4.358**
Tuesday	3.935	4.89	**2.38**	**3.693**	5.684	3.995	4.792
Wednesday	3.352	5.11	**3.28**	4.076	5.601	3.547	4.090
Thursday	3.554	4.33	**2.86**	4.001	7.084	4.315	3.973
Friday	2.768	4.26	**2.78**	4.173	6.572	3.325	3.844
Saturday	4.665	5.04	**2.34**	**3.410**	**5.653**	**4.022**	**3.799**
Sunday	4.093	5.41	**2.48**	**2.919**	6.090	**2.755**	**3.550**
Average MAE	4.355	4.793	**2.652**	**3.926**	6.063	**3.680**	**4.058**

Table 2: Multivariate Results

These results highlight a key challenge in time-series forecasting: adding more variables does not necessarily improve performance, being the case of the SARIMAX and LSTM. It is crucial to carefully select and preprocess the variables to ensure they allow additional valuable information. Surprisingly enough, the multivariate model with the best overall performance is the Prophet model. With an astonishing overall electricity price MAE of 2.652 €/MWh, it has outperformed the other benchmark models.

5. Conclusions & Future Work

This study delivered a valuable vision of the potential of neural ODEs in the short-term Spanish day-ahead EPF problem. This pioneer application brought unique challenges, including the lack of references and comparisons with similar architectures.

A key strength of neural ODEs that has been demonstrated, is their efficiency in the use of data. In contrast to the other benchmark models, that required large training datasets, neural ODEs demonstrated optimal performance with a significantly smaller datasets size. Consequently, neural ODEs arise as an important option for time-series with limited historical data, allowing a quicker reaction to changing dynamics. This could include scenarios related to rare events like natural disasters and new market product sales.

Surprisingly enough, the Facebook Prophet model emerged as the reference model for multivariate EPF. It showed its strength in handling variability and uncertainty in the electricity prices, manifesting its capability to model the holiday effects. Conversely, the study demonstrated that adding more variables does not always enhance forecasting performance. As is the case of the SARIMAX model, which failed to effectively leverage the additional information provided by the exogenous variables.

Moving forward, future work includes the exploration of neural controlled differential equations for real-world applications [7], focusing on improving their generalization capabilities and robustness to noise. However, due to the number of computations required for neural ODEs, the migration to GPU systems will be necessary for exploring more complex time-series and more accurate ODE solvers.

6. References

[1] J. Lago, G. Marcjasz, B. De Schutter, and R. Weron, "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark," *Appl. Energy*, vol. 293, no. December 2020, p. 116983, 2021, doi: 10.1016/j.apenergy.2021.116983.

[2] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future," *Int. J. Forecast.*, vol. 30, no. 4, pp. 1030–1081, 2014, doi: 10.1016/j.ijforecast.2014.08.008.

[3] J. Lago, F. De Ridder, and B. De Schutter, "Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms," *Appl. Energy*, vol. 221, pp. 386–405, 2018, doi: 10.1016/j.apenergy.2018.02.069.

[4] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary

differential equations," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, pp. 6571–6583, 2018.

[5] S. J. Taylor and B. Letham, "Business Time Series Forecasting at Scale," *PeerJ Prepr. 5e3190v2*, vol. 35, no. 8, pp. 48–90, 2017.

[6] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," *Belgian/Netherlands Artif. Intell. Conf.*, no. NeurIPS, pp. 364–366, 2020.

[7] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural controlled differential equations for irregular time series," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, no. 1, 2020.

PREDICCIÓN DE PRECIOS DE LA ELECTRICIDAD CON NEURAL ODE

Autor: Sanz Muñoz, Jaime
Directores: Muñoz San Roque, Antonio
Malpica Morales, Antonio
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

RESUMEN

Este trabajo de fin de máster desarrolla y evalúa el innovador modelo neural ODE (Ecuaciones Diferenciales Ordinarias o EDO) para la predicción a corto plazo del precio de la electricidad en el mercado diario en España en 2019. El modelo neural ODE se implementa en PyTorch y se compara con modelos de referencia que incluyen el SARIMA, Facebook Prophet, MLP, LSTM y CNN-LSTM. La neural ODE univariante logró el error promedio más bajo, superando a los métodos estadísticos y de aprendizaje automático (ML). Sin embargo, el modelo Prophet multivariante proporcionó el mejor rendimiento, aprovechando al máximo las variables exógenas. Aunque son computacionalmente costosas, las neural ODE muestran resultados prometedores.

Palabras clave: Predicción; Precio de la electricidad; EDO; Mercado diario

1. Introducción

La predicción de precios a corto plazo juega un papel crítico en el funcionamiento efectivo y estable de los mercados de energía. Los pronósticos de precios precisos permiten a las empresas energéticas optimizar los horarios de generación, a los operadores maximizar los beneficios y a los consumidores planificar el uso [1]. Sin embargo, los precios de la electricidad muestran una volatilidad sustancial debido a las fluctuaciones en la demanda, la oferta y otros factores exógenos como el clima, lo que dificulta la obtención de predicciones precisas.

A lo largo de los años, se han aplicado diversas técnicas estadísticas y de ML para EPF, incluyendo modelos de media móvil autorregresiva integrada (ARIMA), máquinas de vectores de soporte (SVM) y redes neuronales artificiales [2][3]. A medida que estos métodos evolucionaron, se introdujo el enfoque híbrido, combinando las fortalezas de ambos mundos para ofrecer una forma robusta de lidiar con la creciente volatilidad de los precios. Recientemente, una nueva clase de modelo de aprendizaje profundo llamado neural ODEs ha mostrado resultados prometedores para aprender dinámicas complejas de tiempo continuo [4].

Este artículo proporciona la primera implementación en profundidad de neural ODE para la predicción de precios de la electricidad del mercado diario español. Se desarrollan los enfoques univariante y multivariante. Cabe mencionar que el univariante solo utiliza datos históricos de precios, mientras que el enfoque multivariante incluye tres variables exógenas adicionales: la demanda de energía P48 y la producción eólica (obtenidos del operador del sistema OMIE), y el día de la semana.

Los objetivos principales de este trabajo de fin de máster comienzan con la realización de una revisión exhaustiva de la investigación actual y los estado del arte en el mundo de la predicción de precios de la electricidad, ya que nuevos datos y algoritmos están surgiendo rápidamente en este campo. Además, se implementará una serie de modelos de benchmark que incluyen modelos estadísticos, probabilísticos y de ML para proporcionar una línea de referencia para el análisis posterior de los resultados. Luego, se implementará la neural ODE en PyTorch, y sus resultados serán evaluados utilizando la métrica de error apropiada. Finalmente, los resultados serán resumidos en este artículo académico, describiendo las fortalezas y debilidades del modelo para la predicción de precios de la electricidad. En general, los resultados proporcionarán valiosos conocimientos para todos los agentes del mercado y la comunidad de EPF.

2. Modelos de referencia

Este estudio utiliza datos de precios de electricidad por hora del mercado del día siguiente de España en un período que abarca desde mediados de 2018 hasta finales de 2019. Se ha seleccionado una división de datos consistente para los modelos de referencia, utilizando el año anterior de observaciones como período de entrenamiento, tres meses como conjunto de validación y los siguientes tres meses para el conjunto de prueba, que abarcará de agosto de 2019 a octubre de 2019. Los cinco modelos de referencia que se han desarrollado, para los enfoques univariante y multivariante, se describen a continuación.

El modelo ARIMA estacional (SARIMA) ha sido seleccionado por su capacidad para capturar ciclos estacionales en las series temporales, centrándose para este estudio en las estacionalidades regular, diaria y semanal. Implementado en R, se sigue la metodología Box-Jenkins para encontrar y probar la mejor combinación de hiperparámetros. El modelo se entrena en las 13 semanas anteriores de datos, para predecir los siguientes 24 precios. Finalmente, se analizan los residuos de entrenamiento resultantes.

Caracterizado por su flexibilidad, robustez y eficiencia computacional, el modelo probabilístico Prophet ha sido incluido en este estudio. Este modelo ha sido desarrollado por Facebook para mitigar la falta frecuente de resiliencia de los métodos de predicción [5]. El modelo Prophet utiliza las 6 semanas anteriores de observaciones.

Los modelos de Perceptrón Multicapa (MLP) son redes neuronales artificiales que permiten capturar relaciones no lineales complejas en los datos a través de la retropropagación tradicional y el descenso de gradiente estocástico. El modelo toma los 168 precios horarios anteriores (la semana anterior) como entrada y predice las siguientes 24 horas.

Las redes de Memoria a Largo Plazo (LSTM) son un tipo de Red Neural Recurrente (RNN), que han demostrado ser muy efectivas para modelar datos secuenciales, dada su capacidad para recordar datos a largo plazo. También han sido entrenadas utilizando los datos de la semana anterior.

Finalmente, los modelos híbridos como el modelo de Red Neuronal Convolucional-LSTM (CNN-LSTM), combinan las fortalezas de ambas técnicas, reuniendo sus habilidades para capturar características temporales y espaciales en los datos. Antes de pasar los datos de entrada al modelo LSTM, una capa convolucional de 1D extrae características y patrones, utilizando una secuencia de filtros aprendibles. Esta capacidad de extracción de características es especialmente adecuada para el enfoque multivariante, donde puede haber interconexiones ocultas o patrones recurrentes que no sean inmediatamente reconocibles.

La evaluación del rendimiento de los modelos se ha realizado siguiendo el error absoluto medio estándar (MAE) en las predicciones del conjunto de test.

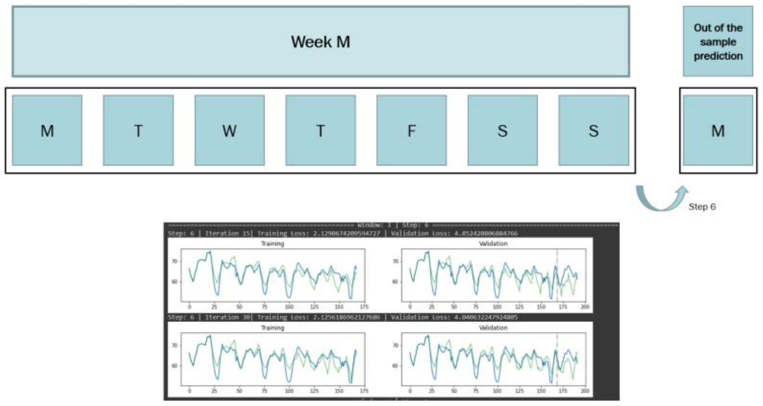
3. Neural ODE

Hace cinco años, en 2018, Ricky T. Q. Chen y sus compañeros publicaron el artículo que introducía esta nueva familia de modelos de redes neuronales profundas, las neural ODEs. Esencialmente, eran una versión continua de los modelos RNN, donde el estado oculto en un momento dado se modela como una función del tiempo, en lugar de una secuencia de pasos de tiempo discretos [4]. Al combinar la flexibilidad de las redes neuronales profundas con las propiedades matemáticas de las EDOs, Chen y sus compañeros demostraron que este nuevo método podía modelar con precisión sistemas complejos y dinámicos, superando en rendimiento a los enfoques tradicionales en ciertos escenarios. Desde entonces, las neural ODEs se han implementado en otros campos, logrando un rendimiento óptimo en áreas como series temporales irregulares y control de robótica [6]. Sin embargo, su aplicación a EPF ha sido limitada y el objetivo principal de este trabajos de fin de máster es implementarlo por primera vez en el mercado eléctrico diario español.

La idea innovadora detrás de una neural ODE es evitar la retropropagación tradicional. Diferenciar a través de las operaciones del 'forward-pass' es sencillo, pero incurre en un alto costo de memoria e introduce un error numérico adicional [4]. Sin embargo, el solver de EDO se representa como una caja negra y calcula la solución del problema de valor inicial (IVP), calculando los gradientes utilizando el llamado método de *adjoint sensitivity*. Este enfoque calcula los gradientes resolviendo una segunda EDO aumentada hacia atrás en el tiempo, y es aplicable a todos los solvers de EDO.

La red neuronal toma como entrada la condición inicial, es decir, la entrada en el primer paso de tiempo, y la desarrolla con el tiempo de acuerdo a la EDO aprendida detrás de la dinámica. Los pasos de tiempo corresponderán a cada día de la semana de entrada, sirviendo el último como la predicción de test. La neural ODE se entrena con 5 pasos de tiempo, que corresponden a la trayectoria de precios de la semana anterior. Después, la predicción fuera de la muestra se realiza en el sexto paso de tiempo.

El solver de ODE implementado, el *dopri5* adaptativo (Dormand-Prince Runge-Kutta de orden 5), calcula un número arbitrario de iteraciones para los datos de cada día, antes de avanzar al siguiente paso de tiempo para la evaluación. La validación de la neural ODE para la predicción de test es la siguiente:



```
============================== Window: 1 | Step: 6 ==============================
Step: 6 | Iteration 15| Training Loss: 2.1290674209594727 | Validation Loss: 4.852420806884766
Step: 6 | Iteration 30| Training Loss: 2.1256186962127686 | Validation Loss: 4.040632247924805
```

Figura 1: Evaluación al final del último paso de tiempo - Predicción de test

Las neural ODE, a diferencia de los modelos de ML tradicionales, requieren que la dimensionalidad de la entrada y la salida coincidan, debido a la naturaleza de las EDOs. Este requisito da forma a la arquitectura del modelo, incluyendo el número de capas y el preprocesamiento de datos de entrada. Para el caso de la neural ODE univariante, la condición inicial del IVP coincide con el tamaño de la predicción deseada, 24. Sin embargo, la neural ODE multivariante calcula y predice los valores para las tres variables (demanda de energía P48, generación de viento P48 y precio de la electricidad). Estos valores se introducen de manera aplanada, dando un total de 72 valores. Sin embargo, para concentrar los recursos computacionales en la tarea principal de predicción de precios de la electricidad, la función de pérdida se aplica solo a los últimos 24 valores de la última capa de la red neuronal. La siguiente figura muestra los valores de salida de la solución de la EDO, habiendo ajustado un rectángulo en las últimas 24 salidas, que representan las predicciones finales de precios para el día siguiente:

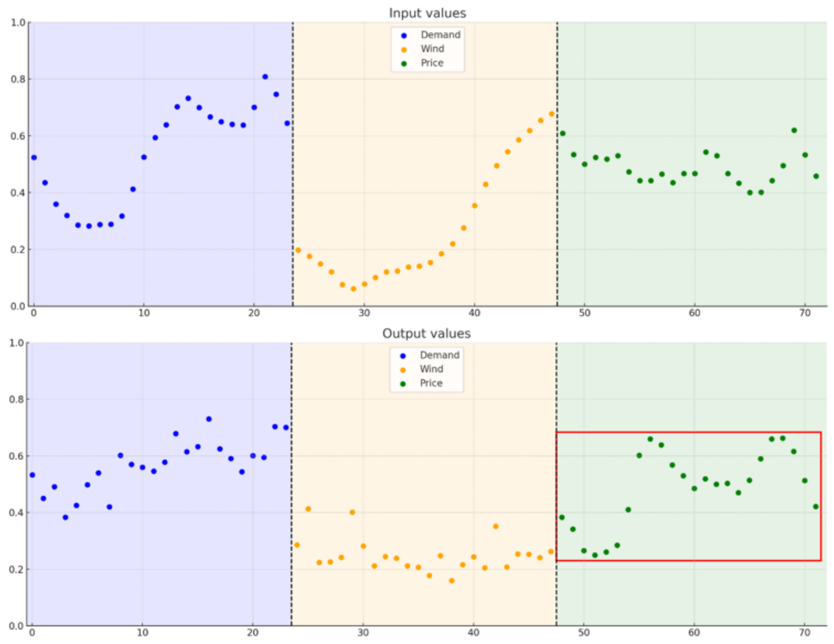


Figura 2: Valores de entrada y salida de la neural ODE multivariante

4. Resultados finales

Los valores resultantes de la función de pérdida de cada modelo univariante para cada día de la semana son los siguientes:

Día a predecir	*Naïve*	*SARIMA*	*Prophet*	*MLP*	*LSTM*	*CNN-LSTM*	*Neural ODE*
Lunes	8.121	4.25	4.94	5.331	5.292	5.639	**3.786**
Martes	3.935	4.49	4.27	3.941	3.961	4.679	**3.460**
Miércoles	3.352	4.59	3.44	4.322	**3.178**	3.239	3.915
Jueves	**3.554**	3.73	4.09	4.636	4.350	4.503	3.727
Viernes	**2.768**	4.01	3.76	3.418	3.309	3.437	3.613
Sábado	4.665	4.56	4.84	3.909	3.908	4.057	**3.381**
Domingo	4.093	6.87	4.68	3.715	3.473	**3.177**	3.648
MAE promedio	4.355	4.643	4.289	4.182	3.924	4.104	**3.647**

Tabla 11: Resultados univariantes

Estos resultados resaltan la importancia de entrenar modelos específicos para cada día de la semana. Esto se debe a que pueden tener diferentes dinámicas de precios debido a factores externos como patrones de demanda, lo que hace adecuado personalizar los modelos para días específicos. Al mirar los valores promedio de la función de pérdida para los diferentes modelos, la neural ODE univariante supera al resto de los modelos de referencia. A pesar de que los modelos se han ejecutado en la CPU, logran los mejores resultados en general. Cabe destacar que esto no se aplica a todos los días, pero en general, la neural ODE es el modelo que más aprovecha la serie temporal de precios de la electricidad, cuando es la única que se considera en el estudio. Los valores resultantes de la función de pérdida de cada modelo multivariante para cada día de la semana son los siguientes:

Día a predecir	*Naïve*	*SARIMAX*	*Prophet*	*MLP*	*LSTM*	*CNN-LSTM*	*Neural ODE*
Lunes	8.121	4.51	**2.45**	5.213	5.756	3.804	4.358
Martes	3.935	4.89	**2.38**	3.693	5.684	3.995	4.792
Miércoles	3.352	5.11	**3.28**	4.076	5.601	3.547	4.090
Jueves	3.554	4.33	**2.86**	4.001	7.084	4.315	3.973
Viernes	**2.768**	4.26	2.78	4.173	6.572	3.325	3.844
Sábado	4.665	5.04	**2.34**	3.410	5.653	4.022	3.799
Domingo	4.093	5.41	**2.48**	2.919	6.090	2.755	3.550
MAE promedio	4.355	4.793	**2.652**	3.926	6.063	3.680	4.058

Tabla 2: Resultados multivariantes

Estos resultados destacan un hecho clave en la predicción de series temporales: agregar más variables no mejora necesariamente el rendimiento, siendo el caso de los modelos SARIMA y LSTM. Es crucial seleccionar y preprocesar cuidadosamente las variables para asegurar que proporcionen información adicional valiosa. Sorprendentemente, el modelo multivariante con el mejor rendimiento global es el Prophet. Con un asombroso MAE global de precios de la electricidad de 2.652 €/MWh, ha superado al resto de modelos de referencia.

5. Conclusiones y Trabajo a Futuro

Este estudio proporcionó una valiosa visión del potencial de las neural ODEs en el problema de la predicción de precios de la electricidad en el mercado diario en España. Esta aplicación pionera ha traído desafíos únicos, incluyendo la falta de referencias y comparaciones con arquitecturas similares.

Una de las principales fortalezas que se ha demostrado de las neural ODE es su eficiencia en el uso de los datos. A diferencia de los otros modelos de referencia, que requerían grandes conjuntos de datos de entrenamiento, las neural ODE demostraron un rendimiento óptimo con un tamaño de conjuntos de datos significativamente menor. En consecuencia, las neural ODE surgen como una opción importante para series temporales con datos históricos limitados, permitiendo una reacción más rápida a las dinámicas cambiantes. Esto podría incluir escenarios relacionados con eventos raros como desastres naturales y ventas de nuevos productos de mercado.

Sorprendentemente, el modelo Prophet de Facebook ha emergido como el modelo de referencia para la predicción de precios multivariante. Ha demostrado tener un gran potencial para manejar la variabilidad e incertidumbre en los precios de la electricidad, manifestando su capacidad para modelar los efectos de los días festivos. Por el contrario, el estudio ha probado que añadir más variables no siempre mejora el rendimiento de la predicción. Como es el caso del modelo SARIMAX, que no pudo aprovechar eficazmente la información adicional proporcionada por las variables exógenas.

Mirando hacia el futuro, el trabajo a futuro incluye la exploración de ecuaciones diferenciales controladas por redes neuronales para aplicaciones del mundo real [7], enfocándose en mejorar sus capacidades de generalización y robustez al ruido. Sin embargo, debido a la cantidad de cálculos requeridos para las neural ODE, la migración a sistemas GPU será necesaria para explorar series temporales más complejas y solvers de ODE más precisos.

6. Referencias

[1] J. Lago, G. Marcjasz, B. De Schutter, and R. Weron, "Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark," *Appl. Energy*, vol. 293, no. December 2020, p. 116983, 2021, doi: 10.1016/j.apenergy.2021.116983.

[2] J. Lago, F. De Ridder, and B. De Schutter, "Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms," *Appl. Energy*, vol. 221, pp. 386–405, 2018, doi: 10.1016/j.apenergy.2018.02.069.

[3] R. Weron, "Electricity price forecasting: A review of the state-of-the-art with a look into the future," *Int. J. Forecast.*, vol. 30, no. 4, pp. 1030–1081, 2014, doi: 10.1016/j.ijforecast.2014.08.008.

[4] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, pp. 6571–6583, 2018.

[5] S. J. Taylor and B. Letham, "Business Time Series Forecasting at Scale," *PeerJ Prepr. 5e3190v2*, vol. 35, no. 8, pp. 48–90, 2017.

[6] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau, "GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series," *Belgian/Netherlands Artif. Intell. Conf.*, no. NeurIPS, pp. 364–366, 2020.

[7] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural controlled differential equations for irregular time series," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, no. 1, 2020.

Acknowledgements

In every life achievement, it is key to have a solid team behind. Therefore, in this crucial point of my life, as I finish the university stage, I feel compelled to express my gratitude towards everyone who made this master’s thesis delivery possible.

First of all, I want to express my acknowledgements to my family. Their constant support in the tough moments have sometimes been taken for granted. On every late night, every stress and accomplishment, they have been there for me. For that, I am very grateful.

I want to thank my friends as well, who have supported me along the way. They remind me the importance of finding balance between life and work, and to always enjoy the journey, no matter the difficulty of the path. For that, I am very grateful.

I want to thank my directors. They have shown much patience and assertiveness, guiding me in this path throughout the year with their useful insights. From the moment they selected me to develop this project, they have demonstrated faith in my abilities and motivated me. For that, I am very grateful.

Finally, I want to thank my university. Although the student path has been bumpy, I will always remember it with affection. Moreover, it has equipped me with the necessary tools for my future career. I feel this master’s thesis is not only an academic achievement, but also a launching pad towards future professional opportunities. And for that, I am very grateful too.

INDEX

Figure Index

Table Index

Chapter 1. INTRODUCTION

1.1. ABSTRACT

Accurate electricity price forecasting (EPF) plays a fundamental role in the decision-making process within the context of a liberalized electricity market landscape. It can be challenging due to the complex and constantly changing nature of the electricity market. Factors such as natural disasters, economic conditions and government policies can all impact electricity prices and make accurate forecasting difficult. Moreover, the disruptive introduction of renewable energies in the energy mix equation has made EPF more difficult to predict than ever. Consequently, in the last years we have seen a more volatile price related to several spikes that has presented at certain times, a negative value.

Over the last years, traditional statistical approaches, which assume linear dependencies between electricity related variables, have been the prevailing techniques for EPF. However, they have been quickly outperformed by a new generation of frameworks namely machine-aided models. Machine Learning methods have become increasingly popular for EPF due to their ability to capture complex relationships and patterns in data.

In this context neural ordinary differential equations (ODEs) have emerged as a promising techniques for handling time-dependent data and capturing intricate underlying dynamics. Following a Machine Learning (ML) architecture, they are capable of modeling complex dynamics systems. Essentially, they are a continuous version of the traditional neural networks, where the hidden state at a given time is modeled as a function of time, instead of a sequence of discrete time-steps.

While research on neural ODEs is still in early stages and has not yet consolidated as a widely known ML framework, there are several studies emphasizing its relevance to many scientific realms. To the best if our knowledge, the implementation of a neural ODE in the context of EPF has not yet been studied and constitute the motivation of the thesis.

In this master's thesis, neural ODEs will be developed and implemented in Python, more specifically, in PyTorch, using the *torchdiffeq* library, implemented by Ricky Chen and his colleagues. The library will be imported to implement all the necessary features of the process. Following the selected error evaluation metrics, it will be compared against various benchmark models, previously developed.

It is worth noting that, as best practices in the EPF field, researchers must be able to easily test their models with shared materials because if not, it would not be possible to properly compare their results (Lago et al., 2021). Previous research indicates that each researcher usually uses a different dataset, with different parameters tuned and different error evaluation procedures, not allowing the models to be properly reproduced. To overcome this limitation, we develop most of the benchmark models and the neural ODEs using the Python programming language, as a common framework.

This master's thesis aims to explore and evaluate neural ODEs for EPF. The primary objectives start conducting an extensive review of the current research and state-of-the-art methods in EPF, as new data and algorithms are rapidly emerging in this evolving field. Moreover, a series of benchmark models including the statistical, probabilistic and ML models will be implemented to provide a baseline for the posterior analysis of results. The neural ODE will then be implemented in PyTorch to forecast the Spanish electricity prices. Its results will be assessed against the benchmark models using the appropriate error metrics, to understand how they perform against stablished methods. Finally, the results will be summarized in an academic paper, describing the strengths and weaknesses of neural ODE for EPF. Overall, the outcomes will provide valuable insights for all the market agents and EPF community.

Therefore, this study is organized in seven chapters. Chapter 1 provides background on the Spanish electricity market, time-series forecasting techniques and the relevance of EPF. Chapter 2 details the benchmark models implemented, including seasonal ARIMA (SARIMA), Prophet, Multi-Layer Preceptron (MLP), Long-Short Term Memory (LSTM) and a hybrid Convolutional neural network - LSTM (CNN-LSTM). Chapter 3 explains the theory and implementation of the neural ODEs. The training strategy for both univariate and multivariate approaches have been also outlined. Chapter 4 analyzes and compares the results of the neural ODEs against the benchmark models. Chapter 5 revisits the key findings and contributions of the thesis, acknowledging limitations and proposing future work. Chapter 6 contains the referenced bibliography that was used. Finally, Chapter 7 represents the Annex, including code samples for the univariate and multivariate neural ODE models and the alignment of the project with the Sustainable Development Goals (SDGs).

1.2. DAY-AHEAD ELECTRICITY MARKET

Energy is one of the most fundamental resources that sustain global economy. Its acquisition, production and use are key elements that influence politics, infrastructure development and the quality of life. Spain, with a varied and complex electricity-power systems, is no exception to this pattern. In this context, OMIE is the 'Nominated Electricity Market Operator' (NEMO) for the Iberian Peninsula. It actively participates in the coupling of the wholesale electricity markets in the EU, together with all the designated NEMOs in each member state.

Europe has established a regulatory framework for the European electricity sector until 2030 based on cross-border marginal energy markets. Under this regulation, OMIE manages the day-ahead and intraday wholesale electricity market (intraday and continuous intraday auctions) for Spain and Portugal. Approximately a 77% of all the energy resources in Spain are allocated through fixed contracts. This leaves the remaining 23% to be dealt with in the day-ahead electricity markets.

The Spanish day-ahead electricity market is a wholesale economic mechanism that allows the transaction between producers and consumers. This market is defined by a series of characteristics that are focused on the efficiency and transparency, which guarantees that all market players can operate on equal terms. The hourly price of electricity energy is determined through a daily bidding process where producers offer the energy they can generate, and consumers bid to acquire it, for each hour of the next day.

The day-ahead market, also known as the Single Day-ahead Coupling (SDAC), as an integral part of the electricity production market, aims to carry out electricity transactions through the submission of bids for the sale and purchase of electricity by market agents for the twenty-four hours of the following day. Every day of the year at 12:00 CET, the day-ahead market session takes place in which electricity prices and energies across Europe are set for the 24 hours of the following day. The price and volume of energy in each hour are established by the cross between supply and demand, following the model agreed and approved by all European markets.

The intraday markets take place on the same day the energy is going to be sold and therefore, used. They are managed by the system operator and though the prices of these markets are related, there can be important differences between them. This system focuses on supply optimization, balancing supply, and demand, to guarantee that the demand is satisfied in the most efficient way. The way the market works is very dynamic and is subject to persistent fluctuations such as the weather, the state of the transmission grid and consumer demand. The rise of renewables is changing the structure of the energy market, diversifying supply, and demand. As a result of this, the competition in the market may intensify, with implications for energy prices and the strategies used by producers and consumers.

This project will focus on EPF in the day-ahead market. Overall, predicting electricity prices in the day-ahead market can have a variety of uses, depending on who is making the predictions and what they are being used for. The ability to accurately predict electricity prices in the day-ahead market offers various benefits for different stakeholders, including electricity producers, consumers, traders, and regulators.

1.3. TIME-SERIES FORECASTING AND EPF

The disruptive introduction of renewable energies in the energy mix equation has made EPF more challenging than ever, due to the complex and constantly changing nature of the electricity market.

There is a large spectrum of factors affecting the price such as the power supply and demand relationship, power generation costs and market structure. While some of these factors can be considered to some extent, there are exogenous market effects that may be difficult to predict. These unforeseeable factor encompass weather conditions, transmissions problems in the power system, causing a high volatility in the price and a certain degree of risk for the agents (Lago et al., 2021).

Because of this, short-term predictions are key in the energy market. The system operator, Red Eléctrica de España (REE), implements advanced forecasting techniques to anticipate supply and demand for the following day. These forecasts include estimations for the clients' consumption, the different energy technologies generation and the weather conditions. By using these forecasts, the different agents (producers and consumers) plan and optimize their operations. In this regard, short-term forecasts are crucial to maintain a reliable and efficient energy supply.

For the electricity producers, accurate EPF is key for an effective planning. They can optimize their operations by knowing when it will be most profitable to generate electricity. Thus, they can understand which energy source is most economically viable at any given time.

Consumers can also benefit from accurate EPF. Each year, as the price rises, it is becoming a general concern. Consequently, strategizing on when to use the washing machine or other high-power appliances is now common and is changing people's habits in terms of energy usage.

Traders, as their work is focused on maximizing their returns in the energy market, they rely heavily on accurate forecasts. The day-ahead market is not the only one, but it is where most of the energy is traded, leaving aside fixed or bilateral contracts. Therefore, it is crucial to make well informed strategic decisions for their buy and sell plan. After the day-ahead window closes, the leftover power gets to be traded in the intraday markets. As the energy delivery deadline approaches, the intraday environment tends to become turbulent and unpredictable. Here, traders depend on accurate EPF to respond to changing conditions.

Regulators, with the task of monitoring and maintaining the integrity and continuity of the electricity market, can use EPF for assessing the market behavior. The forecasts can represent a valuable input to motivate regulatory decisions to ensure a balanced and competitive market.

EPF is also beneficial for the system operator itself as, through better predictions, the power system dispatch could be executed effectively, and the safe and reliable operation of the electricity system would be ensured.

Not surprisingly, short-term predictions are not infallible and are subject to uncertainty. The inherent challenges in EPF have been exacerbated by the significant roll-out of renewable energy generation within the electricity-power system. These renewable-based generation rely on intermittent energy sources. For instance, wind and solar energy production, which represent roughly a 30% of the total production, can vary significantly from one day to another due to changes in the weather conditions.

However, the continuous improvement in the forecasting techniques, along with a broader interconnection between the grids, has facilitated the management of uncertainties and improving the stability and efficiency of the energy system. Looking into the future, it is evident that the Spanish day-ahead electricity market will keep evolving. The efforts to reduce the greenhouse effect, along with the advancements in renewable technologies will reshape the way the electricity is produced and consumed.

To accurately predict electricity prices, researchers and industry professionals are constantly developing advanced forecasting techniques. Broadly, these techniques can be divided into three categories: statistical methods, ML methods, and hybrid methods.

Statistical methods, which form the basis of traditional forecasting, involve using historical data to detect recurring patterns or trends. These methods are adept at capturing linear dependencies in the data. Techniques in this category include autoregressive integrated moving average (ARIMA) models, exponential smoothing, and regression-based approaches. However, these models tend to make assumptions about the data (such as stationarity) and might struggle to capture non-linear relationships and complex interactions between variables, which are often present in electricity price data.

ML methods, on the other hand, have gained popularity for EPF due to their flexibility and ability to capture complex relationships in the data. These methods leverage algorithms that learn from data, enabling them to discern intricate patterns that traditional methods may miss. Techniques in this category include neural networks, support vector machines, random forests, and gradient boosting machines, among others. These methods, unlike their statistical counterparts, can learn and adapt to non-linear relationships and high-dimensional interactions, which are characteristic of electricity price data.

Hybrid methods represent a blend of statistical and ML approaches, aiming to combine the strengths of both. These methods typically involve the use of ML techniques to model the non-linear components and statistical methods to capture the linear trends in the data. By integrating these two approaches, hybrid methods can often outperform either method used in isolation.

A noteworthy advancement in the field of time-series forecasting is the development of neural ODEs. Neural ODEs are an extension of traditional neural networks, falling under the umbrella of ML. They represent a continuous-time model where the hidden state at any given time is modelled as a function of time, as opposed to the sequential discrete time-steps that conventional feedforward neural networks represent. This approach offers several advantages, including the ability to model complex dynamic systems and the flexibility to handle irregularly sampled data. The introduction of neural ODEs represents a significant step forward in the field of time-series forecasting, enabling the potential for enhanced performance across several forecasting applications. Notably, the application of neural ODEs in EPF, an area that remains unexplored, holds considerable promise, which is precisely the scope of our study.

In conclusion, the evolution of forecasting techniques from traditional statistical models to ML methods and, more recently, to hybrid approaches and continuous-time models like neural ODEs, shows the relentless pursuit of more accurate and robust forecasting methods. As the energy markets continue to evolve and grow more complex, these advanced techniques will play an increasingly important role in understanding and accurately predicting electricity prices.

1.4. TIME-SERIES DESCRIPTION

This case study focuses on the years 2018 and 2019 as they represent a crucial turning point for the renewable energy sector in Spain. This period was selected, due to the relative stability observed in the time-series data, providing a solid foundation for a more reliable and feasible forecast. These years were marked by a significant spike in new power capacity installations, particularly in solar and wind energy, influenced by a blend of political, economic, and technological shifts. This was part of Spain's broader commitment to the decarbonization goals in line with European Union's mandate to mitigate climate change and promote a sustainable and resilient economy.

In Spain, there has been an evolution towards green energy, driven by multiple factors, such as government policies, economy considerations and environmental concerns. Specially between years 2018 and 2019, a significant increase in the wind and installed power:

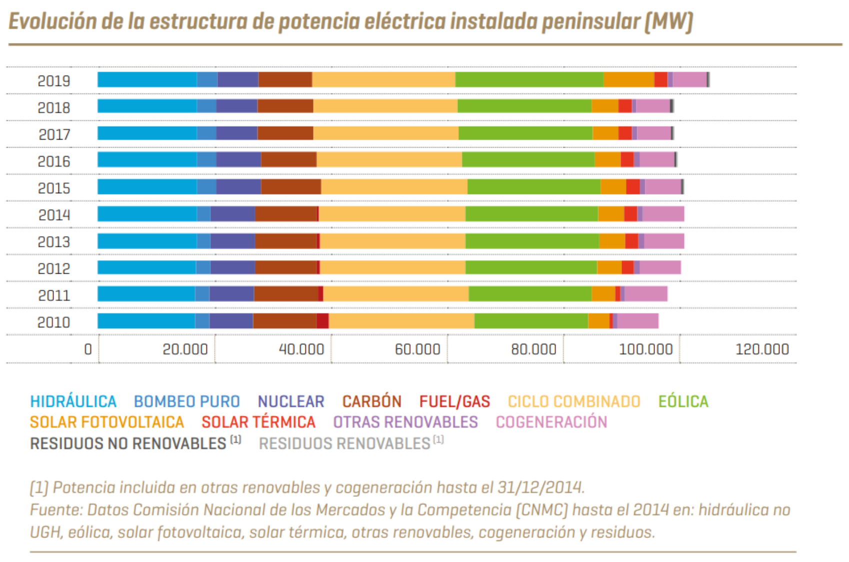


Figure 1: Evolution of the energy generation in Spain (REE, 2020)

Despite the progressive introduction of renewable energies within the Spanish electricity power system in 2018, Spain experienced relatively high electricity prices. This price increase was primarily driven by the escalation of international fossil fuel prices, particularly natural gas, and carbon. The country also faced a series of extreme weather conditions, including droughts and heatwaves, which significantly impacted hydroelectric generation and increased electricity demand. The average daily electricity price is represented in Figure 2:

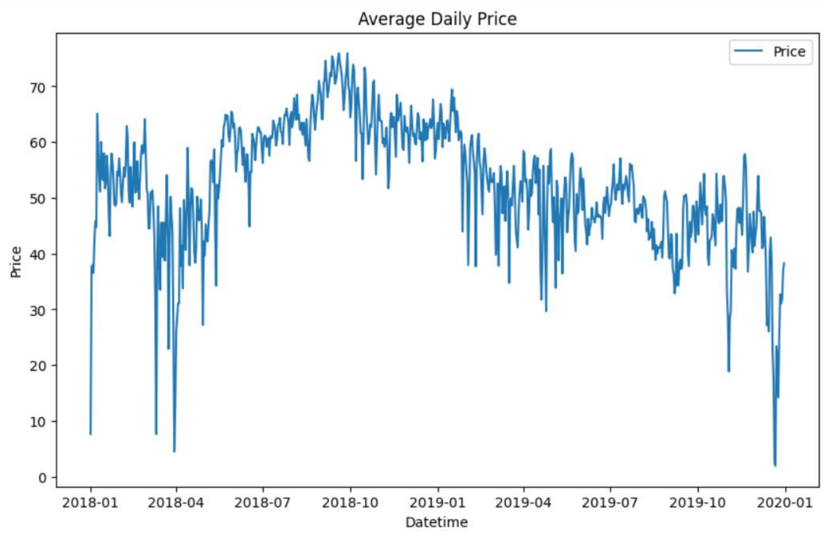


Figure 2: Average daily price 2018-2019

Given that household and office consumption accounts for approximately a 25% of the Spanish electricity demand, intraday fluctuations arise in the electricity demand profile. These fluctuations are directly related to the Spanish consumer's behavior, which is commented below. A closer look at the data reveals that demand typically rises during the early morning and late evening:

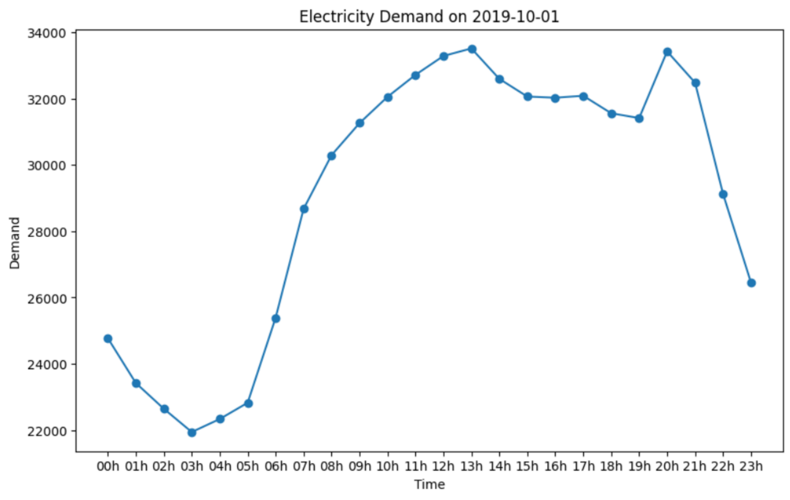


Figure 3: Electricity demand on 2019-10-01

Early in the morning, the price increases as people wake up and begin to use a variety of electrical appliances. Also, the factories and the rest of the heavy electricity consumers initiate their day. As the day progresses, a new spike in the electricity demand is observed. This is the time when people return home and activate their heating or AC (depending on the season). This leads to a surge in the electricity usage. This pattern is commonly observed in many countries and is called bimodal demand pattern. Interestingly, the overall demand pattern was characterized by peaks during the winter and summer months, due to heating and cooling requirements respectively.

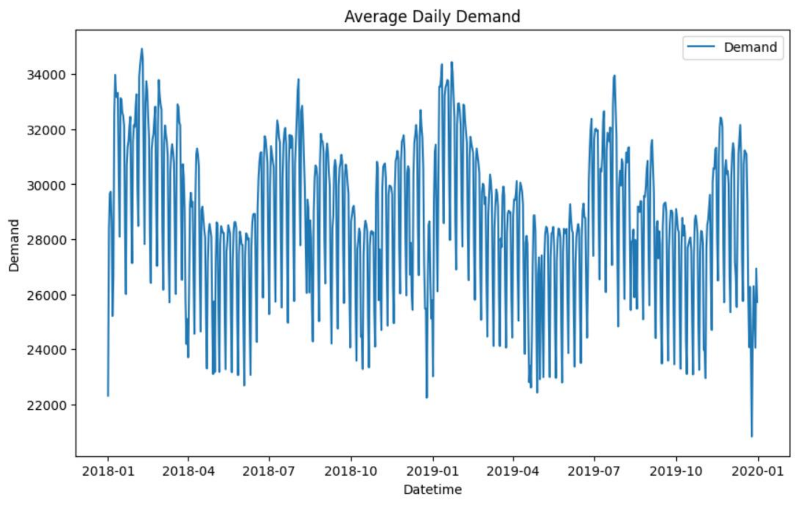


Figure 4: Average daily demand 2018-2019

In 2019, electricity prices experienced a moderate decline, largely due to falling international fossil fuel prices. Moreover, the increased penetration of renewable energies, particularly solar and wind, started to affect the market's price structure, helped by Spain's geographical location.

Focusing first on the PV technology, in 2019 there was almost a 19,8% increase in the installed capacity with respect to 2018, achieving a generation record of 8.841 GWh. It is worth noting that in 2019, the cost of PV technology had decreased by 80% with respect to the previous decade and that the government had promoted solar energy by providing incentives for the installation of PV panels.

Simultaneously, the contribution of wind energy to the Spanish electricity system increased, highlighting the growing confidence in and feasibility of this renewable source. According to data from REE, Spain's national grid operator, wind energy contributed significantly to meeting this demand, with a proportion record in the energy mix.

In 2019, the peninsular wind generation stood at 53.094 GWh, 8,5% higher than in the previous year. It is the most relevant renewable energy source in Spain, as it represented 55,2% of the total production of green energy:

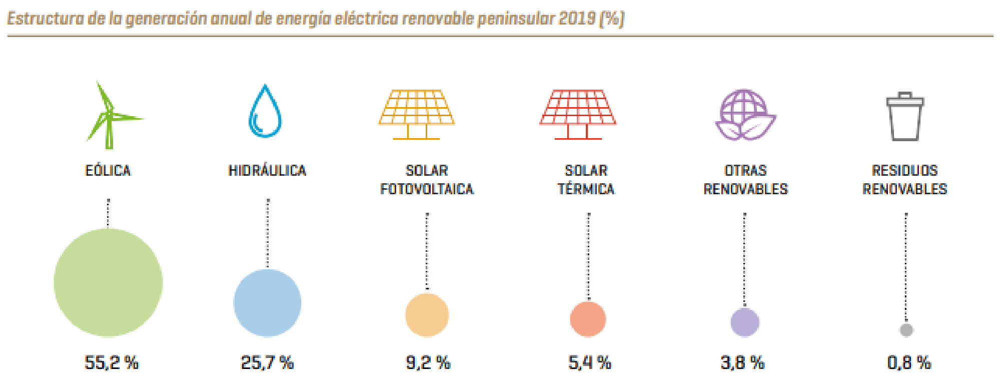


Figure 5: Renewable energy mix in Spain in 2019

Spain is now one of the leading countries in wind generation globally, thanks to onshore and offshore wind farms. This increase in renewable energy production, specially by wind generation, has led to a decrease in electricity prices, due to the low marginal cost of wind power.

One characteristic feature of wind generation data is its volatility, as it's highly dependent on weather conditions. This variability poses challenges to the management of power systems, yet it also presents opportunities for developing advanced forecasting methodologies to enhance accuracy and reliability of supply-demand balancing. The average daily wind generation are represented in Figure 6:

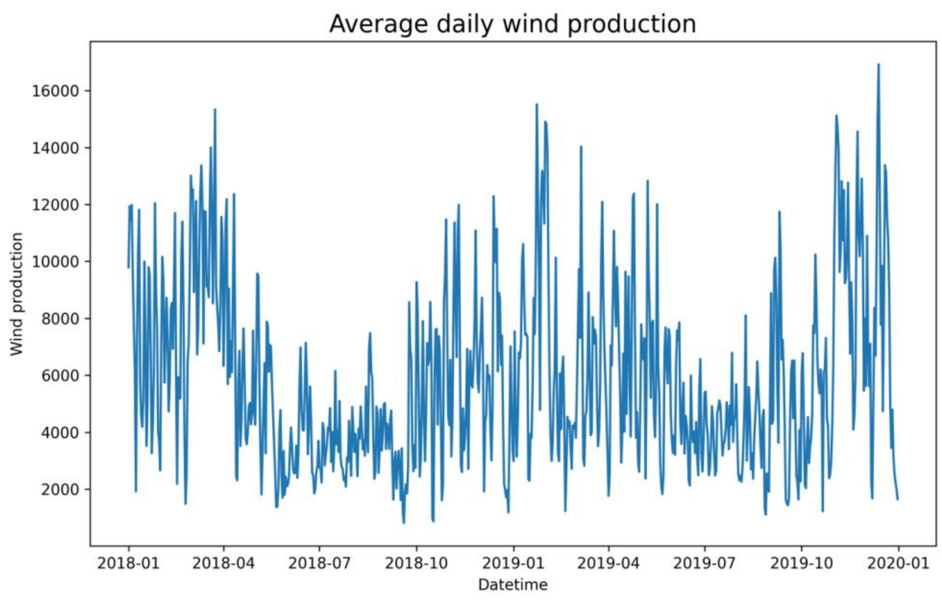


Figure 6: Average daily wind production

As it can be seen, both generation levels and the associated volatility exhibit fluctuations throughout the year, showing peak values in the first months on the year and lower values during the summer season.

The example of fluctuating wind production clearly demonstrates that the transition towards an electricity system with a high share of renewable technologies present some challenges. The characteristic intermittency of renewable sources, presents grid management issues, leading to mandatory investments in energy storage and grid infrastructure.

In the meantime, the available power capacity of fossil-fuel technologies has suffered an important decrease due to the shut-down of coal power plants, which eventually was the backbone of Spain's electricity generation. Ambitious goals have been set by the government, targeting the closure of coal mines, leading to a 69,4% reduction in their respective generation (REE, 2020).

Hydroelectric production would have been much greater, had it rained as the previous year. 2019 was a especially dry year, leading to a 27,6% decrease in hydraulic generation, achieving only 24.709 GWh. The base energy in the energy mix, was substituted by the combined cycle power plants, which experienced a 93,7% increase with respect to 2018, making up a 20,7% contribution to the peninsular energy mix.

Overall, the evolution of the energy generation in Spain has moved towards a greener energy mix. This transition, supported by the government, represents a global trend towards clean energy. Spain, benefiting from its geographic location, is a front-runner in renewable energy generation.

It is important to continue investing in renewables, improving their efficiency, developing infrastructure to facilitate their integration into the power grid, and ensuring a sustainable and resilient future for the energy sector in Spain. Nevertheless, it is equally important to invest in robust forecasting models, as they are crucial in accurately predicting the resulting effect of the green energy integration on electricity prices. These elements are key to provide accurate forecasts and establishing strategies for Spain's path to decarbonization.

1.5. *EXOGENOUS VARIABLES*

The following variables have been included on the multivariate approach, showing significant relevance for the day-ahead market and, therefore, for the electricity price formation.

In the context of the electricity market, the Hourly Operative Energy Program (P48) plays a fundamental role. It is the operative plan that establishes the programming and manages the energy sales and acquisitions in the Spanish electricity peninsular market. P48 represents the planification of the electric system in real time, as it incorporates all the assignations and redepositions of the program, applied by the System Operator (OS) until its publications, which occurs 15 minutes before the beginning of every hour.

Moreover, P48 includes the adjustments of the Daily Functioning Base Program (PDBF) and the results of the different sessions of the intraday market. It adjusts itself to reflect any changes in the market conditions, such as downtime cases or problems communicated by the programmed units. Therefore, P48 reflects a real time picture of the energy supply and demand of the electric system.

In this context, the first exogenous variable is the **P48_demand** (demand) is composed of the real-time energy requirements of the whole electric system. Essentially, it is the hourly energy consumption expectations based on the adjustments made to the PDBF. Fluctuations in the expected demand can derive from changes in consumer behavior, industrial operations, or unexpected events. Nevertheless, thanks to a close monitoring, the system remains resilient and adaptive to the upcoming volatility in the time-series.

The second explanatory variable is the **P48_wind_generation** (wind). As explained before, the wind energy has already become a primary source of energy in Spain, with more than half of the production coming from the wind generators. Unlike traditional power sources, wind is highly dependent on the weather conditions and therefore, far more volatile, and unpredictable. However, this hourly value represents accurately the programmed wind energy. It must be noted that, although the model has been adjusted with these P48 variables for the sake of simplicity, the use of these variables' forecasts as an additional input is widely extended.

Although the **day_of_the_week** variable has not been included in the correlation matrix, it will be taken into account for the benchmark models, as both the probabilistic and ML models will exploit it considering non-linearities. This discrete variable has been included and will assign each day of the week a different number from 0 (Monday) to 6 (Sunday), adding 7 in the cases where that day was a national holiday. For this study, the national holidays that have been considered are as follows:

- 2018-01-01: Año nuevo
- 2018-01-06: Epifanía del Señor
- 2018-03-30: Viernes Santo
- 2018-05-01: Día del Trabajador
- 2018-08-15: Asunción de la Virgen

- 2018-10-12: Día de la Hispanidad
- 2018-11-01: Todos los Santos
- 2018-12-06: Día de la Constitución Española
- 2018-12-08: La Inmaculada Concepción
- 2018-12-25: Navidad
- 2019-01-01: Año nuevo
- 2019-01-07: Epifanía del Señor (Moved)
- 2019-04-19: Viernes Santo
- 2019-05-01: Día del Trabajador
- 2019-08-15: Asunción de la Virgen
- 2019-10-12: Día de la Hispanidad
- 2019-11-01: Todos los Santos
- 2019-12-06: Día de la Constitución Española
- 2019-12-09: La Inmaculada Concepción (Moved)
- 2019-12-25: Navidad

Having introduced these main exogenous variables, their autocorrelations for the training period, from July 2018 to July 2019 are plotted on Figure 7. A key aspect of correlations is determining whether a variable is significant. Typically, two variables are significantly correlated if their correlation value is over 0.3, of course in absolute value. However, this is a general guideline, and the threshold can vary depending on the context. These autocorrelations provide a look into the relationships between the output variable (electricity price) and the predictor variables:

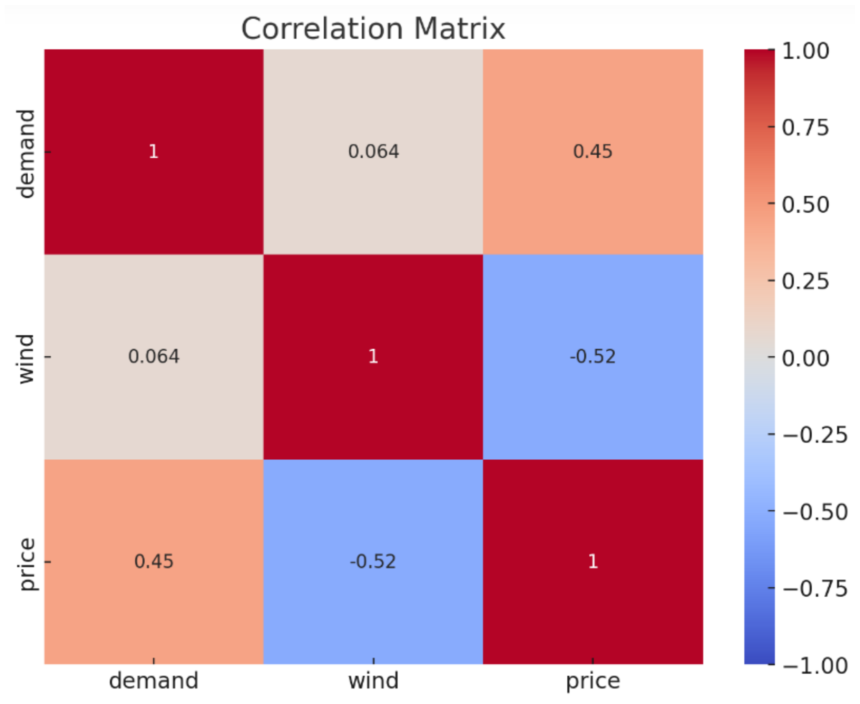


Figure 7: Autocorrelations between variables

As it can be seen on Figure 7, the correlation between the electricity price and the P48 demand had a value of 0.45, -0,52 with the wind production. According to the typical threshold, the P48 demand and wind production would be significant for this study.

It is interesting to see the sign of the correlation values. For the case of the P48 demand, the positive correlation makes sense intuitively, as increased demand leads to an increase in the price. Conversely, the correlation between the price and the P48 wind production is negative. This was also expectable as the wind production takes out of the energy equation the more expensive and contaminating energy sources. Therefore, in conclusion, the variables are significant enough for the models to gain more perspective and context on the electricity price time-series, which will allow for more accurate forecasts.

Chapter 2. BENCHMARK MODELS

The subsection delves into each of the benchmark models included for this study. Guided by an extensive review of the current state-of-the-art for EPF, a selection of the most significant methods will determine a reference baseline for the neural ODE results.

Ranging from traditional statistical methods to modern ML techniques, the selection provides a comprehensive understanding of how these models can be implemented for forecasting electricity prices. Each of them represents a unique approach to time-series forecasting, reflecting the current state-of-the-art in the EPF landscape.

2.1. STATE OF THE ART

The day-ahead EPF problem has been deeply researched for over the last 20 years due to its importance for the different agents in the electricity market. However, the task remains challenging because of the fluctuations associated with the nature of the electricity and the demand-supply dynamics. The complete review of the state-of-the-art models developed by Weron, (2014) indicates five main families, including multi-agent, fundamental, reduced-form, statistical and computer intelligence models. Nevertheless, the most recent review reveals that the evolution of EPF algorithms could be grouped by three main categories: statistical, ML, and hybrid techniques (Lago et al., 2021).

Initially, around the early 2000s, the backbone of the EPF algorithms was the **statistical method**, especially autoregressive models. The essence of this approach is the development of mathematical models, capable of identifying patterns or trends in the historical time-series data. The models do no longer represent an advantage in the task of EPF (Weron, 2014). Moreover, being the electricity price stochastic process, typically heavy-tailed, skewed, and heteroskedastic, new statistical methods have been developed to tackle it. Nevertheless, statistical methods still provide robust and reliable historical data analysis and are an essential tool for EPF.

Statistical methods range from simple linear regressions such as the Lasso Estimated Autoregressive models (LEAR) to more complex models, such as the Autoregressive Integrated Moving Average (ARIMA) or the Generalized Autoregressive Conditional Heteroskedasticity (GARCH). The latter model represents an attempt to capture the dynamic time-series volatility as a function of previous time periods (Cruz et al., 2011). However, it is not an attractive model for short-term EPF by itself and should be coupled with other models to be effective (Weron, 2014). Actually, a study showed that an ARIMA-GARCH model outperformed a generic ARIMA model, whenever the electricity time-series showed high volatility and price spikes (Garcia et al., 2005).

Other methods have been developed such as the exponential smoothing. Cruz et al., (2011) used a double exponential smoothing as a benchmark model, outperforming the naïve and ARIMA models for the hourly day-ahead prices of the Spanish market. However, neural network models proved to be better. This study offered the insight of using the predicted values of the operator's wind generation forecasts, which has resulted in better accuracy.

Given the strong seasonal patterns in the electricity time-series, it has been convenient to include them in the ARIMA models, resulting in a seasonal ARIMA or SARIMA. From the early 2000s, there has been some EPF applications for this kind of method. One of them was developed by Contreras et al., (2003), as they modelled three seasonalities separately on lags 24, 48 and 168h, outperforming the general model for all hours.

Recent advancements have been developed to include a larger number of input features and regularization techniques, currently positioning LEAR as one of the most relevant statistical algorithms. It must be noted that it could be hybrid since Lasso is known to be an ML technique by some authors, but the underlying model is statistical (Lago et al., 2021).

More focused on the probabilistic approach, the Prophet model arises as a flexible and easily interpretable approach for forecasting. Based on an additive model, it accommodates trends, seasonality, and holiday effects in the time-series. Developed by Facebook, it allows non-experts to address business forecasting problems without needing to dominate Data Science knowledge (Taylor & Letham, 2017). This model will later be explained in detail as it has been included in the list of benchmark models.

Progressing into the era of big data, ML techniques took over in the EPF environment. As described in the review developed by (Weron, 2014), these models were generally composed by artificial neural networks, fuzzy systems, Support Vector Machines (SVMs) and evolutionary computation. From the early studies, where neural networks had only one output node to more complex models with additional hidden layers, this model has clearly seen a strong evolution. In this context, González et al., (2005) successfully implemented a hybrid MLP input-output hidden Markov model for the Spanish EPF, providing accurate results and dynamic information about the market.

Where feed-forward networks are characteristic for being static, the Recurrent Neural Networks (RNN) arose to become more dynamic systems. As it will be explained later, on subsection 2.4.2. RNNs have been designed to work with sequential data and can maintain information from previous steps, which can provide information for the current and future time-steps. Nevertheless, they suffer from a problem called *vanishing gradient*, which can cause a slow learning curve for the early layers. To solve this issue, a study developed by Lin et al., (1996) proposed the solution of a *nonlinear autoregressive models with exogenous inputs (NARX)* model. However, it has been outperformed by other recent RNN models such as the Long Short-Term Memory neural networks (LSTM), which was implemented in various studies such as the one from Jiang & Hu, (2018).

A posterior seminal review by Lago et al. (2018) illustrated how **ML methods** like the ones commented above and more, such as the k-Nearest Neighbors (kNN), SVMs, Extreme Learning Machines (ELM), Random Forests (RF), Gradient Boosting Machines (GBM) outperformed traditional statistical techniques for the European electricity market. These models provided capabilities to work with complex and nonlinear time-series like the electricity price.

As explained by Lago et al., (2021), the late trend of published studies of ML methods avoid comparing against well-established benchmark models and their results are very simplistic. Nevertheless, the models keep evolving and adding features for modelling the increasingly complex patterns of the electricity price time-series. Wang et al., (2017) proposed a deep neural network using denoising autoencoders (SDA) for EPF. The authors developed an extended SDA called RS-SDA, which incorporates random sample consensus (RANSAC) to filter the outliers in the training data and stochastic neighbor embedding (SNE) to automatically determine the number of hidden units. This implementation proved to be one of the first studies to successfully use GPU-based frameworks for EPF.

In 2019, Y. Chen et al., (2019) proposed a bidirectional recurrent neural network architecture (RNN) called BRIM for the French EPF. It interestingly incorporates the concept of market integration, as it includes price data from an interconnected market (EXAA in Austria) to the French EPEX data, which is disclosed afterwards. This allows the model to consider past and future data and learn the relationships between them, and ultimately provide better results than the proposed unidirectional benchmark models.

Both classical and ML approaches seek to approximate the electricity price following a loss function, which measures the difference between the actual and the predicted price values. This error metric is progressively updated, adjusting the model hyperparameters in order to minimize the error incurred. The differences between the classical and ML lies in the respective update mechanism; in how they minimize the error in the next iteration. The ML models show an advantage over classical methods as they can ingest larger amounts of data, allowing the algorithms to better capture the dependencies and analyze the most complex patterns, providing more accuracy on the forecasts.

In recent times, the landscape of EPF has abruptly introduced **hybrid models**, that can combine the interpretability of statistical methods and the predictive capabilities of associated with the ML methods. According to Lago et al., (2021), since 2014 more than

100 articles have been written about hybrid methods, which is more than five times the number of papers on deep learning methods.

Usually, each separate algorithm focuses on a specific pre-processing task, such as decomposing data, feature selection and model ensembles. In many methodologies, the hybrid approach is not confined to a single forecasting model. Instead, it comprises a combination of multiple models, broadening the scope of the analysis and delivering a robust forecasting mechanism.

One of the techniques for decomposing a time-series is the *wavelet transformation*, which identifies and segments the signal into different frequencies. The algorithm allows the decomposition of the price series into a high frequency component (which represents the characteristic short-term fluctuations) and a low frequency one (which represents the long-term patterns. This technique is not new as the study developed by Conejo et al., (2006) already merged the wavelet transformation with the ARIMA method, outperforming the standard one. Moreover, it still used nowadays as Chang et al., (2019) develops an LSTM model combined with this pre-processing technique. The resulting decomposed signals were then introduced to the model as additional input features into the model.

Naz et al., (2019) implemented the technique of *Correlation Analysis* to discard the most redundant variables for extreme learning machines to avoid overfitting. These variables were identified because of the high correlation between them. Other similar method is the so-called *Mutual Information*, which measures both linear and non-linear relationships in the data to quantify the amount of information is provided between variables. The latter technique has been applied by Ebrahimian et al., (2018) and Keynia, (2012).

The relentless pursuit for better forecasting accuracy continues to push the boundaries of EPF. The most recent and promising venture in this quest is the development of neural ODEs. As a continuous-depth neural network, it provides a unique perspective of the evolution of a system's state. Instead of requiring a discrete sequence of hidden layers, the derivative of the hidden state is parameterized using a neural network (R. T. Q. Chen et al., 2018).

This master's thesis will develop a neural ODE for the Spanish day-ahead EPF for the first time, as it represents the main goal. Moreover, it will provide an overview of the proposed benchmark models, for posterior results comparison.

2.2. SARIMA MODEL

Traditional algorithms make use of predefined techniques and statistical models such as linear regression, autoregressive integrated moving average (ARIMA), and autoregressive integrated moving average with explanatory variables (ARIMAX). The goal usually involves analyzing data and providing insights. However, they are also used for EPF. One of the features of these models is their transparency, as the outputs provided can easily be traced. This traditional approach is based on simple techniques and linear processes.

Seasonal Autoregressive Integrated Moving Average (SARIMA), is a statistical model with the ability to capture seasonal effects in the original time-series, making it suitable for time-series forecasting problems that show seasonality. Given the nature of the time-series it was necessary to change the approach from ARIMA to SARIMA. The Box-Cox methodology was followed to identify the optimal SARIMA models for this short-term EPF study. Let us first introduce the most general form of the ARIMA model is as follows (G. E. P. Box et al., 2016):

$$\varphi(B)z_t = \phi(B)\nabla^{\mathrm{d}}z_t = \theta_0 + \theta(B)a_t \qquad E\,1$$

Where

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p \qquad \theta(B) = 1 - \theta_1 B - \theta_2 B^2 - \cdots - \theta_q B^q \qquad E\,2$$

Being $\phi(B)$ the autoregressive, $\varphi(B)$ the generalized autoregressive and $\theta(B)$ the moving average polynomials. These polynomial operators have a degree of orders p. d and q, respectively.

Nevertheless, the SARIMA model include the seasonal effects in time-series data, therefore resulting in the following equation (Weron, 2014):

$$\varphi(B)\phi(B^s)\nabla^{\mathrm{d}}\nabla^{D}z_t = \theta(B)\Theta(B^s)\varepsilon_t \qquad E\,3$$

Nevertheless, and as for this case study, the SARIMA model will consider three seasonalities: regular, daily and weekly, using the *msarima* function in R. To identify the best combinations of hyperparameters of the model, the previous year of observations will serve as a training set. Then, as explained before, three months corresponding to the period between August and October 2019 will constitute the test set. It must be noted that the general scheme of the description of the SARIMA model will follow the Box-Jenkins methodology.

However, before the ARMA hyperparameters are selected, we must ensure that the electricity price time-series is first stationary. The autocorrelation (ACF) and partial autocorrelation (PACF) plots are a convenient way to check the stationarity of the time-series. They will be included in the process of identifying the necessary transformations for the time-series.

Before fitting the model, the data must meet the necessary statistical assumptions. As it can be seen in the data, the variance could present heteroscedasticity, meaning that the variance could not be constant enough over time. When the data show different variations, a transformation can be useful. The square root, cube root and the logarithm compose the family of the Box-Cox transformations (Rob J Hyndman, 2014). Named after statisticians George Box and Sir David Cox, it includes powerful means to transform the electricity price data into a variable that follows a normal distribution and has a stable variance. As the studied electricity prices are positive, the equation followed to perform the potential transformation is as follows:

$$y^{(\lambda)} = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda} & (\lambda \neq 0) \\ \log \lambda & (\lambda = 0) \end{cases} \qquad E\ 4$$

This equation is slightly preferable for theoretical analysis as it is continuous at λ=0 (G. E. Box & Cox, 1982).

The decision of applying the Box-Cox transformation is validated with the so-called 'guerrero' method, which has the goal of stabilizing the variance over time. It offers a systematic approach by using local variances to weight observations and then applying the Box-Cox transformation. The optimal lambda is selected by minimizing the variance of the weighted moving averages (Guerrero & Perera, 2004).

Once the variance of the data has been ensured stable, it would be time to check the stationarity, using the KPSS test. The Kwiatkowski-Phillips-Schmidt-Shin test (KPSS) is based on the idea of decomposing a time-series into a deterministic trend, a random walk, and a stationary error term. The null hypothesis of the KPSS test is that the data is stationary, while the alternative hypothesis states that the series is non-stationary. In ARMA models, the stationarity is a crucial assumption, implying that the data's mean variance and autocorrelation does not vary over time. For this case study, the electricity price time-series has been implemented a regular, daily, and weekly seasonal differences. This decision was validated by the autocorrelation plots, as the correlation pattern on the training data, in the presence of a unitary root, is a very slow decay. In cases where there is a predominant seasonality, it must be first applied, before the regular. The equation behind a seasonal difference in a time-series y(t) is as follows:

$$(1 - B^{s})y(t) = y(t) - y(t - s) \qquad E\ 5$$

Where B is the backshift operator and s, the considered seasonal lag.

After a stationary time-series has been achieved, it is time to select the optimal hyperparameters, based on the resulting autocorrelation plots on the training period. The decision process is explained in detail on subsection 4.1. , where the SARIMA implementation for this EPF study is described.

Finally, after training the model, the forecast residuals will be analyzed, through the residuals' autocorrelation plots and their resulting Ljung-Box serial correlation test, to ensure the residuals' independence. The Ljung-Box test checks whether any of a group of autocorrelations are different from zero, instead of testing randomness at each distinct lag. This is why it is often referred to as a 'portmanteau' test (NIST. U.S. Commerce Department, 2003). The Ljung-Box test is a statistical test, whose null hypothesis is that the data is random and the alternative hypothesis is that it is not. The equation behind the it is as follows:

$$Q_{LB} = n(n+2)\sum_{j=1}^{h}\frac{\rho^2(j)}{n-j} \qquad E\ 6$$

Where n is the sample size, ρ_j is the autocorrelation at lag j, and h is the number of lags being tested. Once the residuals have been analyzed, the proper conclusions will be developed.

2.3. PROPHET MODEL

This study includes the recent model developed by Facebook in 2017: Prophet. It is as recent as the award-winning neural ODE paper, and due to its characteristics, it has been selected as a benchmark model. Facebook realized that forecasting methods that are entirely automated can sometimes lack resilience and they are frequently not capable enough to integrate useful assumptions or heuristics. Moreover, the ability of providing accurate forecasts is quite rare is can only be seen in people with substantial experience (Taylor & Letham, 2017).

Prophet was optimized for the business problems that were encountered at Facebook, which involved data with strong seasonality, missing a sensible number of observations and that followed non-linear trends. In essence, Prophet could be described as a generalized additive regression model (GAM) that makes use of four components of the form:

$$y_t = g(t) + s(t) + h(t) + \varepsilon_t \qquad E\ 7$$

where g(t) describes a linear growth trend, s(t) describes the seasonal patterns, h(t) captures the holiday effects and ε_t is a white noise error term.

The main essence of the model is to fit the time-series trajectory, unlike other benchmark models that explicitly account for the temporal dependencies of the data. This formulation provides a number of practical advantages (Taylor & Letham, 2017):

- Flexibility: Seasonality can be easily introduced to the model, in a very accessible way to the user. No matter the knowledge of the user, its configuration makes it easy to extend the model and add new components.
- Robustness: The model work accurately even on non-regularly spaced observations. Therefore, traditional descriptive analysis tasks can be omitted.
- Speed: Prophet is highly computational efficient, which allows it to fit the data very fast.

The Prophet components start with the linear trend, g(t). This trend captures the non-periodic fluctuations in the time-series. Depending on the specific use case, it can be either linear or logistic. In the linear case, it is a piecewise linear function of time. However, for the logistic trend, it models a logistic growth that saturates at a certain capacity. The equations behind g(t) are as follows:

$$Linear: g(t) = (k + a(t)^{\intercal}\delta)t + (m + a(t)^{\intercal}\gamma) \qquad E\ 8$$

Where before k is the growth rate, δ has the rate adjustments, m is the offset parameter, and γj is set to −sjδj to make the function continuous.

$$Logistic: g(t) = \frac{C}{1 + \exp\left(-k(t-m)\right)} \qquad E\ 9$$

With C the carrying capacity, k the growth rate, and m an offset parameter.

The choice between a linear or logistic (logarithmic) trend in the Prophet model validated by the nature of the data. As there is no evident or predictable carrying capacity for the electricity prices, a linear trend can be a more appropriate choice. Moreover, the time-series does not provide a clear saturation point.

Prophet's target time-series involve business, which very often show strong seasonalities. The seasonality is broken down into a Fourier series with periodic effects. Like the following:

$$s(t) = \sum_{n=1}^{N}\left(a_n * \cos\left(\frac{2\pi nt}{P}\right) + b_n * \sin\left(\frac{2\pi nt}{P}\right)\right) \qquad E10$$

Being P the period of the data, which for this study is 24. Fitting these seasonality component means estimating the 2N parameter $\beta = [a_1, b_1, \dots, a_n, b_n]$, which is done by constructing a matrix of seasonality vectors for each *t* in the historical data (Taylor & Letham, 2017).

Moreover, the holidays h(t), will represent the predictable effects of theses specific events. For each holiday, Prophet creates a window around the date and models the effect using a piecewise linear function.

Finally, an error $\varepsilon(t)$ represents a white noise term, assumed to be normally distributed with mean 0.

Prophet leverages a Monte Carlo simulation approach to fit the data. By running multiple simulations with sampled parameters, Prophet generates forecasts, along with their uncertainty intervals. This represents a major advantage of Prophet, as its Bayesian framework allows it to produce full probability distributions over future outcomes.

2.4. MACHINE LEARNING METHODS

On the other hand, ML methods are based on more complex techniques and nonlinear processes. Some of the most notable developments involve the use of LSTM networks, a type of recurrent neural network (RNN) that can effectively model long-term dependencies and capture temporal patterns in data. LSTMs have been widely used for time-series forecasting, natural language processing and other tasks that involve sequential data. Other ML forecasting techniques include gradient boosting, random forests, and support vector machines (SVMs).

The EPF community is also developing hybrid models, which consist of a combination of multiple ML methods to improve accuracy, as they allow for the benefits of multiple methods to be combined into a single model. For example, a hybrid method could consist of an MLP model to capture complex patterns and a statistical method to analyze the temporal dependencies of the data, both long and short-term. For this thesis, a combination of a convolutional neural network and an LSTM model will be developed. The synergy between these methods is known for providing a flexible and adaptive way of learning patterns, especially in data with significant noise.

As EPF best practices indicate, the best loss function depends on every model and forecast configuration. However, the most consistent one has proven to be the MAE. Therefore, the loss function used is MAE of the test set, as implemented in PyTorch's nn.L1Loss.

2.4.1. MLP MODEL

In response to the electricity price time-series complexity, this study implements an MLP model in PyTorch, a type of artificial neural network which is very well suited for the task, as it can model complex and non-linear relationships. Multi-layer Perceptron neural networks are composed of fully connected neurons. The MLP architecture is shown below:

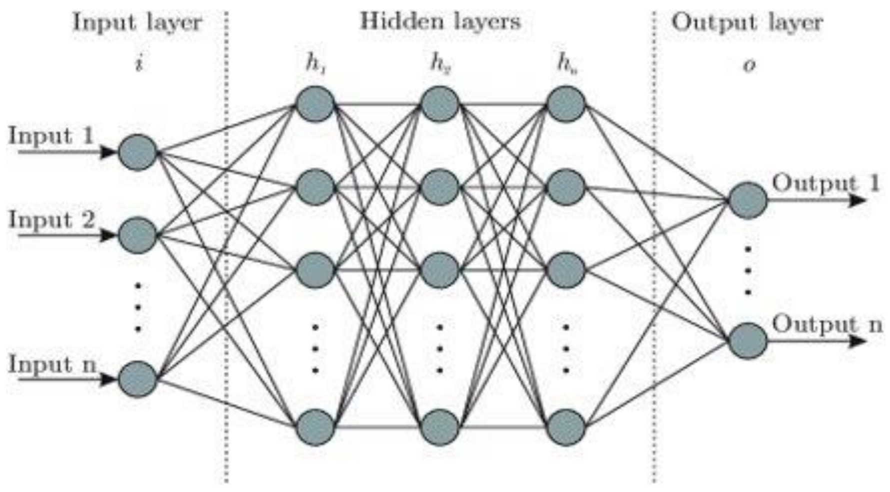


Figure 8: Architecture of an MLP model

The process of training an MLP model implies finding the appropriate weights for each neuron in the net, and the way to achieve it is by backpropagation. In the training process, the multilayer perceptron receives an input vector and given the current set of weights and biases, it provides an output through the layers until the final layer. Once the output has been calculated, it is then compared with the actual values and the loss is obtained. The backpropagation algorithm will adjust the weights in the direction of a lower loss function value. Consequently, after a certain number of iterations, the model can capture the nonlinearities of the time-series and can generalize to provide accurate forecasts with unseen data. The equation that represents the calculation performed in each neuron is as follows:

$$z = f(b + x * w) = f\left(b + \sum_{i+1}^{n} x_i * w_i\right) \qquad E\ 11$$

The nonlinearity is introduced in the model by the activation function $f()$, which comes right after the neuron calculation. It is critical for the ability of modelling complex relationships in the data. As it will later be explained, the selected activation function is ReLU, due to its computational efficiency.

2.4.2. LSTM MODEL

Although MLPs represent a suitable option for the task of forecasting, they struggle with temporal or spatial data, where it is important to capture the relationships across different points in time or space. For this matter, this study includes neural networks with LSTM cells.

LSTM networks are a type of Recurrent Neural Network (RNN), which have proven to be very effective in modelling sequential data, given their ability to remember long-term data.

In order to process sequences of inputs, traditional recurrent neural networks use their internal state, which constitutes its memory. This allows the models to remember and consider patterns in past data. However, the vanishing gradient problem difficulties this process of remembering, as in each iteration the internal states are modified and can lose understanding the past dependencies.

LSTMs were designed to tackle this problem, by having four gates interacting to obtain the final cell state. The net can use its gate units to decide when to keep or override information in the memory cell, decide when to access it and how to prevent other cells from being perturbed by it (Hochreiter & Schmidhuber, 1997).

In Figure 9, the four gates of the LSTM cell are represented:

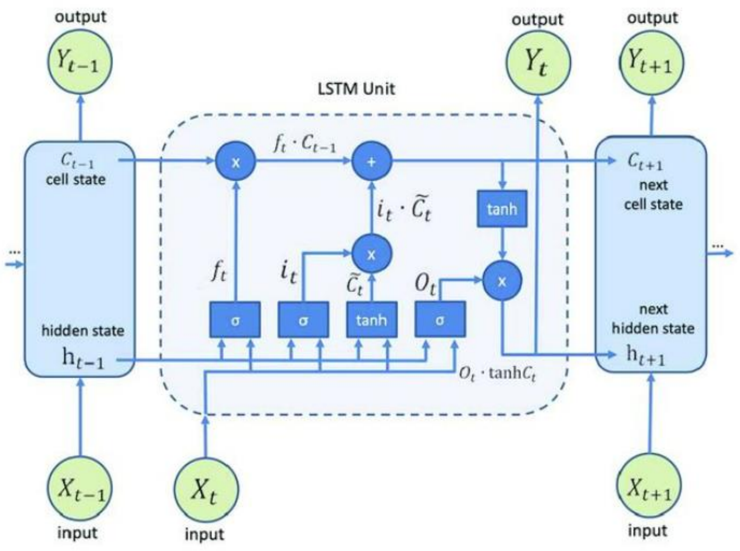


Figure 9: LSTM neuron configuration

At the top of the diagram, the horizontal line represents the main carrier of the information down the entire chain of sequential neurons, which determines the cell state. From below, these four gates can optionally interact with the cell state.

The first gate is the forget gate. Looking at the current input of the cell state, it determines whether to forget pertinent information. In other words, the LSTM cell understands the information that will not be needed in the following cell.

Afterwards, the input gate updates the cell state with new information from the current input. It is composed by two parts: the sigmoid layer and the hyperbolic tangent layer. The sigmoid layer is responsible for deciding which values to update, while the tanh layer develops a vector of candidate values that could be added to the cell state.

Finally, the output gate determines the value of the following hidden state by controlling which pieces of information of the current state to output. While in the MLP model, the backpropagation optimizes the weight and bias of each neuron, in the case of LSTM neural networks, it involves learning the weights for these gate layers. The forward propagation process of an LSTM cell is described by the following formulas:

- Forget gate

$$f_t = \sigma_g * \left(W_f * x_t + U_f * H_{t-1} + b_f\right) \qquad E\ 12$$

- Input gate

$$i_t = \sigma_g * (W_i * x_t + U_i * H_{t-1} + b_i) \qquad E\ 13$$

  - ➢ Cell input activation vector

$$\tilde{c}_t = \sigma_c * (W_c * x_t + U_c * H_{t-1} + b_c) \qquad E\ 14$$

- Output gate

$$o_t = \sigma_g * (W_o * x_t + U_o * H_{t-1} + b_o) \qquad E\ 15$$

- Cell state

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \qquad E\ 16$$

Hidden state

$$h\_t = o_t \odot \sigma_c(c_{t-1}) \qquad E\ 17$$

being $c_0 = 0$ and $h_0 = 0$ are the initial values and $\odot$ represents the element-wise product. x_t is the input vector of the LSTM cell. f_t is the forget gate's activation vector. i_t is the input gate's activation vector. o_t is the output's gate activation vector. h_t is the hidden state of the LSTM cell. $\tilde{c}_t$ is the cell input activation vector. c_t is the cell state vector. W, U and b are the corresponding weights and bias matrices. σ_g represents the sigmoid function. σ_c represents the hyperbolic tangent function. σ_h represents the sigmoid function.

The hidden states are discretely updated in each time-step, not allowing a continuous trajectory as the neural ODE, which will later be explained. The way LSTMs work with hidden states is represented as follows (Kidger et al., 2020).:

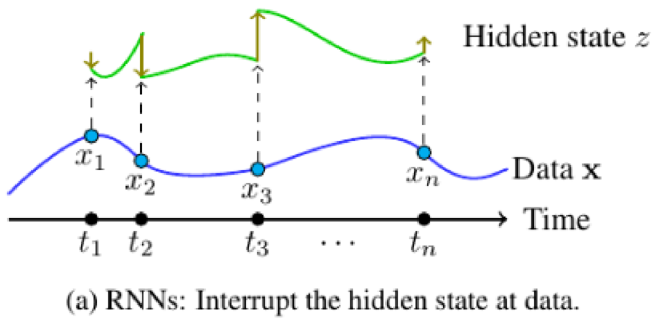

Hidden state z
x_1
x_2
x_3
x_n
Data x
Time
t_1 t_2 t_3 $\cdots$ t_n
(a) RNNs: Interrupt the hidden state at data.

Figure 10: Hidden states in RNNs (Kidger et al., 2020)

As we can see, the hidden state is computed at each time-step in the sequence. This discrete process allows the LSTM to maintain a certain temporal continuity, which allows the model to remember patterns from the past. Afterwards, it will be explained how in Neural ODEs, the hidden states are seen as continuous trajectories.

On the one hand, the need of updating more parameters implies the ability of LSTM models to model more complex temporal dependencies. However, they can then be very computationally intensive and prone to overfitting, especially on small or noisy data. Therefore, as in the rest of the neural network models, certain strategies will be implemented to avoid this problem, such as dropout, early stopping and careful hyperparameter tuning.

2.4.3. CNN-LSTM MODEL

The last DL benchmark model will constitute a hybrid model, consisting of a 1D Convolutional layer preceding an LSTM model, which combines the strengths of both convolutional and recurrent neural networks.

These effective models have been widely used in the field of ML. Combining both, a hybrid CNN-LSTM model is defined, which brings together their abilities to capture temporal and spatial characteristics in the data, in this case, the electricity price time-series. The idea is to provide an additional feature that can help extract information from the data.

A convolution is a mathematical operation performed by a sequence of learnable filters, also known as kernels, which slide through the input data from left to right. The electricity price elements passing through the filter are multiplied elementwise with the corresponding filter parameters and the results are added up. This scanning process allows the model to find matching patterns.

For the univariate case, the output of a convolution is another univariate series that underwent a filtering process (Ismail Fawaz et al., 2019). Thus, applying filters, results in a multivariate time-series whose dimensions are equal to the number of filters used (in this case 64). In each new epoch of the training, the filter's parameters are updated and become more able at recognizing patterns and structures in the data. Moreover, each filter is focused on detecting a different type of feature.

The convolution approach has been the same in both univariate and multivariate approaches: using a 1D convolution. In the context of time-series forecasting, it is common to treat the exogenous variables as how additional color channels in image data. Unlike images, the filters exhibit only one dimension (time), instead of two dimensions (width and height) (Ismail Fawaz et al., 2019). This way, a 1D convolution is typically used, with multiple input channels, instead of a 2D convolution. This is because these variables do not have a spatial relationship that a 2D convolution would exploit. The model architecture would result as follows:

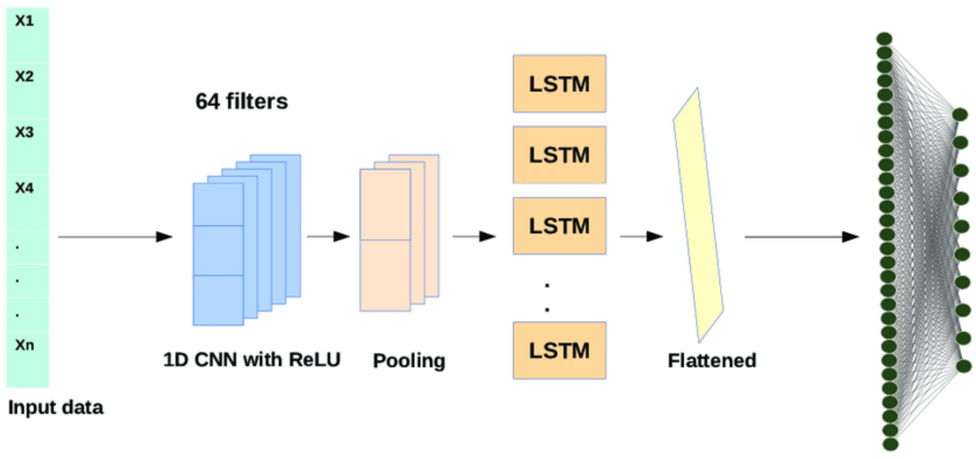


Figure 11: CNN-LSTM model architecture (Hamad et al., 2020)

Although the other benchmark models have performed better being implemented in the PyTorch library, this model has proven to work better using the TensorFlow libraries. The model definition is as follows:

```
# Model definition
CNN_LSTM_model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                      strides=1,
                      padding='causal',
                      input_shape=[window_size, 4]),
  tf.keras.layers.LSTM(500, return_sequences=True),
  tf.keras.layers.LSTM(500, return_sequences=True),
  tf.keras.layers.LSTM(500),
  tf.keras.layers.Dense(24),
])

CNN_LSTM_model.summary()
```

At the beginning of the model lies the convolutional layer. The parameters use 64 filters or kernels, each specialized in a certain type of feature of the input sequence. Each kernel is set to 3, as it performs the convolution in sequences of 3 input elements. The stride is set to 1, which results in convolutional computations spaced 1 input unit. This allows the model to capture all potential features from the data without skipping any prices. The padding is set to causal, so that the model is restricted from looking into future data. This is especially important for time-series data as the test set data is unknown. This padding parameter also ensures that the output size is the same as in the input.

After the convolutional layer, the data passes on to the three LSTM layers, which will focus on its temporal dependencies. Each LSTM layer is composed by 500 neurons, which will allow the model to learn the complex features of the electricity price time-series.

The final Dense layer is responsible for mapping the LSTM outputs to the final forecasts, corresponding to the following day's hourly prices. This way, although the number of neurons in the hidden layers are arbitrary, the input and output layers will have a number of neurons that will correspond to the size of the input and output window, respectively.

2.5. NEURAL NETWORK OPTIMIZATION

The goal is to obtain the most accurate model, i.e., the one that can provide the lowest value for the loss function, in this case, the MAE. This can only be achieved through continuous adjustment over the parameters. In what follows, we will discuss certain parameters that have been navigated to yield the lowest loss difference between the forecasted prices and the actual values.

2.5.1. TRAINING STRATEGY

The objective is to accurately forecast the following 24 hours of prices. So, in this aspect, the output size is fixed. As the neural ODE will train with the prior week of data, this parameter will also apply for the DL methods, as the goal of the project is to compare the performance of these models. Choosing the previous 168 observations for the price as input to forecast the following 24 hours, allows the model to exploit relevant historical information while also maintaining resilience to adapt to possible changes in trends or seasonality patterns over time.

The training data will span from March 1st, 2018, to April 30th, 2019, which constitutes about 70% of the total data. This set allows the model to learn the underlying patterns and structures of the electricity price time-series. It must be noted that there is a trade-off about the size of the training set. The larger it is, the more information is going to provide. However, as the market is continuously growing and adapting to the new times, it does not make the most sense to use huge amounts of training data. As the patterns change with every modification in the configuration of the Spanish energy sources, the evolution of the cost of fossil fuels, etc., it is important to assign more importance to the recent data, consequently not using a huge total dataset. Thus, the selected size of the training data has been assigned to 14 months.

On each epoch in the training set, the model learns the dynamics of the data. However, that does not mean that it will therefore automatically perform better in the test set. Therefore, a validation set is included as a check for overfitting, which will be used to select the resulting hyperparameters of the neural networks. After an arbitrary number of training epochs, the model will validate its efficacy on the validation set. Once the training has finished, the model with the lowest validation error is selected for the final test set. Spanning from May 1st, 2019, to July 31st, 2019, the validation set constitutes 3 months to allow the model to fine-tune its parameters to improve accuracy on future unseen data.

Finally, the test set, which spans from August 1st, 2019, to October 31st, 2019, will be used to evaluate the model's ability to generalize unseen data and provide a forecast for each test window's following 24-hourly prices. It must be noted that the models will be tested within the same period for comparing their accuracy.

This sequential methodology in the dataset split represents a robust way to ensure that the model has enough data to adapt to the new patterns in the price time-series. It must be noted that, although it is appropriate in other ML problems, shuffling the data would destroy the temporal structure. The training-validation-test split would be described in Figure 12:

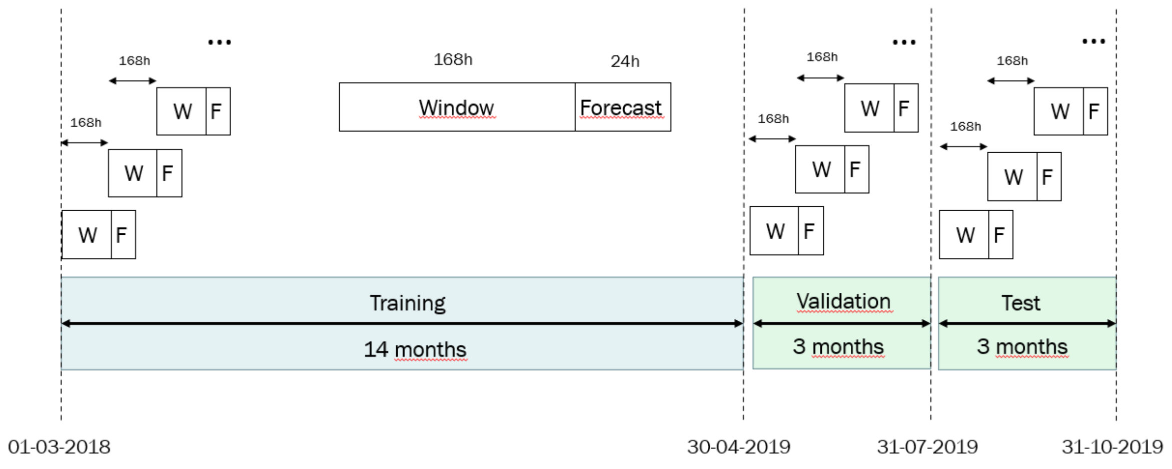


Figure 12: Training-validation-test split

As the neural ODE has proven computationally slow on CPU, it has been appropriate to develop a model for each day of the week. Consequently, except for the statistical (SARIMA) and probabilistic (Prophet) methods, the benchmark models and the neural ODEs will be developed for every day of the week, meaning that there will be seven models per method.

2.5.2. DATA NORMALIZATION

Data normalization is an important preprocessing technique, as it helps neural networks to converge faster and avoid getting stuck in local minima during the training process. It is used to ensure that the time-series falls withing the same scale without losing the data patterns and structure.

Data normalization is key for the learning mechanism of neural networks. They rely on gradient-based algorithms, such as Stochastic Gradient Descent (SGD), that allow the model to update its parameters. If the data has not been normalized the loss function contour may become much more challenging. This could lead to a back-and-forth oscillation that will make the model converge slower or not converge at all.

For this project, both standardization and Minmax scaling of the dataset were tried, providing the latter, better results. It must be noted that the scaler is fit with only the training data, so that the validation and test set serve as unseen data that will help obtain the optimal parameters.

Moreover, scaling from 0 to 1 can mitigate the effect of outliers in the electricity price time-series, contributing to a more robust and accurate model. It also provides a more circular loss function contour, allowing the model to find the global minimum more efficiently.

2.5.3. ACTIVATION FUNCTION

Activation functions fundamentally influence the networks' ability to adapt to complex patterns. The most common ones include ReLU (Rectified Linear Unit), sigmoid, and tanh (hyperbolic tangent). They are placed after a hidden layer so that they can transform the weighted sum of the inputs before passing it to the following layer.

The resulting output value helps in thresholding, i.e., deciding on whether to activate a certain neuron. This is especially crucial to allow neural networks to filter and select the most significant patterns in the data and avoid the influence of noise.

For this study, a ReLU activation function has been implemented as it provided the best results. The ReLU function can be described as max $(0, x)$, it is represented in Figure 13:

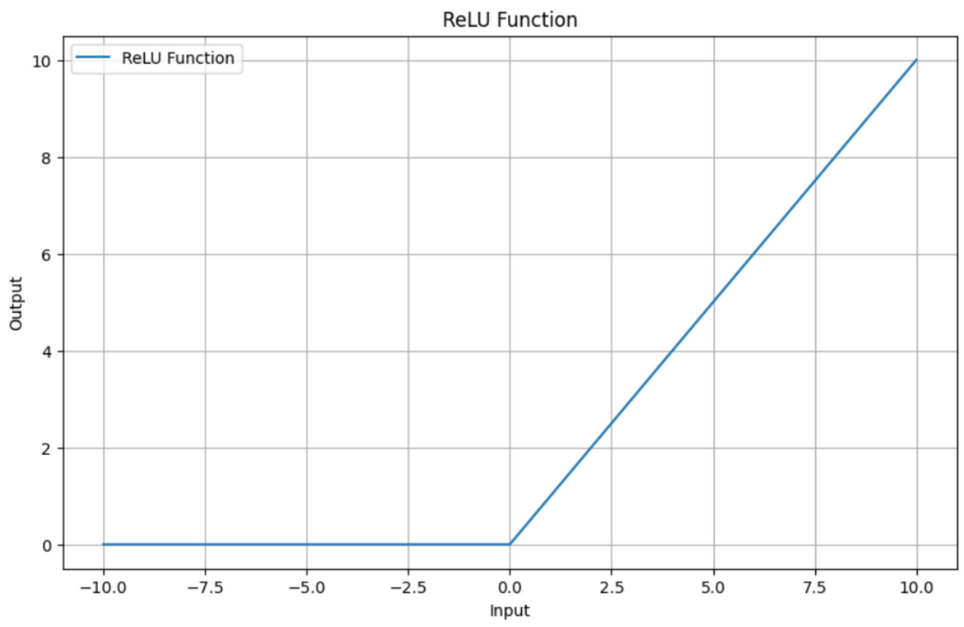


Figure 13: ReLU activation function

2.5.4. LEARNING RATE SCHEDULE

The algorithm responsible for the numerical optimization of the weights during backpropagation is called Gradient Descent, which follows this rule:

$$w_i = w_i - \alpha \frac{dy}{w_i} \qquad E\ 18$$

The weights w_i in the neural network are updated considering the differentiation with respect to the objective, which represents the slope of the loss function. They must be adjusted so that the value of the loss function is minimized.

The step size of the weights update is determined by the learning rate α. Using a higher learning rate will update the weights more drastically and lead to a faster training, although the value of the loss function may oscillate and not find the minimum. On the other hand, using a lower learning rate will lead to a slower training but finding the global minimum will be more likely. Therefore, it is crucial to find the best learning rate for our EPF study.

Although considering a fixed learning rate is a valid option, there is a better way of optimizing this parameter called Learning Rate Schedule. The idea behind it is to start assigning a high learning rate at the beginning, when the weights have just started to be optimized and are not yet capable of capturing the patterns of the series. Progressively, as the model keeps training on each new epoch, a lower learning rate is assigned. Hence, faster learning can be forced at the beginning to help the optimizer to converge and find a more accurate minimum of the loss function.

The Learning Rate will generally start at 0.1 and decrease linearly over the total number of epochs, which largely depends on the model. This type of learning rate schedule is known as a linear learning rate decay. At the start of the training process, i.e., on epoch 1, the high value of the learning rate $\alpha = 0.1$ allows the model to take larger steps and advance faster towards convergence in the minimum of the loss function contour.

By the end of the training, the model needs a smaller value for converging, as it is approaching the minimum. The small value of the learning rate $\alpha = 0.01$ helps to make fine-tuned adjustments to the parameters. In Figure 14 the resulting learning rate schedule is represented:

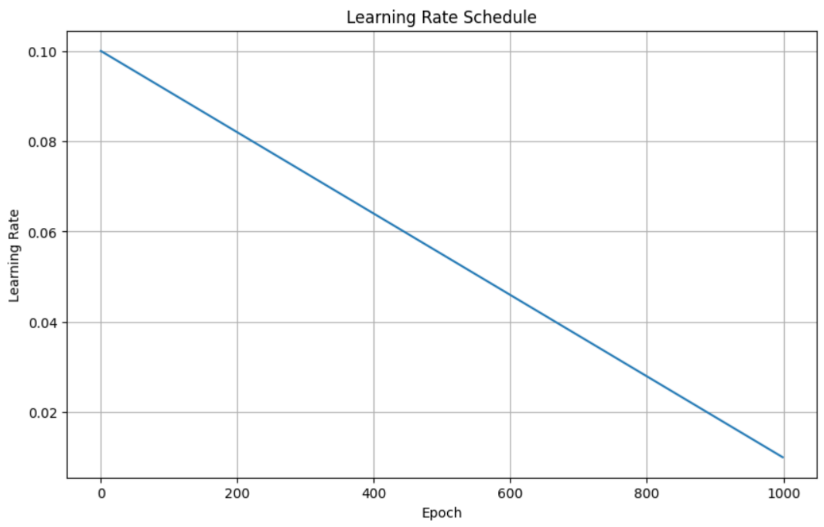


Figure 14: Learning Rate Schedule

2.5.5. EARLY STOPPING

Selecting the number of epochs for training a model is a significant challenge. Choosing a high number of epochs can ensure finding the best set of parameters, but it would surely lead to problems with overfitting and wasted computation. Early Stopping is an optimization technique used to reduce overfitting, which considers the validation error during training.

As the model trains on each new epoch, the training error decreases. This happens as well on the validation set, up to a certain point, where increasing epochs not only does no longer improve the forecasts accuracy, but the validation error starts increasing. It is at this point that the model should stop training and avoid overfitting.

In this aspect, early stopping allows the model to stop training once an arbitrary number of epochs have passed, after the model has found a minimum in the validation error function. However, it could also be implemented that the Early Stopping acts when the rate of change of the validation error is below a certain threshold.

Represented below, is the case of the univariate LSTM training and validation process. As it can be seen, the model stops when an arbitrary number of epochs have passed without outperforming the validation error:

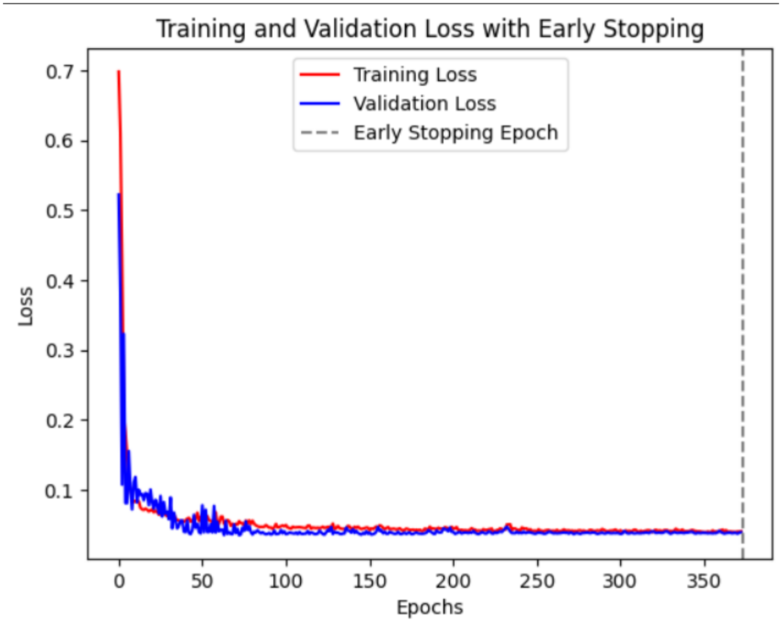


Figure 15: Early Stopping on the LSTM model

2.5.6. OPTIMIZER

The optimizers are responsible for updating the model's parameters during backpropagation to improve accuracy. The ultimate goal is to find the global minimum of the loss function contour, i.e., the minimum MAE between the forecasts and the actual values.

The most common optimizer is the SDG, which can iteratively identify the direction that decreases the most the loss function and update the parameters accordingly. This direction is provided by the negative of the gradient of the function.

SGD is different from the traditional Gradient Descent algorithm, as it does not compute the gradients on the whole dataset, but on a small batch of randomly chosen observations. Thus, making the model faster and more efficient with large datasets.

However, SGD has its limitations. The stochastic nature of the model often leads to significant variances in the parameters' updates. Therefore, converging may not be as fast and could ultimately oscillate around the minimum without reaching it. For this matter, this study includes an SGD optimizer with momentum.

This characteristic allows the model to better navigate through the steep contours, i.e., where the slopes are much different than in other directions. It helps accelerate convergence and dampen oscillations by adding a fraction γ of the past time-step to the current vector (Ruder, 2016). The equation behind the performance of the SGD with momentum is as follows:

$$\theta = \theta - v_t \qquad \textit{E 19}$$

$$v_t = \gamma * v_{t-1} + \eta * \nabla_\theta * J(\theta) \qquad \textit{E 20}$$

The momentum parameter will have a value around 0.9, as it is commonly set. In Figure 16, it is represented how the optimizer works with and without momentum:

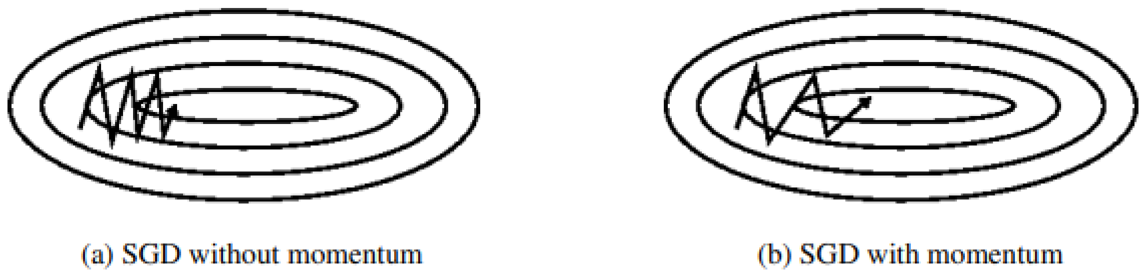


Figure 16: SDG optimizer (Ruder, 2016)

Chapter 3. NEURAL ODE

3.1. INTRODUCTION

Five years ago, in 2018, Ricky T. Q. Chen and his colleagues published the paper introducing a new family of deep neural network model, so-called neural ODEs. It received the best-paper award in NeurIPS in 2018. The main idea behind it is that instead of specifying a discrete sequence of hidden layers, the derivative of the hidden state is parameterized using a neural network. The output of the network is computed using a black-box differential equation solver.

Neural ODEs are a type of ML model that can learn complex dynamics of data. They are an extension of traditional neural networks, which are used to approximate complex functions by composing simpler functions together. Essentially, they are a continuous version of the RNN models, where the hidden state at a given time is modeled as a function of time, instead of a sequence of discrete time-steps. In a neural ODE model, it is assumed that the learned function governs the dynamics of an ODE, which represents the evolution of an observable over time.

By combining the flexibility of deep neural networks with the mathematical properties of ODEs, Chen and his colleagues demonstrated that Neural ODEs could accurately model complex, dynamic systems, outperforming traditional approaches in certain scenarios.

The award-winning paper (R. T. Q. Chen et al., 2018) compares the performances of a ResNet (Residual Network) and a neural ODE. ResNets are a type of neural network characteristic for their 'shortcut connections' (He et al., 2016), which allows the gradient to be more rapidly backpropagated to the first layers. The idea is to mitigate the effect of degradation, which causes the accuracy to get saturated as the network depth increases.

Figure 17 shows the comparison in the residual flows between a ResNet (Residual Network) and a neural ODE. As we can see on the left, the residual network defines a discrete sequence of finite transformations or evaluation locations, each of which taking place in the hidden layers. On the right, the plot represents how the ODE network defines a vector field, which continuously transforms the state:

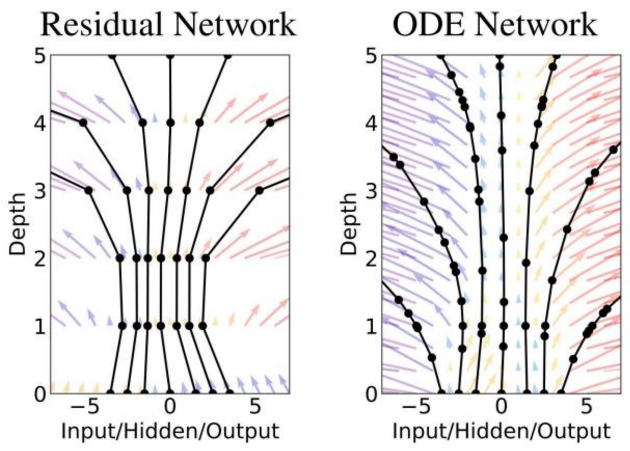


Figure 17: Evaluation locations in a residual network and ODE network

On the left part of Figure 17, the residual network presents a sequence of transformations to a hidden state (h_t), which will eventually form the computed output, in the output layer:

$$h_{t+1} = h_t + f(h_t, \theta_t) \qquad E\ 21$$

These iterative updates can be seen as a Euler discretization of a continuous transformation. This iterative scheme provides an approximate solution, through a local linear computation in discrete steps, to compute the future states of the system's dynamics. Euler's methodology is shown below:

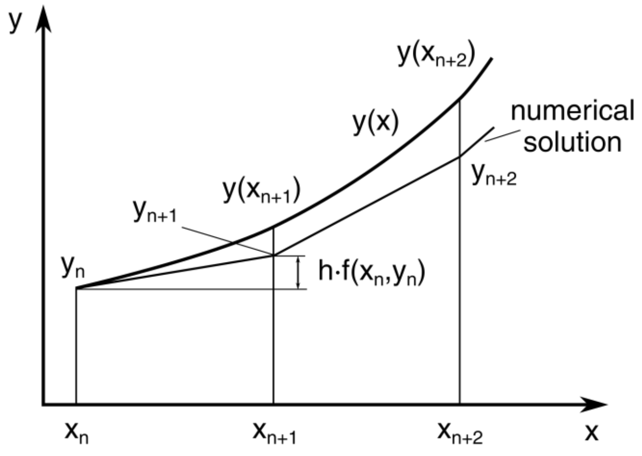


Figure 18: Euler's method

In a neural ODE, as seen on the right part of Figure 17, these evaluation locations are continuous. The effect would resemble having many layers and smaller steps between them. Therefore, the continuous dynamics of hidden states is parameterized using an ODE:

$$\frac{dh(t)}{dt} = f(h(t), t, \theta) \qquad E\ 22$$

The innovative idea behind a neural ODE is to avoid the traditional backpropagation. Differentiating through the operations of the forward pass is straightforward, though it accumulates numerical error and can be computationally expensive (R. T. Q. Chen et al., 2018). Figure 19 represents the reverse-mode differentiation of an ODE solution, as an augmented ODE is solved backwards in time by the adjoint sensitivity method:

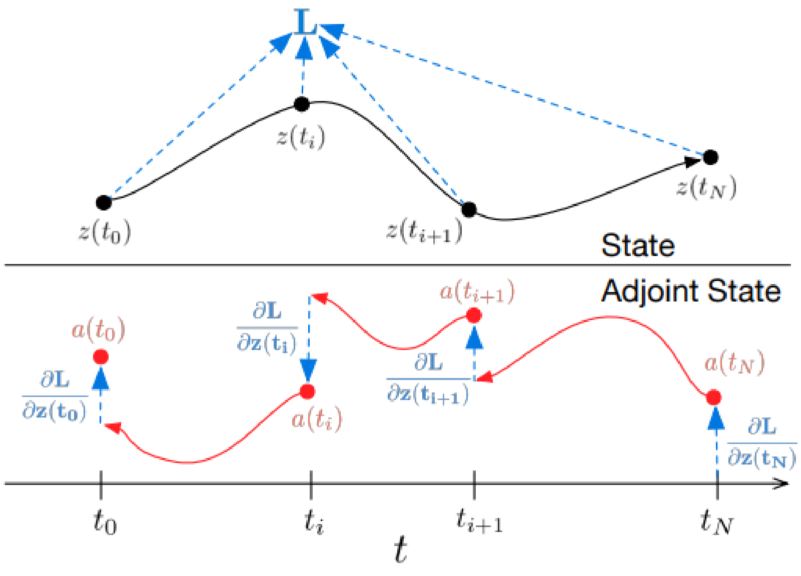

L
$z(t_0)$
$z(t_i)$
$z(t_{i+1})$
$z(t_N)$
State
Adjoint State
$a(t_0)$
$\frac{\partial L}{\partial z(t_0)}$
$\frac{\partial L}{\partial z(t_i)}$
$a(t_i)$
$a(t_{i+1})$
$\frac{\partial L}{\partial z(t_{i+1})}$
$a(t_N)$
$\frac{\partial L}{\partial z(t_N)}$
t_0
t_i
t_{i+1}
t_N
t

Figure 19: Forward and backpropagation of a neural ODE

The adjoint method allows to compute the gradients for any ODE solver, which is treated as a black box, not needing access to the solver's internal operations. Consequently, it scales linearly in computation and memory with the size of the problem, in contrast with traditional backpropagation.

One key element of a neural ODE, like most ML models, is its concept of the loss function. It is a scalar value function, which measures the difference between the model's predictions and the actual values. Optimizing this loss function is an iterative process, with the goal of being able to accurately model the dynamics of the data.

While the optimization is performed by the optimizer (subsectionParte I2.5.6.) which is used twice in every training step – once in the forward propagation to calculate the output (price forecasts of the following day) and during the backpropagation, to compute the gradients.

In a neural ODE, we start from an input layer, denoted by h(0) and we define the output layer, denoted by h(T), which represents the solution to an ODE at a certain time T. More specifically, it is the solution of an Initial Value Problem (IVP), a type of ODE where we know the value of the solution at a specific time. In order to solve this IVP, an ODE solver is needed. This numerical method calculates an approximation of the solution h(t) at discrete time-steps by repeatedly evaluating the derivative of the dynamics of the data.

Training a model by computing gradients of a loss function L with respect to the parameters θ is challenging. A possible approach would be to backpropagate through every step of the ODE solver, but as mentioned before, it incurs high memory cost. Instead, the *adjoint sensitivity method* provides a more efficient way to compute the gradients, by solving a second, augmented ODE backwards in time, applicable to all ODE solvers. The underlying equation is as follows:

$$L\big(z(t_1)\big) = L(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt\,) = L\big(ODESolve(z(t_0), f, t_0, t_1, \theta)\big) \qquad E\ 23$$

To optimize the loss function, the gradients with respect to θ are required. The first step is to understand how the gradient of the loss depends on the hidden state z(t) at each point in time. This quantity is called the adjoint, denoted by a(t), and is mathematically represented as ∂L/∂z(t). Its dynamics can be thought of as the instantaneous analog of the chain rule:

$$\frac{da(t)}{dt} = -a(t)^T * \frac{\partial \mathrm{f}(\mathrm{z}(\mathrm{t}), \mathrm{t}, \theta)}{\partial\, z} \qquad E\ 24$$

To calculate the adjoint, we can compute ∂L/∂z(t0) calling another time to the ODE solver. The solver will run backwards, starting from the initial value of ∂L/∂z(t1). Nevertheless, to solve the ODE, it requires knowing the value of z(t) along its entire trajectory. However, by simply recomputing z(t) backwards in time together with the adjoint, starting from its final value z(t1), the problem would be solved. Now, computing the gradients with respect to the parameters θ requires calculating a third integral, which depends on both z(t) and a(t):

$$\frac{dL}{d\theta} = -\int_{t_0}^{t_1} a(t)^T * \frac{\partial \mathrm{f}(\mathrm{z}(\mathrm{t}), \mathrm{t}, \theta)}{\partial\theta} dt \qquad E\ 25$$

To summarize the mathematical exposition, the paradigm shift that neural ODEs have caused is truly transformative. Their ability to model the trajectory of a dynamic system as a solution to an ODE allows it to provide a unique, continuous-time perspective. Through the adjoint sensitivity method, the neural ODE performs backpropagation, computing the gradients by solving another ODE in reverse-time.

As the paper (R. T. Q. Chen et al., 2018) demonstrates, the number of evaluations in the backward pass is approximately half than in the forward pass. This goes to show the memory and computational efficiency of the adjoint sensitivity method, compared with the traditional backpropagation.

Because of this, neural ODEs are a very suitable option for dealing with complex dynamics, such as the electricity market. Characterized by its high volatility and dependance on external factors like the demand or the renewable generation, the non-stationary time-series exhibit strong fluctuations. While long-term trends are important within EPF, their effect may be negligible when dealing with accurate short-term forecasts. Thus, the model's flexibility and continuous time modelling capabilities make it a fitting choice to generate short-time predictions.

Other potential opportunities involving neural ODE have been studied in recent scientific papers. In 2020, Patrick Kidger and his colleagues combined neural ODEs with signature methods and demonstrated its efficacy with financial time-series forecasting (Kidger et al., 2020). Moreover, in a paper from the same year (De Brouwer et al., 2020), a variant of the model was implemented, the GRU-ODE-Bayes. It yielded more accurate results than standard RNNs.

Overall, although neural ODEs are a relatively new ML model, their characteristics and promising results suggest that they could provide significant improvements over traditional models in forecasting day-ahead electricity prices. Nonetheless, further research is needed to fully comprehend the intricacies of the model, and other potential real-life applications.

3.2. IMPLEMENTATION

The neural ODE will be developed and implemented in Python. More specifically, in the PyTorch library. Following the selected error evaluation metrics, it will be compared against various benchmark models, previously developed. For the case study, both univariate and multivariate neural ODEs have been coded for forecasting electricity prices.

This master's thesis would not have been possible without the efforts of Chen and Kidger, who kindly developed a Python library called *torchdiffeq*. It provides ODE solvers implemented in PyTorch, allowing backpropagation using the adjoint method.

Through its main interface, *odeint*, the algorithms solve the required IVP, which are composed by an ODE and an initial value. The objective of an ODE solver is to find a continuous trajectory that satisfies both:

$$\frac{dy}{dt} = f(t, y) \qquad y(t_0) = y_0 \tag{E 26}$$

In this example, a simple implementation of an IVP problem in the *torchdiffeq* framework is represented:

```
from torchdiffeq import odeint_adjoint as odeint
odeint(func, y0, t)
```

where *func* represents the derivative of the system, y0 is the initial condition and t the 1-D tensor containing the evaluation points and t0 is assumed to be t[0].

As it was explained, the adjoint method, implemented in *torchdiffeq* as *odeint_adjoint*, is the real advantage in comparison with other models that use traditional backpropagation. The difference between the two approaches is emphasized for long sequences or large batch sizes. Specifically, the adjoint method only needs to store the final state and little additional information, which results in a constant memory usage or O(1). Therefore, independently of the size of the input, the amount of memory used does not vary, which makes neural ODEs very efficient compared to other state-of-the-art methods.

One key point in the adjoint method implemented in this library is that the callable function *func*, which represent the derivative of the trajectory, must be an instance of *nn.Module*. *nn.Module* is a base class for neural networks in PyTorch and, therefore can parametrize the ODE. It includes powerful features such as automatic differentiation and GPU acceleration.

For the sake of simplicity and code re-usability, we define a Python class, ODEFunc, to store the neural network that parametrizes the unknown function governing the time-evolution of the day-ahead electricity price:

```
class ODEFunc(nn.Module):

    def __init__(self, input_size: int, hidden_layer_neurons: int, output_size:
int):
        super(ODEFunc, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, hidden_layer_neurons),
            nn.ReLU(),
            nn.Linear(hidden_layer_neurons, hidden_layer_neurons),
            nn.ReLU(),
            nn.Linear(hidden_layer_neurons, hidden_layer_neurons),
            nn.ReLU(),
            nn.Linear(hidden_layer_neurons, output_size),
        )

        for m in self.net.modules():
            if isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, mean=0, std=0.1)
                nn.init.constant_(m.bias, val=0)

        self.nfe = 0  # Initialize number of function evaluations

    def forward(self, t: torch.Tensor, y: torch.Tensor):
        self.nfe += 1  # Increment the counter each time the forward function is
called
        return self.net(y)
```

It must be noted that an identical network architecture, i.e., a feedforward 3-hidden layer network with ReLU as the activation function, applies for both univariate and multivariate models. Within the class constructor, the neural network is stored in the class attribute net. Hence, *self.net* is composed by a series of linear transformations and activation functions:

- nn.Linear(input_size, hidden_layer_neurons): It denotes a fully connected layer, which takes as input the state from the preceding layer and produces a linear transformation for the following layer. This feature is characteristic of MLP models.
- nn.ReLU(): ReLU is the selected nonlinear activation function. The theory of backpropagating through ODEs technically requires that the vector field (and thus the activation function) be continuously differentiable, which ReLUs are not. However, despite this theoretical point, ReLU activations are often successfully implemented (Kidger, 2022), and so is the case.

As it has been proven, initializing the neural vector fields close to zero improves the training. This is accomplished by initializing the weights of the linear layers with random numbers sampled from a normal distribution with mean 0 and std 1. The biases are initialized to zero.

As it is required for all PyTorch modules, the class ends with the forward propagation callable, which in this case consists of passing the initial system's state y, through the neural net, to obtain the final state.

During the training stage, the neural ODE requires a derivative for each one of the dimensions of the state or input as the system's state and the derivative of the state have the same dimensions. Therefore, the output of the neural ODE must match the input size.

Most ODEs cannot be solved analytically, as they represent complex systems of high order or non-linear. Nevertheless, thanks to numerical methods, accurate approximations can be obtained, especially with ODE solvers. These solvers work by discretizing time and computing the system's state at these discretized time-steps. The ODE solvers included in the *torchdiffeq* can be categorized into fixed-step and adaptive-step.

Fixed-step methods, as their name implies, use a fixed step size to calculate the solution over time. Thus, the time vector would be discretized into a certain number of time points equally spaced. The most straightforward and historically significant fixed step solver is the Euler's method. In this method, the loss is proportional to the step size, so that halving the step size would halve the error. Although the Euler's method is known for its simplicity, it may not always provide the most accurate option. Especially with large step sizes or stiff ODEs, involving a rapid rate of variable change over time.

The *torchdiffeq* library includes the following lists of solvers. The fixed-step solvers include:

- Euler: Euler method.
- Midpoint: Midpoint method.
- rk4: Fourth-order Runge-Kutta with 3/8 rule.
- explicit_adams: Explicit Adams-Bashforth.
- implicit_adams: Implicit Adams-Bashforth-Moulton.

Since these solvers use a constant step size, it allows the model to be simpler and therefore, faster. However, it may miss features if step size *h* (depicted in Figure 18) is chosen too large and become slower if it is chosen too small.

For this study, given the complexity of the electricity price time-series, it is a more suitable idea to use the adaptive step solvers, as they adjust the time-step based on the estimated error of the solution. The solver can take smaller time-steps when the trajectory varies quickly and larger ones when they are more static and predictable. Consequently, solving an IVP problem becomes more computationally efficient.

Thus, if the electricity price time-series is changing slowly, it will increase the step size *h* to speed up computation. The following adaptive step solvers are included in the library:

- dopri8: Runge-Kutta 7(8) of Dormand-Prince-Shampine
- dopri5: Runge-Kutta 4(5) of Dormand-Prince [used by default in *torchdiffeq*].
- bosh3 Runge-Kutta 2(3) of Bogacki-Shampine
- fehlberg2: Fehlberg2
- adaptive_heun: Runge-Kutta 1(2)

Runge-Kutta (RK) methods integrate a more complex and accurate approach as they leverage the differential equation for computing the slope of the tangent line to the function *f* (Anonymous & ICLR, 2023). They perform multiple intermediate evaluations using Taylor's Series expansion techniques to determine the general solution of the ODE (Goeken & Johnson, 2000). These evaluations are then combined to a weighted sum to obtain the final approximation at the end of the step.

Backpropagation is not numerically stable for all the solvers. Nevertheless, and as it has proven to be the most precise, the selected adaptive step solver has been the default one: *dopri5*, which is the RK of order 4(5) of the Dormand-Prince method. RK methods evaluate the ODE *func* at the beginning of the step to obtain the first slope. Then, they take several intermediate steps within the current step interval. In the case of *dopri5*, it involves taking six additional steps, making a total of seven evaluations per step.

It computes the two separate solutions and by comparing them, the method can estimate the local truncation error of the solution, to ultimately adapt the step size. If the estimated error is below the previously specified tolerance, the fifth order solution is accepted, and the step size h may increase for the next time-step. However, if it is higher than the tolerance, the step is rejected, and the process is repeated with a smaller step size. As an adaptive step size method, it demonstrates good performance at approximating trajectories with sharp transitions, as it can vary the step size to ensure accuracy while also minimizing computation time.

When compared with a fixed step-size RK4 (Fourth order RK method), *dopri5* requires fewer steps to achieve the same accuracy but it results in a more complex method, so it is a trade-off. Nonetheless, it is an appropriate approach for the case study data.

Dopri5 is described by the following system of equations:

$$K_1 = h * f(t_k, y_k) \tag{E 27}$$

$$K_2 = h * f\left(t_k + \frac{1}{5}h, y_k + \frac{1}{5}K_1\right) \tag{E 28}$$

$$K_3 = h * f\left(t_k + \frac{3}{10}h, y_k + \frac{3}{40}K_1 + \frac{9}{40}K_2\right) \tag{E 29}$$

$$K_4 = h * f\left(t_k + \frac{4}{5}h, y_k + \frac{44}{45}K_1 - \frac{56}{15}K_2 + \frac{32}{9}K_3\right) \tag{E 30}$$

$$K_5 = h * f\left(t_k + \frac{8}{9}h, y_k + \frac{19372}{6561}K_1 - \frac{25360}{2187}K_2 + \frac{64448}{6561}K_3 - \frac{212}{729}K_4\right) \tag{E 31}$$

$$K_6 = h * f\left(t_k + h, y_k + \frac{9017}{3168}K_1 - \frac{355}{33}K_2 - \frac{46732}{5247}K_3 + \frac{49}{176}K_4 - \frac{5103}{18656}K_5\right) \tag{E 32}$$

$$K_7 = h * f\left(t_k + h, y_k + \frac{35}{384}K_1 + \frac{500}{1113}K_3 + \frac{125}{192}K_4 - \frac{2187}{6784}K_5 + \frac{11}{84}K_6\right) \tag{E 33}$$

Based on these equations, we can compute two separate estimations for the next time-step:

- Fourth-order RK solution:

$$y_{k+1} = y_k + \frac{35}{384}K_1 + \frac{500}{1113}K_3 + \frac{125}{192}K_4 - \frac{2187}{6784}K_5 + \frac{11}{84}K_6 \tag{E 1}$$

- Fifth-order RK solution:

$$z_{n+1} = y_n + \frac{5179}{57600}K_1 + \frac{7571}{16695}K_3 + \frac{393}{640}K_4 - \frac{92097}{339200}K_5 + \frac{187}{2100}K_6 + \frac{1}{40}K_7 \tag{E 2}$$

The two solutions are subtracted to obtain the resulting local truncation error. Then, the optimal time-step size h_{opt}:

$$s = \left(\frac{\epsilon * h}{2 * |z_{k+1} - y_{k+1}|}\right)^{\frac{1}{5}} \qquad E\ 34$$

$$h_{opt} = s * h \qquad E\ 35$$

Being h the last step size and ϵ, the desired error tolerance. The latter parameter is used to adaptively modify the step size during computation, in order to balance the existing trade-off between accuracy and computational cost.

Multiple function evaluations and their posterior linear combination make *dopri5* suitable in terms of the ability of handling complex ODEs and rapid change. However, more computation resources are required as they perform seven evaluations for the one that Euler's method does.

Overall, *dopri5* provides a good balance between accuracy and computational cost, which is critical for this project and its limited resources. Nevertheless, the models are computationally expensive and a migration to GPU-based environments results mandatory to compete in the time aspect with the rest of the models.

3.3. TRAINING STRATEGY

3.3.1. UNIVARIATE

The univariate model relies solely on historical electricity price data to forecast the day-ahead prices. This training process involves iteratively optimizing the weights and biases of the neural network to approximate the underlying function governing the dynamic laws of an observed variable. By exploiting the numerical solution of the neural ODE, we can effectively reconstruct the trajectory of the observed variable at hand.

The neural network takes as input the initial condition, i.e., the input at the first time-step, and evolves it over time according to the approximated ODE. In this scenario, the initial condition is composed by the start of a one-week period. The time-steps will correspond to each day of the input week until the last one, which will serve as the out-of-sample prediction. This strategy is more effective than introducing the whole preceding week's data to the neural ODE as it would mitigate the potential risk of model convergence issues during the training stage.

As represented in Figure 20, the ODE is trained with 5 time-steps, which corresponds to the preceding week's price trajectory. Then the out-of-the-sample forecast is performed at time-step 6, configuring the model under evaluation mode:

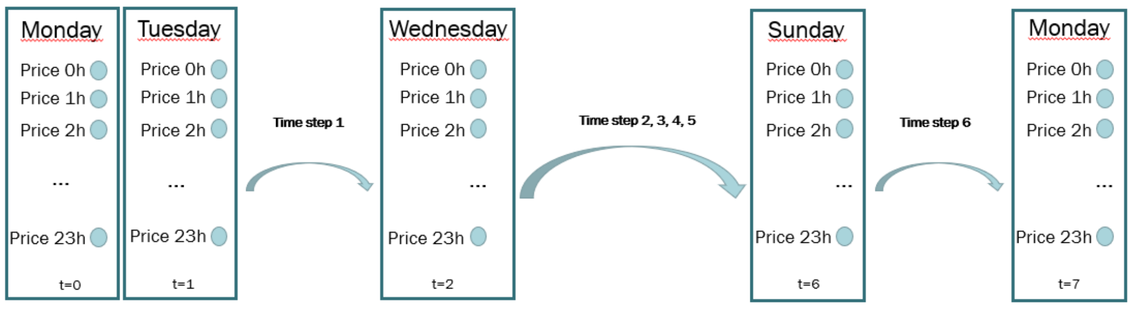
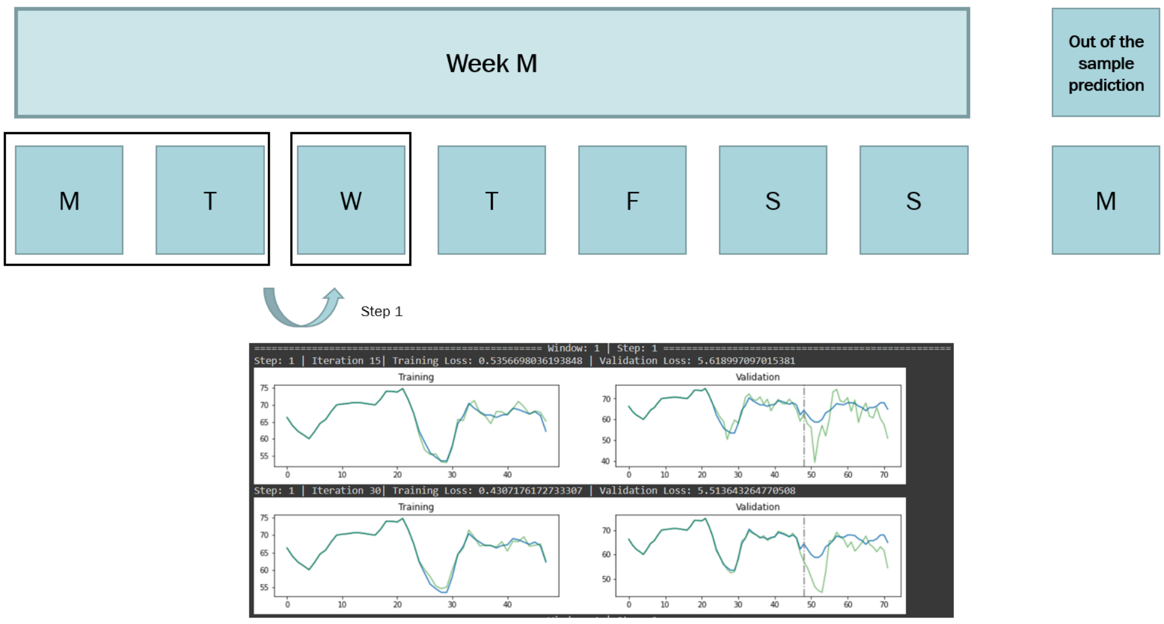


Figure 20: Univariate Neural ODE solver procedure

As mentioned earlier, the input data will consist of each day's 24-hourly electricity prices. Thus, the neural ODE is computed by the solver an arbitrary number of iterations (50, for this study) before passing to the next time-step where another day will be evaluated.

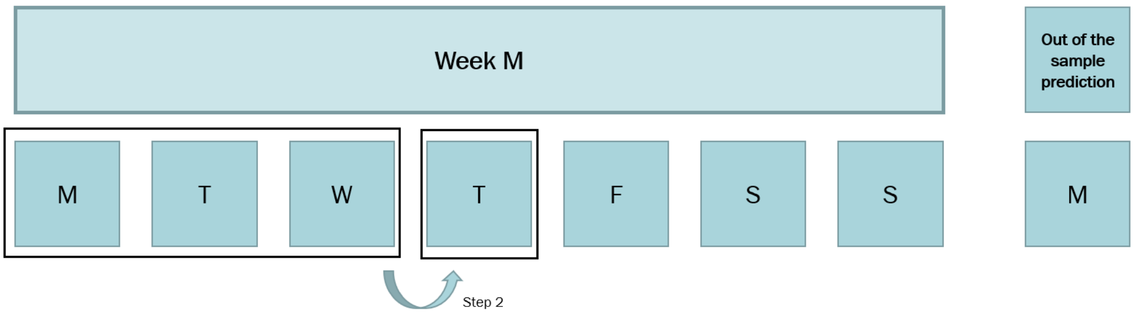
The first time-step is done computing the forecast for the third day, given an input of the first two days, as represented below, for the case of Monday's neural ODE:

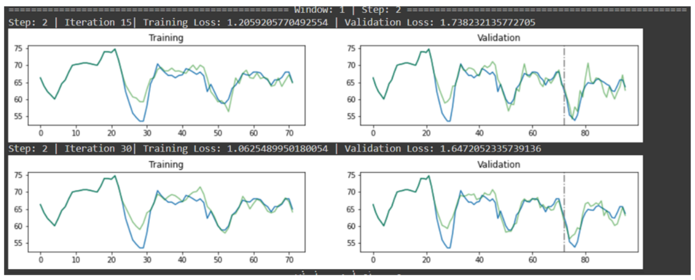


```
============================================== Window: 1 | Step: 1 ==============================================
Step: 1 | Iteration 15| Training Loss: 0.5356698036193848 | Validation Loss: 5.618997097015381
Step: 1 | Iteration 30| Training Loss: 0.4307176172733307 | Validation Loss: 5.513643264770508
```

Figure 21: Evaluation at the end of the first time-step

On the second time-step, the neural ODE is trained with the first three days to forecast the fourth day or time-step.

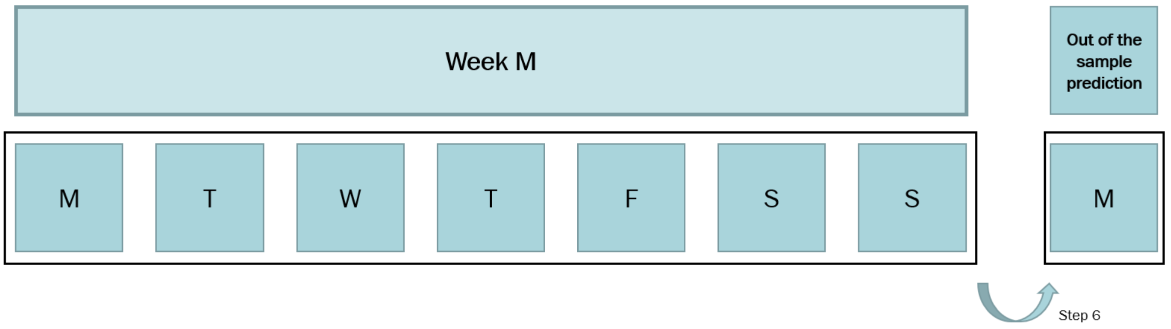


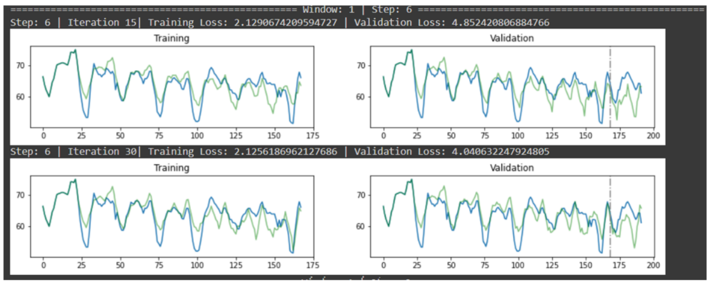


```
============================================= Window: 1 | Step: 2 =============================================
Step: 2 | Iteration 15| Training Loss: 1.2059205770492554 | Validation Loss: 1.738232135772705
Step: 2 | Iteration 30| Training Loss: 1.0625489950180054 | Validation Loss: 1.6472052335739136
```

Figure 22: Evaluation at the end of the second time-step

This process continues for the rest of windows so that in the final out-of-the-sample prediction, the model is trained with the whole prior week. As it can be seen in the image, the loss function value (MAE) for that first window out-of-the-sample forecast is 4.040:





```
============================================ Window: 1 | Step: 6 ============================================
Step: 6 | Iteration 15| Training Loss: 2.1290674209594727 | Validation Loss: 4.852420806884766
Step: 6 | Iteration 30| Training Loss: 2.1256186962127686 | Validation Loss: 4.040632247924805
```

Figure 23: Evaluation at the end of the last time-step - Out-of-the-sample prediction

This way, the architecture of the model, i.e., the derivative of the trajectory, is optimized to match the most recent patterns in the day-ahead electricity price series.

One of the main limitations of neural ODEs is that, as training processes and flow gets increasingly complex, the number of steps required to solve the ODE increases (R. T. Q. Chen et al., 2018) and the model becomes very computationally expensive. The following plot represents the number of function evaluation points (NFE), computed by the solver in the first univariate window:

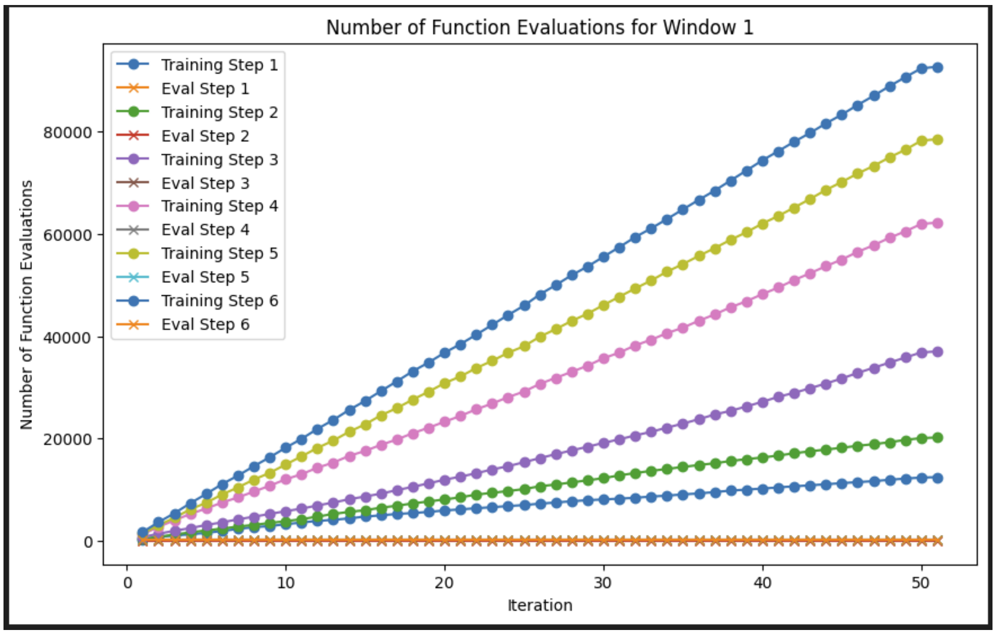


Figure 24: Evolution of the number of evaluation points during training

A way of determining the 'depth' of the Neural ODE is the number of evaluation points of the hidden state dynamics, dependent on the initial state or input (R. T. Q. Chen et al., 2018). As the number of epochs increases and the model is trained with more days, as it can be seen in Figure 24, the number of evaluation points increases. It must be noted, though, that the number of evaluations is not strictly proportional to the model size, due to the adaptive nature of the solver. For this specific case, the ΔNFEs between the training step 3 and 4 show an spike in the volatility of the price on training step 4, therefore requiring more function evaluations to accurately represent the trajectory. As this model represents the training for forecasting Fridays (Training Step 6), the corresponding week appears to have a tricky Wednesday (Training Step 4).

Nevertheless, and though the resources have been limited and neural ODEs have taken long to train, they have proven to be promising in terms of accuracy.

3.3.2. MULTIVARIATE

In this scenario, the neural ODE will follow the evolution of the electricity price, based on the provided historical data, including two exogenous variables: the P48 demand and the P48 wind generation. These variables are particularly relevant for the electricity market and, therefore, for the electricity price formation.

As mentioned earlier, one requirement of neural ODEs that distinguishes them from more traditional ML models is the condition that the dimensionality of the input and output data must match. This is due to the nature of ordinary differential equations. This requirement influences key decisions such as the model's architecture, including the number and size of the layers and the preprocessing and data feeding into the model. This constitutes no problem for the univariate neural ODE, as the initial condition of the IVP matches the size of the desired forecast.

Thus, similarly to the univariate approach, the initial condition of the ODE will consist of the flattened values of the variables, which including the exogenous variables, it has now increased to a dimensionality of 72 (24 times 3). Nevertheless, in the multivariate approach, to meet the latter condition, the output will also be composed of 72 values. The resulting multivariate training strategy is represented as follows:

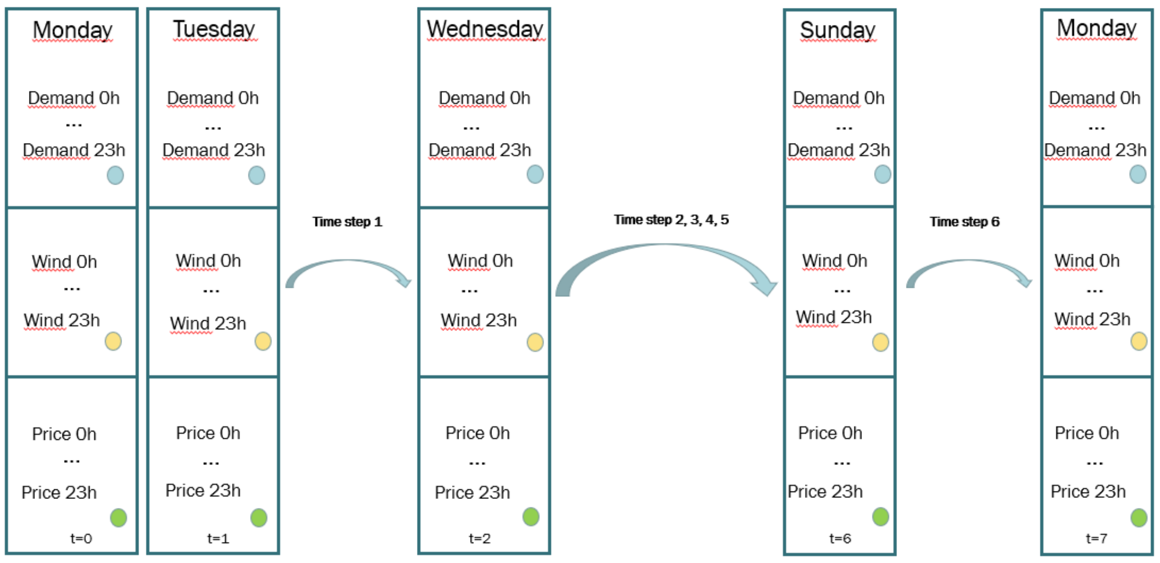


Figure 25: Multivariate Neural ODE solver procedure

It could be confusing that, to forecast the following 24-hourly prices, the neural ODE has to compute and predict the values of all the variables, 72. Nevertheless, this study provides a creative solution to avoid this rather avoidable problem. The solution lies on the loss function.

Deep networks are included in the supervised learning paradigm, as they must be indicated the actual values of the expected output in the training process. This includes indicating the loss function that will compute the difference between the actual values and the obtained output. For this study, the main loss function is the MAE. This loss function is usually applied to all the outputs of the last layer of the network. However, it must be noted that it could very much not, as it is the case.

As mentioned before, the output size is fixed to 72. But the use case of the model is to forecast the following 24 prices. Therefore, the loss function has been programmed to compute the output of the last 24 Linear neurons of the last layer. It must be noted that, instead of the last ones, any 24 outputs could have been selected to be revised by the loss function. After all, the neural ODE will learn to optimize those selected 24 outputs (of the whole 72) to approximate optimally to the following 24 electricity prices.

This way, the neural ODE considers all the variables, although it does not provide their forecasts, allowing the model to focus only on the electricity price time-series. As it has been proven afterwards, having all the outputs revised by the loss function does not provide better results on the price forecasts.

Chapter 4. ANALYSIS OF RESULTS

This chapter presents the obtained results from the application of the neural ODE and the benchmark models, which have been divided into statistical, probabilistic, and ML categories. These models, which include the SARIMA, Prophet, MLP, LSTM and CNN-LSTM, have been selected to establish a solid reference point for the newly developed central model of this master's thesis: the neural ODE.

At the end, the gathered results, applied to the Spanish day-ahead EPF for the period between 2018-2019, will be analyzed. The evaluation of the performance of the models has been carried out by using standard metrics, such as the MAE, whose equation is represented below:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \qquad E\ 36$$

Where N represents the total number of data points, y_i is the actual value of the i-th instance and $\hat{y}_i$ is the predicted value of the i-th instance.

In this equation, we compute the absolute difference between the actual and predicted values of each data point, to then obtain the average value. The result is a non-negative number, where lower values indicate better model performance, i.e., lower forecast error.

4.1. SARIMA MODEL

For this study, a SARIMA model will be implemented for both univariate and multivariate approaches. The statistical approach follows a training strategy that allows obtaining the optimal parameters for the test phase. A training period, spanning from July 2018 to July 2019 is implemented to fine-tune the model. This includes obtaining the optimal hyperparameter combination (autoregressive, integrated and moving average) for the three main seasonalities involved in the time-series: regular, daily, and weekly. Also, it will serve as a cross-validation test for obtaining the optimal window size to forecast the following 24-hourly electricity prices. Finally, a test set, spanning from August 1st to October 31st, 2019, was selected, to avoid important outliers in November 2019. It must be noted that the input windows will be separated by 24 hours, having a window per test day, therefore composing a total of 92 windows.

The training strategy followed by the SARIMA(X) model is described on Table 9:

Training set		**Test set**	
Start	***End***	***Start***	***End***
July 1st, 2018	July 31st, 2019	August 1st, 2019	October 31st, 2019

Table 1: Training-Validation-Test dataset division for the SARIMA(X) model

As part of the exploratory analysis, the autocorrelation plots of the training price time-series are represented in Figure 26:

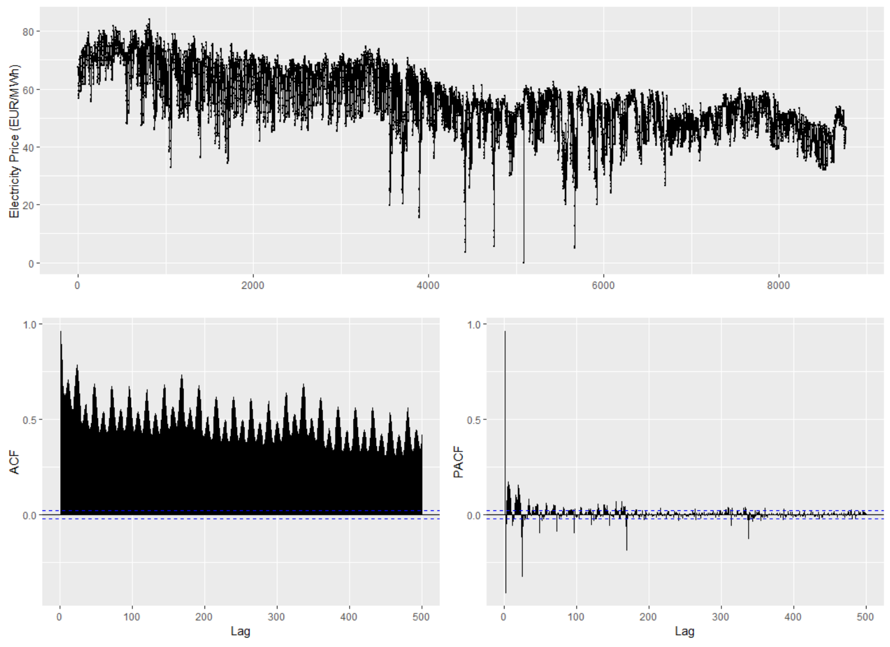


Figure 26: Training period for hyperparameter tuning (without outliers)

As it can be seen, the time-series shows significant outliers. Eliminating them, ensures that the model is trained on high quality data, which accurately represent the dynamics of electricity prices. Outliers can have a disproportionate impact on the model, causing it to overfit and leading to a poor generalization performance. Moreover, by removing these extreme values, the integrity of the multivariate relationships is maintained. For this study, all training electricity price observations whose value differed at least 3,5 standard deviations from the mean were considered outliers. To maintain the structure of the time-series, they have adopted the average value of that day, without considering the original outlier value. Therefore, the average price is calculated with the remaining 23 hourly prices. The resulting training period for hyperparameter tuning will be as follows:

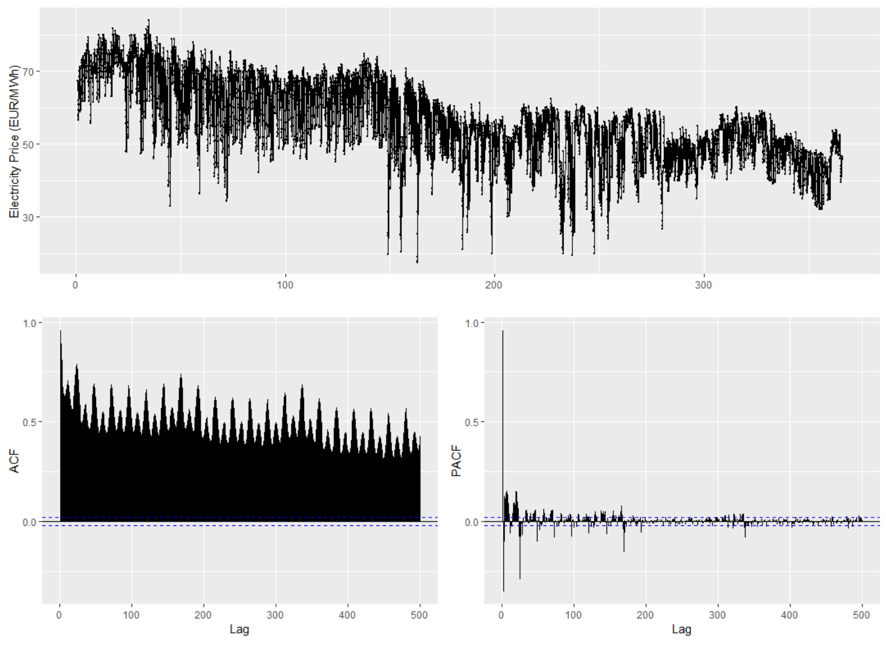


Figure 27: Training period for hyperparameter tuning (without outliers)

Once the outliers have been removed, it is convenient to ensure that the variance of the data is stable, remaining constant over time. Thus, the mean/sd scatterplot will decide, jointly with the obtained value of lambda, whether the Box-Cox transformation must be applied to the data. The mean/sd scatterplot represents the logarithm of the standard deviation with respect to the logarithm of the moving average, i.e., the relation between the variance and the mean. The obtained lambda parameter indicates the kind of transformation that should be applied to the data in order to stabilize the variance:

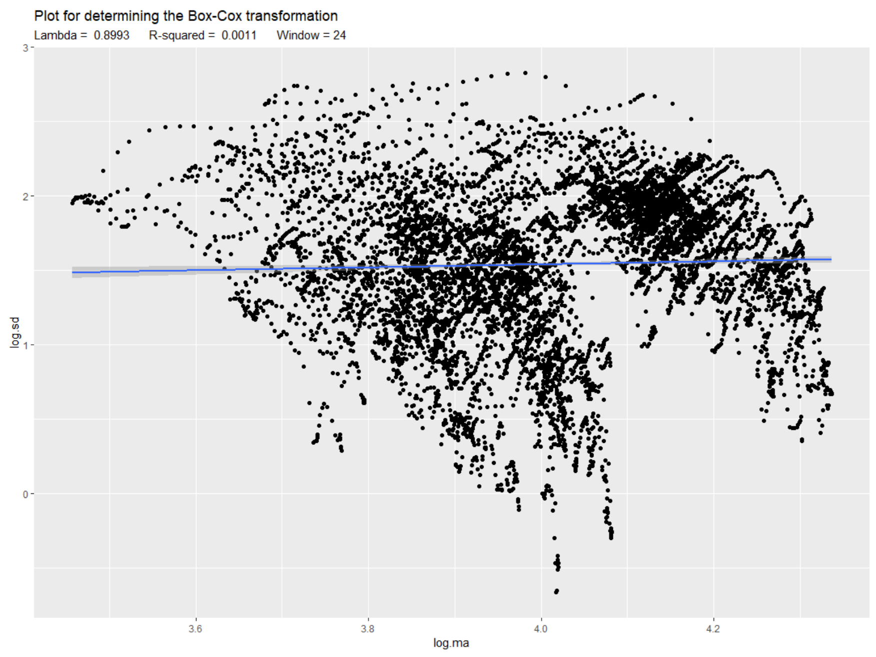
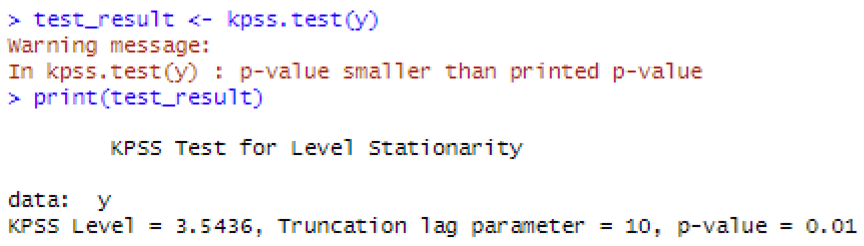


Figure 28: Mean/sd scatterplot

In our study, the training time-series shows a $\lambda = 0.8993$, meaning that the need for any transformation is dubious. Also, the low R-squared value suggests that there is little linear relationship between the log-transformed moving average and the log-transformed standard deviation of the time-series. Therefore, the time-series will be treated as is, with no transformations applied.

The following step in the SARIMA process is to ensure the stationarity of the time-series. Performing the KPSS test on our original training data, the following results are obtained:


```
> test_result <- kpss.test(y)
Warning message:
In kpss.test(y) : p-value smaller than printed p-value
> print(test_result)

        KPSS Test for Level Stationarity

data:  y
KPSS Level = 3.5436, Truncation lag parameter = 10, p-value = 0.01
```

Table 2: KPSS test on the original data

As the obtained KPSS p-value is 0.01, but as the test indicates, this value could be even smaller. Therefore, the null hypothesis cannot be rejected, meaning that the data is not stationary and must be differentiated. It must be noted that for seasonal time-series, a seasonal differencing must first be applied. Then, once the autocorrelations have been stabilized, the regular differencing would follow, if needed. As the autocorrelation plots show strong daily seasonality, the data will first suffer a differentiation at lag 24, resulting in the following autocorrelation plots:

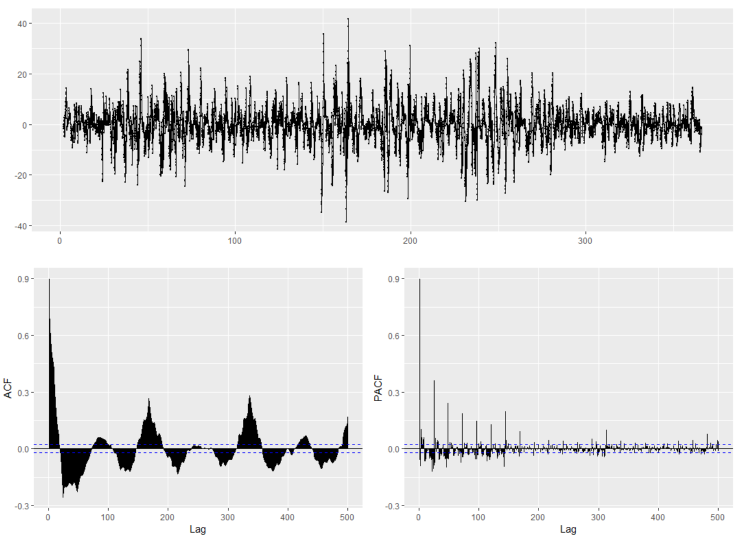


Figure 29: Autocorrelation plots after the daily seasonality differentiation

Once the daily seasonality has been performed, the time-series continues to show seasonal patterns. As expected, as the first lags exceed the significance interval, a regular differencing will be applied. Nevertheless, and as shown in Figure 29, the time-series has a significant seasonality on lag 168, which corresponds to the weekly seasonality. Therefore, both weekly and regular differencing will be applied. The resulting autocorrelation plots are shown on Figure 30:

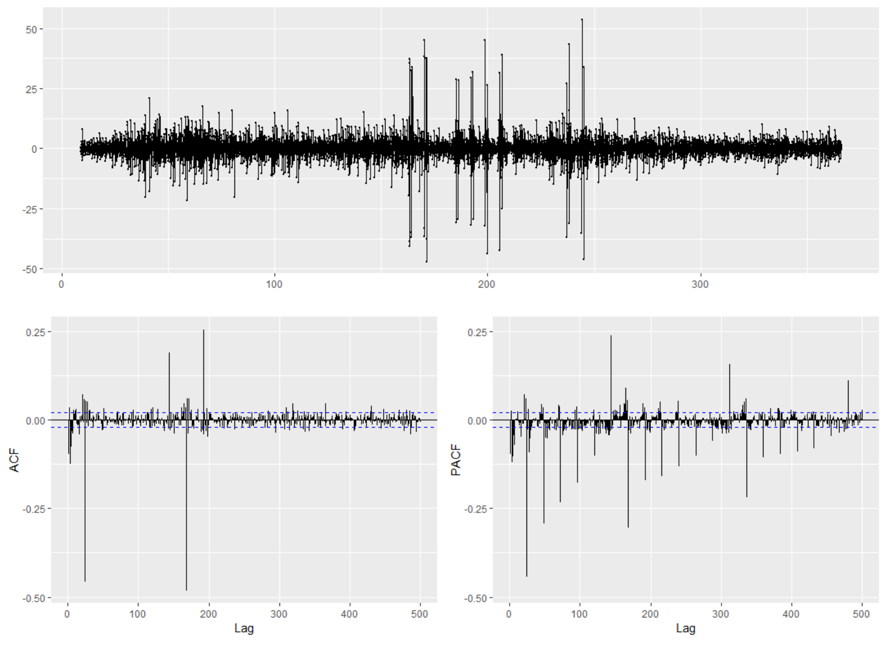
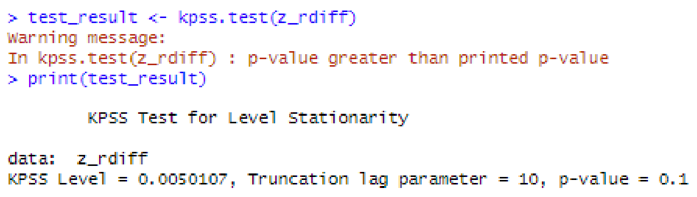


Figure 30: Autocorrelation plots after regular, daily and weekly differencing

Although the data continues to show some seasonality, the data can be now considered stationary. After performing a new KPSS test on the three-times differentiated time-series, the following results are obtained:



```
> test_result <- kpss.test(z_rdiff)
Warning message:
In kpss.test(z_rdiff) : p-value greater than printed p-value
> print(test_result)

        KPSS Test for Level Stationarity

data:  z_rdiff
KPSS Level = 0.0050107, Truncation lag parameter = 10, p-value = 0.1
```

Figure 31: KPSS test after regular, daily, and weekly differencing

The obtained KPSS p-value is 0.1 (or greater), meaning that the null hypothesis cannot be rejected. It can be concluded that the series is stationary enough to proceed with the hyperparameter selection.

The identification of the optimal SARMA hyperparameters of the series include the autoregressive (AR) term (p), the order of the moving average (MA) term (q) and the corresponding P and Q of the regular, daily and weekly seasonalities.

To identify the rest of the regular and seasonal parameters, as their integrated parameter has been fixed to 1, the autocorrelation plots from Figure 30 will be analyzed. The autoregressive parameter p is identified by looking at the decay on the ACF. For an AR(p) process, the ACF decays after lag p. However, and as Figure 30 shows, the decay is produced in the PACF plot. Therefore, we are treating with moving average processes. This way, the preliminary selected univariate autoregressive hyperparameters are:

p	P_{24}	P_{168}
0	0	0

Table 3: Univariate Autoregressive Hyperparameters

In order to achieve a stationary time-series and obtain uncorrelated training residuals, a regular, daily and weekly differences were applied to the original time-series. Therefore, the resulting integrated hyperparameters are:

d	D_{24}	D_{168}
1	1	1

Table 4: Univariate Integrated Hyperparameters

Similarly, to identify the moving average term q, the same procedure is followed but on the ACF plot. As the decay is exponential, the order of the hyperparameter q will be set to 1. It must be noted that the decay is produced visibly on both daily and weekly seasonalities (lags 24 and 168). When analyzing the autocorrelation plot with more detail, a slight moving average behavior is also identified on the regular component. Consequently, the selected univariate moving average hyperparameters will be as follows:

q	Q_{24}	Q_{168}
1	1	1

Table 5: Univariate Moving Average Hyperparameters

Once the combination of hyperparameters has been identified on the training period, i.e., the year prior to the test set, it is time to test them on the test rolling windows. It must be noted that the training data also served as a cross-validation test to identify the best window size, which resulted in 13 weeks prior to the posterior 24-hourly electricity price forecasts. Therefore, just to remind the reader, for every day to forecast (out of the 92 test days), a different SARIMA model will be fitted with the same hyperparameters. The first input window will correspond to the 13 prior weeks to the first test set day, August 1st, 2019.

For the multivariate approach, the same strategy for hyperparameter selection has been applied. Nevertheless, the best candidate combination on the training data proved to be a simpler model, as ignoring the slight moving average of the regular component leaded to better results. The final hyperparameters of the multivariate SARIMA models and their resulting equations are indicated on equations *E 37* and *E 38,* based on the notation found on the study developed by Weron, (2014):

- **Univariate SARIMA: (0, 1, 1) (1), (0, 1, 1) (24), (0, 1, 1) (168)**

$$(1-B)\left(1-B^{24}\right)\left(1-B^{168}\right)y(t) = (1-\theta_1 B)\left(1-\theta_{24}B^{24}\right)\left(1-\theta_{168}B^{168}\right)\varepsilon(t) \qquad E\ 37$$

- **Multivariate SARIMA: (0, 1, 0) (1), (0, 1, 1) (24), (0, 1, 1) (168), (0, 0, 0) ($X^{(1)}$), (0, 0, 0) ($X^{(2)}$), (0, 0, 0) ($X^{(3)}$)**

$$(1-B)(1-B^{24})(1-B^{168})y(t) = (1-\theta_{24}B^{24})(1-\theta_{168}B^{168})\varepsilon(t) + \sum_{i=1}^{3} \psi^i(B)X^i(t) \qquad E\ 38$$

Where:

- B is the backshift or lag operator
- y_t is the original time-series
- C is the constant term
- θ_1 is the regular MA term
- θ_{24} is the daily seasonality MA term
- θ_{168} is the weekly seasonality MA term
- ϵ_t is the error term
- $X^{(1)}$ is the P48 power demand
- $X^{(2)}$ is the P48 wind generation
- $X^{(3)}$ is the day of the week
- $\psi^i(B)$ is a shorthand notation for $\psi^i(B) = \psi_0^i + \psi_1^i B + \psi_2^i B^2 + \psi_3^i B^3$ with $\psi_j^i s$ being the corresponding coefficients

For the SARIMA implementation, the R framework has been used. This was convenient as the *msarima* function is only implemented in this coding language and is a must for modelling more than one seasonality, which is our case. This example of the multivariate implementation is reflected below, as this line of code is responsible for training a model per input window:

```
  # Model

  model <- msarima(y[i:(i + window_size - 1)], xreg=exogenous_window,
orders=list(ar=c(0,0,0),i=c(1,1,1),ma=c(0,1,1)), lags=c(1,24,168), h=24, holdout
= TRUE, silent = FALSE, initial = 'backcasting')
```

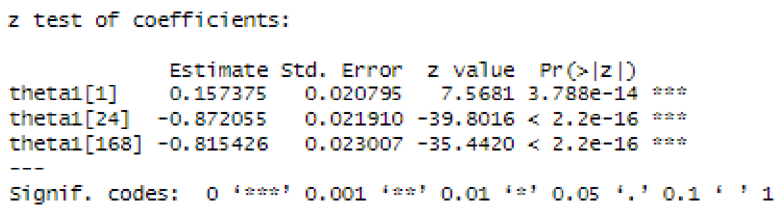
Being the window size, the preceding 13 weeks of data, the model accounts for regular, daily and weekly seasonalities with a combination of hyperparameters composed by the *ar*, *i* and *ma* tuples. The *xreg* parameter includes the exogenous variables.

After having selected the hyperparameters for both approaches, it is time for training a different model per input window and computing the corresponding 24-hourly forecasts. The MAE results from both univariate and multivariate test models are gathered in Table 6:

Day to forecast	***Naïve***	***SARIMA***	***SARIMAX***
Monday	8.12	**4.25**	4.51
Tuesday	**3.93**	4.49	4.89
Wednesday	**3.35**	4.59	5.11
Thursday	**3.55**	3.73	4.33
Friday	**2.76**	4.01	4.26
Saturday	4.66	**4.56**	5.04
Sunday	**4.09**	6.87	5.41

Table 6: SARIMA(X) Results

These results show the difficulty behind EPF. Even the 2019 electricity price time-series, when the renewables were just starting to infiltrate the energy market, show complex patterns that the ARIMA models fail to capture. Once the forecasts for the test set have been obtained, it is time to check the significance of the model coefficients and analyze the training residuals. Performing the *coeftest* function, the significance of the coefficients is checked. Plotted on Figure 32 are the univariate coefficients in the last test window, which show are significant, i.e., their p-value is much lower than 0.05, which means that they are contributing to the model:

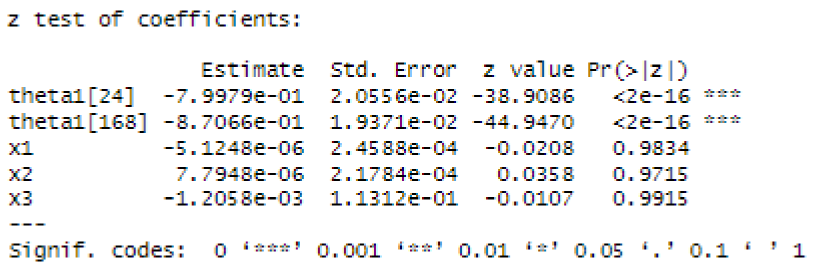


```
z test of coefficients:

              Estimate Std. Error  z value  Pr(>|z|)    
theta1[1]     0.157375   0.020795   7.5681 3.788e-14 ***
theta1[24]   -0.872055   0.021910 -39.8016 < 2.2e-16 ***
theta1[168]  -0.815426   0.023007 -35.4420 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 32: Significance of the univariate SARIMA hyperparameters

The same happens with the multivariate approach. It must be noted that, although in other models, the correlation between the exogenous and the output variables have not proven useful, the SARIMA statistical model is able to find strong linear correlations:



```
z test of coefficients:

               Estimate  Std. Error  z value Pr(>|z|)    
theta1[24]  -7.9979e-01  2.0556e-02 -38.9086   <2e-16 ***
theta1[168] -8.7066e-01  1.9371e-02 -44.9470   <2e-16 ***
x1          -5.1248e-06  2.4588e-04  -0.0208   0.9834    
x2           7.7948e-06  2.1784e-04   0.0358   0.9715    
x3          -1.2058e-03  1.1312e-01  -0.0107   0.9915    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Figure 33: Significance of the multivariate SARIMA hyperparameters

The p-values of the orders of the model reveal that, although the moving average hyperparameters are significant, the exogenous variables are providing noise. This is understandable due to the relatively low correlation between them and the output variable. While the obtained MAE values are very competitive, once the out-of-sample forecasts have been obtained, it is time to check their resulting residuals. Looking at the autocorrelation plots on Figure 34, there are certain seasonal patterns that have not been accurately:

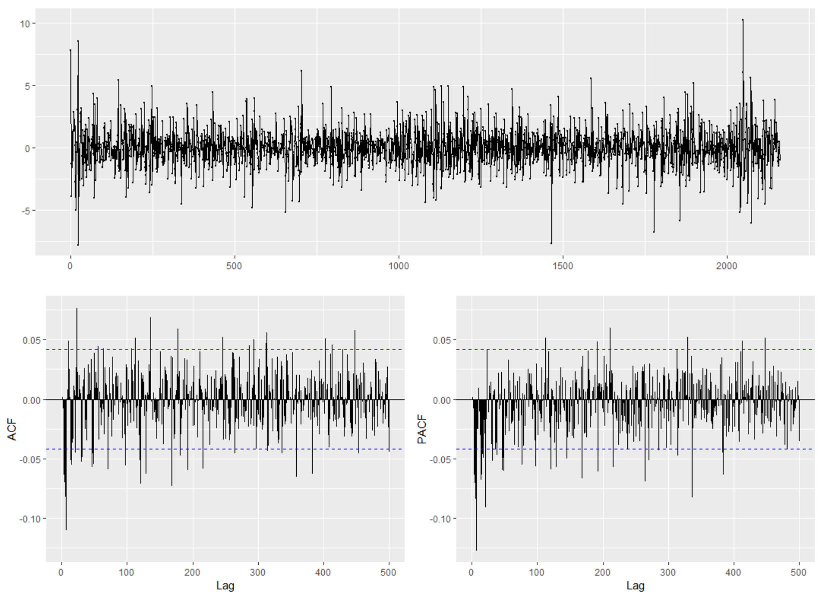


Figure 34: Generic univariate model residuals

As it can be seen, there are some marginal lags that overpass the significance thresholds, indicating that this model residuals may not fully capture the temporal dependencies, specially the initial lags. This is because they do not strictly resemble white noise, meaning that the residuals are not completely independent and distributed with a mean of zero and constant variance. However, due to the complexity and volatility of the time-series, these residuals are considered acceptable.

When performing the residuals analysis on the multivariate approach, it can be seen that these residuals show similar properties as in the univariate approach. The multivariate residuals are as follows:

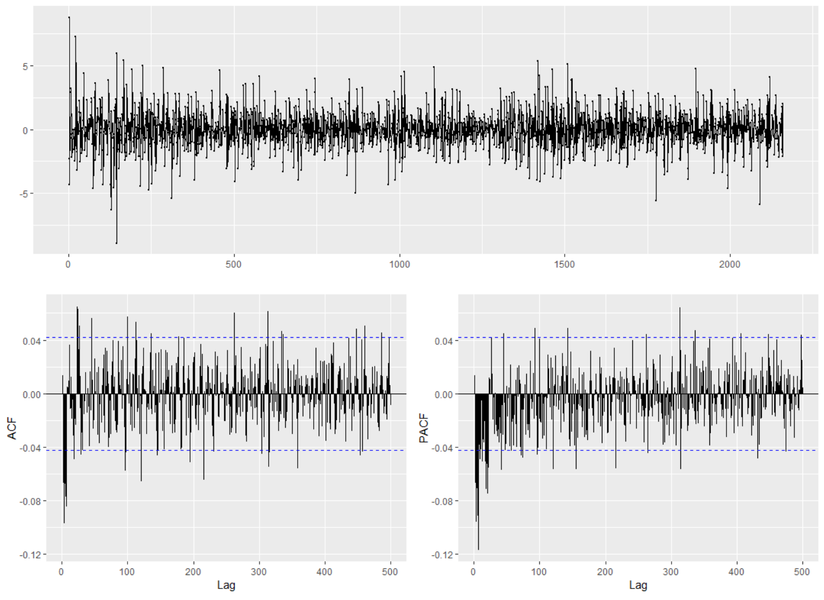
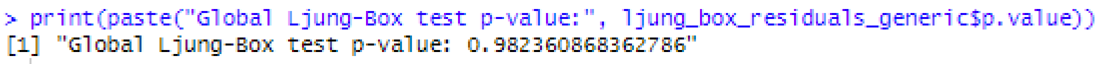


Figure 35: Generic multivariate model residuals

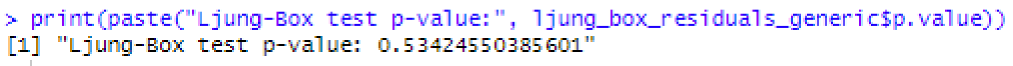
Even though the residuals seem uncorrelated visually, a posterior Ljung-Box statistical test is crucial to ensure the residuals' independence. As explained before, the Ljung-Box test checks whether any of a group of autocorrelations are different from zero, instead of testing randomness at each distinct lag. After performing the test on univariate generic models, we obtain the following p-values:



```
> print(paste("Global Ljung-Box test p-value:", ljung_box_residuals_generic$p.value))
[1] "Global Ljung-Box test p-value: 0.982360868362786"
```

Figure 36: Ljung-Box test on the univariate residuals

And as for the multivariate residuals, Figure 26 shows the result of the serial correlation test:



```
> print(paste("Ljung-Box test p-value:", ljung_box_residuals_generic$p.value))
[1] "Ljung-Box test p-value: 0.53424550385601"
```

Figure 37: Ljung-Box test on the multivariate residuals

The Ljung-Box tests p-values show that neither of the approaches fail to reject the null hypothesis of no autocorrelation, as they are way above 0.05. This suggests that the residuals of the model are independently distributed with no autocorrelation structure.

In conclusion, although the models have shown poor results, the residuals are uncorrelated, demonstrating that the models have not captured all the information in the data. As the models are very simple, there is little room for considering overfitting. The most plausible explanation at this point is that the exogenous variables provided noise, degrading the true variable, the electricity price. This is validated with the high p-value of the exogenous variables on Figure 33. It must be noted that the statistical methods, which include the SARIMA model, are known for not performing on high frequency data, which is our case. The underlying volatility of the electricity price time-series adds non-linearities which makes it difficult to predict (Lago et al., 2018).

4.2. PROPHET MODEL

As it was explained, Facebook Prophet has been included in the list of benchmark models for its capability to effectively handle datasets with complex seasonal patterns and holiday effects. The following results demonstrate its robustness.

This model has been implemented in R, for efficiently dividing the computation between the different frameworks (VS Code, Google Colab and RStudio). Thus, the corresponding libraries have been used. It must be noted that the dataset is reformatted to suit the requirements of the libraries of Prophet.

The Prophet model will follow a similar strategy as in the statistical approach. As explained later, the model does not need manually integrated parameters. Therefore, the only variable to consider will be the input window size to forecast the following 24-hourly prices. This parameter is obtained during the training phase, based on the preceding year of observations from July 2018 to July 2019. Then, a different model will be developed to compute the forecasts for the 92 days composing the test set, from August 1st to October 31st, 2019.

The training strategy followed by the Prophet model is described on Table 7:

Training set		**Test set**	
Start	***End***	***Start***	***End***
July 1st, 2018	July 31st, 2019	August 1st, 2019	October 31st, 2019

Table 7: Training-Validation-Test dataset division for the Prophet model

The sliding window approach involves shifting the training windows by 24 hours and iteratively fitting the Prophet model on each of them, followed by an evaluation of the following 24-hour period. In this cross-validation process of selecting the optimal window size, several options are analyzed, ranging from 1 to 11 weeks. For each window size, the MAE is computed. This way, the window size that provided the most accurate results on the training data was 6 weeks.

Once the optimal window size has been selected, the final model proceeds to the test part. To forecast each day's 24-hourly prices, a different Prophet model will be fitted and trained with the preceding 6 weeks of data. Throughout the process, the residuals are computed and stored for further analysis.

The following code represents the way a multivariate Prophet models per input window is fitted. The data is first split into rolling train/test sets, as occurs in the rest of the models. As for the size of the input window for the Prophet models, *best_window* is set to 6 weeks, as it has been explained. The regressor variables will of course consist of the P48 demand, P48 wind production and the day of the week. Moreover, the *holidays* parameter will allow the model to consider these effects.

```
  # Diving the data into training and test sets
  train_temp <- df[(i-best_window+1):i, ]
  test_temp <- df[(i+1):(i+24), ]

  # Initializing the Prophet model
  m <- prophet(holidays=holidays, daily.seasonality = TRUE, weekly.seasonality =
TRUE)  # Enable daily and weekly seasonalities

  # Adding 'demand' and 'wind' as regressors
  m <- add_regressor(m, 'demand')  # Adding demand as a regressor
  m <- add_regressor(m, 'wind')  # Adding wind as a regressor
  m <- add_regressor(m, 'day_of_week')  # Adding day of the week as a regressor

  # Fitting the model with the training data
  m <- fit.prophet(m, df = train_temp)
```

The obtained MAE results for both univariate and multivariate test models are gathered in Table 8:

Day to forecast	***Naïve***	***Univariate Prophet***	***Multivariate Prophet***
Monday	8.121	4.94	**2.45**
Tuesday	3.935	4.27	**2.38**
Wednesday	3.352	3.44	**3.28**
Thursday	3.554	4.09	**2.86**
Friday	2.768	3.76	**2.78**
Saturday	4.665	4.84	**2.34**
Sunday	4.093	4.68	**2.48**

Table 8: Prophet Results

The naïve model provides as forecasts the electricity prices of the preceding day. It is used as a baseline model for the rest of the models to be compared with. As the Table 8 shows, the univariate model is already more accurate than the naïve model, using only the past values of the electricity price for forecasting.

As for the multivariate model, also taking as input the exogenous variables (P48 demand, P48 wind generation and the day of the week), it is able to provide the best result of the whole study. Apart from the demand and the wind generation, it can accurately model the trajectory considering all the complex patterns caused by the strong seasonalities of the time-series.

Given the superior performance of the model, considering the holiday effects of the country, it has been decided that the *day_of_the_week* exogenous variable will be included in the rest of the benchmark models.

4.3. MACHINE LEARNING METHODS

In this subsequent subsection of our EPF study, an overview of the three core ML models will be provided, including MLP, LSTM, and CNN-LSTM. We will provide a detailed explanation of how each model has been implemented.

It must be noted that the multivariate ML models will include three exogenous variables: power demand, wind production and the day of the week. This discrete variable has been included and will assign each day of the week a different number from 0 (Monday) to 6 (Sunday), adding 7 in the cases where that day was a national holiday, mentioned on subsection 1.5.

The ML models have been trained on 14 months of historical data from March 2018 to April 2019. Right afterwards, a validation set consisting of 3 months from May to July 2019, allowed it to optimize its parameters to perform accurately on unseen data. Finally, a test set, spanning from August to October 2019 was selected, to avoid important outliers in November 2019. Thus, the dataset training-validation-test set division is a commonly used 70-15-15%. It must be noted that again, the optimal input window size will be obtained during the training phase.

The training strategy followed by the ML methods is described on Table 9:

Training set		**Validation set**		**Test set**	
Start	***End***	***Start***	***End***	***Start***	***End***
March 1st, 2018	April 30th, 2019	May 1st, 2019	July 31st, 2019	August 1st, 2019	October 31st, 2019

Table 9: Training-Validation-Test dataset division for ML methods

4.3.1. MLP MODEL

For this study, both univariate and multivariate MLP models have been implemented. The univariate MLP calculates the forecast based only on past prices, basically modelling the autocorrelation in the time-series. The multivariate model, on the other hand, leverages additional exogenous variables in its predictions.

For the univariate approach, although a different model for every day of the week has been trained, the most common configuration has been as follows:

Number of linear layers	*Neurons per layer*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
4	100	0.1-0.001	1000	0.1	0.9

Table 10: Univariate MLP model architecture

This approach considers only the historical price data, i.e., the preceding 168 observations prior to the window forecasts. On the univariate implementation in Python, the example of the architecture definition is coded as follows:

```python
# Model definition
class MLPModel(nn.Module):
    def __init__(self, input_size, hidden_layer_neurons, output_size):
        super(MLPModel, self).__init__()

        self.mlp = nn.Sequential(
            nn.Linear(input_size, hidden_layer_neurons),
            nn.BatchNorm1d(hidden_layer_neurons),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(hidden_layer_neurons, hidden_layer_neurons),
            nn.BatchNorm1d(hidden_layer_neurons),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(hidden_layer_neurons, hidden_layer_neurons),
            nn.BatchNorm1d(hidden_layer_neurons),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(hidden_layer_neurons, hidden_layer_neurons),
            nn.BatchNorm1d(hidden_layer_neurons),
            nn.ReLU(),
            nn.Dropout(0.1),

            nn.Linear(hidden_layer_neurons, output_size),
        )

    def forward(self, x):
        x = self.mlp(x)
        return x
```

A stack of 4 linear layers composed by 100 neurons has been configured to provide the best results. Using the neural network optimization techniques (Data normalization, Learning Rate Schedule, Early Stopping, etc.) allowed the model to understand the data dynamics. This architecture specifically, was implemented on the Tuesday model. In Figure 38, the univariate forecasts for the first test window are represented:

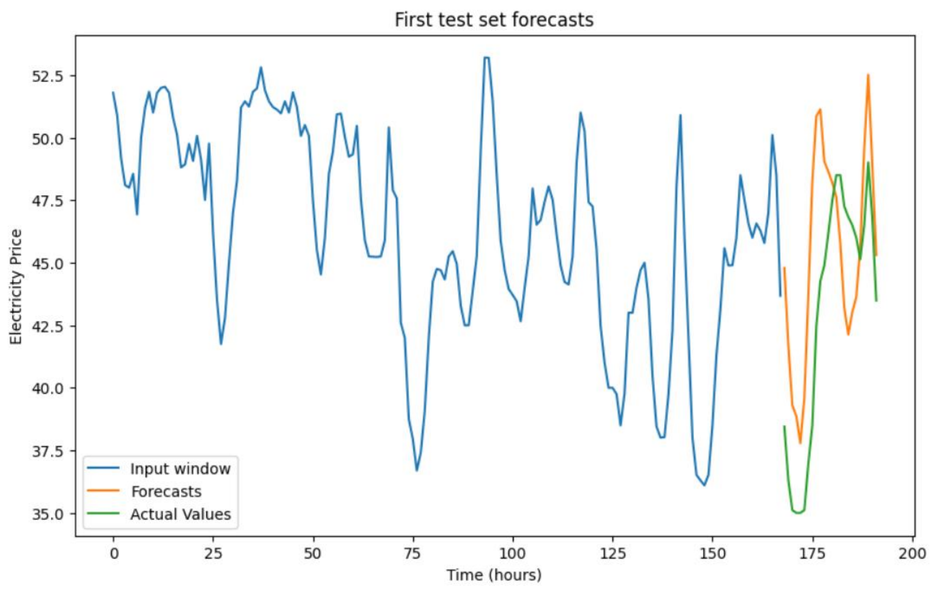


Figure 38: Univariate MLP forecasts

On the case of the multivariate model, as the input data can be 2D, not only will it consider the historical data of price and exogenous variables, but it will also take into account the values of the exogenous variables for the following 24-hours to forecasts. Consequently, these additional 24 values per variable serve as forecasts for the resulting window, as it is commonly done for EPF. Therefore, each sample of the input data is a 1D array of length 744, which correspond to 168 electricity prices and 192 values for each exogenous variable ('price', P48 power demand', 'P48 wind production' and 'day_of_the_week'). Although the models present slight difference in architectures, the most used configuration on the multivariate approach has been the following:

Number of linear layers	*Neurons per layer*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
4	300	0.1-0.001	2000	0.1	0.95

Table 11: Multivariate MLP model architecture

Alike the univariate approach, the multivariate architecture will also be composed of 4 stacked layers. However, it includes three times more units, which makes sense as the input has now more observations to analyze. The results show a better performance from the multivariate approach, compared with the univariate approach. Although the exogenous variables are not that much correlated with the output variable, the model is able to extract information about the intrinsic pattern of the time-series. The univariate and multivariate MAE results for the test set are represented in Table 12:

Day to forecast	***Naïve***	***Univariate MLP***	***Multivariate MLP***
Monday	8.121	5.331	**5.213**
Tuesday	3.935	3.971	**3.693**
Wednesday	**3.352**	4.322	4.076
Thursday	**3.554**	4.636	4.001
Friday	**2.768**	3.418	3.369
Saturday	4.665	3.909	**3.410**
Sunday	4.093	3.715	**2.919**

Table 12: MLP Results

As it will happen with other models, forecasting is especially convenient on Mondays, Saturdays, and Sundays. These are the weekdays which most differ from their preceding day, in terms of electricity usage. In the case of Sundays, it must be noted that, although it should be similar to Saturdays, there could be important events like, for example, football matches, that could significantly alter the energy consumption. Consequently, their naïve MAE is higher than in the middle weekdays. In fact, as the middle weekdays have a more accurate naïve MAE, it provides less slack for the MLP models to outperform it.

Beating the naïve model on Thursday's forecasts is very challenging. It is the day of the week with the poorest MLP model performances, as none of them have been able to outperform the naïve model.

From the results of Table 12, the multivariate MLP model consistently outperforms its univariate counterpart. Therefore, it would be always suitable to choose a multivariate forecasting model for the case of MLP models.

4.3.2. LSTM MODEL

The LSTM model has been trained for both univariate and multivariate approaches. The training-validation-test distribution is the same as the MLP and the CNN-LSTM models.

For implementing the latter model, as well as the LSTM, the TensorFlow framework was chosen. While PyTorch was used for the MLP and the neural ODE, TensorFlow provided the better results in these cases. Its extensive configuration of the model architecture, along with a more user-friendly environment was also considered. The architecture of the univariate LSTM model is as follows:

Number of LSTM layers	*LSTM cells per layer*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
2	500	0.1-0.01	2000	0.1	0.95

Table 13: Univariate LSTM model architecture

This sequential model begins with two LSTM layers. The first one is composed by 500 units, and for this study, it has been fixed that it returns sequences. It means that it outputs all the hidden states, in contrast to return the hidden state of the last time-step. Thus, the following layer is provided with more information about the dynamics of the price data. Moreover, an additional feature has been implemented, the *Reset States* custom callback class. It allows to reset the states of the model at the beginning of each epoch, which avoids overfitting.

The second layer, on the other hand, does not return sequences and it appropriately outputs only the hidden state of the last step, which represent the final output of the LSTM block. Following the LSTM layers, a Dropout layer with a rate of 10% is added. This layer randomly sets 10% of the input units (output of the final LSTM layer) to 0 at each epoch during training. Thus, the model is well equipped with another tool for avoiding overfitting. Finally, a Dense layer with 24 units provides the final forecast of the model.

On the multivariate approach, the LSTM model has two starting layers with 2000 units each. Alike the univariate approach, the model is equipped with both *return sequences* and *return states* terms activated. As it includes the exogenous variables (demand, wind generation and day of the week, the model expects an input of (batch_size, 744,1), which depends on the number of windows on the training, validation and test sets. For the consisting training strategy and model, the inputs will be:

- X_train.shape: (60, 744, 1)
- X_validation.shape: (12, 744, 1)
- X_test.shape: (12, 744, 2)

The multivariate model architecture will be configured as follows:

Number of LSTM layers	*LSTM cells per layer*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
2	800	0.01-0.001	2000	0.1	0.95

Table 14: Multivariate LSTM model architecture

For the LSTM model, the example of the multivariate model architecture is coded as follows:

```
# Model declaration
LSTM_model = tf.keras.models.Sequential([
    tf.keras.layers.LSTM(800, return_sequences=True,
input_shape=(X_train.shape[1], X_train.shape[2])),
    tf.keras.layers.LSTM(800, return_sequences=True),
    tf.keras.layers.LSTM(800),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(forecast_size)
])

LSTM_model.summary()
```

The results presented by the LSTM models demonstrate that the LSTM accurately captures the temporal dependencies on the price time-series data, yielding more precise predictions than the naïve and MLP models. The MAE results for both univariate and multivariate LSTM test models are as follows:

Day to forecast	***Naïve***	***Univariate LSTM***	***Multivariate LSTM***
Monday	8.121	**5.292**	5.756
Tuesday	**3.935**	3.961	5.684
Wednesday	3.352	**3.178**	5.601
Thursday	**3.554**	4.350	7.084
Friday	**2.768**	3.309	6.572
Saturday	4.665	**3.898**	5.653
Sunday	4.093	**3.473**	6.090

Table 15: LSTM Results

Nevertheless, the multivariate model, which included the exogenous variables, did not perform as well as the univariate model. This shows the limitation of the LSTM cells. Although they perform approximately three times more calculations than the linear layers, it is very dependent on correlated variables. In this case, the variables presented a mild correlation with the price, and consequently, they added noise to the model, making it more challenging for the LSTM to accurately predict future prices.

In conclusion, only the univariate LSTM model exhibited improved forecasting performance over the naïve model for specific days. This suggests that for this specific case, a simpler univariate model may be more effective than the multivariate LSTM framework. It must be noted that the complex LSTM cells allowed a more accurate test set univariate forecasts than all the MLP weekdays models, constituting the best ML model for the challenging forecast case of Wednesdays.

4.3.3. CNN-LSTM MODEL

This model, as well as the LSTM, has been coded in the TensorFlow framework. Not only did it provide better results, but it also included a more user-friendly architecture, which allowed access to more parameters in the convolution.

As explained on subsection 2.3.3., this model includes, for both approaches, a 1D Convolution before the LSTM deep neural network. This allows the model to effectively capture the temporal dependencies in the data while also considering the different variables.

The CNN-LSTM model is a sequential model that begins with a convolutional layer. This layer uses 64 filters or kernels, which focus on detecting a certain type of feature in the input sequence. Setting the kernel size to 3, the convolution operation is performed on sequences of 3 input elements at a time. Moreover, setting the stride to 1 means that the convolutional computations will be spaced one input unit, allowing the model to capture all potential features. Finally, the padding parameter is set to *causal*, to prevent the model to look into future data.

Following the convolutional layer, there are 3 LSTM layers with 500 units each. As in the LSTM model, the first two layers have the *return sequences* parameter set to *True*. Thus, the final layer will access all the hidden states of the LSTM cells. The univariate architecture is gathered as follows:

Number of CNN filters	*Kernel size*	*Number of LSTM cells*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
64	3	3x500	0.1	2000	0.1	0.9

Table 16: CNN-LSTM univariate architecture

The input of the model is designed in a specific way to accommodate the structure of the LSTM network, which expects a 3D array as input. The dimensions of this array are [*samples*, *time-steps*, *features*].

On the multivariate approach, the *samples* correspond to the already scaled electricity price observations. The *time-steps* are the window of consecutive data points used for computing the forecasts. For this case study, a window size of 168 observations is used, as the preceding week is considered for the following 24-hourly prices. Finally, the *features* value is 4, as the model exploits the 'price', P48 power demand', 'P48 wind production' and 'day_of_the_week' variables. The final output of the CNN-LSTM model is a 1D array with 24 values, representing the final forecasts.

The model is compiled using an SDG optimizer and a MAE loss function. During training, the model's weights and biases are saved as the validation loss improves and the Early Stopping callback allows it to avoid overfitting and computation efficacy. For the CNN-LSTM model, the example of the univariate model architecture is coded as follows:

```
# Model definition
CNN_LSTM_model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=64, kernel_size=3,
                      strides=1,
                      padding='causal',
                      input_shape=[window_size, 1]),
  tf.keras.layers.LSTM(500, return_sequences=True),
  tf.keras.layers.LSTM(500, return_sequences=True),
  tf.keras.layers.LSTM(500),
  tf.keras.layers.Dense(24),
])

CNN_LSTM_model.summary()
```

For the multivariate approach, more LSTM cells have been added, increasing the number to 800 per layer. Nevertheless, the rest of the convolution configuration remains constant:

Number of CNN filters	*Kernel size*	*Number of LSTM cells*	*Learning Rate*	*Epochs*	*Dropout*	*Optimizer Momentum*
64	3	3x800	0.1	2000	0.1	0.9

Table 17: CNN-LSTM multivariate architecture

The MAE results of the CNN-LSTM test models are gathered in Table 18:

Day to forecast	***Naïve***	***Univariate CNN-LSTM***	***Multivariate CNN-LSTM***
Monday	8.121	5.639	**3.804**
Tuesday	**3.935**	4.679	3.995
Wednesday	3.352	**3.239**	3.547
Thursday	**3.554**	4.503	4.325
Friday	**2.768**	3.437	3.325
Saturday	4.665	4.057	**4.022**
Sunday	4.093	3.177	**2.755**

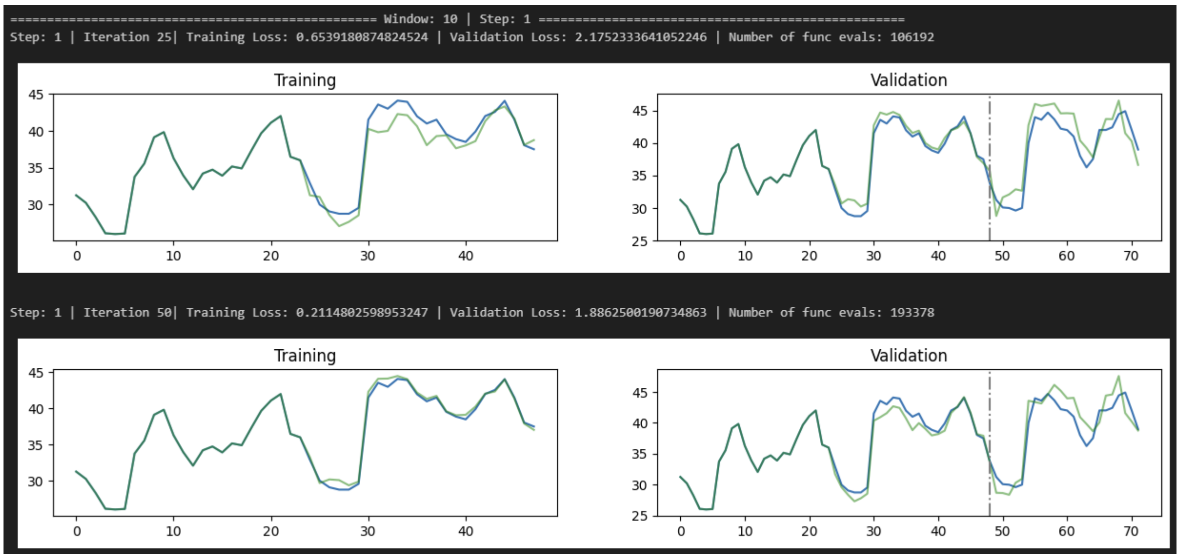
Table 18: CNN-LSTM Results

As it can be seen, the univariate model does not outperform the Naïve model on Tuesdays, Thursdays and Fridays. This might suggest that the data patterns are relatively simple and a complex model like CNN-LSTM model could be overfitting it or just not capturing the patterns effectively. On the other hand, the multivariate approach provides overall outperformance over the univariate, except for Wednesdays.

In summary, these results highlight the trade-offs in model selection and feature engineering in time-series forecasting- While more complex models like CNN-LSTM are more capable to capture patterns in the data, they also risk ending overfitting and not generalizing well. Conversely, incorporating additional relevant exogenous variables can significantly improve model performance, as the multivariate model has demonstrated.

4.4. NEURAL ODE MODEL

For the neural ODE to be computationally competitive with the rest of the models, a time-period of 7 days prior to the out-of-the-sample forecast has been selected. As in the univariate approach, the model considers only the historical price data, each daily time-step will be composed of its 24-hourly prices. In the first window of training, the neural ODE is trained with the first two days of the window. Then, it produces the forecast of the third day, as represented below:



```
============================================== Window: 10 | Step: 1 ==============================================
Step: 1 | Iteration 25| Training Loss: 0.6539180874824524 | Validation Loss: 2.1752333641052246 | Number of func evals: 106192
```

```
Step: 1 | Iteration 50| Training Loss: 0.2114802598953247 | Validation Loss: 1.8862500190734863 | Number of func evals: 193378
```

Figure 39: Training of the first univariate neural ODE window

The architecture of the neural ODE is composed by a neural network, which approximate the derivative of the dynamics of day-ahead electricity price time-series. This neural network has been designed with a feedforward linear layers architecture. The univariate configuration is as follows:

Number of linear layers	*Neurons per layer*	*Learning Rate*	*Epochs*	*Dropout*
4	100	0.01	2000	0.1

Table 19: Univariate neural ODE architecture

As for the multivariate model, as it takes the demand and wind generation exogenous variables, the input and output sizes will be fixed to 72, as the exogenous variables are concatenated with the day-ahead prices. Nevertheless, as it was explained, only 24 outputs are needed. Therefore, the loss function of the neural network will only focus on 24 outputs corresponding to the day-ahead prices. The multivariate neural ODE architecture is as follows:

Number of linear layers	*Neurons per layer*	*Learning Rate*	*Epochs*	*Dropout*
4	200	0.01	2000	0.1

Table 20: Multivariate neural ODE architecture

It must be noted that, as in every multivariate approach, all the variables have been appropriately scaled (using MinMaxScaler from 0 to 1), using the preceding year of data. As it is a good practice, the data should only be scaled on past data. Thus, the problem of data leakage is avoided. Thus, the input window, or initial condition to the IVP problem, will be composed of the flattened three daily time-series, as represented below:

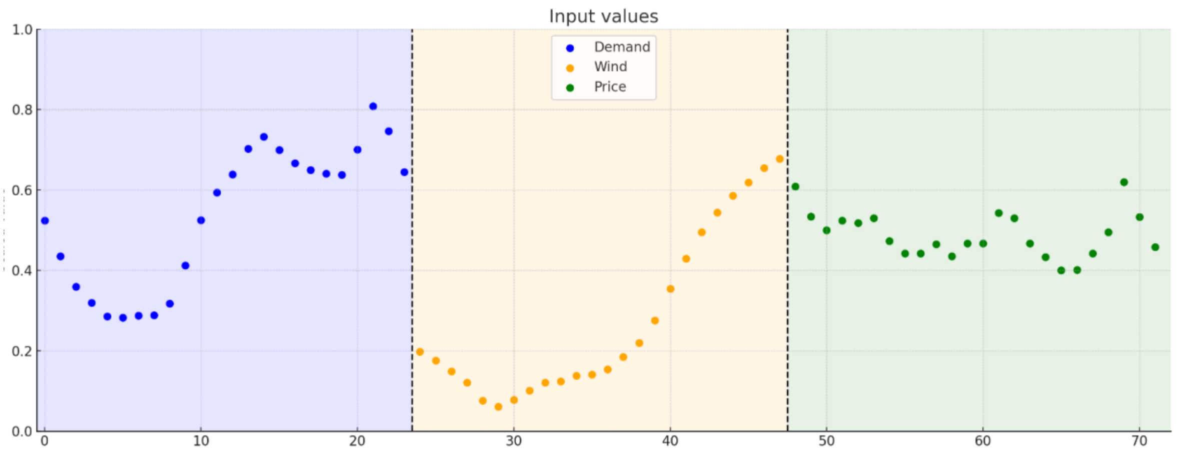


Figure 40: Input values of the multivariate neural ODE

The output of the neural ODE represents the solution of the IVP problem. The model also provides the 'forecast' of the expected energy demand and wind generation, given the dynamics of the artificially flattened time-series. The figure below shows the output values, having fit a rectangle in the last 24 outputs, which represents the final price forecasts for the following day:

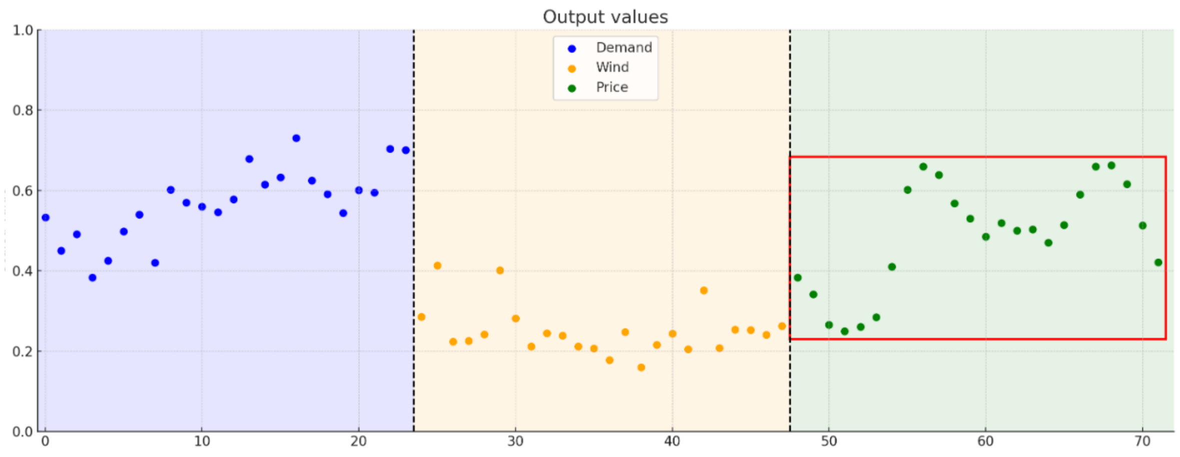


Figure 41: Output of the multivariate neural ODE

During the fitting stage of the model, the training and evaluation processes were plotted. In this case, only the price time-series has been plotted to check that the neural ODE was reconstructing the day-ahead trajectories. This specific model's training consists of 3 days (72 hourly electricity prices), represented in Figure 42:

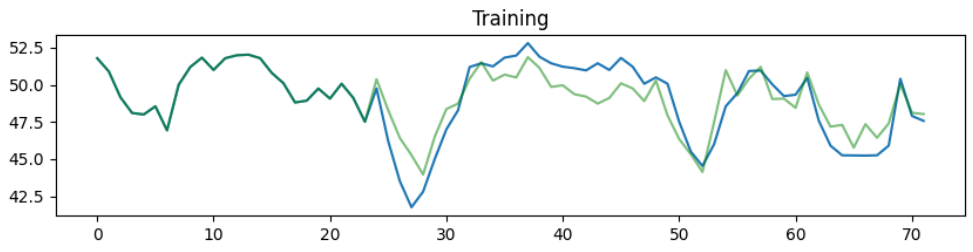


Figure 42: Neural ODE Training

Once the training is complete, the performance of the neural ODE is evaluated. At this point, the time-generative feature of the neural ODE provides the predicted day-ahead price trajectory for the out-of-sample time-steps. The evolution of the first four days' trajectory is as follows:

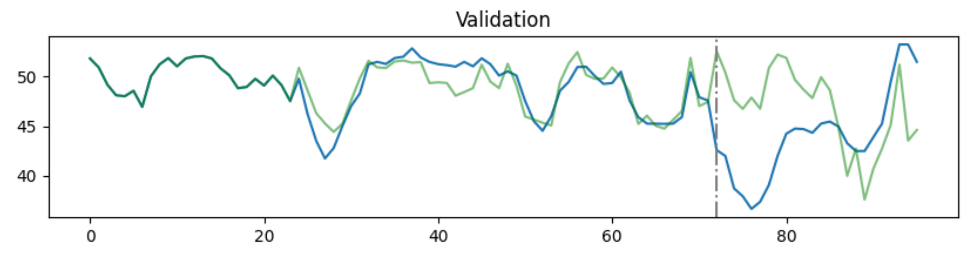


Figure 43: Neural ODE Evaluation

In the context of the rolling window training strategy, the neural ODE performs several training iterations for each time-step. In each iteration, the derivative of the price dynamics is computed in adaptive-size time-steps to optimize the loss function. As commented in the original paper, Ricky T. Q. Chen and his colleagues discussed the depth of neural ODEs and the fact that the number of function evaluations increases throughout training, presumably adapting to the increasing complexity of the dynamics (R. T. Q. Chen et al., 2018). To prove it, the number of times the solver computes the gradient in each time training epoch has been plotted:

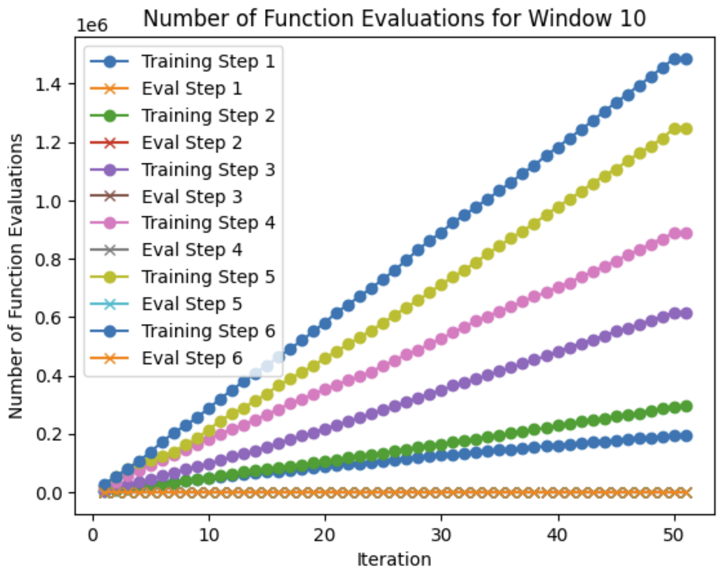


Figure 44: Evolution of the number of function evaluations

Looking at Figure 44, not only does the number of evaluations increase with each training window, but it also does not do it proportionally to the number of steps of the forecast horizon. For example, the second training epoch ends with the solver computing around 300.000 evaluations. This training step is trained with three days and evaluates on the fourth. Then, on the following training step, which includes one more time-step, the model is trained with four days and evaluates on the sixth. Nevertheless, in the third training step, the solver ends evaluating 600.000 times, twice as many as in the second training step which accounts for one less time-step than the third training step.

This is due to the adaptive nature of the solver, the *dopri5*. It can select the time-step size depending on the trajectory of the time-series. If the time-evolution of the electricity price is rapidly oscillating, then it will perform more evaluations than in cases where the price dynamics are more stable.

Neural ODE tackle the IVP problems by parametrizing the function that governs a hypothetical ODE driving the dynamics of the observed variables. As an adaptive solver, *dopri5* continuously chooses the step, to ensure the outputs' error do not exceed an arbitrary tolerance. For this study, the solver will have the default tolerances of the *odeint* library configured. Being the absolute tolerance atol 1e-6 and the relative tolerance rtol 1e-3. However, although the tolerance remains constant for training, the solver adaptivity means that the actual error and the number of function evaluations may change over time. On one hand, the solver adjusts the step size and number of steps based on local error estimates. Moreover, it rejects and retries steps that exceed tolerance, so more evaluations may be done for challenging evaluations. Therefore, the evolution of the number of evaluation points depends on the model setup.

Similarly, as in the DL methods, a model for each day of the week has been studied. The MAE results of the neural ODE's test models are as follows:

Day to forecast	***Naïve***	***Univariate Neural ODE***	***Multivariate Neural ODE***
Monday	8.121	**3.786**	4.358
Tuesday	3.935	**3.460**	4.792
Wednesday	**3.352**	3.915	4.090
Thursday	**3.554**	3.727	3.973
Friday	**2.768**	3.613	3.844
Saturday	4.665	**3.381**	3.799
Sunday	4.093	3.648	**3.550**

Table 21: Neural ODE Results

Interestingly, the multivariate approach, which includes information from the predicted power demand and wind generation, does not generally provide better results than the univariate models. This also happened in the multivariate LSTMs, which have not been capable of extracting new information about the demand and wind generation and use it to compute more accurate forecasts. In fact, the only day in which it performs better is on Sundays.

Moreover, the neural ODE struggles beating the naïve model, which has proven to be competitive against other methods. However, the time-series on weekends and on Mondays seem to be stable and the model can capture the dynamics of the electricity price.

While neural ODEs are designed to run effectively on graphics processing unit (GPU), the computations for this master's thesis have been carried out on central processing unit (CPU), achieving significant results. We would expect to obtain superior performance and inferior computational time required in future work, where the model will be trained on a GPU-based framework.

4.5. SUMMARY

In the following part, the results of the different models are gathered and analyzed depending on their training approach. A summary of the training strategies used for the three families of models is described on Table 22:

Training strategy per method	***Training set***		***Validation set***		***Test set***	
	Start	**End**	**Start**	**End**	**Start**	**End**
Statistical	July 1st, 2018	July 31st, 2019	-		August 1st, 2019	October 31st, 2019
Probabilistic	July 1st, 2018	July 31st, 2019	-		August 1st, 2019	October 31st, 2019
ML	March 1st, 2018	April 30th, 2019	May 1st, 2019	July 31st, 2019	August 1st, 2019	October 31st, 2019
Neural ODE	-		-		August 1st, 2019	October 31st, 2019

Table 22: Summary of training strategies

4.5.1. UNIVARIATE SCENARIO

The resulting MAE loss function values of each univariate model for every day of the week are as follows:

Day to forecast	*Naïve*	*SARIMA*	*Prophet*	*MLP*	*LSTM*	*CNN-LSTM*	*Neural ODE*
Monday	8.121	4.25	4.94	5.331	5.292	5.639	**3.786**
Tuesday	3.935	4.49	4.27	3.941	3.961	4.679	**3.460**
Wednesday	3.352	4.59	3.44	4.322	**3.178**	3.239	3.915
Thursday	**3.554**	3.73	4.09	4.636	4.350	4.503	3.727
Friday	**2.768**	4.01	3.76	3.418	3.309	3.437	3.613
Saturday	4.665	4.56	4.84	3.909	3.908	4.057	**3.381**
Sunday	4.093	6.87	4.68	3.715	3.473	**3.177**	3.648
Average MAE	4.355	4.643	4.289	4.182	3.924	4.104	**3.647**

Table 23: Univariate Results

These results highlight the importance of running day-specific models in day-ahead EPF. This is because different days of the week may have different price dynamics due to external factors like demand patterns, making it suitable to tailor models to specific days.

Looking at the average values of the loss function for the different models, the neural ODE shows dominance. Recall that neural ODE models were run on CPU, limiting the exploration of more complex network architectures. However, they still achieve the best overall results. It must be noted that this does not apply for all days but generally, the neural ODE is the model that makes the most out of the electricity price time-series.

Moreover, there is a day where the naïve model outperforms the rest, and that is on Fridays. This suggests that the electricity prices on Fridays are particularly stable and show a strong one-day lag correlation. In other words, the prices on Fridays are very similar to the prices on Thursdays. This could be due to specific market dynamics, and demand and supply factors. Also, Fridays and Thursdays are candidates of national holidays. It must be noted that some of the holidays that have been considered in most of the multivariate models, have been rearranged to fit on a Thursday or Friday, when the dates fell on weekends.

Apart from that, there is always a univariate model that is going to beat the naïve forecasts. For the specific days where the neural ODE is not the most accurate solution, both LSTM and CNN-LSTM models have shown a good performance. It is the case of Wednesdays, Fridays and Sundays.

In conclusion, even though the models only consider historical price data they have provided more accurate forecasts than the naïve model. Moreover, the neural ODE arises as the model which presented more capabilities to model a single variable. This serves as an important insight as they become extremely interesting for modelling time-series with no significant exogenous variables or with little historical data.

4.5.2. MULTIVARIATE SCENARIO

The resulting loss function values of each univariate model for every day of the week are as follows:

Day to forecast	***Naïve***	***SARIMAX***	***Prophet***	***MLP***	***LSTM***	***CNN-LSTM***	***Neural ODE***
Monday	8.121	4.51	**2.45**	5.213	5.756	3.804	4.358
Tuesday	3.935	4.89	**2.38**	3.693	5.684	3.995	4.792
Wednesday	3.352	5.11	**3.28**	4.076	5.601	3.547	4.090
Thursday	3.554	4.33	**2.86**	4.001	7.084	4.315	3.973
Friday	**2.768**	4.26	2.78	4.173	6.572	3.325	3.844
Saturday	4.665	5.04	**2.34**	3.410	5.653	4.022	3.799
Sunday	4.093	5.41	**2.48**	2.919	6.090	2.755	3.550
Average MAE	4.355	4.793	**2.652**	3.926	6.063	3.680	4.058

Table 24: Multivariate Results

These results highlight a key challenge in time-series forecasting: adding more variables does not necessarily improve its performance. It is crucial to carefully select and preprocess the variables to ensure they allow additional valuable information. Surprisingly enough, the multivariate model with the best overall performance is the Facebook Prophet model. With an astonishing daily average MAE of 2.652 €/MWh, it has outperformed the rest of benchmark models.

Notably, the multivariate LSTM, which although it provided very competitive results on the univariate approach, not only was it not able to capture the underlying dynamics and patterns of the exogenous variables, but it also introduced noise and led to worse performance. Conversely, the LSTM architecture itself might not have been well-suited to modelling the relationships between the variables.

Also, it is interesting to see how the statistical (SARIMAX) did not take advantage of the information provided by the exogenous variables, which underscores how important it is to try different models for our forecasting problem. Although the forecast residuals could be assumed to be uncorrelated, and the hyperparameter selection proved to be significant, the SARIMAX model was not able to catch up with the rest of the models (except for the LSTM).

Therefore, it has been proven that understanding the effects of seasonal patterns is the key for providing more accurate forecasts. This premise brought the Facebook Prophet model to the list of benchmark models, and it has certainly demonstrated it was worth it. Prophet's ability to successfully incorporate additional variables, is particularly beneficial when the present distinct trends or seasonal patterns. Thus, it shows strong performance handling uncertainty and variability in the electricity prices.

Chapter 5. CONCLUSIONS AND FUTURE WORK

In this chapter, the key findings and contributions of this master's thesis are revisited, and the implications of the work are discussed. Also, the limitations of the current study are acknowledged and possible directions for future research are proposed.

5.1. CONCLUSIONS

This study has provided a valuable vision of the capability of neural ODEs in the short-term Spanish day-ahead EPF problem. It must be noted that, as this is the first case study involving neural ODEs for EPF in general, there have been various challenges, such as the absence of references, no possible contrast with other neural ODEs architectures, etc. Nevertheless, they have delivered more than reasonable results, even beating some of the benchmark models developed for the same purpose in certain scenarios.

An important aspect to highlight in the neural ODE approach is its efficiency in the use of data. Unlike other methods, like the benchmark models, that require bigger amounts of data for training, neural ODEs have proven to work with an optimal performance using a considerably smaller dataset size. This characteristic is indeed valuable, showing that neural ODEs, being less data hungry than state-of-the-art models, can generalize with the same level of accuracy as the traditional models.

On one hand, it makes neural ODEs an important option for time-series with limited historical data. Traditional modelling methods often rely on huge datasets to accurately forecast future events. Not surprisingly, data augmentation has become a widespread preprocessing technique, like the employment of generative adversarial networks for time-series. With these data augmentation techniques, the training dataset is artificially extended by applying transformations to the existing data, generating modified copies with the aim of improving the model's performance. Nevertheless, in many practical situations, there is simply not enough historical data.

On the other hand, it also allows a quicker reaction to changing dynamics, like the electricity market. In this regard, one of the drawbacks of the benchmark models is that they were trained mostly with 2018 data and, although the validation set was just before the test set (August-October 2019). Therefore, it surely missed important aspects of the time-series for the recent period prior to the forecasts.

For this project, we selected a dataset spanning the years 2018 and 2019, as this timeframe presented a relatively stable period, when the renewable energies were beginning to penetrate the Spanish electricity market. More recent periods, such as the COVID-19 pandemic or the current year 2023 exhibit more fluctuations and could potentially complicate the initial application and evaluations of the model. As our primary goal was to establish a baseline understanding how neural ODEs performed on EPF problems, we found the 2018-2019 dataset to be more suitable.

Nevertheless, in future work we plan to extend the application of neural ODEs to more challenging periods, including times of significant market volatility, economic recession, and global events like the COVID-19 pandemic. By doing this, we aim to push the boundaries of the capabilities of the model, testing its adaptability and resilience under a broader set of circumstances.

Overall, the obtained results highlight the strengths and weaknesses associated with different state-of-the-art models, used as a benchmark, both from univariate and multivariate perspectives. Remarkably, the naïve model still provides a better estimation for Fridays' prices on the univariate approach. Nevertheless, for the rest of the days and for the whole multivariate approach, there is always a better option. Surprisingly enough, the Facebook Prophet model, equipped with capabilities to model seasonal patterns and trends, emerges as the reference model for multivariate EPF, showing its strength in handling variability and uncertainty in electricity prices.

When considering the multivariate models, this study has proven that the mere addition of variables does not always improve forecasting performance. However, it highlights the necessity for careful variable selection and preprocessing to ensure that it will make a valuable and meaningful input. In this context, it must be noted that the SARIMAX model was unable to leverage the additional information provided by the exogenous variables effectively, emphasizing the importance of exploring different methods for a forecasting problem.

As the univariate neural ODE showed dominant performance over the rest of the models, it could be interesting to analyze other time-series where this innovative method could significantly improve short-term forecasts. This is the case, for example, of the sales of new market products, where there is limited historical data available. Another case would be the time-series related to rare events, such as natural disasters. Neural ODEs could be a valuable tool to work with these smaller datasets.

Finally, and as it has been discussed recently, Neural ODEs are very suitable for real-world irregularly sampled data (Kidger et al., 2020), from medical data, where the measurements are taken depending on the patient's condition and treatment schedule to financial data, where trading volumes can fluctuate throughout the day.

5.2. FUTURE WORK

The exploration of Neural ODEs for time-series forecasting, particularly in EPF, has yielded promising results. However, there is a vast potential for further research and development in this area.

As mentioned, neural ODEs are especially suitable for irregularly sampled time-series or with limited historical data. Extensive research is being carried out to introduce Neural Controlled Differential Equations to real world applications. So far, with its capability of introducing incoming data, its memory-efficient adjoint-based backpropagation, and its state-of-the-art performance (Kidger et al., 2020), neural CDEs have proven to be flexible and robust to handle this kind of time-series.

Moreover, recent developments in the field of neural ODEs have focused on improving their generalization capabilities, robustness to noisy data, and interpretability of the learned representations. For example, researchers have explored the use of normalizing flows to model complex distributions in the hidden state, as well as the integration of causal inference techniques to better understand the relationships between inputs and outputs (Habiba & Pearlmutter, 2020).

Another attractive feature of Neural ODEs is their compatibility with other ML architectures, such as the benchmark models proposed in this study. This was demonstrated in previous studies (Lago et al., 2018) as modelling non-linearity in the electricity price data is key. For example, they can be combined with CNNs to effectively capture spatial patterns in data, or with RNNs to better handle temporal dependencies (De Brouwer et al., 2020). These hybrid models allow for a greater exploitation of the strengths of each approach, potentially leading to improved predictive performance.

However, the computational demands of neural ODEs represent a significant challenge. The power of neural ODEs for time-series forecasting is proportional to the amount of available computation. For this master's thesis, it has been a major limitation and, as a future step, the migration to GPUs systems is mandatory. Its use allows for the exploration of more complex and computationally expensive models. Also, it could enable the modelling of larger neural network architectures, more sophisticated training techniques and more accurate ODE solvers.

Chapter 6. BIBLIOGRAPHY

Anonymous, & ICLR. (2023). *S-SOLVER : NUMERICALLY STABLE*. *5*(4), 1–11.

Box, G. E., & Cox, D. R. (1982). An analysis of transformations revisited, rebutted. *Journal of the American Statistical Association*, *77*(377), 209–210. https://doi.org/10.1080/01621459.1982.10477788

Box, G. E. P., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2016). *Time Series Analysis. Forecasting and Control*.

Chang, Z., Zhang, Y., & Chen, W. (2019). Electricity price prediction based on hybrid model of adam optimized LSTM neural network and wavelet transform. *Energy*, *187*, 115804. https://doi.org/10.1016/j.energy.2019.07.134

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, *2018-Decem*, 6571–6583.

Chen, Y., Wang, Y., Ma, J., & Jin, Q. (2019). BRIM: An accurate electricity spot price prediction scheme-based bidirectional recurrent neural network and integrated market. *Energies*, *12*(11). https://doi.org/10.3390/en12122241

Conejo, A., Plazas, M., & Espínola, R. (2006). Day-ahead electricity price forecasting using the wavelet analysis and MPMR models. *IET Conference Publications*, *20*(523 CP), 1035–1042. https://doi.org/10.1049/cp:20062176

Contreras, J., Espínola, R., Nogales, F. J., & Conejo, A. J. (2003). ARIMA models to predict next-day electricity prices. *IEEE Transactions on Power Systems*, *18*(3), 1014–1020. https://doi.org/10.1109/TPWRS.2002.804943

Cruz, A., Muñoz, A., Zamora, J. L., & Espínola, R. (2011). The effect of wind generation and weekday on Spanish electricity spot price forecasting. *Electric Power Systems Research*, *81*(10), 1924–1935. https://doi.org/10.1016/j.epsr.2011.06.002

De Brouwer, E., Simm, J., Arany, A., & Moreau, Y. (2020). GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. *Belgian/Netherlands Artificial Intelligence Conference, NeurIPS*, 364–366.

Ebrahimian, H., Barmayoon, S., & Mohammadi, M. (2018). The price prediction for the energy market based on a new method. *Economic Research-Ekonomska Istraživanja*, *31*(1), 1–25. https://doi.org/10.1080/1331677X.2018.1429291

Garcia, R. C., Contreras, J., van Akkeren, M., & Garcia, J. B. C. (2005). A GARCH forecasting model to predict day-ahead electricity prices. *IEEE Transactions on Power Systems*, *20*(2), 867–874. https://doi.org/10.1109/TPWRS.2005.846044

Goeken, D., & Johnson, O. (2000). Runge-Kutta with higher order derivative approximations. *Applied Numerical Mathematics*, *34*(2), 207–218. https://doi.org/10.1016/S0168-9274(99)00128-2

González, A. M., San Roque, A. M., & García-González, J. (2005). Modeling and forecasting electricity prices with input/output hidden Markov models. *IEEE Transactions on Power Systems*, *20*(1), 13–24. https://doi.org/10.1109/TPWRS.2004.840412

Guerrero, V. M., & Perera, R. (2004). Variance stabilizing power transformation for time series. *Journal of Modern Applied Statistical Methods*, *3*(2), 357–369. https://doi.org/10.22237/jmasm/1099267740

Habiba, M., & Pearlmutter, B. A. (2020). Neural Ordinary Differential Equation based Recurrent Neural Network Model. *2020 31st Irish Signals and Systems Conference, ISSC 2020*, *1*, 1–8. https://doi.org/10.1109/ISSC49989.2020.9180182

Hamad, R. A., Yang, L., Woo, W. L., & Wei, B. (2020). Joint learning of temporal models to handle imbalanced data for human activity recognition. *Applied Sciences (Switzerland)*, *10*(15). https://doi.org/10.3390/APP10155293

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., & Muller, P. A. (2019). Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, *33*(4), 917–963. https://doi.org/10.1007/s10618-019-00619-1

Jiang, L., & Hu, G. (2018). Day-Ahead Price Forecasting for Electricity Market using Long-Short Term Memory Recurrent Neural Network. *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, *November*, 949–954. https://doi.org/10.1109/ICARCV.2018.8581235

Keynia, F. (2012). Engineering Applications of Artificial Intelligence A new feature selection algorithm and composite neural network for electricity price forecasting. *Engineering Applications of Artificial Intelligence*, *25*(8), 1687–1697.

https://doi.org/10.1016/j.engappai.2011.12.001

Kidger, P. (2022). *On Neural Differential Equations*. http://arxiv.org/abs/2202.02435

Kidger, P., Morrill, J., Foster, J., & Lyons, T. (2020). Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, *2020-Decem*(1), 1–12.

Lago, J., De Ridder, F., & De Schutter, B. (2018). Forecasting spot electricity prices: Deep learning approaches and empirical comparison of traditional algorithms. *Applied Energy*, *221*, 386–405. https://doi.org/10.1016/j.apenergy.2018.02.069

Lago, J., Marcjasz, G., De Schutter, B., & Weron, R. (2021). Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark. *Applied Energy*, *293*(December 2020), 116983. https://doi.org/10.1016/j.apenergy.2021.116983

Lin, T., Horne, B. G., Tino, P., & Giles, C. L. (1996). Learning long-term dependencies in NARX recurrent neural networks. *IEEE Transactions on Neural Networks*, *7*(6), 1329–1338. https://doi.org/10.1109/72.548162

Naz, A., Javed, M. U., & Javaid, N. (2019). *Short-Term Electric Load and Price Forecasting Using Smart Grids*. https://doi.org/10.3390/en12050866

NIST. U.S. Commerce Department, N. (2003). *Ljung-Box Test*. Dataplot. Statistical Engineering Division. https://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/ljungbox.htm

REE. (2020). *Informe Sistema Electrico Español*. 100. http://www.ree.es/es/publicaciones/sistema-electrico-español/informe-del-sistema-electrico-espanol-2012

Rob J Hyndman. (2014). *Forecasting: Forecasting: Principles & Practice*. *September*, 138. robjhyndman.com/uwa%5Cnhttp://robjhyndman.com/papers/forecasting-age-specific-breast-cancer-mortality-using-functional-data-models/

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. 1–14. http://arxiv.org/abs/1609.04747

Taylor, S. J., & Letham, B. (2017). Business Time Series Forecasting at Scale. *PeerJ Preprints 5:E3190v2*, *35*(8), 48–90.

Wang, L., Zhang, Z., & Chen, J. (2017). Short-Term Electricity Price Forecasting with Stacked Denoising Autoencoders. *IEEE Transactions on Power Systems*, *32*(4), 2673–2681. https://doi.org/10.1109/TPWRS.2016.2628873

Weron, R. (2014). Electricity price forecasting: A review of the state-of-the-art with a look into the future. *International Journal of Forecasting*, *30*(4), 1030–1081. https://doi.org/10.1016/j.ijforecast.2014.08.008

Chapter 7. ANNEX

7.1. UNIVARIATE NEURAL ODE CODE

```
# Check if CUDA is available
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

# Model and optimizer definition
func = ODEFunc(input_size=24, hidden_layer_neurons=100, output_size=24)
optimizer = optim.RMSprop(func.parameters(), lr=LEARNING_RATE)

# Mean Absolute Error loss function
loss_fn = nn.L1Loss()

train_spot_tensor = torch.from_numpy(train_sc).float()

train_losses = [] # For storing the loss value during training
valid_losses = [] # For storing the loss value during eval
best_eval_accuracy = []
best_train_loss = float('inf') # initialize to infinity
best_valid_loss = float('inf') # initialize to infinity
nfe_history = {} # For storing number of function evaluations during training
nfe_history_eval = {} # For storing number of function evaluations during
evaluation

for j in range(num_windows):
    nfe_history[j] = {}
    nfe_history_eval[j] = {}

    for i in range(1, TRAIN_STEPS+1):
        # Resetting the function evaluations counter at the beginning of each
step within window
        func.nfe = 0
        nfe_history[j][i] = {}

        # Train
        train_loss = 0
        func.train()
        print("="*50 + f" Window: {j+1} | Step: {i} " + "="*50)
        batch_y0 = train_spot_tensor[24*j:24*(j+1)]  # y0
        batch_t = torch.from_numpy(np.arange(0, i+1, dtype=float))  # [0, ..., i]

        # Train neural ODE N_ITR times
        for itr in range(1, N_ITR+1):
            optimizer.zero_grad() # initializing gradients

            # Next time-step ODE computation
            pred_spot_sc = odeint(func=func, y0=batch_y0, t=batch_t, method =
'fehlberg2').to(device)  # pred_spot shape: (i+1, 24)
            pred_spot_flat = pred_spot_sc.flatten()
            target_sc = train_spot_tensor[24*j:24*(j+i+1)]

            # Computing the training loss
            step_loss = loss_fn(pred_spot_flat, target_sc)
            step_loss.backward()
            optimizer.step()

            # Inverse scaling the forecast and actual values to compute the loss
in the original scale
            pred_spot_sc = pred_spot_flat.detach().cpu().numpy()
            pred_spot = scaler_price.inverse_transform(pred_spot_sc.reshape(-
1,1))
            pred_spot_tensor = torch.from_numpy(pred_spot)

            target_sc_numpy = target_sc.detach().cpu().numpy()
            target_sc_numpy =
scaler_price.inverse_transform(target_sc_numpy.reshape(-1,1))
            target_sc_tensor = torch.from_numpy(target_sc_numpy)

            loss = loss_fn(pred_spot_tensor, target_sc_tensor)
            train_loss += loss.item()
            nfe_history[j][i][itr] = func.nfe

            if itr % (N_ITR // 2) == 0:
                # Evaluation
                valid_loss = 0

                with torch.no_grad():
                    nfe_before_eval = func.nfe
                    func.eval() # eval mode
                    batch_t_eval = torch.from_numpy(np.arange(0, i+1+TEST_SIZE,
dtype=float))  # [0, ..., i+TEST_SIZE]

                    # After training for 50 epochs, it is time to evaluate on the
following time-step (out-of-sample forecast)
                    pred_spot_eval = odeint(func, batch_y0, batch_t_eval, method
= 'fehlberg2')  # pred_spot_eval shape: (i+1+TEST_SIZE, 24)

                    # Inverse scaling the forecast and actual values to compute
the loss in the original scale
                    pred_spot_eval_1 = pred_spot_eval.cpu().numpy()
                    pred_spot_1 =
scaler_price.inverse_transform(pred_spot_eval_1.reshape(-1,1))
                    pred_spot_eval_1_tensor = torch.from_numpy(pred_spot_1)
                    pred_spot_eval_sc = pred_spot_eval[-
TEST_SIZE:].flatten().cpu().numpy()
                    pred_spot =
scaler_price.inverse_transform(pred_spot_eval_sc.reshape(-1,1))
                    pred_spot_eval_tensor = torch.from_numpy(pred_spot)

                    target_eval_sc =
train_spot_tensor[24*(j+i+1):24*(j+i+1+TEST_SIZE)].cpu().numpy()
                    target_sc_numpy =
scaler_price.inverse_transform(target_eval_sc.reshape(-1,1))
                    target_eval_tensor = torch.from_numpy(target_sc_numpy)

                    # Computing the eval loss
                    loss_eval = loss_fn(pred_spot_eval_tensor,
target_eval_tensor)
                    valid_loss += loss_eval.item()

                    nfe_after_eval = func.nfe  # storing the number of function
evaluations after evaluation
                    nfe_eval = nfe_after_eval - nfe_before_eval  # calculating
the number of function evaluations during evaluation
                    nfe_history_eval[j][i] = nfe_eval  # storing the number of
function evaluations during evaluation

                    print(f"Step: {i} | Iteration {itr}| Training Loss:
{loss.item()} | Validation Loss: {loss_eval.item()} | Number of func evals:
{nfe_history[j][i][itr]}")

                    if itr == N_ITR and i == TRAIN_STEPS:
                        best_eval_accuracy.append(valid_loss)

                    # Inverse transforming before plotting Training
                    fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(15, 2))
                    ax0.plot(train_spot[24*j:24*(j+i+1)])
                    ax0.plot(pred_spot_tensor.detach().numpy(), color="green",
alpha=.5)
                    ax0.set_title("Training")

                    # Inverse transforming before plotting Validation
                    ax1.plot(train_spot[24*j:24*(j+i+1+TEST_SIZE)])
                    ax1.plot(pred_spot_eval_1_tensor.detach().numpy(),
color="green", alpha=.5)
                    ax1.set_title("Validation")
                    ax1.axvline(x=24*(i+1), color="grey", ls="dashdot")
                    plt.show()

                func.train() # back to training mode where backpropgation will be
perfomed (via the adjoint sensitivity method)

        # Appending the number of function evaluations to the history at the end
of each step of a window
        nfe_history[j][i]["total"] = func.nfe
        train_losses.append(best_train_loss)
        valid_losses.append(best_valid_loss)
```

7.2. MULTIVARIATE NEURAL ODE CODE

```
# Checking if CUDA is available
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")

# Model and optimizer definition
func = ODEFunc(input_size=72, hidden_layer_neurons=200, output_size=72)
optimizer = optim.RMSprop(func.parameters(), lr=LEARNING_RATE)

# Mean absolute error loss function
loss_fn = nn.L1Loss()

train_losses = [] # For storing the loss value during training
valid_losses = [] # For storing the loss value during eval
loss_eval_original = []
best_eval_accuracy = []
best_train_loss = float('inf')  # initializing to infinity
best_valid_loss = float('inf')  # initializing to infinity
nfe_history = {}  # For storing number of function evaluations during training
nfe_history_eval = {}  # For storing number of function evaluations during
evaluation

for j in range(train_num_windows):
    nfe_history[j] = {}
    nfe_history_eval[j] = {}

    for i in range(1, TRAIN_STEPS+1):
        # Resetting the function evaluations counter at the beginning of each
step within window
        func.nfe = 0
        nfe_history[j][i] = {}

        # Train
        train_loss = 0
        func.train()
        print("="* 50 + f" Window: {j+1} | Step: {i} " + "="*50)
        batch_y0 = train_spot_tensor[72*j:72*(j+1)] # y0
        batch_t = torch.from_numpy(np.arange(0, i+1, dtype=float)) # [0, ..., i]

        # Train neural ODE N_ITR times
        for itr in range(1, N_ITR+1):
            optimizer.zero_grad()

            # Next time-step ODE computation
            pred_data = odeint(func=func, y0=batch_y0, t=batch_t, method =
'fehlberg2').to(device) # pred_data shape: (i+1, 72)
```

```
            # Computing the training loss
            loss = loss_fn(pred_data[:, -24:].flatten(), 
price_tensor[j*24:(j+i+1)*24])
            loss.backward()
            optimizer.step()

            # Inverse scaling the forecast and actual values to compute the loss 
in the original scale
            loss_train_original = 
loss_fn(torch.tensor(inverse_transform(pred_data[:, -24:].flatten(), 
scaler_price)), torch.tensor(inverse_transform(price_tensor[j*24:(j+i+1)*24], 
scaler_price)))
            train_loss += loss_train_original.item()
            nfe_history[j][i][itr] = func.nfe

            if itr % (N_ITR // 2) == 0:
                # Evaluation
                valid_loss = 0
                loss_eval_original = 0

                with torch.no_grad():
                    nfe_before_eval = func.nfe
                    func.eval() # eval mode
                    batch_t_eval = torch.from_numpy(np.arange(0, i+1+TEST_SIZE, 
dtype=float))  # [0, ..., i+TEST_SIZE]

                    # After training for N_ITR epochs, it is time to evaluate on 
the following time-step (out-of-sample forecast)
                    pred_data_eval = odeint(func, batch_y0, batch_t_eval, method 
= 'fehlberg2')  # pred_data_eval shape: (i+1+TEST_SIZE, 72)
                    loss_eval = loss_fn(pred_data_eval[-TEST_SIZE:, -
24:].flatten(), price_tensor[24*(j+i+1):24*(j+i+1+TEST_SIZE)])
                    valid_loss += loss_eval.item()

                    # Inverse scaling the forecast and actual values to compute 
the loss in the original scale
                    loss_eval_transformed = 
loss_fn(torch.from_numpy(inverse_transform(pred_data_eval[-TEST_SIZE:, -
24:].flatten(), scaler_price)), 
torch.tensor(inverse_transform(price_tensor[24*(j+i+1):24*(j+i+1+TEST_SIZE)], 
scaler_price)))
                    nfe_after_eval = func.nfe  # storing the number of function 
evaluations after evaluation
                    nfe_eval = nfe_after_eval - nfe_before_eval  # calculating 
the number of function evaluations during evaluation
                    nfe_history_eval[j][i] = nfe_eval  # storing the number of 
function evaluations during evaluation

                    # Computing the eval loss
                    loss_eval_original += loss_eval_transformed.item()
                    print(f"Step: {i} | Iteration {itr}| Training Loss:
{loss_train_original.item()} | Validation Loss: {loss_eval_transformed.item()} |
Number of func evals: {nfe_history[j][i][itr]}")

                    if itr == 30 and i == TRAIN_STEPS:
                        best_eval_accuracy.append(loss_eval_original)

                    fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(15, 2))

                    # Inverse transforming before plotting Training
                    ax0.plot(inverse_transform(price_tensor[j*24:(j+i+1)*24],
scaler_price))
                    ax0.plot(inverse_transform(pred_data[:, -
24:].flatten().detach(), scaler_price), color="green", alpha=.5)
                    ax0.set_title("Training")

                    # Inverse transforming before plotting Validation
                    ax1.plot(inverse_transform(price_tensor[24*j:24*(j+i+1+TEST_S
IZE)], scaler_price))
                    ax1.plot(inverse_transform(pred_data_eval[:, -
24:].flatten().detach(), scaler_price), color="green", alpha=.5)
                    ax1.set_title("Validation")
                    ax1.axvline(x=24*(i+1), color="grey", ls="dashdot")
                    plt.show()

                func.train() # back to training mode where backpropgation will be
perfomed (via the adjoint sensitivity method)

        train_losses.append(best_train_loss)
        valid_losses.append(best_valid_loss)

        # Appending the number of function evaluations to the history at the end
of each step of a window
        nfe_history[j][i]["total"] = func.nfe
        train_losses.append(best_train_loss)
        valid_losses.append(best_valid_loss)
```

7.3. ALIGNMENT WITH SDGS

7.3.1. SDG 7

This project aligns with the seventh Sustainable Development Goal (SDG) by *helping develop a more sustainable market and a more affordable electricity price.*

Figure 45: SDG 7

Although technology is continuously improving and the energy mix is greener than ever, there is still much work to be done to improve access to affordable and clean energy. To combat this issue, countries have accelerated their transitions to an affordable, reliable, and sustainable energy system by adopting clean energy technologies and infrastructure.

In a world increasingly committed to caring for the environment, it is essential to create models that better predict resources to help make decisions about the energy mix. With more accurate predictions, utility companies, which are very dependent on electricity prices, can obtain an optimal production schedule. This would result in lower energy costs as the risk of facing unexpected conditions that would cause a disruption in price forecasts decreases. Moreover, energy demand is increasing every year. In 2021, there was a 6% increase due to strong economic growth, combined with the effects of global warming, which produced hotter summers and colder winters.

The first target of SDG7 is "to ensure universal access to affordable, reliable, and modern services by 2030." There are still 789 million people worldwide lacking access to electricity. Without it, women and girls in sub-Saharan Africa must spend hours fetching water, and people cannot maintain competitive businesses to make a living. In 2020, during the Covid-19 pandemic, there were problems with vaccine distribution in Africa as some clinics in smaller cities could not store them due to a lack of necessary energy. Energy services are crucial for preventing disease and fighting pandemics, providing power to healthcare facilities, supplying clean water for essential hygiene, and enabling communications and IT services that connect people while maintaining social distancing. Lack of access to energy resources is a significant impediment to tackling the pandemic and the next health crisis to come.

The second target of SDG7 is "to increase substantially the share of renewable energy in the global energy mix by 2030." Earth Overshoot Day marks the date when humanity has used all the biological resources that Earth can regenerate during the entire year. In 2022, it landed on the 28th of July, meaning resources were demanded almost twice as fast as they should have been to be sustainable. Furthermore, government financial assistance for developing countries in renewable energy has decreased for the second consecutive year, and this impressive progress has slowed. Current predictions indicate that by 2030, the number will have decreased to 679 million.

The third target of SDG7 is "by 2030, double the global rate of improvement in energy efficiency." In Spain, household energy bills have increased by 70%, and energy poverty is on the rise, affecting 4.5 million people. A significant part of the solution lies in energy efficiency. It is crucial to address this issue, as much energy is lost due to inadequate building climatization. Apart from relying on greener energies, governments should promote investment in clean energy infrastructure and technology.

In conclusion, through better EPF, there would be a non-negligible cost reduction in market operation, and energy would become more accessible to people. This project also aligns with other SDGs, such as SDG9 (Industry, Innovation, and Infrastructure) and SDG13 (Climate

Action), by promoting innovative forecasting methods, reducing carbon emissions, and encouraging a more sustainable energy infrastructure.

7.3.2. SDG 9

This project supports SDG9 by *promoting innovation for a more sustainable industry*.

Figure 46: SDG 9

By studying the development of a potentially more accurate, efficient, and reliable energy forecasting model such as Neural ODE, this project will drive technological progress in the energy sector. As energy companies adopt these advanced models, they will be better equipped to optimize their operations, reduce inefficiencies, and invest in clean energy infrastructure.

The adoption of innovative forecasting methods can lead to the development of smarter energy grids and more efficient energy management systems. This, in turn, can improve the overall stability and reliability of the energy network, benefitting both consumers and businesses. Furthermore, the project's alignment with SDG9 has the potential to create new job opportunities in the energy sector, particularly in research, development, and implementation of recent forecasting technologies.

By contributing to the advancement of the energy sector, this project supports the development of resilient infrastructure, promotes inclusive and sustainable industrialization, and fosters innovation, which are the main objectives of SDG9.

7.3.3. SDG 13

This project also aligns with SDG13 by *encouraging a more sustainable energy infrastructure and reducing carbon emissions*.

Figure 47: SDG 13

With more accurate electricity price forecasts, utility companies can better plan their production schedules and more efficiently allocate resources to renewable energy sources. By promoting the use of cleaner energy, the project supports efforts to mitigate climate change and reduce greenhouse gas emissions.

Accurate forecasting can also help identify and prioritize investment opportunities in renewable energy and energy efficiency measures. This can lead to a decrease in the reliance on fossil fuels, which is a significant contributor to climate change. By providing the tools to optimize energy production and consumption, this project can indirectly contribute to the implementation of climate change mitigation measures and the development of more sustainable energy policies.

Moreover, the project supports the global efforts to meet the targets set by the Paris Agreement, which aims to limit global warming to well below 2°C above pre-industrial levels and to pursue efforts to limit the temperature increase to 1.5°C. By reducing emissions and fostering renewable energy adoption, the project contributes to the overall goal of building a low-carbon future.

To summarize, the project is not only aligned with SDG7, but it also supports SDG9 and SDG13 by promoting innovative technologies and contributing to climate action. More accurate electricity price forecasts can lead to a more sustainable and efficient energy infrastructure, helping to reduce climate change. Additionally, the project encourages innovation in the energy sector. By aligning with these crucial Sustainable Development Goals, the project has the potential to make an impact on both the energy sector and the global environment.