

APLICACIÓN DE ECUACIONES DIFERENCIALES NEURONALES PARA ESTABILIZAR EL VOLTAJE EN LA RED ELÉCTRICA ESPAÑOLA

¿CÓMO PUEDEN APLICARSE LAS ECUACIONES DIFERENCIALES
NEURONALES AL CONSUMO DE LA RED ELÉCTRICA ESPAÑOLA PARA
ASEGURAR UN VOLTAJE CONSTANTE?

CÓMPUTO DE PALABRAS: 3999

Índice

1. **Introducción** – 3
2. **Conocimientos básicos sobre Machine Learning** – 3
3. **Los Neural ODE como modelo matemático** – 7
4. **Ejemplo teórico de los Neural ODE** – 8
5. **Ejemplo práctico de los Neural ODE aplicados a los datos de la REE** – 10
6. **Las cuatro piezas de código** – 17
7. **Entrenamiento y resultados** – 17
8. **Conclusiones** – 19
 - 8.1 **Limitaciones, mejoras y ampliaciones** – 20
9. **Bibliografía** – 21
10. **Anexo** – 22

1-Introducción

Las redes eléctricas operan a un voltaje constante, esto es un hecho en todas las partes del mundo. Si este voltaje cambia lo suficiente, los dispositivos enchufados a la corriente dejan de funcionar. Para mantener el voltaje constante, se realizan complejas predicciones para equilibrar la producción eléctrica con el consumo. El objetivo de esta monografía es predecir el consumo de la red eléctrica española (REE) utilizando ecuaciones neuronales diferenciales, pues al equiparar el suministro a la demanda el voltaje de la red se mantiene estable.

Esto es importante pues tradicionalmente la predicción de consumo eléctrico se ha hecho manualmente, sin embargo, un modelo de inteligencia artificial como “long short term memory (LSTM)” o, en este caso, ecuaciones diferenciales ordinales neuronales “neural ODE’s” son especialmente útiles cuando se trata de predecir modelos continuos en el tiempo, ya que modelan el sistema que rige a los datos, una función, y la utilizan para ver cómo van a ser los datos en el futuro. Por tanto, un sistema de predicción automática puede ser más preciso y servir de ayuda en la tarea de mantener el voltaje estable en una red eléctrica.

Para medir el éxito de la red neuronal vamos a tomar un enfoque cuantitativo. Se examinarán los datos reales y se compararán con las predicciones del modelo, de ahí obtendremos un porcentaje de acierto dentro de un margen de error estándar en el mundo real.

Esta monografía consecuentemente va a tener un alcance explorativo, ya que las ecuaciones diferenciales neuronales son un concepto novedoso y que hasta día de hoy tiene aplicaciones limitadas. Pero también analítico, pues vamos a determinar si una inteligencia artificial puede alcanzar el nivel de predicción que se obtiene hoy en día con humanos programas de análisis de datos.

2-Conocimientos básicos sobre machine learning

Machine learning es un aspecto del campo de estudios de la inteligencia artificial. Es la base de todos los modelos y arquitecturas que se pueden usar. El objetivo de machine learning es utilizar datos para crear un modelo capaz de realizar una tarea, y medir la eficiencia de ese modelo numéricamente. El aprendizaje es el proceso donde uno encuentra el modelo adecuado cambiando parámetros como la arquitectura, las dimensiones de los datos o el número de pasadas que el modelo le da a los datos, midiendo este aprendizaje de forma numérica con varios indicadores. El objetivo de este proceso no es conseguir el modelo que mejor se ajuste a los datos que se tienen, sino el que sea capaz de mejor predecir los datos futuros.

La información que ya se tiene y se le da al modelo para que aprenda se conoce como “input data”. Esta va a ser un vector con un número de dimensiones positivas y enteras. Los datos que el modelo trata de predecir se conocen como “output data”. Estos, si bien pueden ser un vector, en la mayoría de los casos van a ser un valor numérico escalar.

Cuando uno está entrenando un modelo en machine learning, los datos se dividen en tres tipos, entrenamiento, test y validación. El más importante por ahora es el set de entrenamiento, pues los otros son para asegurarse que el modelo se ha entrenado correctamente. El set de entrenamiento viene en parejas $(x_1, y_1) \dots (x_n, y_n)$. Donde x es el conjunto de variables que se suministran al modelo, combinadas en forma de vector con tantas dimensiones como variables haya, y es el dato esperado del modelo, en valor numérico.

Una vez el modelo esta entrenado, se mide la diferencia entre los datos reales y las predicciones del modelo con una función llamada “loss function” o función de pérdidas, L . El problema entonces se reduce a optimización matemática, donde se busca minimizar esta función encontrando los valores ideales para varias variables de las que hablaremos más adelante. Tal que $\mu = \arg \min_{\theta} L(\hat{y})$. Donde μ es el valor que minimiza la función de pérdidas, $\arg \min$ es notación para representar el argumento que minimiza, es decir, estamos buscando el valor del parámetro θ que minimiza $L(\hat{y})$, siendo $L(\hat{y})$ la propia función de pérdidas. Cabe anotar que hay muchas funciones de pérdidas dependiendo de la arquitectura elegida, esto es una generalización para comprender su funcionamiento.

En un problema de regresión, se busca predecir valores continuos, es decir, un valor numérico para cada valor de tiempo individual, a partir de valores de entradas, como hemos discutido anteriormente. La función de pérdidas más popular para problemas de regresión y consecuentemente la que vamos a aplicar va a ser MSE “Mean Square Error” (Error cuadrático medio). Su fórmula matemática es la siguiente:

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

Esta fórmula representa el valor medio de la diferencia entre el valor real y_i y el valor predicho \hat{y}_i , elevada al cuadrado. Donde N representa el número total de datos observados hasta ese punto. Al estar elevado al cuadrado, el valor resultante siempre va a ser positivo, y por la misma razón los errores más grandes se penalizan más duramente.

Ahora que se han expuesto los conocimientos necesarios sobre aprendizaje automático, vamos a dar una base teórica sobre ecuaciones diferenciales ordinarias (ODE's) para entender el resto de este trabajo.

Lo primero que es necesario entender es el problema del valor inicial. El problema del valor inicial es aquel en el que se define una ecuación diferencial ordinaria de forma

$$\frac{d\varphi}{dt} = f(t, x(t)), \quad x(t_0) = x_0$$
 lo que significa que $f(t, x(t))$ es la función que describe como

cambia x con respecto al tiempo. El problema del valor inicial IVP por sus siglas en inglés, busca una función que, vamos a considerar, por convención, $\varphi(t) \cdot \varphi(t) \equiv x(t)$.

$\varphi(t)$ tiene que satisfacer las siguientes condiciones:

$$\varphi(t_0) = x_0, \text{ es decir, la función en } t_0 \text{ debe comenzar en } x_0,$$

$(t, \varphi(t)) \in D$, la función debe mantenerse dentro del dominio D , siendo D la región de valores con dimensiones \mathbb{R}^{n+1} , n es la cantidad de dimensiones que tiene el vector x (provenientes del set de entrenamiento) y el +1 viene de la otra variable, el tiempo.

$$\frac{d\varphi}{dt} = f(t, \varphi(t)),$$
 la derivada de $\varphi(t)$ con respecto al tiempo está determinada por la función

f . Es decir, f describe cómo debe cambiar $\varphi(t)$ a medida que el tiempo avanza, de acuerdo con la ecuación diferencial.

La solución única de un IVP se anota tal que $\varphi(t; t_0, x_0)$, lo que indica que la solución depende del tiempo y de las condiciones iniciales.

Por último, es necesario conocer el teorema de Picard, este teorema, que ahora explicaremos, es imperativo para verificar la validez de nuestros resultados, ya que va a comprobar si la solución que hemos encontrado al teorema del valor inicial es determinada y única, por lo que nunca se va a dar el caso de dos posibles consumos para una única demanda, pues la solución correcta va a ser la que cumpla con este teorema. El teorema de Picard dice que nuestra solución debe de cumplir con las condiciones de Lipschitz, es decir, la función con respecto a y tiene que cumplir la desigualdad:

$$\sqrt{\sum_{i=1}^n (f_i(t, x_1) - f_i(t, x_2))^2} \leq L \sqrt{\sum_{i=1}^n (x_{1,i} - x_{2,i})^2}, \quad L > 0$$

Esta desigualdad quiere decir que la función con respecto a su segunda variable, x , no cambia más rápido que la constante de Lipschitz, L . Esto nos asegura que no ocurre un crecimiento infinito por lo que la solución es real para cualquier par de valores de x , como x_1 y x_2 .

3-Los neural ODE'S como modelo matemático

Si bien los Neural ODE'S pueden abordarse desde un punto de vista de código, vamos a plantearlo tal como para conseguir una definición matemática. En este caso, vamos a partir de una ecuación diferencial ordinal y utilizar técnicas de machine learning para parametrizar el campo vectorial. Consideremos un problema de valor inicial cualquiera

$$\frac{dx}{dt}(t) = f_{\theta}(t, x(t))$$

$$x(0) = x_0$$

Estas son las condiciones iniciales que hemos discutido antes.

En una ecuación diferencial ordinaria neuronal, el campo vectorial f_{θ} sería parametrizado por una red neuronal, de cualquier tipo de arquitectura, comúnmente LSTM.

En este mismo contexto, la solución de la ecuación va a venir dada por una función $\varphi(t)$, como hemos visto anteriormente. Esta función va a estar dentro del conjunto $D \subseteq \mathbb{R}^2 \times \mathbb{R}^d \times \mathbb{R}^p$. Esto viene de la dimensionalidad de $\frac{dx}{dt}(t)$, pues \mathbb{R}^2 viene dado por t y t_0 ,

\mathbb{R}^d viene dado por las dimensiones de x y \mathbb{R}^p por las dimensiones de θ . En este caso, uno puede proyectar la solución donde se toma un vector de entrada x y se devuelve otro de salida en las mismas dimensiones $\varphi: \mathbb{R}^d \rightarrow \mathbb{R}^d$. Entonces la solución con condiciones iniciales x_0 con respecto al tiempo viene dada por $x \mapsto \varphi(t; t_0, x, \theta)$, donde θ son los parámetros del modelo.

El inconveniente de esta idea es que estamos asumiendo implícitamente que la transformación interna mantiene la misma dimensionalidad, es decir, el input y output tienen el mismo número de dimensiones, lo que muchas veces, como en nuestro caso, no es cierto, ya que la entrada tiene varios parámetros, que se representan en un espacio de altas dimensiones y la salida va a ser un escalar.

4-Ejemplo teórico de los neural ODE

Pongamos el siguiente ejemplo. Tenemos unos datos representables con un tensor $\mathbb{R}^{128 \times 128 \times 3}$.

Donde 128×128 es el tamaño de la imagen y el 3 representa la intensidad de cada color, rojo, verde o azul. Podemos imaginarlo como una esfera en tres dimensiones, donde un vector representa los colores dependiendo de su longitud en cada dimensión. Y queremos clasificar la imagen en 7 distintas categorías, por lo que el output tiene que ser \mathbb{R}^7 . En este caso nuestro flujo sería tal que:

$$\text{Entrada: } \mathbb{R}^{128 \times 128 \times 3}$$

Esto es el tensor inicial que entra al modelo.

$$\mathbb{R}^{128 \times 128 \times 3} \rightarrow \varphi: \mathbb{R}^{128 \times 128 \times 3}$$

Aquí la ecuación neuronal diferencial φ , que habríamos entrenado para resolver este tipo de problemas, toma la imagen inicial y la reduce a una imagen de la misma dimensionalidad, pero donde solo las características relevantes para su clasificación están expuestas.

$$\mathbb{R}^{128 \times 128 \times 3} \rightarrow l_{\theta}: \mathbb{R}^7$$

Aquí tomamos el resultado de φ y lo pasamos por una capa transformadora l_{θ} . Esta capa sirve para convertir nuestra imagen simplificada a un vector en siete dimensiones, donde su longitud en cada dimensión representa la probabilidad de que la imagen caiga dentro de la categoría representada por esa dimensión.

$$\mathbb{R}^7 \rightarrow \text{softmax}: \text{Categorías}$$

Por último, se toma del espacio en siete dimensiones la dimensión donde el vector tenga mayor longitud y se clasifica la imagen como perteneciente a esa categoría.

Aquí, la imagen pasa primero por una Neural ODE (representada por φ) que la transforma en una representación del mismo tamaño, pero con características más fáciles de interpretar por el modelo. Esta aproximación se logra considerando φ como una función que evoluciona con el tiempo, tal como se discutió en el problema del valor inicial.

Posteriormente, la salida de $\varphi(x)$ se introduce en la capa lineal l_{θ} , que reduce dicha representación a un vector de \mathbb{R}^7 . Finalmente, la función *softmax* convierte este vector en una distribución de probabilidad sobre las 7 categorías, indicando cuál es la más probable según el modelo.

Hay una técnica muy útil para estos modelos conocida como aumentación. Esta técnica inserta un mapa afín entre el input y los valores iniciales para aumentar la dimensionalidad

de los datos, permitiendo así al modelo trabajar en más dimensiones y por consecuencia predecir mejor.

Esta técnica, matemáticamente hablando podemos definirla de la siguiente manera. Pensemos en un input con forma $x \in \mathbb{R}^d$, entonces, los valores iniciales del modelo van a tener la forma $g_0(x)$ en vez de x , donde $g_0(x)$ es una función con también una cantidad de dimensiones, pero aumentada con respecto a x porque las funciones implícitamente tienen más dimensiones que un valor numérico. El objetivo de esto entonces es aumentar las dimensiones que llegan al modelo. Esto ocurre pues el flujo de una ODE no puede cambiar la forma de sus inputs, por lo que el espacio en el que trabaja va a ser de las mismas dimensiones que sus valores iniciales.

5-Ejemplo práctico de los neural ODE aplicando a los datos de la REE

Ahora que hemos visto cómo podríamos realizar este modelo desde un punto de vista matemático, y las técnicas que vamos a aplicar, es momento de realizarlo. Para comenzar, voy a tomar un ejemplo de mis datos obtenidos de la Red Eléctrica Española (Red eléctrica Española, 2024) y explicar cómo funciona el modelo en detalle, sin embargo, esto tiene dos limitaciones, la primera es que una parte fundamental del modelo es el campo parametrizado por el LSTM, en este se ajustan unos pesos hasta encontrar los pesos óptimos para el sistema dado. Para encontrar los pesos óptimos hace falta iterar con una cantidad altísima de datos, y, ante la imposibilidad de hacer esto manualmente, voy a ejemplificar con unos pesos arbitrariamente elegidos, aunque no sean representativos de los pesos finales que el modelo encuentre, ya que no puedo calcularlos manualmente.

Una aclaración necesaria es que para este ejemplo vamos a ignorar el teorema de Picard que hemos mencionado antes, esto no es arbitrario. Con una cantidad de datos tan pequeños y unos pesos aleatorios es extremadamente probable que la solución que encontremos, si bien sea un número real localmente, tienda hacia el infinito positivo o negativo en un intervalo de tiempo mayor.

Lo primero que necesitamos entonces son datos limpios, si vemos el anexo 1, esta pieza de código lo que está haciendo es combinar todos los datos brutos extraídos de la red eléctrica y procesándolos a unos datos de más calidad y con la forma que nuestro modelo necesita. Nosotros nos vamos a basar en los datos para el 17 de septiembre de 2024. Esto de forma arbitraria, podría haberse escogido cualquier otro día. El código va a eliminar las características que no son interesantes para nuestro modelo y va a convertir ciertas características temporales a senos y cosenos. Esto se hace ya que, por regla general, el consumo de electricidad suele aumentar y disminuir tanto a las mismas horas del día como en tendencias más generales como meses. La naturaleza cíclica de las funciones trigonométricas es perfecta para esta aplicación. Como el consumo es parecido año tras año, este no se trata como una función cíclica, sino como un valor escalar.

Tabla 1: Seguimiento de la demanda de energía eléctrica, 17 de septiembre de 2024.

Real (MW)	Prevista (MW)	Hora_seno	Hora_coseno	Mes_seno	Mes_coseno	Año
26679	26783	0	1	-1	0	2024
25066	25312	0,2588	0,9659	-1	0	2024
24056	24250	0,5	0,866	-1	0	2024
23688	23632	0,7071	0,7071	-1	0	2024
23430	23383	0,866	0,5	-1	0	2024
23443	23280	0,9659	0,2588	-1	0	2024
24449	24570	1	0	-1	0	2024
27830	27812	0,9659	-0,2588	-1	0	2024
30217	30329	0,866	-0,5	-1	0	2024
30705	30985	0,7071	-0,7071	-1	0	2024
31133	30716	0,5	-0,866	-1	0	2024
30437	30432	0,2588	-0,9659	-1	0	2024
30229	30404	0	-1	-1	0	2024
30390	30533	-0,2588	-0,9659	-1	0	2024
29757	30337	-0,5	-0,866	-1	0	2024
29367	29676	-0,7071	-0,7071	-1	0	2024
29373	29184	-0,866	-0,5	-1	0	2024
29682	29683	-0,9659	-0,2588	-1	0	2024
30864	30637	-1	0	-1	0	2024
31085	31276	-0,9659	0,2588	-1	0	2024
32406	32855	-0,866	0,5	-1	0	2024
34338	34290	-0,7071	0,7071	-1	0	2024
31733	32077	-0,5	0,866	-1	0	2024
28636	28660	-0,2588	0,9659	-1	0	2024

Para este ejemplo, vamos a centrarnos en uno de los datos de la muestra, el primero.

26679	26783	0	1	-1	0	2024
-------	-------	---	---	----	---	------

Aquí el valor real de consumo: 26679MW

Y el valor que se predijo: 26783MW

Ahora entonces necesitamos calcular la media y desviación estándar de estos valores sobre el resto de los datos de nuestra muestra. Si realizamos estos cálculos podemos ver que la

media es 27268 MW y la desviación estándar es de 3163 MW. Estos valores son necesarios para la normalización que tenemos que realizar sobre los datos. Los datos se normalizan porque para el modelo es mucho más fácil procesar datos entre -1 y 1 que datos con valores elevados, realmente en un escenario hipotético el resultado debería de ser el mismo, pero esto reduce gratamente el tiempo de computación necesario para entrenar este modelo. Además, trae la ventaja de aumentar la facilidad al mirar a los cálculos que se van a realizar. La normalización se realiza con la siguiente formula

$$X_{std} = \frac{X - \bar{X}}{\sigma_s(X)}$$

Siendo X_{std} el valor de X normalizado, X el valor original, \bar{X} es el valor medio del consumo y $\sigma_s(X)$ la desviación estándar de X .

Para el valor real

$$X_{std} = \frac{26679 - 27268}{3163} \approx -0,186$$

Para el valor previsto

$$X_{std} = \frac{26783 - 27268}{3163} \approx -0,153$$

Ahora tenemos un input, siendo este el valor real de la hora anterior, y un output, el valor previsto.

El siguiente paso es entrenar el modelo, como ya he mencionado, vamos a realizar una representación simplificada pues los pesos reales y la forma final de la ecuación es el resultado de un proceso de afinación muy largo y complejo.

Un modelo simple de una ODE sobre la que podemos resolver sería

$$\frac{dx}{dt} = W \cdot x + b$$

Donde W y b serían los pesos que definen el sistema y que se ajustarían tras cada dato.

Como es una ecuación diferencial simple, vamos a resolverla con un factor de integración, aunque en un caso real esto no fuera el método real.

Asumiendo que W y $b \neq 0$, el procedimiento sería el siguiente.

Tenemos que convertir la ecuación a algo con el siguiente aspecto.

$$\frac{dx}{dt} + P(t) \cdot x = Q(t)$$

En nuestro caso, la ecuación se quedaría tal que:

$$\frac{dx}{dt} - W \cdot x(t) = b$$

Donde $P(t)$ es igual a $-W$ y $Q(t)$ es igual a b .

Para resolverlo, vamos a utilizar el factor de integración.

$$\mu(t) = e^{\int P(t) dt} = e^{\int -W dt} = e^{-Wt}$$

Ahora multiplicamos ambos lados de la ecuación por el factor de integración

$$e^{-Wt} \frac{dx}{dt} - W e^{-Wt} \cdot x(t) = b e^{-Wt}$$

Entonces, el lado izquierdo de la ecuación es la derivada del producto $e^{-Wt} \cdot x(t)$

$$\frac{d}{dt}(e^{-Wt} \cdot x(t)) = b e^{-Wt}$$

Integramos con respecto a t

$$\int \frac{d}{dt}(e^{-Wt} \cdot x(t)) dt = \int b e^{-Wt} dt$$

El lado izquierdo es la integral de la derivada, por lo que

$$\int \frac{d}{dt}(e^{-Wt} \cdot x(t)) dt = e^{-Wt} \cdot x(t)$$

El lado derecho requiere calcular

$$b \int e^{-Wt} dt$$

$$\int e^{-Wt} dt = -\frac{1}{W} e^{-Wt} \quad (W \neq 0)$$

$$b \int e^{-Wt} dt = -\frac{b}{W} e^{-Wt} + C$$

Al final resulta en

$$e^{-Wt} \cdot x(t) = -\frac{b}{W} e^{-Wt} + C$$

Si ahora despejamos $x(t)$ resulta en

$$x(t) = -\frac{b}{W} + C e^{Wt}$$

Ahora determinamos la constante C para t_0

$$x(t_0) = x_0 \Rightarrow -\frac{b}{W} + C e^{Wt_0} = x_0 \Rightarrow C = \left(x_0 + \frac{b}{W}\right) e^{-Wt_0}$$

Como t_0 va a ser cero, la ecuación general resulta en:

$$x(t) = -\frac{b}{W} + \left(x_0 + \frac{b}{W}\right) e^{Wt}, \quad W \neq 0$$

Donde x_0 significa $x(0)$

Si $W = 0$, como partimos de la ecuación $\frac{dx}{dt} - W \cdot x(t) = b$

$$\frac{dx}{dt} = b \Rightarrow x(t) = x_0 + b(t - t_0)$$

Esta solución es válida para la ecuación diferencial del ejemplo que se plantea, pero esto nos deja con un sistema cerrado. Es perfectamente válido si los datos se ajustan a esta ecuación diferencial, pero por regla general cualquier sistema con un cierto grado de complejidad no lo hace, por tanto, en vez de utilizar datos para encontrar valores de W y b óptimos y luego basar nuestras predicciones, vamos a utilizar el método de Euler con unos valores de W y b arbitrarios y ver cómo, con cada iteración, estos valores cambian para ajustarse más cercanamente a los datos.

Vamos entonces a aplicar un método de Euler para actualizar W y b así como realizar predicciones sobre el consumo de la red eléctrica en base a los datos de muestra. Vamos a tratar con las tres primeras horas. Asumimos la misma media y desviación estándar calculada previamente.

Para la primera iteración, de la hora 0 a la hora 1:

Comenzamos por dar unos valores arbitrarios a W y b

$$W = 0,001$$

$$b = -0,1$$

Ahora hemos de escoger una tasa de aprendizaje (η) para W y b , esto esencialmente es lo mismo que el paso de tabla en un método de Euler, pero representa la magnitud de los cambios que ocurren en cada iteración a W y b . Vamos a tomar $\eta = 0,1$. En este contexto, las abreviaciones pred y std significan predicho y estandarizado respectivamente.

El dato al comenzar la hora 0, cuando es estandarizado

$$x_{\text{real,std}}(0) = \frac{26679 - 27268}{3163} \approx -0,186$$

El dato que buscamos al terminar la hora 0, es decir, hora 1, cuando es estandarizado

$$y_{\text{objetivo,std}}(1) = \frac{25066 - 27268}{3163} \approx -0,696$$

Realizando una predicción con el método de Euler, de paso $\Delta t = 1$

$$x_{\text{pred,std}}(1) = x_{\text{real,std}}(0) + (W(0)x_{\text{real,std}}(0) + b(0))$$

Si tomamos los valores iniciales arbitrarios de W y b

$$x_{\text{pred,std}}(1) = -0,1862 + (0,001 \times (-0,186) - 0,1) \approx -0,286$$

Entonces el MSE con respecto a $-0,696$ es

$$\text{MSE} = (y_{\text{objetivo,std}}(1) - x_{\text{pred,std}}(1))^2 = (-0,696 - (-0,286))^2 = (-0,410)^2 \approx 0,168$$

Aunque aparecen los valores aproximados a tres cifras significativas, las operaciones se han realizado con todos sus decimales.

Ahora derivamos la ecuación del MSE con respecto a la predicción para actualizar los pesos W y b

$$\frac{\partial \text{MSE}}{\partial x_{\text{pred}}} = 2 (y_{\text{objetivo,std}} - x_{\text{pred}}) \times (-1) = 2 \times (-0,410) \times (-1) = 0,819$$

Ahora, determinamos las derivadas respecto a W y b

$$\frac{\partial x_{\text{pred}}}{\partial W} = x_{\text{real,std}}(0) = -0,186, \quad \frac{\partial x_{\text{pred}}}{\partial b} = 1$$

Si aplicamos la regla de la cadena, obtenemos los nuevos valores de W y b

$$\frac{\partial \text{MSE}}{\partial W} = 0,819 \times (-0,186) \approx -0,153, \quad \frac{\partial \text{MSE}}{\partial b} = 0,819$$

Ahora que tenemos las derivadas, utilizamos el paso de Euler para calcular los nuevos valores.

$$W_{\text{nuevo}} = W_{\text{viejo}} - \eta \frac{\partial \text{MSE}}{\partial W}$$

Se resta el producto de la derivada pues esta representa la dirección de aumento del error, y queremos minimizarlo.

Calculamos W_{nuevo} y b_{nuevo} bajo esa fórmula y obtenemos

$$W_{nuevo} = 0,016 \text{ aproximadamente}$$

$$b_{nuevo} = -0,182 \text{ aproximadamente}$$

Si repetimos este proceso dos veces más, obtenemos la siguiente tabla,

Tabla 2: Cálculo de datos con el método de Euler para las tres primeras horas.

Parámetro	Iteración 1	Iteración 2	Iteración 3
Intervalo	0 → 1	1 → 2	2 → 3
x(real,std) (inicial)	-0,1862	-0,696	-1,016
y(objetivo,std) (objetivo)	-0,696	-1,016	-1,132
W	0,001	0,01626	0,03390
b	-0,1	-0,18192	-0,20727
x(pred,std)	-0,2863862	-0,8892648	-125,772
MSE	0,16779	0,01606	0,01580
W nuevo	0,01626	0,03390	0,00840
b nuevo	-0,18192	-0,20727	-0,18213

6-Las cuatro piezas de código

Ahora se va a brevemente explicar que hace cada una de las 4 piezas de código que se encuentran en el anexo. Para comenzar, código 1. Esta pieza de código simplemente se dedica a limpiar los datos que he descargado de la red eléctrica española, además, los reduce de un dato cada 5 minutos a un dato cada hora, y transforma los datos a la forma que hemos tratado antes, convirtiendo los datos horarios en función de senos y cosenos, así como limpiando las columnas innecesarias. Respecto a la pieza de código 2, esta se encarga de entrenar el modelo. Aquí es donde se utilizan los datos para entrenar el modelo e iterar los pesos hasta su forma final, además de comprobar la solución con el teorema de Picard. El código 3 se

encarga de utilizar el modelo para realizar predicciones y obtener gráficos y métricas que nos permitan medir su validez y el código 4 simplemente se encarga de verificar que todas las otras piezas de código hayan funcionado efectivamente. Se pueden ver estas piezas de código en el anexo.

7-Entrenamiento y resultados

Podemos entonces ejecutar el código y entrenar el modelo. Tras alrededor de 6 horas, llegamos a un modelo con las siguientes métricas.

Mean Squared Error (MSE): 103,0861562MW \approx 103MW

El modelo predijo correctamente dentro de un margen de $\pm 2\%$ en el 94,6% de los casos.

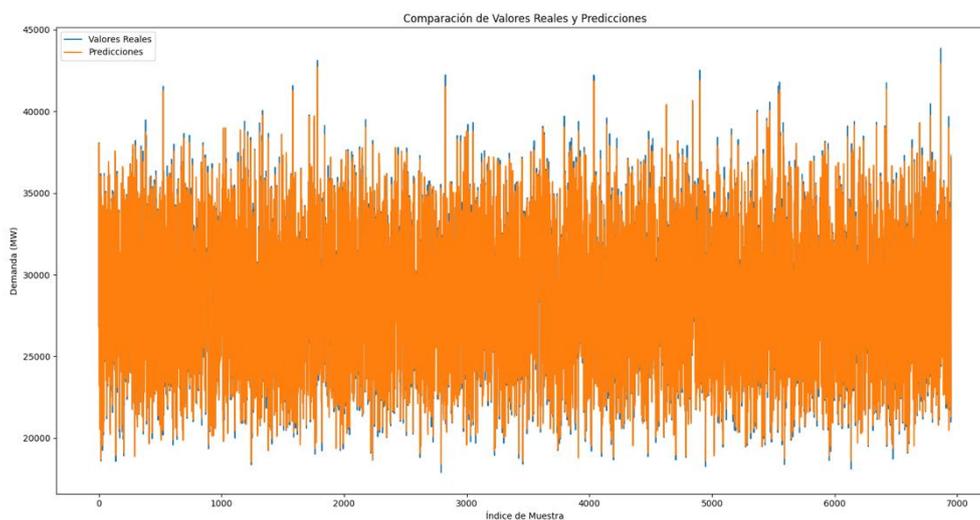
El modelo predijo correctamente dentro de un margen de $\pm 10\%$ en el 99,99994% de los casos.

Esto es el output del código, se puede verificar ejecutando los códigos de los anexos y entrenando el modelo. Es importante anotar que, por la naturaleza aleatoria de los modelos basados en redes neuronales, cabe esperar una ligera variación en los números exactos, pero si se usan los mismos datos y el mismo código deberían de ser resultados extremadamente parecidos.

Estos resultados son muy interesantes, el modelo no solo ha caído dentro del margen de 10% estándar en la industria, sino que ha tenido un error cuadrado promedio de aproximadamente 103 MW. Muy por debajo de los 3000MW de error que constituirían un 10% de 30000MW, una demanda aproximada a la media diaria. Pero es posible que el modelo esté, por ejemplo, prediciendo muy precisamente a ciertas horas del día e incorrectamente a

otras, y la media del error parezca impresionante, por esto, vamos a utilizar la pieza de código 3 para poner estos resultados en gráficas y representar un día aleatorio.

Gráfica 1: Comparación de valores reales y predichos realizado por el modelo para 9000 horas. Obtenida con Python. Autoria propia.

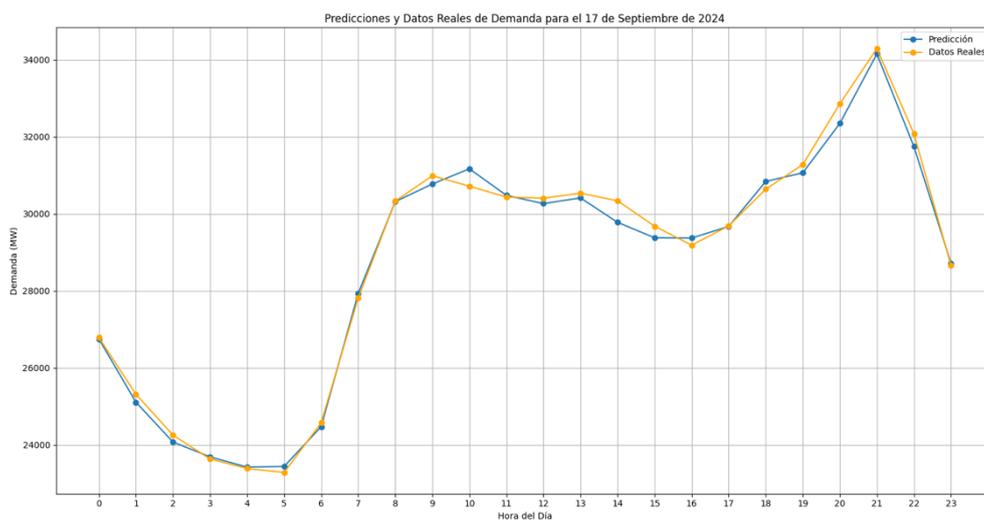


El primer gráfico nos enseña el valor real contra el predicho para las más de 7000 horas de las que tenemos datos. Como podemos observar, los valores predichos contra los valores reales son muy similares, no parece haber ninguna disparidad periódica lo que significa que nuestra idea de utilizar senos y cosenos para las características temporales ha sido preciso, podemos ver que el modelo puede llegar a tener problemas en los casos más extremos tanto con los datos muy elevados como los datos anormalmente bajos, viendo que el modelo infra predice sobre los datos excepcionalmente grandes y sobre predice sobre los datos anormalmente bajos. Esto es normal pues es imposible que el modelo espere estos datos anormales sin tener

un contexto de qué está ocurriendo ese día a esa hora para que ocurra una demanda anormalmente elevada.

Gráfica 2: Predicciones y datos reales de demanda para el 17 de septiembre de 2024.

Obtenida con Python. Autoría propia.



Respecto al segundo gráfico, podemos ver en detalle un día concreto, en este caso el 17 de septiembre de 2024. El modelo ha sido realmente preciso al predecir el consumo para este día estándar, demostrando como es muy preciso para predecir un día cotidiano.

8-Conclusiones

Tras haber validado el modelo frente a los datos reales, podemos extraer varias conclusiones.

Comenzando, podemos decir que mezclar las técnicas de modelado matemático tradicional con las redes neuronales de los LSTM para crear las neural ODE ha resultado en un modelo que ha podido predecir los datos con una tasa de éxito dentro de los márgenes estándar para la industria aproximadamente el 100% de las veces. Mezclando los beneficios de aplicar un

modelaje tradicional con una red neuronal para gestionar mejor las tendencias a corto plazo, mientras que las tendencias a largo plazo son gestionadas por la ecuación diferencial, combina lo mejor de ambas áreas para dar un resultado satisfactorio.

Las predicciones del modelo son sorprendentemente estables. Si bien es cierto que hemos aplicado el teorema de Picard para descartar soluciones que tiendan al infinito, la naturaleza de las redes neuronales que son subyacentes a estas ecuaciones hace que sean comunes saltos aleatorios en las predicciones, cosa que aquí, no ocurre.

Podríamos entonces decir, y para responder a la pregunta de investigación, que las ecuaciones diferenciales neuronales ordinarias pueden aplicarse a los sistemas de predicción ya existentes como una nueva herramienta de parcial o total control supervisado, estos sistemas son igual o más efectivos que las soluciones tradicionales a este problema, y completamente capaces de mantener el voltaje de la red eléctrica estable. No solo esto, sino que sería extremadamente simple integrar un sistema de este estilo con las herramientas ya existentes, el concepto desde un punto de vista matemático es complejo, pero desde un punto de vista computacional es código bastante básico para alguien que conozca cómo crear modelos de inteligencia artificial. Es cierto que una versión final requeriría ciertas mejoras, como ahora trataremos, pero conceptualmente es perfectamente viable implementar esta solución.

8.1-Limitaciones, mejoras y ampliaciones

Si bien esta implementación ha sido exitosa, es cierto que no ha sido sin sus compromisos y posibles mejoras. Para comenzar, solo hemos tomado un dato por cada hora cuando la resolución de los datos oficiales que ofrece la red eléctrica española es de 1 dato cada 5

minutos, lo que implicaría un nivel de detalle 12 veces mayor. Este volumen de datos hubiera sido demasiado para mis capacidades computacionales, y, por encima de esto, no es realmente necesario para un ejercicio demostrativo como este que lo que busca es probar la idea y estudiar su viabilidad. Pero para una implementación a un sistema real se tendría que utilizar datos con la mayor resolución posible.

Respecto a posibles mejoras. Una forma de haber mejorado este modelo hubiera sido utilizando ecuaciones diferenciales parciales. Las ecuaciones diferenciales parciales neuronales son un concepto aún más novedoso que las ecuaciones que hemos tratado en este informe, pero debido a su complejidad no solo computacional sino matemática, no eran precisas para este informe. Es un campo de investigación relacionado muy interesante, pero por las restricciones planteadas y el nivel de profundidad y rigor matemático que es necesario, no es posible tratar este tema.

9-Bibliografía

- Cui, W., Zhang, H., Chu, H., Hu, P., & Li, Y. (2023, junio). *On robustness of neural ODEs image classifiers*. ScienceDirect. <https://www.sciencedirect.com/science/article/abs/pii/S0020025523003444>
- Gutermuth, D. (s.f.). *Picard's Existence and Uniqueness Theorem*. <https://ptolemy.berkeley.edu/projects/embedded/eecs44/lectures/Spring2013/Picard.pdf>
- Kidger, P. (2022, 4 de febrero). *On Neural Differential Equations*. arXiv.org. <https://arxiv.org/abs/2202.02435>
- Lee, K., & J. Parish, E. (2021, 15 de septiembre). *Parameterized neural ordinary differential equations: applications to computational physics problems*. The Royal Society. <https://royalsocietypublishing.org/doi/10.1098/rspa.2021.0162>
- Melton, R. (2019, 3 de agosto). *Initial-Value Problems | Calculus II*. Lumen Learning – Simple Book Production. <https://courses.lumenlearning.com/calculus2/chapter/initial-value-problems/>
- Paoletti, M., M Haut, J., Plaza, J., & Plaza, A. (2019, 12 de noviembre). *Neural Ordinary Differential Equations for Hyperspectral Image Classification*. ResearchGate. https://www.researchgate.net/publication/337071873_Neural_Ordinary_Differential_Equations_for_Hyperspectral_Image_Classification
- Red Eléctrica Española. (2024). *Nacional - Seguimiento de la demanda de energía eléctrica*. <https://demanda.ree.es/visiona/seleccionar-fecha>
- Stewart, K. (2023, 22 de marzo). *Mean squared error (MSE) | Definition, Formula, Interpretation, & Facts | Britannica*. Encyclopedia Britannica. <https://www.britannica.com/science/mean-squared-error>
- T. Q. Chen, R., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018, 19 de junio). *Neural Ordinary Differential Equations*. arXiv.org. <https://arxiv.org/abs/1806.07366>

10-Anexo

Codigo 1

```
import os
import pandas as pd

# Función para limpiar las comas al final de las líneas
def limpiar_archivo(ruta_archivo):
    with open(ruta_archivo, 'r', encoding="ISO-8859-1") as file:
        lineas = file.readlines()

    # Eliminar la coma al final de cada línea si existe
    lineas_limpias = [linea.rstrip().rstrip(',') + "\n" for linea in lineas]

    # Guardar el archivo limpio temporalmente
    with open(ruta_archivo, 'w', encoding="ISO-8859-1") as file:
        file.writelines(lineas_limpias)

# Función para filtrar los datos del día correcto y eliminar registros fuera
# del rango de 00:00 a 23:55
def filtrar_por_dia_y_hora(df, dia):
    # Verificar que no haya encabezados adicionales
    df = df[df["Hora"] != "Hora"]

    # Filtrar las filas que coinciden con el día especificado en el nombre del
    # archivo
    df = df[df["Hora"].str.startswith(dia)]

    # Filtrar las filas que tienen horas entre 00:00 y 23:55
    filtro = df["Hora"].str.slice(11, 16) >= "00:00" # Seleccionar desde
    00:00 en adelante
    filtro &= df["Hora"].str.slice(11, 16) <= "23:55" # Seleccionar hasta
    23:55

    # Aplicar el filtro y devolver el DataFrame filtrado
    return df[filtro]

# Directorio relativo donde están los archivos CSV
directorio_csv = os.path.join(os.getcwd(), "data")
```

```

# Lista para almacenar Los dataframes
lista_datos = []

# Leer todos Los archivos en el directorio
for archivo in sorted(os.listdir(directorio_csv)):
    if archivo.endswith(".csv") and "Custom-Report" in archivo:
        # Extraer el día desde el nombre del archivo, asumiendo que el formato
        es 'Custom-Report-YYYY-MM-DD...'
        dia = archivo.split("-")[2] + "-" + archivo.split("-")[3] + "-" +
archivo.split("-")[4]

        # Limpiar el archivo antes de procesarlo
        ruta_archivo = os.path.join(directorio_csv, archivo)
        limpiar_archivo(ruta_archivo)

        # Cargar el CSV actual, saltando las dos primeras líneas y sin incluir
        el encabezado
        try:
            datos_csv = pd.read_csv(ruta_archivo, skiprows=2, header=None,
encoding="ISO-8859-1")
        except Exception as e:
            print(f"Error al leer el archivo {archivo}: {e}")
            continue

        # Asignar nombres de columnas temporalmente para poder manipular los
        datos
        datos_csv.columns = ["Hora", "Real", "Prevista", "Programada"]

        # Filtrar los datos para que solo incluyan el día correcto y las horas
        entre 00:00 y 23:55
        datos_filtrados = filtrar_por_dia_y_hora(datos_csv, dia)

        # Añadir los datos filtrados a la lista
        lista_datos.append(datos_filtrados)

# Concatenar todos Los dataframes en uno solo
if lista_datos:
    datos_combinados = pd.concat(lista_datos, ignore_index=True)

    # Añadir encabezado a los datos combinados solo una vez
    datos_combinados.columns = ["Hora", "Real", "Prevista", "Programada"]

    # Guardar el dataframe combinado en un nuevo archivo CSV con codificación
    UTF-8
    datos_combinados.to_csv("seguimiento_demanda_combinado.csv", index=False,
encoding="utf-8")

```

```

    print("Archivos combinados y guardados correctamente en formato UTF-8.")
else:
    print("No se encontraron archivos CSV válidos.")

```

Codigo 2

```

import torch
import torch.nn as nn
import pandas as pd
import numpy as np
from torchdiffeq import odeint
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib
from datetime import datetime

# Configuración del dispositivo (CPU o GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 1. Cargar Los datos
data = pd.read_csv('data_transformed_filtered.csv')

# 2. Preprocesamiento de datos
# Seleccionar las columnas relevantes
# Vamos a ajustar 'Año_num' para que sea relativo al año mínimo en los datos
anio_minimo = data['Año_num'].min()
data['Año_relativo'] = data['Año_num'] - anio_minimo

features = ['Real', 'Hora_sin', 'Hora_cos', 'Mes_sin', 'Mes_cos',
'Año_relativo']
target = 'Prevista'

X = data[features].values
y = data[target].values.reshape(-1, 1) # Aseguramos que y sea una matriz
columna

# Dividir en conjuntos de entrenamiento y prueba antes de la normalización
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Normalización de los datos
scaler_X = StandardScaler()
scaler_y = StandardScaler()

# Ajustar el escalador en el conjunto de entrenamiento y transformar

```

```

X_train = scaler_X.fit_transform(X_train)
y_train = scaler_y.fit_transform(y_train)

# Usar el mismo escalador para transformar el conjunto de prueba
X_test = scaler_X.transform(X_test)
y_test = scaler_y.transform(y_test)

# Convertir a tensores de PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32).to(device)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).to(device)

X_test_tensor = torch.tensor(X_test, dtype=torch.float32).to(device)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).to(device)

# 3. Definir el modelo de Neural ODE aumentada

# Definir La función de aumento
class Augmenter(nn.Module):
    def __init__(self, augment_dim):
        super(Augmenter, self).__init__()
        self.augment_dim = augment_dim

    def forward(self, x):
        # Aumentar La dimensionalidad agregando ceros
        batch_size = x.size(0)
        aug = torch.zeros(batch_size, self.augment_dim).to(x.device)
        x_aug = torch.cat([x, aug], dim=1)
        return x_aug

# Definir La función ODE
class ODEFunc(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super(ODEFunc, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.Tanh(),
            nn.Linear(hidden_dim, input_dim),
        )

    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)

    def forward(self, t, x):
        return self.net(x)

```

```

# Definir el modelo completo
class NeuralODE(nn.Module):
    def __init__(self, input_dim, augment_dim, hidden_dim, output_dim):
        super(NeuralODE, self).__init__()
        self.augmenter = Augmenter(augment_dim)
        self.odefunc = ODEFunc(input_dim + augment_dim, hidden_dim)
        self.linear = nn.Linear(input_dim + augment_dim, output_dim)

    def forward(self, x):
        x_aug = self.augmenter(x)
        # Resolver la ODE desde t=0 hasta t=1
        t = torch.tensor([0., 1.]).to(x.device)
        out = odeint(self.odefunc, x_aug, t, method='rk4')
        # Tomar el último paso de tiempo
        out = out[1]
        out = self.linear(out)
        return out.squeeze()

# Parámetros del modelo
input_dim = X_train.shape[1]
augment_dim = 2 # Puedes ajustar este valor
hidden_dim = 50
output_dim = 1 # Ya que estamos prediciendo una variable continua

model = NeuralODE(input_dim, augment_dim, hidden_dim, output_dim).to(device)

# 4. Definir la función de pérdida y el optimizador
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), Lr=0.001)

# 5. Entrenar el modelo
num_epochs = 1000 # Puedes ajustar el número de épocas

for epoch in range(num_epochs):
    model.train()
    optimizer.zero_grad()
    outputs = model(X_train_tensor)
    loss = criterion(outputs, y_train_tensor.squeeze())
    loss.backward()
    optimizer.step()

    if (epoch+1) % 50 == 0:
        # Evaluar en el conjunto de prueba
        model.eval()
        with torch.no_grad():
            test_outputs = model(X_test_tensor)
            test_loss = criterion(test_outputs, y_test_tensor.squeeze())

```

```

        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}, Test
Loss: {test_loss.item():.4f}')

# 6. Predicción final
model.eval()
with torch.no_grad():
    predictions_scaled = model(X_test_tensor)

# Invertir la normalización para obtener los valores originales
predictions = predictions_scaled.cpu().numpy()
predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()

y_test_actual = y_test.flatten()
y_test_actual =
scaler_y.inverse_transform(y_test_tensor.cpu().numpy()).flatten()

# 7. Visualización de resultados
# Gráfico 1: Comparación de valores reales y predicciones en el conjunto de
prueba
plt.figure(figsize=(10,6))
plt.plot(y_test_actual, Label='Valores Reales')
plt.plot(predictions, Label='Predicciones')
plt.legend()
plt.title('Comparación de Valores Reales y Predicciones en el Conjunto de
Prueba')
plt.xlabel('Índice de Muestra')
plt.ylabel('Demanda (MW)')
plt.show()

# 8. Evaluación del modelo

# Calcular el error absoluto y porcentual
error_absoluto = np.abs(predictions - y_test_actual)
error_porcentual = (error_absoluto / y_test_actual) * 100

# Manejar posibles divisiones por cero en y_test_actual
error_porcentual = np.where(y_test_actual != 0, error_porcentual, 0)

# Calcular el MAPE y La precisión
mape = np.mean(error_porcentual)
print(f"\nError Porcentual Medio Absoluto (MAPE): {mape:.2f}%")

precision = 100 - mape
print(f"Precisión del modelo (1 - MAPE): {precision:.2f}%")

# Métricas adicionales
mse = mean_squared_error(y_test_actual, predictions)

```

```

mae = mean_absolute_error(y_test_actual, predictions)
r2 = r2_score(y_test_actual, predictions)

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Coeficiente de Determinación (R²): {r2:.4f}")

# **Guardando Los escaladores**
# Guardar Los escaladores en archivos
joblib.dump(scaler_X, 'scaler_X.save')
joblib.dump(scaler_y, 'scaler_y.save')

```

Codigo 3

```

import pandas as pd
import numpy as np
from datetime import datetime

# Datos proporcionados para el 17 de septiembre de 2024
data_17sep2024 = """
Real,Prevista,Hora_sin,Hora_cos,Mes_sin,Mes_cos,Año_num
26679,26783,0.0,1.0,-1.0,-1.8369701987210297e-16,2024
25066,25312,0.25881904510252074,0.9659258262890683,-1.0,-1.8369701987210297e-
16,2024
24056,24250,0.49999999999999994,0.8660254037844387,-1.0,-1.8369701987210297e-
16,2024
23688,23632,0.7071067811865476,0.7071067811865476,-1.0,-1.8369701987210297e-
16,2024
23430,23383,0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-
16,2024
23443,23280,0.9659258262890683,0.25881904510252074,-1.0,-1.8369701987210297e-
16,2024
24449,24570,1.0,6.123233995736766e-17,-1.0,-1.8369701987210297e-16,2024
27830,27812,0.9659258262890683,-0.25881904510252063,-1.0,-1.8369701987210297e-
16,2024
30217,30329,0.8660254037844387,-0.49999999999999998,-1.0,-1.8369701987210297e-
16,2024
30705,30985,0.7071067811865476,-0.7071067811865475,-1.0,-1.8369701987210297e-
16,2024
31133,30716,0.49999999999999994,-0.8660254037844387,-1.0,-1.8369701987210297e-
16,2024
30437,30432,0.258819045102521,-0.9659258262890682,-1.0,-1.8369701987210297e-
16,2024
30229,30404,1.2246467991473532e-16,-1.0,-1.0,-1.8369701987210297e-16,2024
30390,30533,-0.2588190451025208,-0.9659258262890683,-1.0,-1.8369701987210297e-
16,2024

```

```

29757,30337,-0.4999999999999997,-0.8660254037844388,-1.0,-1.8369701987210297e-
16,2024
29367,29676,-0.7071067811865471,-0.7071067811865479,-1.0,-1.8369701987210297e-
16,2024
29373,29184,-0.8660254037844385,-0.5000000000000004,-1.0,-1.8369701987210297e-
16,2024
29682,29683,-0.9659258262890683,-0.25881904510252063,-1.0,-
1.8369701987210297e-16,2024
30864,30637,-1.0,-1.8369701987210297e-16,-1.0,-1.8369701987210297e-16,2024
31085,31276,-0.9659258262890684,0.2588190451025203,-1.0,-1.8369701987210297e-
16,2024
32406,32855,-0.8660254037844386,0.5000000000000001,-1.0,-1.8369701987210297e-
16,2024
34338,34290,-0.7071067811865477,0.7071067811865474,-1.0,-1.8369701987210297e-
16,2024
31733,32077,-0.5000000000000004,0.8660254037844384,-1.0,-1.8369701987210297e-
16,2024
28636,28660,-0.25881904510252157,0.9659258262890681,-1.0,-1.8369701987210297e-
16,2024
""""

# Leer los datos en un DataFrame
from io import StringIO

df_17sep2024 = pd.read_csv(StringIO(data_17sep2024))

# Añadir la columna 'Año_relativo'
df_17sep2024['Año_relativo'] = df_17sep2024['Año_num'] - anio_minimo

# Crear la columna 'Fecha'
df_17sep2024['Hora'] = range(24) # Asignar las horas de 0 a 23
df_17sep2024['Fecha'] = pd.to_datetime({
    'year': df_17sep2024['Año_num'],
    'month': 9, # Septiembre
    'day': 17,
    'hour': df_17sep2024['Hora']
})

# Seleccionar las características necesarias
features_new = ['Real', 'Hora_sin', 'Hora_cos', 'Mes_sin', 'Mes_cos',
'Año_relativo']
X_new = df_17sep2024[features_new].values

# Normalizar las características usando el escalador entrenado
X_new_scaled = scaler_X.transform(X_new)

# Convertir a tensor

```

```

X_new_tensor = torch.tensor(X_new_scaled, dtype=torch.float32).to(device)

# Realizar Las predicciones
model.eval()
with torch.no_grad():
    predictions_scaled = model(X_new_tensor)

# Invertir La normalización
predictions = predictions_scaled.cpu().numpy()
predictions = scaler_y.inverse_transform(predictions.reshape(-1, 1)).flatten()

# Datos reales
real_values = df_17sep2024['Prevista'].values # Asumiendo que 'Prevista' son
Los valores reales

# Horas para el gráfico
horas = df_17sep2024['Hora'].values

# Gráfico: Predicciones y datos reales para el 17 de septiembre de 2024
plt.figure(figsize=(10,6))
plt.plot(horas, predictions, marker='o', Label='Predicción')
plt.plot(horas, real_values, marker='o', Label='Datos Reales', color='orange')
plt.title('Predicciones y Datos Reales de Demanda para el 17 de Septiembre de
2024')
plt.xlabel('Hora del Día')
plt.ylabel('Demanda (MW)')
plt.xticks(horas)
plt.grid(True)
plt.legend()
plt.show()

# Imprimir Las predicciones y Los valores reales para cada hora
for hour, pred, real in zip(horas, predictions, real_values):
    print(f"Hora {hour:02d}:00 - Predicción: {pred:.2f} MW - Real: {real:.2f}
MW")

```

Codigo 4

```

import os
import pandas as pd

# Función para Limpiar Las comas al final de Las líneas
def limpiar_archivo(ruta_archivo):
    with open(ruta_archivo, 'r', encoding="ISO-8859-1") as file:
        lineas = file.readlines()

```

```

# Eliminar la coma al final de cada línea si existe
lineas_limpias = [linea.rstrip().rstrip(',') + "\n" for linea in lineas]

# Guardar el archivo limpio temporalmente
with open(ruta_archivo, 'w', encoding="ISO-8859-1") as file:
    file.writelines(lineas_limpias)

# Función para filtrar los datos del día correcto y eliminar registros fuera
del rango de 00:00 a 23:55
def filtrar_por_dia_y_hora(df, dia):
    # Verificar que no haya encabezados adicionales
    df = df[df["Hora"] != "Hora"]

    # Filtrar las filas que coinciden con el día especificado en el nombre del
archivo
    df = df[df["Hora"].str.startswith(dia)]

    # Filtrar las filas que tienen horas entre 00:00 y 23:55
    filtro = df["Hora"].str.slice(11, 16) >= "00:00" # Seleccionar desde
00:00 en adelante
    filtro &= df["Hora"].str.slice(11, 16) <= "23:55" # Seleccionar hasta
23:55

    # Aplicar el filtro y devolver el DataFrame filtrado
    return df[filtro]

# Directorio relativo donde están los archivos CSV
directorio_csv = os.path.join(os.getcwd(), "data")

# Lista para almacenar los dataframes
lista_datos = []

# Leer todos los archivos en el directorio
for archivo in sorted(os.listdir(directorio_csv)):
    if archivo.endswith(".csv") and "Custom-Report" in archivo:
        # Extraer el día desde el nombre del archivo, asumiendo que el formato
es 'Custom-Report-YYYY-MM-DD...'
        dia = archivo.split("-")[2] + "-" + archivo.split("-")[3] + "-" +
archivo.split("-")[4]

        # Limpiar el archivo antes de procesarlo
        ruta_archivo = os.path.join(directorio_csv, archivo)
        limpiar_archivo(ruta_archivo)

        # Cargar el CSV actual, saltando las dos primeras líneas y sin incluir
el encabezado
        try:

```

```

        datos_csv = pd.read_csv(ruta_archivo, skiprows=2, header=None,
encoding="ISO-8859-1")
    except Exception as e:
        print(f"Error al leer el archivo {archivo}: {e}")
        continue

    # Asignar nombres de columnas temporalmente para poder manipular los
datos
    datos_csv.columns = ["Hora", "Real", "Prevista", "Programada"]

    # Filtrar los datos para que solo incluyan el día correcto y las horas
entre 00:00 y 23:55
    datos_filtrados = filtrar_por_dia_y_hora(datos_csv, dia)

    # Añadir los datos filtrados a la lista
    lista_datos.append(datos_filtrados)

# Concatenar todos los dataframes en uno solo
if lista_datos:
    datos_combinados = pd.concat(lista_datos, ignore_index=True)

    # Añadir encabezado a los datos combinados solo una vez
    datos_combinados.columns = ["Hora", "Real", "Prevista", "Programada"]

    # Guardar el dataframe combinado en un nuevo archivo CSV con codificación
UTF-8
    datos_combinados.to_csv("seguimiento_demanda_combinado.csv", index=False,
encoding="utf-8")

    print("Archivos combinados y guardados correctamente en formato UTF-8.")
else:
    print("No se encontraron archivos CSV válidos.")

```