UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO DE FIN DE MÁSTER

Predicción de consumo de energía en viviendas mediante IoT e Inteligencia Artificial

Prediction of energy consumption in homes through IoT and Artificial Intelligence

---

Tutor 1: Luis Felipe Romero Gómez
Dpto. Arquitectura de Computadores

Tutor 2: Luis Felipe Romero Caparrós
Ingeniero Industrial, externo

Alumno: Javier Romero Caparrós

Curso: Máster en Ingeniería Industrial
Curso 2023 /2024

MÁLAGA, 2 de noviembre de 2023

Resumen

El presente proyecto desarrolla un sistema basado en inteligencia artificial para predecir el consumo de energía en viviendas con instalación trifásica. Utilizando los datos recopilados como series temporales (almacenadas en una base de datos noSQL), y utilizando diferentes técnicas de aprendizaje profundo en GPUs, se ha logrado una predicción de excepcional calidad sobre el comportamiento futuro del consumo eléctrico. Para ello, se han considerado diferentes factores colaterales que afectan al consumo, como son la hora del día, el día de la semana, si es un día festivo o no, la meteorología, e incluso la presencia o ausencia de personas.

Para alcanzar estos objetivos, se ha diseñado una completa arquitectura de recopilación de datos, en la que se combinan diferentes tecnologías IoT de alto rendimiento y reducido ancho de banda, y entre las que se incluyen un protocolo de comunicaciones basado en mensajería por suscripción, sistemas de información mediante servicios RESTful, y el almacenamiento e intercambio de información en formato JSON.

Adicionalmente, se ha desarrollado una aplicación Android que permite la calibración del consumo eléctrico de los diferentes aparatos de la vivienda con el simple objeto de etiquetar los causantes de las variaciones del consumo, lo cual permitiría, por ejemplo, mejorar la predicción o la implementación de un sistema de retroalimentación que identifique los principales responsables de la factura eléctrica.

El sistema se ha planteado para que sea, en un futuro, extrapolable a comunidades de vecinos o agrupaciones de mayor entidad. Para ello, las tecnologías empleadas, como la base de datos MongoDB o el protocolo de comunicación MQTT, han sido elegidas por su escalabilidad y su diseño especialmente orientado al big data. Finalmente, y sin que fuera un objetivo expreso del presente proyecto, se han realizado algunos experimentos de predicción, utilizando los datos recopilados durante los apenas dos meses que ha permitido el proyecto. Los resultados alcanzados son de una extraordinaria precisión, especialmente si se tiene en cuenta la naturaleza caótica del propio sistema modelado y el relativamente reducido tamaño de los conjuntos de datos.

Palabras clave: Aprendizaje Profundo, Internet de las Cosas, Consumo eléctrico, Inteligencia Artificial, Predicción de series temporales, Instalaciones eléctricas.

Abstract

This project develops an artificial intelligence-based system to predict energy consumption in homes with three-phase power installation. Using the data collected as a time series (stored in a NoSQL database) and different deep learning techniques on GPUs, an exceptional quality prediction of the future electricity consumption behavior has been achieved. To do this, various collateral factors affecting consumption have been considered, such as the time of day, the day of the week, whether it is a holiday, the weather, and even the presence or absence of people.

To achieve these objectives, a complete data collection architecture has been designed, combining different high-performance IoT technologies with reduced bandwidth, including a communications protocol based on subscription messaging, information systems through RESTful services, and storing and exchanging information in JSON format.

Additionally, an Android application has been developed that allows the calibration of the electrical consumption of the different appliances in the home with the simple purpose of labeling the causes of consumption variations, which would allow, for example, to improve the prediction or the implementation of a feedback system that identifies the main responsible for the electricity bill.

The system is designed to be extrapolated to communities of neighbors or larger groupings in the future. For this purpose, the technologies used, such as the MongoDB database or the MQTT communication protocol, have been chosen for their scalability and design, mainly oriented to big data.

Finally, and without being an express objective of this project, some prediction experiments have been carried out using the data collected during the barely two months the project has allowed. The results achieved are of extraordinary precision, especially considering the chaotic nature of the system itself and the relatively small size of the data sets.

Keywords: Deep Learning, Internet of Things, Electricity Consumption, Artificial Intelligence, Time-series forecasting, Electrical installations.

Agradecimientos

"A mi padre y tutor de este proyecto, por la gran dedicación y pasión que demuestra en todo lo que hace y por enseñarme que la perseverancia y la entereza son claves para superar las dificultades del camino."

"A mi madre, por su amor y cariño incondicional, por soportar día a día las apasionantes conversaciones entre ingenieros y, sobre todo, por ser el mejor ejemplo de vida que un hijo pueda tener."

"A mi hermano y cotutor, por su gran ayuda en mis primeros pasos en el Deep Learning y por ser la persona más feliz que conozco, cuya alegría reside en hacer felices a los que le rodean."

"A mi querida Paula, mi compañera y amiga, por estar siempre a mi lado pase lo que pase y por enseñarme que si las cosas no salen como esperamos, es porque algo mejor está en camino."

"A mi Abuelo Lupi y mi Abuela Mari, por ser mis mayores orgullos y mis referentes en esta vida. Gracias a ellos soy quien soy ahora, y no hay nada que me haga más feliz en el mundo que parecerme a ellos."

"A mi Abuela Yolanda y mi Abuelo Federico, por siempre demostrar su absoluta confianza en mí y por enseñarme, con su ejemplo, la importancia vital de mantener la familia unida."

"A los amigos que me llevo del Máster, Alonso, Josemi y Virginia, con quienes he vivido momentos increíbles que siempre llevaré en el corazón."

Acrónimos

API: *Application Programming Interface*, interfaz de programación de aplicaciones

ARIMA: *Autoregressive Integrated Moving Average*, modelo autorregresivo integrado de promedio móvil

CNN: *Convolutional Neural Network*, red neuronal convolucional

DNN: *Dense Neural Network*, Redes neuronales densamente conectadas

GRU: *Gated Recurrent Unit*, unidad recurrente cerrada

IA: Inteligencia Artificial

IoT: *Internet of Things*, Internet de las Cosas

JSON: *JavaScript Object Notation*, anotación de objetos JavaScript

LSTM: *Long short-term memory*, memoria a largo plazo o a corto plazo

MAE: *Mean Absolute Error*, error absoluto medio

MQTT: *Message Queuing Telemetry Transport*, transporte de telemetría de cola de mensajes

QoS: *Quality of Service*, calidad del servicio

ReLU: *Rectified Linear Unit*, unidad lineal rectificada

REST: *Representational State Transfer*, transferencia de estado representacional

RNN: *Recurrent Neural Network*, red neuronal recurrente

TCN: *Temporal Convolutional Network*, red temporal convolucional

TLS: *Transport Layer Security*, seguridad en la capa de transporte

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Los eventos recientes a nivel global, como la recuperación económica tras la pandemia de Covid-19 y conflictos como las guerras en Ucrania y Gaza, han provocado un cambio importante en la situación energética mundial. El aumento significativo en los precios de la energía ha causado un aumento preocupante en la inflación y ha llevado a que muchas familias caigan en la pobreza. Estos desafíos económicos y sociales se han visto agravados por la creciente conciencia pública sobre la necesidad de reducir la huella de carbono para abordar el acelerado cambio climático.

En este contexto, surge una imperiosa necesidad colectiva de buscar soluciones mediante el aprovechamiento de las nuevas tecnologías emergentes que permitan un uso más eficiente de la energía. Es precisamente esta urgencia y la búsqueda de soluciones tecnológicas innovadoras lo que motiva el presente Trabajo de Fin de Máster.

Este documento se enfoca en desarrollar un sistema de inteligencia artificial que utiliza *Machine Learning* para predecir el consumo de energía en viviendas con instalaciones eléctricas trifásicas. Se analizan series temporales y datos de potencia, considerando factores como la hora del día y las condiciones ambientales. También se incorpora un servicio de IoT (*Internet of Things*) para conectar con una aplicación móvil a través de MQTT (*Message Queuing Telemetry Transport*), facilitando la identificación de dispositivos eléctricos por parte de los usuarios. El objetivo es promover el consumo energético responsable y ofrecer soluciones tecnológicas innovadoras para el sector energético.

1.1. Estado del arte

El análisis de series temporales tradicionalmente utiliza técnicas estadísticas para examinar características como tendencias, periodicidad y estacionalidad en datos temporales.

Numerosos estudios previos se basan en predicciones de consumo energético empleando métodos ARIMA[1] para mejorar las predicciones de demanda energética en países tanto desarrollados como en desarrollo [1], o para la predicción del consumo energético en países asiáticos para los próximos años [2, 3].

Otros autores se han encargado de combinar métodos ARIMA junto con técnicas de inteligencia artificial, para pronosticar el consumo de energía, abordando los patrones lineales y no lineales en los datos y mejorando la precisión de la predicción en comparación con el uso de los modelos aislados [4, 5].

Aunque las técnicas tradicionales de predicción han demostrado históricamente su eficacia, en la actualidad, las técnicas de Machine Learning han surgido como una alternativa prometedora para potenciar aún más la precisión en la predicción de series temporales. Estas técnicas aprovechan la capacidad de los algoritmos de Machine Learning para identificar patrones complejos y relaciones no lineales en los datos, lo que representa un avance significativo en la mejora de las proyecciones.

Es el caso de algunos estudios centrados en realizar predicciones del consumo energético en viviendas mediante técnicas de Machine Learning, utilizando datos recogidos de una *smart grid* [6], o, estudios centrados en analizar la eficacia de modelos de aprendizaje profundo mediante la aplicación de redes neuronales convolucionales (CNN) o recurrentes (RNN) en la predicción de series temporales de energía [7, 8].

Otros estudios de elevado interés para la realización de este proyecto, debido a la similitud tanto de los datos tratados (en periodicidad, tendencias y estacionalidad) como de los datos de consumo de una vivienda trifásica y los modelos de predicción empleados, son dos artículos publicados por la Universidad de Sevilla. El primer estudio [9] se centra en predecir los precios de la electricidad en el mercado español utilizando modelos de *Deep Learning*, destacando la efectividad de estos modelos y cómo diferentes períodos de tiempo afectan los resultados. El segundo estudio [10] utiliza Redes de Convolución Temporales (TCN) para mejorar la precisión en la predicción de la demanda eléctrica nacional y la demanda de energía en estaciones de carga para vehículos eléctricos en España, demostrando que los TCN superan a las redes recurrentes LSTM (*Long short-term memory*), que son el estándar en el campo de la predicción de series temporales relacionadas con la energía.

La obtención de datos relevantes para el entrenamiento de modelos de aprendizaje profundo destinados a predecir el consumo energético en viviendas energéticas puede ser un desafío significativo. Esto se debe a la necesidad de contar con datos históricos precisos de consumo eléctrico de la vivienda, así como datos ambientales que puedan influir en dicho consumo. Aquí es donde surge el interés en la tecnología IoT (Internet de las cosas) y el protocolo MQTT. Utilizar IoT para obtener estos datos es atractivo porque permite

[1] El Método ARIMA (*Autoregressive Integrated Moving Average*) es una técnica estadística utilizada en el análisis de series temporales. Combina componentes de autorregresión, integración y media móvil para modelar y predecir datos secuenciales, siendo útil tanto para predicciones a corto como a largo plazo.

una recopilación continua y en tiempo real de información relevante, como el consumo eléctrico de la vivienda y las condiciones ambientales. Esto puede mejorar la precisión de los modelos de predicción al proporcionar datos actualizados y detallados, lo que a su vez contribuye a una gestión más eficiente de la energía en viviendas energéticas y, en última instancia, ahorra recursos y reduce costos.

Algunos estudios demuestran la utilidad de las tecnologías IoT y el análisis de series temporales para mejorar la gestión y eficiencia energética en entornos residenciales y de hogares inteligentes. Uno de ellos emplea el protocolo MQTT para obtener datos sobre temperatura, humedad y presión en tiempo real [11]. Otro estudio se centra en el uso de datos de consumo de energía y condiciones climáticas para detectar anomalías en electrodomésticos [12].

1.2. Alcance del proyecto y objetivos

El presente Trabajo de Fin de Máster tiene como objetivo establecer las bases para la creación de un modelo sólido destinado a predecir con precisión el consumo energético en viviendas, tanto trifásicas como monofásicas. Los modelos basados en técnicas de inteligencia artificial requieren una gran cantidad de datos estructurados para alcanzar su máximo potencial. Por este motivo, este trabajo plantea una nueva metodología para la recopilación de una variedad significativa de datos y su organización estructurada, preparándolos para el posterior entrenamiento de redes neuronales a través del uso de tecnologías IoT. Durante el desarrollo de este proyecto, se ha implementado esta innovadora forma de adquirir datos, si bien la cantidad de datos recopilados, hasta el momento de entrega de esta memoria, es limitada en comparación con lo requerido para desarrollar un modelo de inteligencia artificial plenamente funcional. Con el tiempo y la recopilación continua de datos, se espera que la precisión de las predicciones mejore significativamente. En este contexto, se plantean los siguientes objetivos:

- Desarrollar un sistema de predicción de consumo de energía en viviendas basado en inteligencia artificial.
- Crear una base de datos para el almacenamiento de la información generada por los dispositivos, electrodomésticos, etc., así como cualquier otra información relevante para el sistema.
- Determinar los patrones de encendido y apagado de los diferentes aparatos eléctricos presentes en la vivienda, priorizando el tratamiento especial de algunos electrodomésticos con patrón periódico o con más relevancia en el consumo.
- Integrar un servicio MQTT para intercambiar información e interactuar con una aplicación móvil que facilite el etiquetado de los aparatos eléctricos.

- Mejorar la precisión de la predicción del consumo futuro de la vivienda considerando la hora del día, el día de la semana, el mes del año, los días festivos, la potencia de los aparatos eléctricos y datos ambientales como la temperatura, humedad y estado del cielo.

Capítulo 2

Fundamento teórico

2.1. Fundamentos básicos de las series temporales

Una serie temporal es una secuencia de N datos ordenada y equidistante cronológicamente sobre una característica (serie univariante) o sobre M características (serie multivariante) [13]. La ecuación (2.1) muestra una representación matemática frecuente de las series temporales.

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,N} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{M,1} & y_{M,2} & \cdots & y_{M,N} \end{bmatrix}, \tag{2.1}$$

donde $y_{i,j}$ representa el valor de la característica i en el instante j. Un ejemplo de series temporales multivariantes[1], que serán abordadas en este proyecto, puede ser la potencia desglosada por fases consumida en una vivienda trifásica, como se aprecia en la Figura 2.1.

Otros conceptos de interés en el análisis de series temporales [14] son los siguientes:

- **Tendencias**: Representan trayectorias generales a largo plazo en una serie temporal. Por ejemplo, un aumento constante en el precio de la energía durante varios años.
- **Estacionalidad**: Oscilaciones que se producen y repiten a corto plazo. Por ejemplo, el precio de la energía es más alto en invierno debido a la mayor demanda de calefacción.
- **Patrones Cíclicos**: Oscilaciones que se producen a largo plazo debido a factores subyacentes no relacionados con patrones estacionales. Por ejemplo, ciclos económicos que afectan al precio de la energía.

[1] En el caso de series temporales univariantes, simplemente se eliminaría el subíndice i.

- **Residuos**: Representa la variabilidad aleatoria de una serie temporal. Por ejemplo, la influencia de una tormenta de verano en el precio de la luz ante la ausencia de producción de energía solar.

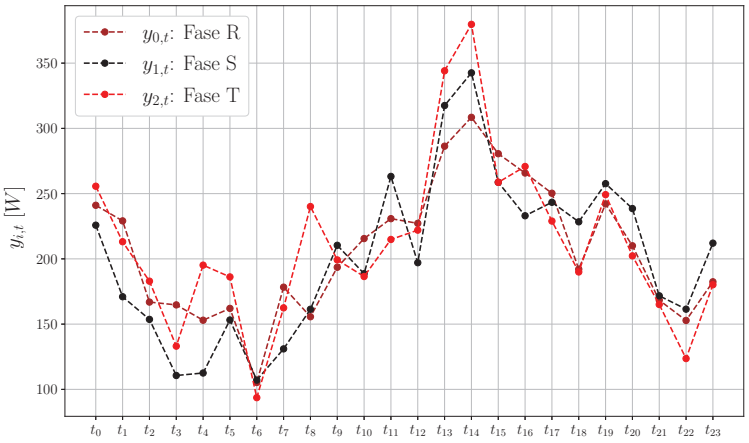


Figura 2.1: Representación de una serie temporal multivariante de la potencia consumida en una vivienda trifásica, medida en vatios.

2.2. Redes neuronales y Deep Learning

En primer lugar es importante situar el Deep Learning dentro del Marco de Inteligencia Artificial ya que se trata de un subconjunto de Machine Learning, que a su vez es solo una parte de la Inteligencia Artificial [15], aunque en estos momentos es quizás la rama más dinámica que esta produciendo que las Inteligencias Artificiales estén en pleno auge. Con el objetivo de situar estos campos, se muestra a continuación una aproximación simple [16] de cada uno de estos conceptos.

La **Inteligencia artificial** se refiere al esfuerzo de automatizar tareas intelectuales normalmente realizadas por humanos, representando la inteligencia que muestran las máquinas en contraste con la inteligencia natural de los humanos.

El **Machine Learning** es un subcampo de la inteligencia artificial, permite a las maquinas aprender sin programación explícita, encontrando patrones en los datos para construir algoritmos de predicción específicos para problemas concretos. El Machine Learning se agrupa en tres grandes categorías.

- Aprendizaje supervisado: Los datos de entrenamiento incluyen la soluciones deseadas, denominas etiquetas.
- Aprendizaje no supervisado: Los datos de entrenamiento no incluyen etiquetas, y será el algoritmo el que intentará clasificar la información por sí mismo.

- *Reinforcement Learning*: El modelo se implementa como un agente que explora un espacio desconocido y toma acciones basadas en prueba y error.

El **Deep Learning** es un caso especial de Machine Learning, donde se emplean algoritmos basados en las neuronas humanas (redes neuronales artificiales) que emplean múltiples capas de procesamiento para aprender representaciones de los datos, realizando transformaciones tanto lineales como no lineales a partir de los datos de entrada para obtener una salida cercana a la deseada (etiquetas). El aprendizaje supervisado se enfoca en ajustar los parámetros de estas transformaciones, como los pesos y sesgos, para que la salida predicha sea óptima y difiera muy poco de la salida esperada.

Una **red neuronal**, en el contexto de aprendizaje supervisado no deja de ser una "caja negra" que aprende a hacer predicciones. Introduciéndole una serie de ejemplos de entradas o características (x), junto con sus resultados o etiquetas (y), la red ajusta sus conexiones internas (pesos) para que, con el tiempo, pueda realizar predicciones del resultado ($\hat{y}$) cuando se introduzcan nuevas entradas que nunca haya visto antes.

En el contexto de redes neuronales, existen dos tipos de problemas principales: regresión y clasificación. La regresión se centra en predecir valores numéricos continuos, mientras que la clasificación se enfoca en asignar categorías a los datos. Este documento se enfoca exclusivamente en modelos de regresión debido a que se aborda un problema relacionado con series temporales para la predicción de valores numéricos.

2.2.1. Neurona artificial simple

Una neurona artificial simple [17] es el componente fundamental de las redes neuronales artificiales y actúa como una unidad de procesamiento que realiza operaciones matemáticas en los datos de entrada. Una neurona recibe una serie de datos de entrada para realizar una combinación algebraica de los mismos para producir una salida, pasando previamente dicha combinación por una función de activación que se encargará de "apagar o encender" dicha neurona.

Una neurona recibe un vector de M datos de entrada $X = (x_1, x_2, \ldots, x_M)$ y establece a cada entrada un conjunto de pesos $W = (w_1, w_2, \ldots, w_M)$ y un sesgo común b, de manera que establece la siguiente relación algebraica

$$z = \sum_{i=1}^{M} w_i \cdot x_i + b = W^T \cdot X + b, \tag{2.2}$$

una vez realizada esta combinación, la neurona producirá la salida y introduciendo z en una función de activación

$$y = f(z). \tag{2.3}$$

La función de activación introduce no linealidad en la salida de la neurona artificial, actuando como un interruptor que decide cuándo enviar una señal. Esto permite a la neurona capturar relaciones complejas en los datos de entrada. En problemas de regresión, se utilizan diferentes tipos de funciones de activación.

- **Linear:** En algunas ocasiones, no interesa modificar la señal de salida de la neurona. Su ecuación es:
$$f(z) = linear(z) = z. \tag{2.4}$$
- **Sigmoid:** Es útil cuando queremos que la salida de la neurona esté en un rango entre 0 y 1. Reduce los valores atípicos extremadamente grandes o pequeños sin eliminarlos. Su ecuación es:
$$f(z) = sigmoid(z) = \frac{1}{1 + e^{-z}}. \tag{2.5}$$
- **Tanh:** La función tangente hiperbólica (tanh) también limita la salida de la neurona, pero en un rango entre -1 y 1. Es útil cuando queremos que la salida sea simétrica alrededor de cero. Su ecuación es:
$$f(z) = tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.6}$$
- **ReLU (Rectified Linear Unit):** ReLU es simple y ampliamente utilizada. Si la entrada es positiva, la neurona se activa; de lo contrario, no. Su ecuación es:
$$f(z) = ReLu(z) = \begin{cases} z & \text{si } z \geq 0 \\ 0 & \text{si } z < 0 \end{cases}. \tag{2.7}$$

En la Figura 2.2, se representan las señales de activación, mientras que la Figura 2.3 muestra el funcionamiento de una neurona para transformar entradas en salidas. En la Figura 2.4, se muestra cómo se crea un modelo que puede predecir P características diferentes utilizando M características de entrada. Esto se logra mediante la agrupación de múltiples neuronas en una red.[2]

[2]Por ejemplo, al proporcionar información como la potencia consumida en cada fase (RST), el porcentaje de humedad y la temperatura durante una hora h (donde $M = 5$), la red es capaz de predecir la potencia de cada fase ($P = 3$) para la siguiente hora $h + 1$.

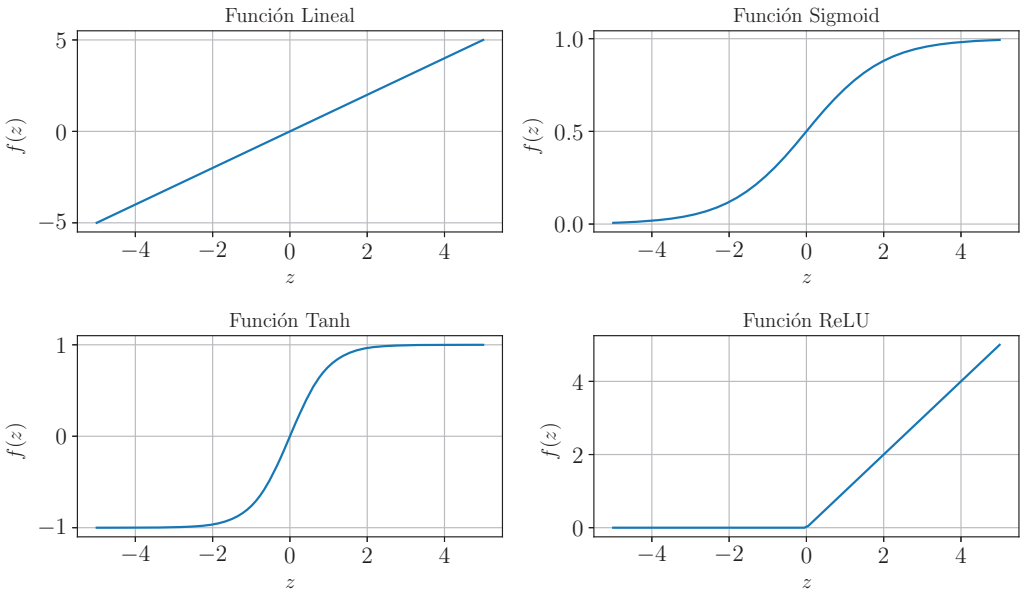


Figura 2.2: Representación de las distintas señales producidas por diferentes funciones de activación.

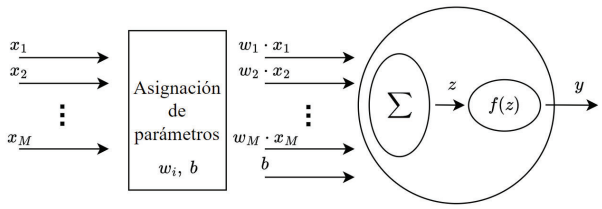


Figura 2.3: Esquema del funcionamiento de una neurona artificial simple.

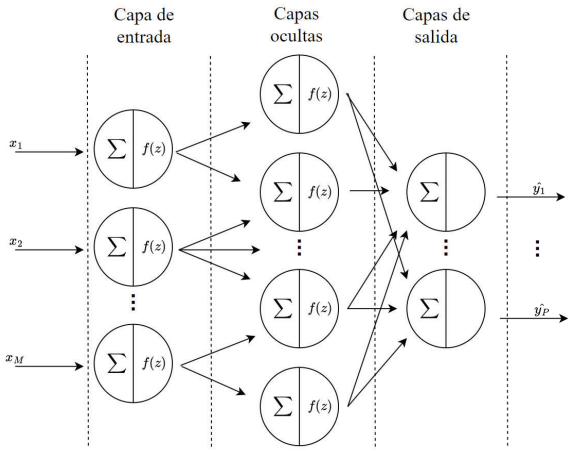


Figura 2.4: Esquema del funcionamiento de una red neuronal.

2.2.2. Entrenamiento de una red neuronal

En el proceso de configuración y evaluación de modelos en Machine Learning y Deep Learning, generalmente se dividen los datos en tres conjuntos: entrenamiento, validación y prueba. En la Figura 2.5 se observa un esquema de la división de los datos y el propósito de la misma, obtenida del libro "Python deep learning: introducción práctica con Keras y TensorFlow 2" de Jordi Torres (2020) [18].

- **Datos de Entrenamiento:** se utilizan para ajustar los parámetros del modelo. Son los datos con los que el modelo ha sido entrenado y que conoce a fondo, ya que ha aprendido de ellos.
- **Datos de Validación:** se utilizan las métricas para evaluar el desempeño del modelo ante datos de entrada que no conoce, se emplean para ajustar los hiperparámetros, propios del algoritmo de aprendizaje o la estructura, antes de repetir el proceso de entrenamiento.
- **Datos de Prueba:** se reservan exclusivamente para evaluar el rendimiento final del modelo después de haber sido entrenado y validado.

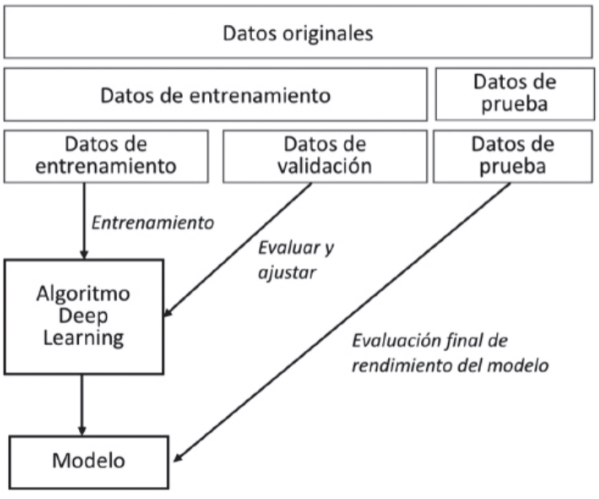


Figura 2.5: Esquema de la segmentación de los datos y el propósito de cada uno de ellos.

Recapitulando y empleando notación para problemas de series temporales, una red neuronal ante una muestra de entradas $X_t = (x_{1,t}, x_{2,t}, \ldots, x_{t,M})$ y una etiqueta de una característica en concreto $y_{j,t}$, es capaz de realizar una predicción acorde a la siguiente ecuación

$$\hat{y_{j,t}} = f(x_{1,t}, x_{2,t}, \ldots, x_M). \tag{2.8}$$

Puesto que se desea conseguir que los valores de $y_{j,t}$ e $\hat{y_{j,t}}$ sean prácticamente los mismos, es necesario emplear el siguiente algoritmo de aprendizaje (mostrado esquematizado en la Figura 2.6) que recoge Jordi Torres en su libro [18]:

1. Inicializar los valores (normalmente de manera aleatoria) de los pesos y el sesgos.
2. Introducir en la red un conjunto de datos de entrada (lotes) y pasarlos a la red para obtener su predicción.
3. Comparar las predicciones obtenidas con las etiquetas esperadas y calcular el error cometido mediante la función de pérdida.
4. Propagar hacia atrás el error para que llegue a cada uno de los parámetros que componen la red neuronal.
5. Usar esta información propagada con un algoritmo de descenso de gradiente para actualizar los parámetros de la red neuronal buscando reducir el error.
6. Continuar iterando los siguientes pasos, hasta que consideremos que tenemos un buen modelo.

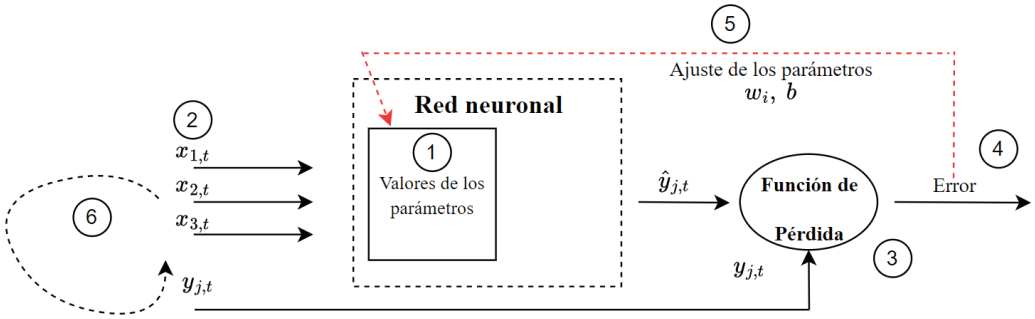


Figura 2.6: Esquema de los pasos del proceso iterativo de aprendizaje de una red neuronal de una neurona.

2.2.3. Función de pérdida

Es importante cuantificar lo cercana que es la predicción realizada por la red neuronal ($\hat{y_{j,t}}$) frente a su valor ideal ($y_{j,t}$), y para ello se emplea en el algoritmo de aprendizaje la función de pérdida (o *loss* en ingles) $L(y_{j,t}, \hat{y_{j,t}})$.

Existen numerosas funciones de pérdida que se pueden emplear, por ejemplo en problemas de clasificación binaria es comúnmente usada la función binaria *binary_ crossentropy* mientras que en problemas de regresión es común el empleo de funciones como *Mean Squared Error*.

Este documento se centra únicamente en los modelos de regresión, los cuales emplean frecuentemente las siguientes funciones de pérdida: Error cuadrático medio [19], error absoluto medio [20] y Función de pérdida de Huber [21].

- **Error Cuadrático Medio (MSE)**:
  - Ventajas: Sensible a errores grandes, fácil de derivar.
  - Inconvenientes: Sensible a valores atípicos.
  - Fórmula:
$$L(y, \hat{y}) = (y_i - \hat{y}_i)^2 \tag{2.9}$$
- **Error Absoluto Medio (MAE)**:
  - Ventajas: Robusto a valores atípicos, fácil de interpretar.
  - Inconvenientes: No penaliza errores grandes de la misma manera que el MSE.
  - Fórmula:
$$L(y, \hat{y}) = |y_i - \hat{y}_i| \tag{2.10}$$
- **Pérdida de Huber**:
  - Ventajas: Combina las ventajas del MSE y el MAE, menos sensible a valores atípicos.
  - Inconvenientes: Tiene un parámetro de ajuste que debe ser seleccionado.
  - Fórmula:
$$L(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{si } |y - \hat{y}| \leq \delta \\ \delta|y - \hat{y}| - \frac{1}{2}\delta^2 & \text{si } |y - \hat{y}| > \delta \end{cases} \tag{2.11}$$

Estas ecuaciones muestran el valor de pérdida para un solo punto, si se promedian en todo el conjunto de los datos se obtiene la función de coste:

$$J(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} L(y, \hat{y}). \tag{2.12}$$

2.2.4. Descenso del gradiente

En el Deep Learning se utiliza un proceso de optimización iterativo donde se ajusta gradualmente los parámetros del modelo con el objetivo de minimizar la función de coste sobre el conjunto de datos de entrenamiento. El descenso del gradiente usa la primera derivada (el gradiente) de la función de coste para realizar la actualización de parámetros. Esto implica aplicar la regla de la cadena [22] para calcular cómo cambia la función de coste con respecto a cada parámetro del modelo.

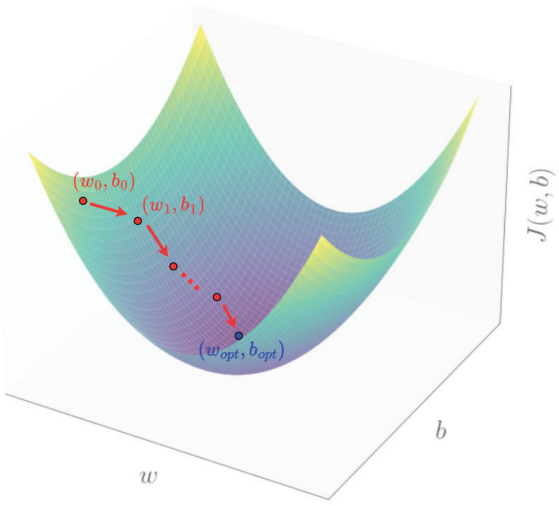


Figura 2.7: Esquema del descenso del gradiente de la función de coste de un modelo con una única neurona donde $J(y, \hat{y}(w, b)) = J(w, b)$.

El modelo dispone de una serie de parámetros iniciales, y mide el gradiente de la función de coste en ese punto con respecto a los parámetros del modelo, lo que requiere aplicar nuevamente la regla de la cadena, y va en la dirección del gradiente descendente[3]. En la Figura 2.7 se muestra un esquema del proceso del descenso del gradiente para una red que dispone de los parámetros de solo una neurona artificial.

Comúnmente, se emplea la técnica de mini lotes (*batches*), que consiste en dividir los datos de entrenamiento en grupos pequeños. En lugar de utilizar todos los datos a la vez para ajustar un modelo, se calcula y aplica el gradiente a cada uno de estos grupos más pequeños, lo que acelera el entrenamiento y mejora la generalización si los datos están bien distribuidos.

Optimizadores

Los optimizadores son variantes o optimizaciones del algoritmo de de descenso del gradiente. En `TensorFlow`[4], con la API de `Keras`, se emplean optimizadores como: SGD, TMSprop, AdaGrad o Adam, entre muchos otros. Este proyecto utiliza únicamente el algoritmo optimizador Adam [24], que actualmente es el más frecuentemente usado en Deep Learning.

[3] El descenso del gradiente es, intuitivamente, como encontrar el camino más rápido, para bajar de una montaña, estando totalmente a ciegas, y siguiendo la pendiente más pronunciada para llegar a la parte más baja dando pequeños pasos.

[4] En este trabajo se han utilizado `Keras` y `TensorFlow` [23], dos bibliotecas de aprendizaje automático y redes neuronales muy populares, que se implementaron en los lenguajes de programación Python y C++. Keras es una interfaz de alto nivel que facilita la construcción y entrenamiento de modelos de aprendizaje automático en Python, mientras que `TensorFlow` es una plataforma de código abierto que proporciona herramientas para implementar y ejecutar modelos de aprendizaje automático en Python y C++.

Adam[5] utiliza una estimación del momento y de la magnitud de los gradientes anteriores para actualizar los parámetros del modelo en cada iteración. Sin embargo, en lugar de utilizar una tasa de aprendizaje constante para todos los parámetros, Adam adapta la tasa de aprendizaje de cada parámetro individualmente en función de su estimación del momento y de la magnitud del gradiente[6]. Esto permite que el modelo se ajuste de manera más eficiente y efectiva a los datos de entrenamiento, lo que puede llevar a una mayor precisión de la predicción en comparación con otros métodos de optimización.

2.2.5. Parámetros e hiperparámetros

Los **parámetros** del modelo son los parámetros internos que se obtienen o se estiman durante el proceso de entrenamiento, y son, por ejemplo, los pesos o sesgos de las neuronas. Estos parámetros, una vez el modelo ya esta entrenado, se utilizan para realizar predicciones.

Los **hiperparámetros** son parámetros externos al modelo establecidos por el programador, por ejemplo las funciones de activación a emplear o el tamaño del lote. Tienen un gran impacto en la red, y encontrar sus valores óptimos, no es algo trivial y en la mayoría de los casos se consigue por prueba y error.

A continuación, se muestran los principales hiperparámetros, clasificados en dos grandes grupos:

Hiperparámetros de estructura y topología de la red neuronal

- **Funciones de activación:** Las funciones utilizadas para calcular la salida de cada neurona en una capa de la red.
- **Número de capas de neuronas:** La cantidad de capas ocultas en la red, junto con el número de neuronas en cada capa, define la profundidad y complejidad de la red, lo que puede influir en su capacidad para modelar relaciones en los datos.
- **Inicialización de pesos:** La manera en que los pesos de las conexiones entre las neuronas se establecen al principio del entrenamiento, normalmente de manera aleatoria.
- **Tamaño de la capa de salida:** El número de neuronas en la capa de salida de la red, que depende del tipo de problema que se esté. En el contexto de series temporales su valor debe de coincidir con el número de características de salida a predecir.

[5] Empleando el entorno de TensorFlow, nunca tendremos que emplear directamente este algoritmo, por lo que ahondar en su parte matemática carece de sentido, mas allá de comprender que la optimización esta basada en gradientes.

[6] Términos como tasa de aprendizaje o momento, son ejemplos de hiperparámetros de una red neuronal, en el siguiente apartado se entrará en más detalle.

- **Función de pérdida:** La función utilizada para medir la discrepancia entre las predicciones del modelo y los valores reales del conjunto de entrenamiento.
- **Regularización:** La técnica utilizada para prevenir el sobreajuste, como la regularización L1 o L2, y los hiperparámetros asociados, como el factor de regularización (λ)).
- ***Dropout*:** La fracción de unidades que se "apaga" aleatoriamente durante el entrenamiento para prevenir el sobreajuste.
- **Arquitectura de red especializada:** Especifica arquitecturas de redes neuronales especializadas, como redes recurrentes (LSTM, GRU), redes convolucionales (tamaño de filtro, cantidad de filtros).

Hiperparámetros de algoritmo de aprendizaje

- **Número de *epochs*:** Es el número de veces que los datos de entrenamiento han pasado por la red neuronal. Encontrar el número óptimo es crucial, ya que un valor muy bajo puede limitar el potencial del modelo, mientras que un valor muy alto puede provocar sobreajuste u *overfitting*, donde el modelo se vuelve demasiado especializado en los datos de entrenamiento y no generaliza bien a nuevos datos.
- **Tamaño de ventana (*window size*):** En el análisis de series temporales, el tamaño de ventana se refiere al número de puntos de datos pasados que se utilizan para predecir un valor futuro. El tamaño de ventana óptimo debe estar relacionado con la estacionalidad de la serie temporal, mencionada en el Apartado 2.1.
- **Tamaño de lote (*batch size*):** A la hora de entrenar un modelo, se suelen introducir los datos de entrenamiento en lotes. Por ejemplo, en el contexto de series temporales, el *batch size* se refiere a el número de ventanas que se utilizan en cada lote durante el entrenamiento. El valor óptimo del *batch size* depende de varios factores, entre ellos la capacidad de memoria del computador o el número total de datos de entrenamiento.
- **Tasa de aprendizaje (*learning rate*):** El vector de gradiente tiene una dirección y una magnitud. Los algoritmos de gradiente descendiente multiplican la magnitud del gradiente por un escalar conocido como *learning rate*. Si se elige una tasa de aprendizaje (*learning rate*) muy alta, el entrenamiento puede entrenar muy rápido, pero puede llevar a ubicaciones completamente de la curva y no converger; en cambio, si se elige una tasa de aprendizaje muy baja, el entrenamiento puede ser muy lento y quedar atrapado en mínimos locales.
- **Decaimiento de la Tasa de aprendizaje (*learning rate decay*):** La mejor tasa de aprendizaje es aquella que va disminuyendo el *learning rate* a medida que van pasando las *epochs*. Este hiperparámetro reduce en cada época la tasa de aprendizaje

para obtener un entrenamiento más grande al principio, y a medida que avanza, ajustes más pequeños.

- **Momento (*Momentum*):** En situaciones donde la función objetivo tiene múltiples mínimos locales, el uso de momentum ayuda a evitar quedar atrapado en mínimos locales subóptimos. El momentum permite que el optimizador "recuerde" las actualizaciones de peso anteriores y las utiliza para mantener el impulso y superar obstáculos en el camino hacia el mínimo global[7].

2.3. Tipos de Redes Neuronales de pronóstico de series temporales

En el campo del Deep Learning, existen diversos tipos de redes neuronales son utilizados para abordar tareas específicas. Sin embargo, en esta sección, se prestará atención exclusivamente a un subconjunto particular de estas redes: las redes de pronóstico de series temporales.

Las arquitecturas de Deep Learning para la predicción de series temporales son modelos que ante un tamaño de ventana de entrada de ws_i,

$$x_{i,t} \quad \text{para } i \in \{1, 2, \ldots, M\} \text{ y } t \in [t - w_{s_i}, t], \tag{2.13}$$

realizan una predicción de salida de tamaño de ventana ws_o desplazado s pasos de tiempo hacia el futuro (*shift*).

$$\hat{y}_{i,t} \quad \text{para } i \in \{1, 2, \ldots, P\} \quad \text{y } t \in [t - ws_i + s, t - ws_i + s + ws_o]. \tag{2.14}$$

Una representación esquemática de los modelos de pronostico de series temporales se muestra en la Figura 2.8. La ecuación que relaciona la matriz de entrada X y la matriz de salida $\hat{Y}$ resulta:

$$\hat{Y} = F(X), \tag{2.15}$$

donde:

- $\hat{Y}$ es una matriz de tamaño $P \times ws_o$ que contiene las predicciones $\hat{y}_{i,t}$.
- X es una matriz de tamaño $M \times ws_i$ que contiene las observaciones de entrada $x_{i,t}$.

[7] Este hiperparámetro se puede entender como una canica que rueda cuesta abajo. Cuando la canica encuentra un obstáculo en el camino, su impulso acumulado le ayuda a superarlo en lugar de quedarse atascada.

- F representa una función optimizada por la red neuronal que procesa la matriz de entrada X para generar la matriz de salida $\hat{Y}$.

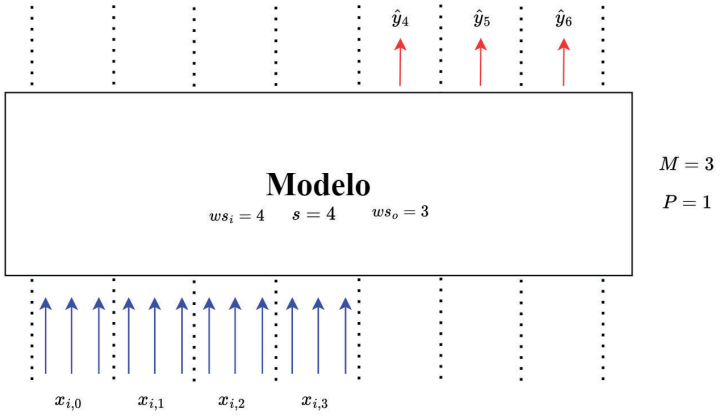


Figura 2.8: Diagrama de un modelo de series temporales con ventanas de entrada y salida de tamaño 4 y 3, respectivamente, y un desplazamiento de 4 pasos de tiempo. Este modelo, opera con 3 características de entrada, genera una única característica de salida.

2.3.1. Métodos de predicción de datos futuros

Existen dos enfoques principales para la predicción de series temporales utilizando deep learning: modelos iterativos y modelos directos [25, 26].

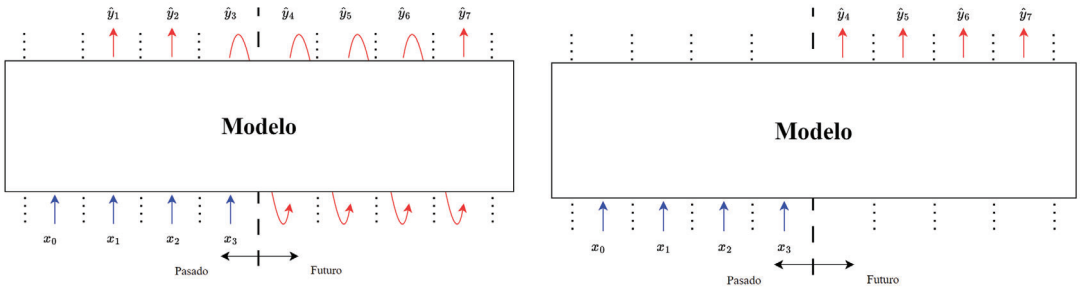


Figura 2.9: Representación esquemática de los métodos (a) iterativos y (b) directos.

- **Métodos iterativos:** Realizan predicciones a futuro de un solo paso de tiempo. Estos métodos utilizan las nuevas predicciones como nuevos datos de entrada para poder seguir realizando predicciones. Sin embargo, se produce un pequeño error en cada iteración que se va acumulando en cada paso de tiempo. Es un enfoque interesante para modelos auto regresivos que pueden ayudar a mitigar este problema.

- **Métodos directos:** Estos métodos utilizan únicamente una ventana de datos *a priori* conocidos, para producir una ventana de datos futuros sin necesidad de iterar e introducir salidas como entradas, al menos si se desea predecir un horizonte futuro de datos no muy extenso.

Una representación más visual del funcionamiento de estos modelos se observa en la Figura 2.9.

2.3.2. Redes neuronales densamente conectadas (DNN)

Como fue mencionado en la Sección 2.2.1, las redes neuronales se componen de numerosas capas que contienen gran cantidad de neuronas simples (perceptrones) cada una. Cuando todas las neuronas de una capa están conectadas a todos las neuronas de la capa anterior, la capa se denomina como capa densa.

En las redes densas usadas para el pronóstico de series temporales, comúnmente las capas ocultas suele emplear una función de activación "ReLu", mostrada en la ecuación (2.7), mientras que las capas de salida emplean la función de activación "linear" (2.4).

Cabe destacar de este tipo de redes neuronales que, ante una ventana de datos de entrada, solo son capaces de introducir por entrada un único paso de tiempo. Para solucionar este problema si se desea, por ejemplo, introducir 24 pasos de tiempo para solo producir un único valor futuro , será necesario realizar un aplanado de la entrada (*flatten*). como se muestra en la siguiente expresión:

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,N} \\ x_{2,1} & x_{2,2} & \dots & x_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M,1} & x_{M,2} & \dots & x_{M,N} \end{bmatrix},$$

$$\text{flatten}(X) = [x_{1,1}, x_{1,2}, \dots, x_{1,N}, x_{2,1}, x_{2,2}, \dots, x_{2,N}, \dots, x_{M,1}, x_{M,2}, \dots, x_{M,N}]. \quad (2.16)$$

2.3.3. Redes neuronales convolucionales (CNN)

Las redes neuronales convolucionales muestran muchas similitudes con las redes neuronales densas, están formadas por neuronas que tienen parámetros como pesos y sesgos. La principal diferencia es que están diseñadas principalmente para el procesamiento de imágenes y datos con una dependencia espacial, como imágenes en dos dimensiones, donde el color de un píxel depende enormemente de los píxeles de alrededor.

Las CNNs son capaces de encontrar relaciones en datos que son consistentes sin importar dónde se encuentren en una imagen. Para aplicar CNNs a conjuntos de datos de series temporales, los investigadores utilizan múltiples capas de convoluciones causales [27, 28, 29], como las que se usan en *WaveNet*. Estas convoluciones causales son filtros diseñados de tal manera que solo utilizan información pasada para hacer pronósticos.

El funcionamiento de una capa convolucional 1D consiste en pasar una ventana de datos de entrada de una característica de entrada i por una capa de neuronas de entrada. Posteriormente, no se conectarán todas las neuronas de entrada a una capa de neuronas ocultas (como en las redes densas), solo se realizará por pequeñas zonas localizadas. Por ejemplo, si se dispone de la característica x_1 una ventana de 12 pasos de tiempos, cada neurona de la capa oculta será conectada a una pequeña región de 4 neuronas. Esta pequeña región va deslizándose a lo largo de toda la capa de neuronas de entrada. Por cada posición de la ventana hay una neurona en la capa oculta que procesa esta información. En la Figura 2.10 se muestra el funcionamiento de una capa de neuronas convolucional 1D para una sola característica, mientras que en la Figura 2.11 se muestra el funcionamiento de 1 filtro convolucional para varias características.

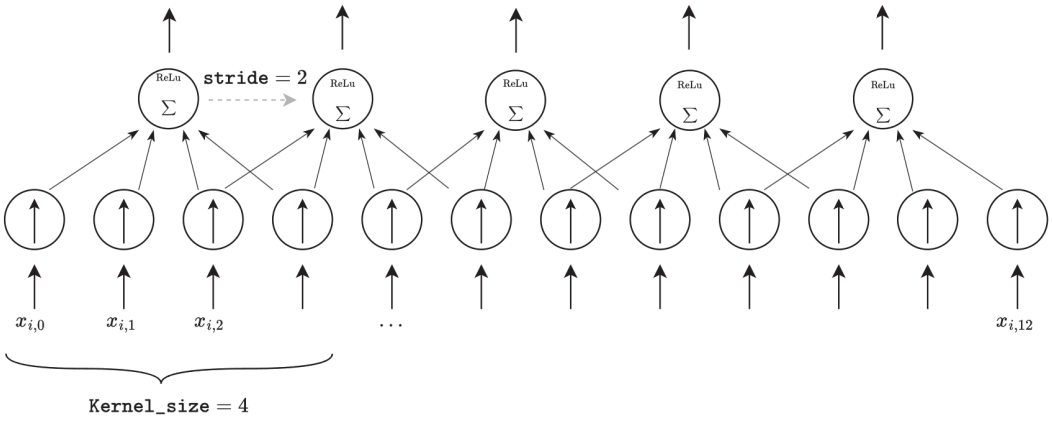
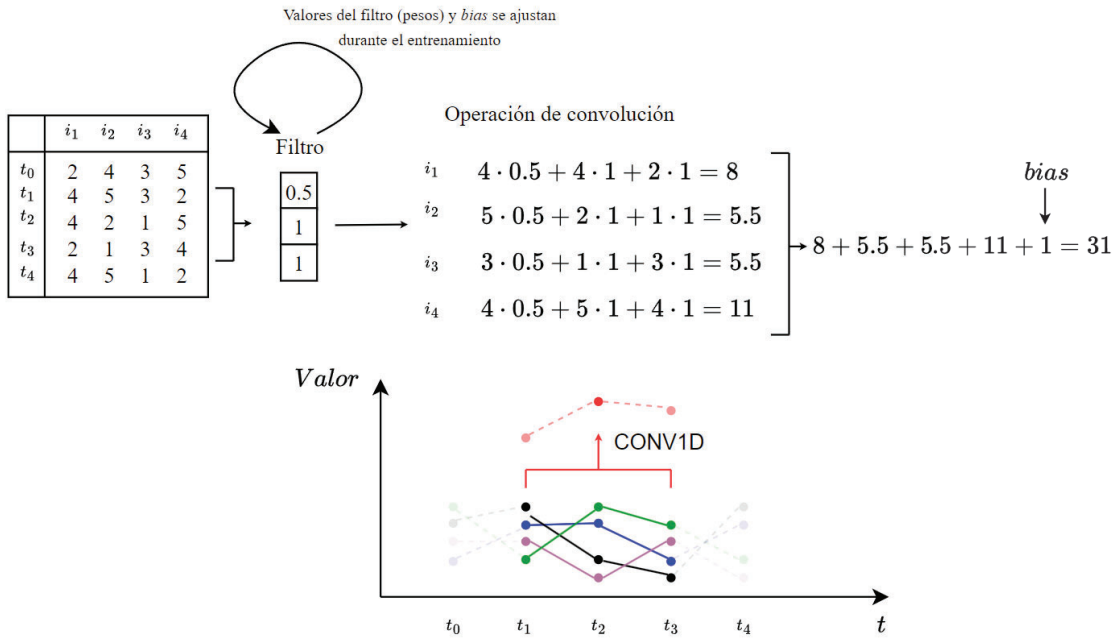


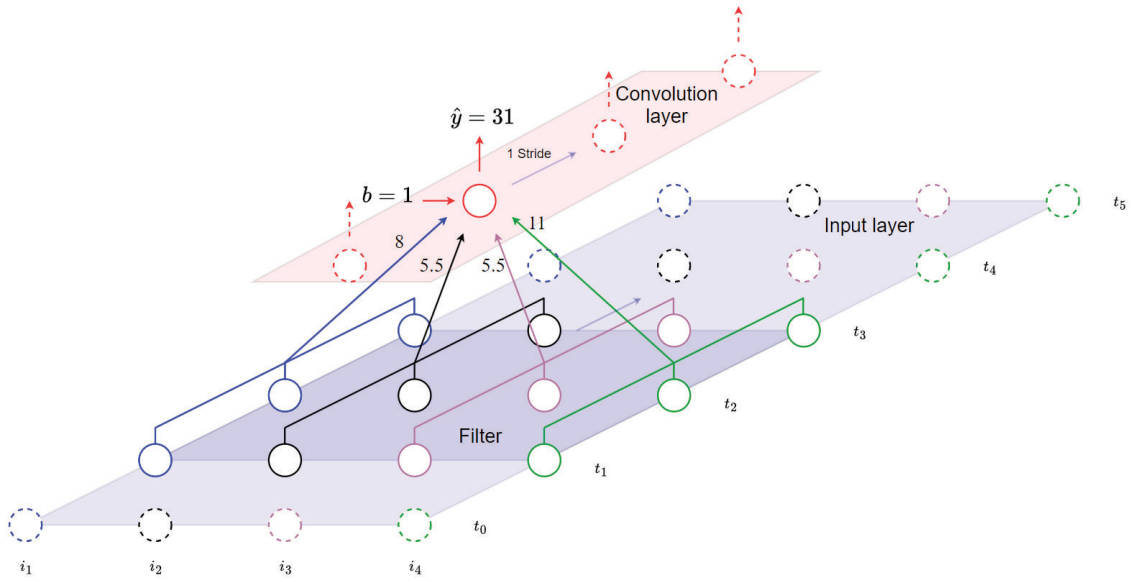
Figura 2.10: Representación del funcionamiento de una capa de neuronas convolucional.

Como se observa en la Figura 2.10, aparecen una serie de hiperparámetros como Stride o Kernel_size. A continuación, se presentan los principales hiperparámetros de las Capas Convolucionales 1D en el contexto de series temporales multivariantes:

- **Kernel_size:** determina el tamaño de la ventana o filtro convolucional que se desliza a lo largo de la serie temporal. El valor de Kernel_size afecta la cantidad de información local que se captura en cada paso de la convolución.
- **Stride:** es la cantidad de pasos que da el filtro o ventana convolucional cuando se desliza sobre los datos de entrada.



(a) Operación de convolución 1D



(b) Representación de una capa de convolución 1D en una red neuronal.

Figura 2.11: Representación del proceso de convolución en redes de predicción de series temporales multivariables.

- **Número de Filtros:** Especifica cuántos filtros convolucionales se aplicarán en paralelo a la entrada. Cada filtro captura diferentes patrones y características en la serie temporal, permitiendo que la red aprenda representaciones más ricas. Para permitir el pronóstico de series temporales multivariantes es necesario utilizar tantos filtros como salidas se quieran pronosticar (en la última capa de salida).
- **Función de Activación:** La función de activación se aplica después de la convolución.
- **Padding:** El relleno (padding) se utiliza para controlar el tamaño de la salida después de la convolución. Puede apreciarse en la Figura 2.10 que ante una entrada de 12 pasos de tiempo, se reduce a 5 pasos de tiempo. *Padding* consiste en añadir ceros a la entrada añadiendo más pasos de tiempo. Los tipos de padding son:
  - **Same:** Añade ceros a los datos de entrada tanto en el pasado como en el futuro, para que la salida tenga las mismas dimensiones que la entrada en una convolución.
  - **Valid:** No agrega relleno, lo que resulta en una salida más pequeña que la entrada en una convolución.
  - **Casual:** Agrega relleno solo en el pasado de una serie temporal para evitar que la convolución "vea" el futuro en aplicaciones de predicción de series temporales.

2.3.4. Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes son una clase de redes neuronales pensadas especialmente para analizar datos de series temporales, y permiten tratar la dimensión de tiempo, que hasta ahora no había sido considerada por ninguno de los anteriores modelos.

Las redes neuronales recurrentes (RNN), las conexiones entre neuronas no solo actúan hacia adelante desde la capa de entrada hasta la de salida, sino que incluyen conexiones que apuntan "hacia atrás en el tiempo", es decir una especie de retroalimentación.

Esto se debe a que las redes RNNs disponen de **neuronas recurrentes**, las cuales son una pequeña variación de las neuronas artificiales simples, las cuales reciben una entrada, produce una salida y envía a sí misma esa salida.

En cada instante de tiempo, esta neurona recibe una entrada x_t, así como su propia salida del instante anterior $\hat{y}_{t-1}$. Una representación de esta neurona [8], así como su despliegue a lo largo del tiempo, se muestra en la Figura 2.12.

Sin entrar en formulación excesivamente compleja e implementando la notación anteriormente empleada, el funcionamiento de esta neurona para la predicción de un nuevo

[8] No se debe confundir la representación temporal de esta neurona como si fuesen un conjunto de neuronas; simplemente se representa una sola neurona, evolucionando a lo largo del tiempo.

valor para una característica de salida, en concreto j, podría expresarse de la siguiente manera:

$$\hat{y}_{j,t} = tanh(W^T \cdot X_t + u \cdot \hat{y}_{t-1} + b). \tag{2.17}$$

donde, ante M características de entrada para predecir la característica concreta j, se emplea la secuencia de entrada $X_t = (x_{1,t}, \ldots, x_{M,t})$, la correspondiente matriz de pesos correspondientes a cada entrada $W = (w_1, \ldots, w_M)$, el peso u_j que opera sobre el estado de la red en el instante anterior $y_{j,t}$ y el sesgo correspondiente a la capa b.

Se puede apreciar que la función de activación empleada es $f(z) = tanh(z)$, no se emplea la función de activación ReLu, esto se debe a que las RNNs tienen tendencia a producir gradientes inestables que pueden desaparecer o incrementarse, exageradamente especialmente si se usa ReLu, que no satura la salida como si hace Tanh al limitar entre valores contenidos entre -1 y 1.

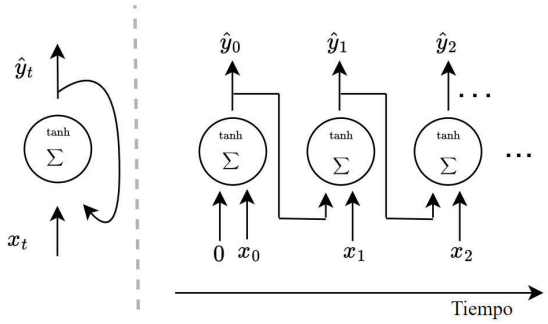


Figura 2.12: Representación del funcionamiento de una única neurona recurrente (imagen de la izquierda), y de su despliegue a lo largo del tiempo (imagen de la derecha).

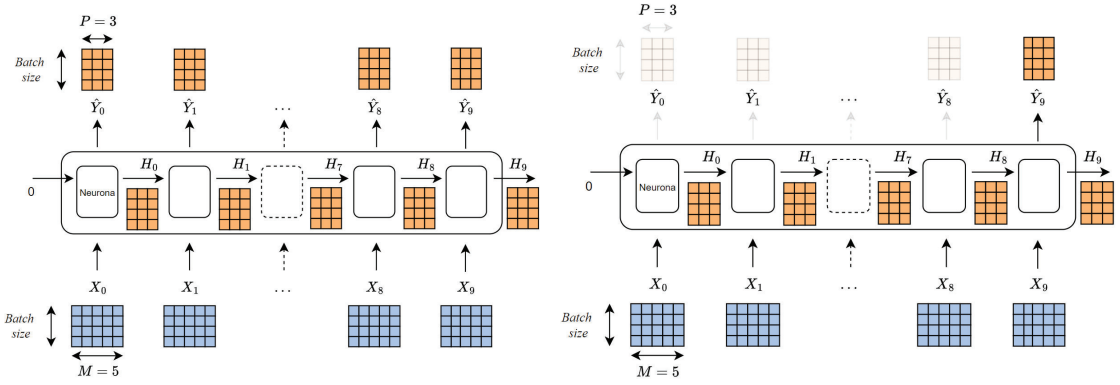
A continuación, se mostrará un pequeño ejemplo con el objetivo de entender las distintas maneras de construir una red neuronal recurrente. Se desea implementar una red neuronal que ,introduciendo una ventana de 10 pasos de tiempo de 5 características de entrada $M = 5$ (la potencia consumida en cada fase R, S y T, la temperatura y la humedad ambientes), sea capaz de predecir la potencia de cada fase en el futuro $P = 3$. Para conseguir esto será necesario una capa de neuronas recurrentes de 3 unidades o neuronas. Se dispone de datos de entrenamiento de estas 5 características en 40 pasos de tiempo. Existen dos enfoques de como tratar este problema.

El primer enfoque se denomina ***Stateless*** el cual consiste en reiniciar el estado interno H después de cada iteración. En este ejemplo, se dicide los datos en ventanas 10 pasos de tiempo de una hora en una hora ($v_1 = (0, \ldots, 9),\ v_2 = (1, \ldots, 10), \ldots$). Para facilitar el entrenamiento, como se mencionó en la sección 2.2.4 se tomarán pequeños lotes de 4 ventanas normalmente desordenadas (v_3, v_{10}, v_7, v_1) o *batches* de manera que la dimensión

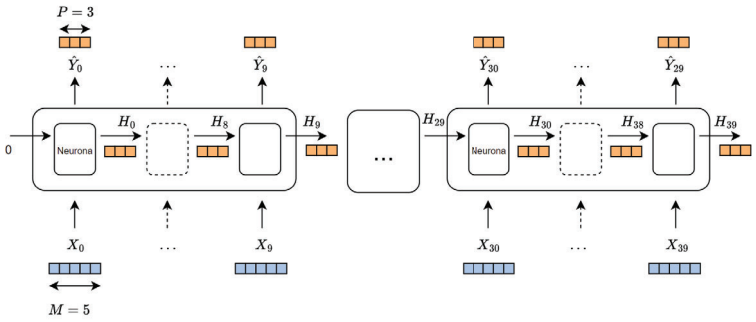
de los datos de entrada sera: 4 ventanas de 10 pasos de tiempo cada una y 5 características (4,10,5). En los modelos *stateless* se mantiene el estado interno de cada paso de tiempo H_t hasta que se realiza la predicción final de la ventana, posteriormente se reinicia el estado interno para calcular una nueva ventana. Este punto se dispone de dos opciones:

- ***Sequence to sequence***: Realiza una predicción para cada paso de tiempo de la ventana. Dimensión de los datos a la salida (4,10,3).
- ***Sequence to vector***: Realiza una única predicción. Dimensión de los datos a la salida (4,1,3).

El segundo enfoque se denomina ***stateful***, se conserva el estado interno H entre iteraciones de entrenamiento. En este ejemplo, dividimos los datos en ventanas de 10 pasos de tiempo de 10 en 10 horas cada una ($v_1 = (0, \dots, 9),\ v_2 = (10, \dots, 19), \dots$). También se organizan en lotes de 4 ventanas ordenadas (v_1, v_2, v_3, v_4) En los modelos ***stateful***, se mantiene el estado interno de cada paso de tiempo H_t hasta que se realiza la predicción final de la ventana. Luego, se conserva el estado interno para calcular la siguiente ventana.



(a) *RNN stateless sequence to sequence* (b) *RNN stateless sequence to vector*



(c) *RNN stateful*

Figura 2.13: Esquema de funcionamiento de las redes neuronales recurrentes *stateless* y *stateful*.

Long-Short Term Memory

Una célula LSTM es una variante de las redes neuronales recurrentes (RNN) que se utiliza para aprender patrones a largo plazo en secuencias de datos. A diferencia de las RNN tradicionales, las células LSTM tienen una memoria más larga y pueden capturar dependencias a largo plazo en los datos de entrada.

La célula LSTM consta de varias partes clave: la memoria a largo plazo, la memoria a corto plazo y tres puertas: la puerta de olvido (forget gate), la puerta de entrada (input gate) y la puerta de salida (output gate).

La memoria a largo plazo es donde se almacena la información relevante a largo plazo. Esta memoria se actualiza en cada paso de tiempo utilizando la información de la puerta de olvido y la puerta de entrada.

La puerta de olvido decide qué información se debe eliminar de la memoria a largo plazo. Recibe como entrada el estado anterior de la memoria a largo plazo y la entrada actual, y produce un vector de valores entre 0 y 1 que indica cuánta información se debe olvidar.

La puerta de entrada determina qué nueva información se debe agregar a la memoria a largo plazo. Al igual que la puerta de olvido, recibe el estado anterior de la memoria a largo plazo y la entrada actual, y produce un vector de valores entre 0 y 1 que indica cuánta información se debe agregar.

La memoria a corto plazo es una versión filtrada de la memoria a largo plazo. Se actualiza utilizando la información de la puerta de salida y la memoria a largo plazo actualizada.

La puerta de salida decide qué información de la memoria a corto plazo se debe utilizar como salida. Recibe el estado anterior de la memoria a largo plazo y la entrada actual, y produce un vector de valores entre 0 y 1 que indica cuánta información se debe utilizar como salida.

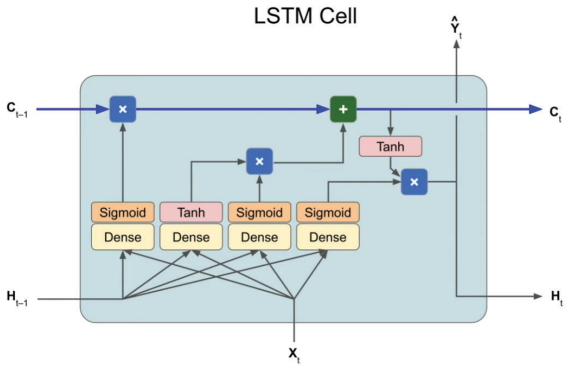


Figura 2.14: Funcionamiento de una célula, o neurona LSTM.

2.4. MQTT: un protocolo ligero de mensajería

MQTT (Message Queuing Telemetry Transport) es un protocolo de mensajería de publicación-suscripción ligero, ideal para la comunicación de máquina a máquina (M2M). Está diseñado para ser simple y eficiente, lo que facilita su implementación y escalamiento. MQTT se utiliza ampliamente en una variedad de aplicaciones, incluido Internet de las cosas (IoT), automatización industrial y sistemas integrados.

MQTT es un protocolo de mensajería de publicación-suscripción. Esto significa que hay dos tipos de participantes en una red MQTT: editores y suscriptores. Los editores publican mensajes, que luego se entregan a los suscriptores que han expresado interés en esos mensajes.

Los mensajes MQTT son muy ligeros (ocupan muy pocos bytes), lo que los hace ideales para la transmisión a través de redes de bajo ancho de banda. También son muy eficientes, lo que los hace muy adecuados para aplicaciones en las que la duración de la batería o el consumo son un inconveniente.

Especificación del protocolo MQTT

El protocolo MQTT está definido por el estándar OASIS MQTT 5.0 [30]. La especificación define los siguientes elementos del protocolo:

- Clientes: Los clientes MQTT son las entidades que participan en la red MQTT. Los clientes pueden ser editores, suscriptores o ambos.
- Corredores, o *brokers*: Los brokers MQTT son los servidores que almacenan y entregan mensajes.
- Temas, o *topics*: Los topics son los nombres de los canales en los que se publican y suscriben los mensajes.
- Mensajes: Los mensajes son las unidades de datos que se transmiten a través de la red MQTT.

Clientes MQTT

Los clientes MQTT son las entidades que participan en la red MQTT. Como se mencionó anteriormente, pueden ser de dos tipos:

- *Publishers*, o editores: Publican mensajes en la red.
- *Subscribers*, o suscriptores: Se suscriben a ciertos temas para recibir mensajes.

Los clientes MQTT pueden ser implementados en una variedad de lenguajes de programación, como C, C++, Java, Python, JavaScript, etc. También hay una serie de APIs y herramientas disponibles para ayudar con el desarrollo de clientes MQTT. Los clientes MQTT pueden ser implementados en dispositivos de cualquier tamaño, desde grandes servidores hasta pequeños microcontroladores. Esto hace que MQTT sea una buena opción para una amplia gama de aplicaciones, incluyendo IoT, automatización industrial y sistemas embebidos. A continuación se presentan algunos ejemplos de clientes MQTT:

- PubSubClient: Una biblioteca de cliente MQTT de código abierto para Arduino y ESP8266.
- Paho: Una biblioteca de cliente MQTT de código abierto para Python, PHP y C++.
- MQTT.js: Una biblioteca de cliente MQTT de código abierto para JavaScript.
- MQTT.NET: Una biblioteca de cliente MQTT de código abierto para .NET.
- MQTT X: Un cliente MQTT gráfico para Windows, macOS y Linux.

Los clientes MQTT proporcionan una serie de funciones y características, incluyendo:

- Publicación de mensajes: Los clientes MQTT pueden publicar mensajes en la red MQTT.
- Suscripción a temas: Los clientes MQTT pueden suscribirse a temas para recibir mensajes.
- Gestión de suscripciones: Los clientes MQTT pueden gestionar sus suscripciones, por ejemplo, para añadir o eliminar temas.
- Gestión de mensajes: Los clientes MQTT pueden gestionar los mensajes que reciben, por ejemplo, para almacenarlos, procesarlos o eliminarlos.
- Seguridad: Los clientes MQTT pueden usar una serie de mecanismos de seguridad para proteger los mensajes, como el cifrado.

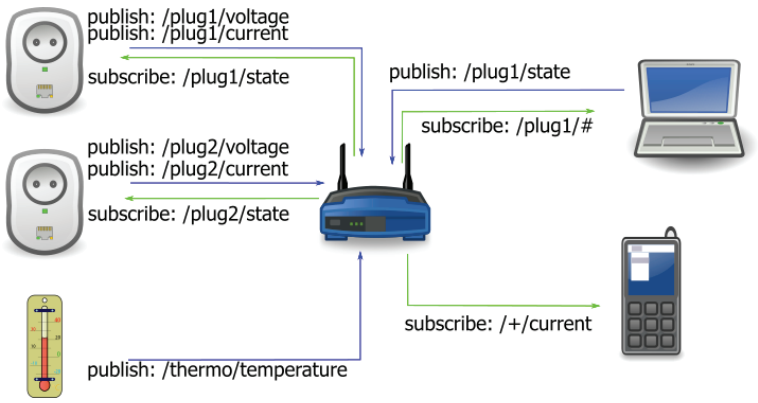


Figura 2.15: Esquema de mensajerías MQTT.

Brokers MQTT

Los brokers MQTT son los servidores que almacenan y entregan mensajes en la red MQTT. Son una parte esencial de la red, ya que proporcionan un punto central para la comunicación entre los clientes. Los brokers MQTT pueden ser implementados en una variedad de formas, desde software de código abierto hasta hardware dedicado. Entre los brokers MQTT disponibles se encuentran Mosquitto, RabbitMQ o Azure IoT Hub.

En este proyecto hemos usado Mosquitto [30]. Mosquitto es un broker MQTT de código abierto y de alto rendimiento. Es una buena opción para una amplia gama de aplicaciones, incluyendo IoT, automatización industrial y sistemas embebidos. Mosquitto es ligero y eficiente, lo que lo hace adecuado para dispositivos con recursos limitados. También es muy escalable, lo que lo hace adecuado para aplicaciones de gran tamaño. Mosquitto también es un broker MQTT seguro, ya que admite una serie de mecanismos de seguridad, como el cifrado TLS.

Topics o temas en el sistema MQTT

Los temas (topics) en MQTT desempeñan un papel fundamental en la comunicación dentro del mundo IoT. Estos temas son cadenas de texto que actúan como canales de comunicación entre dispositivos conectados a la red MQTT. Un tema se utiliza para indicar la dirección de destino de los mensajes que se envían a través del protocolo MQTT. Los temas son esenciales para la organización y la distribución eficiente de la información en una red de IoT. Si comparamos el mensaje MQTT con un correo electrónico, el tema no sólo sería el equivalente al destinatario, sino también al asunto, ya que serán destinatarios del mensaje todos aquellos dispositivos que se hayan suscrito a un determinado asunto.

En MQTT, los temas siguen una estructura jerárquica y se componen de múltiples niveles separados por barras diagonales (“/”). Esta jerarquía facilita la suscripción selectiva

a los temas y la publicación de datos a grupos específicos de dispositivos. Por ejemplo, si tenemos una red de sensores de temperatura, los temas podrían organizarse de la siguiente manera:

- `casa/salon/temperatura`
- `casa/cocina/temperatura`
- `oficina/sala1/temperatura`

En este caso, "casa" y "oficina" representan ubicaciones, "salon", "cocina" y "sala1" podrían ser diferentes áreas en esas ubicaciones, y "temperatura" indica el tipo de datos que se está enviando. Los dispositivos pueden suscribirse a temas específicos, por lo que, por ejemplo, un termostato en la sala de estar podría suscribirse a `casa/salon/temperatura`, mientras que otro en la cocina se suscribiría a `casa/cocina/temperatura`. Esto permite una administración eficiente de datos y una comunicación selectiva en la red IoT.

En resumen, los temas en MQTT son elementos esenciales para definir rutas de comunicación en una red de IoT, y su estructura jerárquica y organización permite una flexibilidad significativa en la gestión de datos y dispositivos en la red.

2.4.1. Mensajes y Payloads en MQTT

Los mensajes en MQTT son la unidad básica de comunicación entre dispositivos y se envían a través de los temas (topics) previamente definidos. Cada mensaje consiste en dos partes principales: el encabezado y el payload.

- **Encabezado MQTT:** El encabezado de un mensaje MQTT contiene información esencial para la entrega y el procesamiento del mensaje. Incluye elementos como el QoS (Calidad de Servicio), el identificador de mensaje, las banderas RETAIN y DUP, y otros campos necesarios para el control de flujo y la garantía de entrega.
- **Payload:** El payload es la carga útil del mensaje, que lleva los datos específicos que se desean transmitir. El payload, en MQTT, es el mensaje en sí. La longitud y el formato del payload pueden variar según la aplicación, lo que hace que MQTT sea muy versátil. Por ejemplo, en una aplicación de monitoreo de sensores de temperatura, el payload podría contener el valor de temperatura actual en grados Celsius.

La estructura modular de los mensajes MQTT, con un encabezado que controla la entrega y un payload que lleva los datos, permite una gran flexibilidad en la implementación de aplicaciones IoT. Los desarrolladores pueden adaptar la longitud y el contenido del

payload según sus necesidades específicas, lo que facilita la transmisión de una variedad de tipos de datos, desde información de sensores hasta comandos de control.

Aunque muchos mensajes en MQTT sólo necesitan el topic (por ejemplo: casa/cocina/-luz/enciende), hay muchos casos en que este es necesario (casa/cocina/luz, con payload "encender"). El payload puede enviar cadenas de texto ASCII sin formato o datos binarios (en crudo). Sin embargo, es frecuente que la información más compleja se envíe en formato JSON:

JSON en MQTT

JSON (JavaScript Object Notation) es un formato de datos ligero y ampliamente utilizado en la programación y la comunicación de datos en la web y en aplicaciones de IoT. Un objeto JSON es una secuencia de texto que se compone de pares clave-valor, donde la clave es una cadena de texto y el valor puede ser cualquier tipo de dato válido en JSON, como cadenas de texto, números, booleanos, arrays o incluso otros objetos JSON. MQTT y JSON a menudo van de la mano, ya que MQTT es un protocolo de comunicación que se integra fácilmente con datos estructurados en formato JSON. A continuación, se exploran los aspectos clave de la relación entre MQTT y JSON:

- **Flexibilidad en el Payload:** MQTT permite enviar datos en formato JSON como parte del payload de un mensaje. Esto significa que puedes incorporar información estructurada de manera jerárquica, como configuraciones de dispositivos, lecturas de sensores o datos de control, dentro de un mensaje MQTT. El payload JSON se ajusta perfectamente a la filosofía de MQTT, que es transportar información útil en un formato eficiente y legible.

- **Interoperabilidad:** Dado que JSON es un formato ampliamente reconocido y compatible con una amplia variedad de lenguajes de programación, MQTT facilita la interoperabilidad entre dispositivos y sistemas. Esto es especialmente útil en aplicaciones de IoT, donde diferentes dispositivos y plataformas pueden necesitar comunicarse entre sí de manera eficiente.

- **Ejemplo de Payload JSON en MQTT:** Supongamos que estás monitoreando un conjunto de sensores de calidad del aire. Puedes enviar lecturas de los sensores en un formato JSON como payload en un mensaje MQTT de la siguiente manera:

```
{   "dispositivo": "SensorA",
    "temperatura": 25.5,
    "humedad": 45.2,
    "pm2.5": 12.3   }
```

En este ejemplo, el payload JSON contiene datos relacionados con el dispositivo, la temperatura, la humedad y las lecturas de partículas PM2.5. Esto permite una representación estructurada de los datos, que es fácil de analizar y procesar.

La combinación de MQTT y JSON ofrece una solución poderosa para la comunicación de datos en aplicaciones de IoT y sistemas distribuidos. La flexibilidad de MQTT y la facilidad de uso de JSON hacen que esta combinación sea ideal para el intercambio de información en un formato estructurado y legible en una amplia variedad de aplicaciones.

Calidad de servicio y seguridad del protocolo MQTT

MQTT admite tres niveles de calidad de servicio (QoS):

- QoS 0: como máximo una vez entregado. Los mensajes pueden perderse o entregarse fuera de orden.
- QoS 1: Al menos una entrega. Se garantiza que los mensajes se entregarán al menos una vez, pero es posible que se entreguen varias veces.
- QoS 2: Entrega exactamente una vez. Se garantiza que los mensajes se entregarán exactamente una vez.

Además, MQTT admite una serie de funciones de seguridad, que incluyen:

- Texto sin formato: los mensajes se transmiten en texto sin formato.
- TLS: los mensajes se cifran mediante el protocolo Transport Layer Security (TLS).
- MQTT-SN: una versión ligera de MQTT diseñada para su uso en redes de bajo ancho de banda.

Aplicaciones del protocolo MQTT

MQTT se utiliza ampliamente en una variedad de aplicaciones, que incluyen:

Internet de las cosas (IoT): MQTT es una opción popular para aplicaciones de IoT, como dispositivos domésticos inteligentes, automatización industrial y dispositivos portátiles. Automatización industrial: MQTT se utiliza para conectar sensores y actuadores en sistemas de automatización industrial. Sistemas integrados: MQTT se utiliza para comunicarse entre dispositivos integrados.

En resumen, MQTT es un protocolo de mensajería de publicación-suscripción liviano que es ideal para la comunicación de máquina a máquina. Está diseñado para ser simple y

eficiente, lo que facilita su implementación y escalamiento. MQTT se utiliza ampliamente en una variedad de aplicaciones, incluidas IoT, automatización industrial y sistemas integrados.

2.5. Servicios REST y RESTful

Los servicios REST (Representational State Transfer) son una arquitectura de comunicación que utiliza el protocolo HTTP para transmitir datos entre aplicaciones. Estos servicios se utilizan para acceder y publicar información, con frecuencia obtenida de bases de datos como MongoDB y MySQL. A continuación, se exploran las funciones de los servicios REST, ejemplos de aplicaciones prácticas, y su relación con JSON.

2.5.1. Publicación de Información con servicios REST

Los servicios REST permiten la publicación de información de manera eficiente y estandarizada en la web. Estos datos pueden provenir de diversas fuentes, siendo comunes las bases de datos. Por ejemplo, servicios de pronóstico del tiempo pueden publicar datos meteorológicos actualizados, mientras que compañías eléctricas o aéreas pueden proporcionar información sobre consumos o precios de billetes. Los servicios REST, se basan en la arquitectura *Representational State Transfer* (REST), y utilizan HTTP para transmitir datos entre aplicaciones. Sin embargo, paradójicamente, los servicios REST no necesariamente siguen todos los principios de REST, mientras que los servicios RESTful se adhieren estrictamente a estos principios. REST por tanto es simplemente una transmisión de datos utilizando HTTP, mientras que los servicios RESTful siguen rigurosamente sus convenciones, garantizando una comunicación más coherente y estructurada entre aplicaciones.

Ejemplos de Aplicaciones

- **Servicios Meteorológicos:** Los servicios meteorológicos utilizan REST para ofrecer pronósticos y datos climáticos en tiempo real. Los clientes pueden realizar solicitudes a través de URLs con parámetros GET para obtener información como la temperatura actual, la previsión de lluvia o la velocidad del viento. Ejemplo:

- **Compañías Eléctricas o Aéreas:** Empresas de servicios públicos y aerolíneas utilizan REST para proporcionar a sus clientes datos sobre consumos de electricidad o precios de boletos. Los clientes pueden consultar la información relevante a través de servicios RESTful.

2.5.2. Consulta a un Servicio REST mediante URL GET

Un ejemplo de una consulta a un servicio REST sería la solicitud de datos de una estación meteorológica a través de una URL con parámetros GET. Por ejemplo, para obtener el pronóstico del tiempo para una ubicación específica, podríamos utilizar una URL como esta:

https://api.aemet.es/pronostico?ciudad=Madrid&fecha=2023-10-30

Aquí, "ciudad" y "fecha" son parámetros GET que especifican la ubicación y la fecha para la consulta. El servicio responderá con los datos meteorológicos relevantes en formato JSON.

2.5.3. Servidores REST en PHP

Es muy habitual usar PHP para los servicios Web. PHP es un lenguaje de programación Web "del lado del servidor". Esto quiere decir que, con PHP, el servidor Web no se limita simplemente a publicar una página HTML previamente escrita, sino que la construye en el momento en que se la solicita algún cliente, y con los parámetros concretos que le interesan al cliente. Así, un servidor no muestra la misma información de una página a todo el mundo. Utiliza parámetros GET (del estilo de "?ciudad=x&fecha=y") o cookies para diferenciar la página que muestra.

Para implementar un servidor, PHP se encarga de manejar las solicitudes HTTP, interactuar con la base de datos y responder con datos en formato JSON. Sin embargo, el código específico puede variar según el proyecto y la base de datos utilizada. Normalmente un código PHP que implementa un servicio RESTful incluye una secuencia relativamente sencilla de pasos:

- En las primeras líneas, el código PHP obtiene los parámetros a través de la variable $_GET
- El código analiza dicha estructura para identificar qué le están pidiendo y con qué parámetros adicionales (por ejemplo, le pueden estar preguntando por el número de personas en una vivienda, o la temperatura a mediodía del pasado 1 de enero)
- El código ejecuta una función específica, que podría consistir en una consulta a una base de datos, la lectura de un archivo, la ejecución de un cálculo matemático, un mensaje procedente de MQTT, e incluso la consulta a un servicio REST externo.
- El servidor escribe un documento JSON con la información obtenida de las funciones, y lo publica como texto plano en el servidor web.

Uso de JSON

JSON es el formato de datos más utilizado en servicios REST para estructurar la información. La mayoría de los servicios RESTful utilizan JSON para enviar y recibir datos debido a su simplicidad y facilidad de lectura. Esto facilita también la interoperabilidad y el procesamiento de datos en aplicaciones cliente.

En resumen, los servicios REST y RESTful son fundamentales para la publicación y acceso eficiente de información en aplicaciones web y móviles. Se utilizan para obtener datos de diversas fuentes, como bases de datos, y los presentan en formato JSON, lo que permite a los clientes acceder y utilizar la información de manera efectiva.

Capítulo 3

Metodología

La optimización del consumo energético y la gestión de costes en entornos residenciales se ha convertido actualmente en una tarea absolutamente imprescindible para garantizar un futuro sostenible, y la mejor forma de conseguir este objetivo es, sin duda, conocer a los actores responsables del consumo, así como predecir su comportamiento. En este capítulo se describe la metodología integral empleada para alcanzar este ambicioso objetivo, y que consiste básicamente en desarrollar un sistema predictivo robusto para el consumo de energía y la previsión de costos en una residencia familiar con una instalación eléctrica trifásica. El sistema integra datos de diversas fuentes, incluidas mediciones de energía, condiciones ambientales y precios históricos de la electricidad. El objetivo es crear un modelo altamente preciso que no sólo prediga el consumo de energía sino que también anticipe los picos de potencias, lo que lleve a decisiones de consumo de energía más informadas y eficientes.

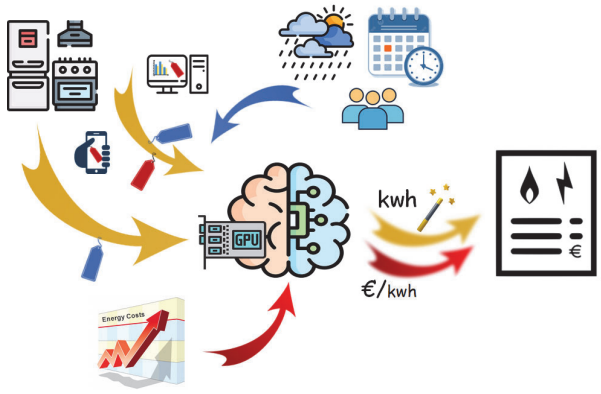


Figura 3.1: Un esquema del sistema propuesto, donde las flechas indican flujos de datos eléctricos (amarillas), ambientales (azules) y de costes (rojas).

Este trabajo aprovecha una combinación de tecnologías, que incluyen herramientas y dispositivos IOT, como MQTT, para la recopilación de datos de sensores eléctricos; la programación de móviles con Android Studio y Kotlin para el etiquetado asistido de dichos

datos, C++ y programación Web con PHP, para el acceso a servicios REST de datos meteorológicos disponibles públicamente, así como para la implementación de un servicio REST propio con datos de sensores ambientales; MongoDB y MySQL para el almacenamiento de datos y, finalmente, TensorFlow y Keras, para la implementación de redes neuronales artificiales. Este enfoque multifacético nos permite aprovechar el poder del aprendizaje automático y los datos en tiempo real para crear una solución integral para la gestión de la energía. Siguiendo esta metodología integral, nuestro objetivo es desarrollar un sistema sofisticado que no solo prediga con precisión el consumo de energía y el uso máximo de energía, sino que también proporcione información valiosa para una gestión energética global eficiente en entornos residenciales.

La metodología propuesta considera las siguientes etapas:

- En primer lugar, la instalación o adaptación de los sensores y dispositivos que detecten tanto el consumo instantáneo de energía como las condiciones ambientales presentes en cada momento que pudieran afectar al consumo.
- En segundo lugar, la preparación de las herramientas software para la recogida y el almacenamiento de datos. Estas dos primeras etapas debían abordarse, forzosamente, en la fase más temprana del proyecto, ya que el volumen de datos recogidos debía ser muy elevado ("big data") por las propias características que requiere el aprendizaje profundo. Es evidente que los tiempos académicos propios de un TFM asfixian las necesidades del proyecto para recoger suficientes datos. Aún así, a sabiendas de que será imposible validar plenamente el sistema hasta que trascurran varios meses o años después de la finalización de esta memoria, se continuarán con las siguientes etapas para cumplir objetivos:
- En una tercera fase, se propone la implementación de una red neuronal artificial que sea capaz de predecir el precio de la electricidad en el mercado español, utilizando la información histórica existente en bases de datos públicas. Si bien es sabido que ya existen numerosas herramientas basadas en Inteligencia Artificial para predecir el "precio de la luz", se ha considerado elaborar una herramienta propia por dos motivos: en primer lugar, para la propia formación del proyectando, y en segundo lugar, para no depender de una información externa, que no siempre es accesible y gratuita.
- En la cuarta fase se elaborarán y pondrán en marcha las aplicaciones software para completar el etiquetado de los datos. Si bien la información de las condiciones ambientales presentes en el momento en que se consume la electricidad ya son conocidas, no ocurre lo mismo con la información sobre los aparatos responsables del consumo. Esta fase concluye con el etiquetado masivo de la información histórica.
- En la quinta fase, se elaborará una arquitectura de aprendizaje profundo capaz de predecir el consumo futuro, y que, gracias al etiquetado, tenga en cuenta las cos-

tumbres de las personas que habitan la vivienda y las características eléctricas y temporales de los aparatos.

- Finalmente, se validarán los resultados utilizando las propias herramientas que proporciona el software TensorFlow, siempre con las limitaciones temporales del proyecto, como se comentó anteriormente.

A continuación, en las siguientes subsecciones profundizaremos en los detalles de cada una de estas fases.

3.1. Instalación de sensores

3.1.1. Sensores de potencia y energía

Para medir la potencia hemos elegido el sensor Shelly 3EM. El sensor Shelly 3EM es un dispositivo de monitorización de energía diseñado para medir el consumo eléctrico en un entorno residencial o comercial. Este sensor está fabricado por la empresa Shelly y se integra en sistemas de automatización del hogar y sistemas de gestión de energía.

Figura 3.2: Sensor de potencia Shelly 3EM.

El Shelly 3EM permite medir la energía eléctrica consumida en tres circuitos diferentes de forma simultánea. Algunas de sus características clave incluyen la medición de la corriente, voltaje y potencia activa en tiempo real en cada uno de estos circuitos. Además, puede calcular la energía acumulada y proporcionar datos sobre la calidad de la energía eléctrica, como la frecuencia y el factor de potencia.

Una característica relevante del Shelly 3EM es su capacidad para integrarse en sistemas de automatización a través del protocolo MQTT. Descrito exhaustivamente en el capítulo anterior, es un protocolo de comunicación ligero que permite la publicación y suscripción de datos en tiempo real. Al utilizar MQTT, el Shelly 3EM puede enviar los datos de medición de energía a otros dispositivos o sistemas que sean compatibles con MQTT, como sistemas de domótica o aplicaciones de gestión de energía como la que se propone en este proyecto.

En resumen, el sensor Shelly 3EM es un dispositivo diseñado para medir y monitorizar el consumo eléctrico en tres circuitos diferentes, lo que lo hace útil para el control y la gestión de la energía en el hogar o en entornos comerciales. Su capacidad de publicar datos a través

del protocolo MQTT facilita su integración en sistemas de automatización y control más amplios.

3.1.2. Sensores meteorológicos

La recogida de datos meteorológicos se realiza mediante la combinación de una estación FWS-20 del fabricante español PCE, y una aplicación para recogida y publicación de datos en red (Meteobridge) instalada en una Raspberry Pi, model 4. En este caso, no fue necesaria la instalación de la estación, pues lleva años en funcionamiento en dicha vivienda. Tan sólo hubo que modificar el software para que hiciera públicos los datos en un servicio REST para que pudieran ser utilizados en este proyecto utilizando un protocolo estándar.

Figura 3.3: Estación meteorológica.

3.1.3. Detección de presencia de personas en la vivienda

La propia Raspberry Pi que ejecuta el software de recogida de datos meteorológicos será utilizada como dispositivo para la detección de presencia de los miembros de la familia en casa, en combinación de otros dispositivos ya existentes y bien conocidos: los móviles. En este caso, se ha utilizado un sencillo truco: hacer *ping* (comprobar si hay una respuesta en la red) a cada uno de los móviles, y utilizar un fichero de la Raspberry para guardar la marca (en los archivos /tmp/1 y siguientes, como se muestra en el Listado 3.1, para una sola persona). El script se ha programado con la ayuda de ChatGPT.

```
#!/bin/bash
ip="192.168.1.145"
archivo="/tmp/1"
ping -c 1 $ip > /dev/null
if [ $? -eq 0 ]; then
    echo "La IP $ip responde" > $archivo
else
    if [ -f $archivo ]; then
        rm $archivo
    fi
fi
```

Código 3.1: Script de detección de una persona con ping en Bash

3.2. Recogida y almacenamiento de datos

Además de los datos que proceden de los sensores, y que se han mencionado en el apartado anterior, hay parámetros que se pueden obtener de servicios públicos de datos u otras fuentes de información de lo más variada, como podrían ser los propios manuales de instrucciones de los aparatos de la vivienda. En este apartado resumimos los distintos orígenes de datos, la forma de recopilarlos, así como el método de almacenamiento.

La gráfica 3.4 representa gráficamente los orígenes y destinos de los datos, así como las dos aplicaciones que se han desarrollado específicamente para la recogida y publicación de datos de este proyecto, y que serán descritas en posteriores apartados:

- Servidor REST, para la publicación de datos ambientales
- DemonioMQTT, para la recolección y almacenamiento en bases de datos de los datos ambientales y eléctricos

3.2.1. Recogida de datos

Datos eléctricos

La inmensa mayoría de los datos eléctricos recopilados proceden del sensor Shelly. Dichos datos son publicados mediante el protocolo MQTT [30], con una periodicidad aproximada de un minuto. El propio dispositivo almacena dicha información en un archivo *csv*. Dichos datos pueden visualizarse en la web [31]. Sin embargo, hemos preferido almacenarlos en una base de datos para optimizar la búsqueda de información. En particular, hemos elegido MongoDB [32], ya que esta base de datos noSQL brinda ventajas significativas para el tratamiento de grandes series de datos temporales gracias a su escalabilidad, el intuitivo modelo de documentos JSON (JavaScript Object Notation) para representación de los datos, indexación eficiente, replicación, y capacidad de análisis en tiempo real, lo que facilita la gestión y obtención de información valiosa de series temporales extensas. Además, existe mucha documentación sobre la forma de consultar y almacenar en este tipo de bases de datos desde distintos lenguajes de programación, como los utilizados en este proyecto (Python, C++, PHP y Matlab).

Para almacenar la información en la base de datos se ha creado una aplicación *ad hoc* que se ejecuta en una máquina virtual protegida con un SAI: demonioMQTT, descrito en la sección 3.2.2. En informática, un "demonio" (también conocido como "daemon", del inglés) es un programa o proceso que se ejecuta en segundo plano en una computadora o servidor sin necesidad de interacción directa con el usuario. El demonio está suscrito a las publicaciones MQTT de Shelly, y los almacena en la base de datos MongoDB.

Otros datos eléctricos se obtendrán de los manuales de instrucciones de los aparatos, así como de un medidor de consumo que se inserta entre el aparato y la toma de la pared, similar al de la figura.

Datos ambientales

El sentido común nos dice que el consumo eléctrico en una vivienda no sólo depende de la potencia de los aparatos que están instalado y su patrón de uso, sino también de una serie de condiciones ambientales que, en mayor o menor medida, modifican dichos patrones. Quizás, el más evidente es el tiempo atmosférico, ya que es bien sabido que las situaciones térmicas extremas disparan el consumo eléctrico por el uso del aire acondicionado y los radiadores. También la radiación solar juega un papel importante en el consumo eléctrico, especialmente en instalaciones con placas solares.

En este proyecto hemos catalogado como datos ambientales al conjunto de factores externos que consideramos que pueden tener un impacto significativo. Y entre los que más pueden afectar significativamente al consumo eléctrico de la vivienda, a largo plazo, hemos considerado la siguiente lista:

- Hora del día y día de la semana
- Festividades y vacaciones
- Presencia o ausencia de miembros de la unidad familiar
- Información meteorológica básica: Temperatura, viento y humedad (especialmente por su contribución a la sensación térmica)
- Datos de luminosidad y radiación solar, básicamente determinados por la elevación del sol y la nubosidad

Todos estos datos, a excepción de la presencia/ausencia de uno o varios miembros de la familia, pueden obtenerse fácilmente con unos cálculos relativamente simples, con la ayuda de librerías (como la posición del sol) o mediante el acceso a servicios externos, como por ejemplo, los proporcionados por la API de AEMET [33].

3.2.2. Aplicaciones de preprocesamiento y almacenamiento de datos

Servidor REST de datos ambientales

Un servicio RESTful, o REST (de Representational State Transfer), es una arquitectura web que utiliza solicitudes HTTP (como GET, POST, PUT y DELETE) para permitir la

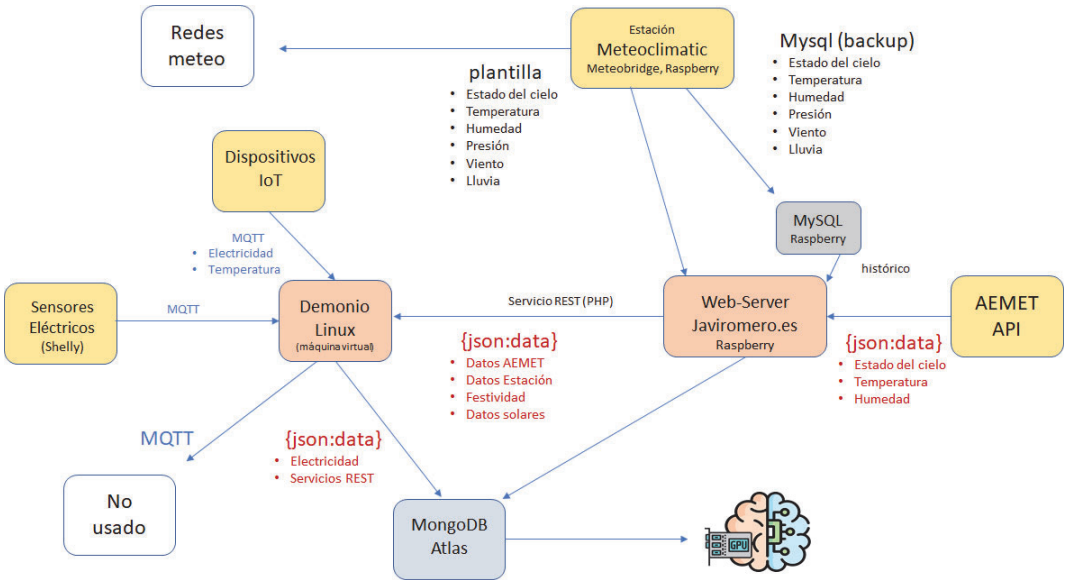


Figura 3.4: Esquema del sistema de recogida de datos ambientales y eléctricos que alimentan la red neuronal. El amarillo representa los orígenes de datos, y en naranja, las dos aplicaciones que recuperan datos. En gris, las bases de datos.

comunicación y transferencia de datos entre sistemas a través de la web. Un aspecto clave de los servicios REST es su capacidad para representar datos en formato JSON, que es ampliamente utilizado debido a su simplicidad y facilidad de lectura tanto para humanos como para máquinas. JSON se utiliza para estructurar y transmitir datos entre sistemas, lo que lo convierte en un formato común en servicios web REST. En este proyecto se ha programado, en lenguaje PHP, un servicio REST para la publicación de datos ambientales [34]. El servicio utiliza cinco parámetros: cuatro de ellos para obtener información de un día y hora específicos: y, m, d, h, así como un quinto parámetro (secreto, por motivos de seguridad, pues permite conocer qué personas están presentes en la vivienda) que simplifica la información más relevante con este formato:

```
{
"fechaUTC": "2023-10-10T08:33:00+00:00",
"festivo": false,
"alturaSol": 24.3,
"cielo":"17",
"temperatura":23,
"humedad":77,
"personas":11
}
```

Código 3.2: Ejemplo de datos del servicio REST compacto

Según el día y la hora de la solicitud, la información meteorológica distingue cinco tipos de consulta, obteniendo, en función de ella, datos procedentes de observación, predicción, o climáticos.

- Futuro lejano: No hay predicción meteorológica. Sólo climática.
- Futuro próximo: Desde la hora actual (más dos), y mientras haya predicciones meteorológicas, incluyendo nubosidad.
- Actual. La consulta solicita una hora que es posterior a la publicación del último parte de predicción, en el momento en que se genera la respuesta, y anterior a la hora actual, más dos.
- Pasado reciente: AEMET no ha hecho pública la información de observación (suele tardar un par de días), pero la estación meteorológica de la vivienda la ha almacenado en una base de datos MySQL (temperatura, humedad y lluvia).
- Pasado. Existen datos parciales de observación, aunque es posible que falten algunos datos de nubosidad.

Para el estado del cielo, se ha utilizado la codificación de iconos de AEMET, donde, por ejemplo, el número 17 significa que hay nubes altas.

Además, se han programado en PHP dos sencillas funciones para obtener los siguientes datos:

- Cálculo de las festividades, utilizando una función que determina el día de la semana, así como los festivos nacionales, autonómicos y locales, las excepciones, así como la función PHP *easter_ date($year)* para el cálculo de la Semana Santa. En la documentación Doxygen del servicio pueden consultarse el código [35].
- Cálculo de las personas presentes en la vivienda. A partir del script 3.1, se ha codificado en binario las personas presentes en la vivienda: por ejemplo, si el campo *personas* tiene el valor 11 (1011 en binario), significa que solo la persona número dos está ausente en ese momento.

Los datos obtenidos del servidor REST, al estar en formato JSON, se almacenan directamente en una colección MongoDB sin necesidad de procesamiento.

El servicio demonioMQTT

La principal aplicación de recogida y almacenamiento de datos es el demonioMQTT. Dicha aplicación es un servicio Linux, programado en C++, que se encarga de:

- Recoger datos ambientales mediante consultas al servidor REST, utilizando una librería *curl* para la descarga.

- Recoger datos eléctricos mediante subscripción MQTT, utilizando la librería *Paho C++*.
- Almacenar, cada 10 minutos, los datos ambientales en la base de datos MongoDB, utilizando la librería *mongocxx*.
- Preprocesar y etiquetar datos eléctricos.
- Almacenar datos eléctricos en la base de datos MongoDB.
- Publicar en MQTT los boletines y alertas que necesite el sistema, o que pudieran necesitarse en futuros proyectos.

El preprocesamiento y pre-etiquetado de datos eléctricos que ejecuta el demonio se encarga, básicamente, de determinar y etiquetar cuándo se produce un cambio en la potencia de cada fase eléctrica de la vivienda y en qué cantidad (actualmente el umbral para que se almacene está en 5W). Además, cualquier dato de potencia registrado se etiqueta con los datos ambientales más recientes, si existen. Estos datos tienen una antigüedad inferior a 10 minutos.

La documentación del código, con una descripción detallada de variables y funciones, puede consultarse en [36].

3.2.3. Bases de datos

Este proyecto utiliza tres bases de datos:

- Datos csv del dispositivo Shelly EM.
- Base de datos MySQL con datos meteorológicos de la estación.
- Base de datos MongoDB Atlas.

Sin embargo, sólo la tercera forma una parte esencial de este proyecto, y es además la única creada *ad hoc* para el mismo. La base de datos tiene cuatro tablas (o colecciones, según la nomenclatura MongoDB): *ambientes, aparatos, periodicos y saltos*. En cada fila de la tabla (o documento, según la nomenclatura MongoDB), encontramos:

- En la tabla *ambientes*, el documento compacto recuperado del servicio REST (véase como ejemplo el listado 3.2).
- En la tabla *aparatos*, el nombre de cada aparato registrado de la vivienda, su potencia de referencia, si la tiene, y la, o las fases en la que se conecta.

- En la tabla *periodicos*, el valor de potencia en cada fase, en cada minuto exacto (segundo cero), y el ambiente más reciente.

- En la tabla *saltos*, los incrementos significativos de potencia que se detecten, con la información adicional conocida:


```
{
  "_id": {"$oid": "65251db200211de7c20dd960"},
  "tiempo": {"$date": "2023-10-10T09:47:30.727Z"},
  "ambiente": "65251d5800211de7c20dd95b",
  "fase": 1,
  "desde": 449,
  "hasta": 428,
  "aparato": 9
}
```


Código 3.3: Ejemplo de un documento de la colección *saltos* de la base de datos MongoDB

3.3. Previsión del precio de la electricidad

El objetivo de este apartado es realizar predicciones futuras del precio marginal de la electricidad en España y que contribuye a determinar el precio futuro de la factura eléctrica. Los modelos que se presentan a continuación tendrán en cuenta, durante su entrenamiento, el precio de la electricidad hora a hora desde el 25/08/2022 hasta el 25/08/2023, considerando la hora del día, el día de la semana y el mes del año correspondientes a cada precio de la electricidad registrado. Al tener en cuenta todos estos factores, se pretende que los modelos consideren las horas punta y valle, así como la importancia de días ciertamente atípicos que se repiten periódicamente, como los fines de semana o el día de Navidad.

3.3.1. Preprocesamiento de los datos

Los datos del precio del mercado diario de la electricidad en €/MWh se han obtenido directamente de OMIE (Operador del Mercado Ibérico de Energía) [37]. Estos datos muestran el precio de la electricidad a cada hora del día recopilado desde hace un año. En la Tabla 3.1 se muestra el formato original de los datos recogidos en OMIE, los cuales requieren de un preprocesamiento para poder ser introducidos a la red neuronal.

Primero, se realiza la conversión de las fechas y horas en el conjunto de datos. Las fechas originales en formato de texto se transforman en objetos de fecha y hora. A continuación, se convierten estas fechas en *timestamps* Unix, que representan el tiempo en segundos desde el 1 de enero de 1970.

Tabla 3.1: Formato original de los datos.

Índice	**Fecha**	**Hora**	**Precio**
0	25/08/2022	00:00:00	P_0
1	25/08/2022	01:00:00	P_1
⋮	⋮	⋮	⋮
8760	25/08/2023	23:00:00	P_{8760}

Tabla 3.2: Formato preprocesado de los datos.

Índice	**Precio**	**dia**	**semana**	**mes**
0	P_0	day_0	$week_0$	$year_0$
1	P_1	day_1	$week_1$	$year_1$
⋮	⋮	⋮	⋮	⋮
8760	P_{8760}	day_{8760}	$week_{8760}$	$year_{8760}$

Para capturar patrones cíclicos en los datos, se convierten las horas del día en representaciones sinusoidales. Se utilizan funciones trigonométricas para transformar las horas en señales sinusoidales, lo que facilita la detección de patrones diarios, semanales y anuales en los datos. Estas trasformaciones trigonométricas son exactamente las mismas que se realizarán para la predicción del consumo de la vivienda cuyas expresiones vienen recogidas en la ecuación (3.2), mostradas más adelante.

De esta manera, los datos preprocesados adquieren el formato que se muestra en la Tabla 3.2.

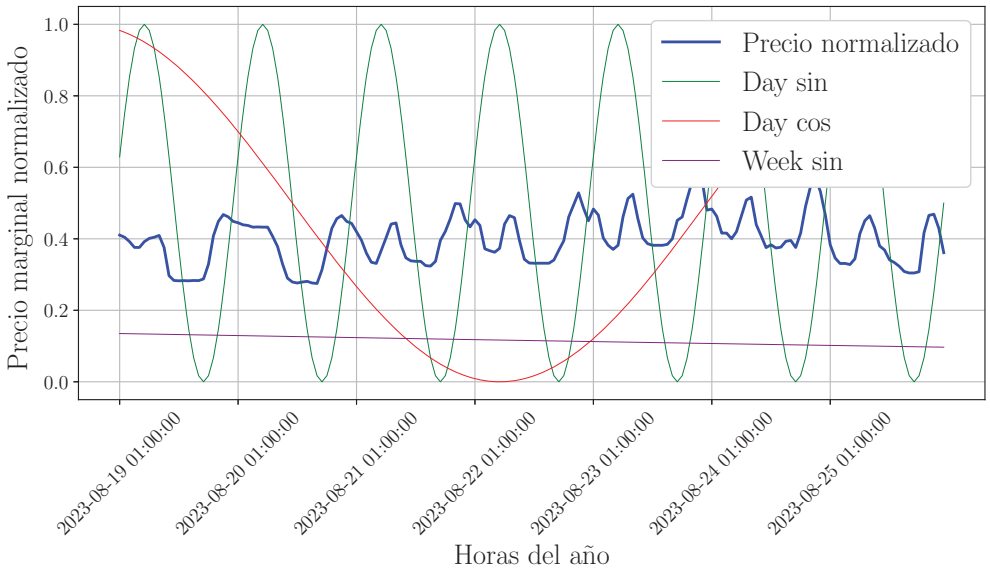


Figura 3.5: Señales de entrada a los modelos de predicción del precio marginal de la electricidad.

3.3.2. Segmentación y normalización de los datos

En este proceso, se dividen los datos de la serie temporal en tres conjuntos distintos con el propósito de desarrollar y evaluar modelos de predicción. Las razones detrás de esta división son las siguientes:

1. **Conjunto de Entrenamiento**: Este conjunto representa el **70 %** de los datos totales.

2. **Conjunto de Validación**: Se reserva un **29.7 %** de los datos para validar los datos de entrenamiento.

3. **Conjunto de Prueba**: La porción restante, un **0.3 %** que corresponde al ultimo día de datos se utiliza como conjunto de prueba.

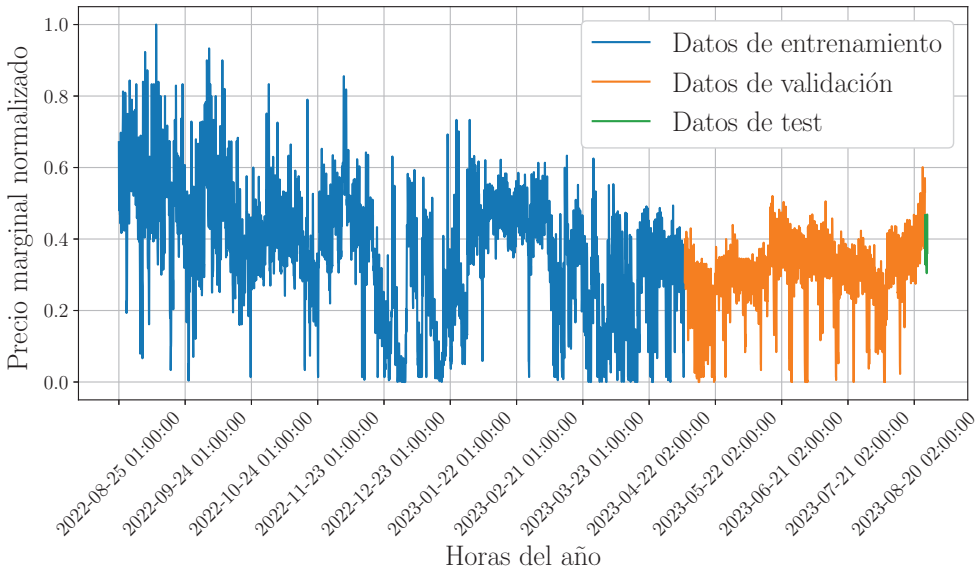


Figura 3.6: Representación fragmentada de los datos normalizados del precio de la electricidad.

La normalización de los datos se realizará empleando el método máximomínimo, el cual es un método bastante sencillo que consiste en que todos los valores de la serie temporal se encuentren comprendidos entre 0 y 1, donde 0 corresponde al menor valor de la serie y 1 al mayor valor. La ecuación empleada es la siguiente

$$P_{norm} = \frac{P - min(P)}{max(P) - min(P)}. \tag{3.1}$$

3.3.3. Creación de las ventanas de datos

Una vez obtenidos los datos en un formato apropiado, es conveniente agrupar los datos en ventanas temporales mas pequeñas para facilitar el entrenamiento. En los modelos utilizados para las predicciones, ha sido necesario crear dos tipos de ventana diferentes que se adapten a la naturaleza de los distintos algoritmos empleados. El procedimiento empleado para la creación de ambos tipos de ventana es el siguiente:

- **Ventanas secuencia a secuencia**
  1. Crear un Dataset en `TensorFlow` a partir de una matriz de datos.
  2. Crear ventanas de datos de tamaño 24+1 con un desplazamiento de 1.
  3. Aplanar las ventanas para convertirlas en forma de "vector".
  4. Mezclar las ventanas de manera aleatoria.
  5. Dividir cada ventana en una ventana de entrada de tamaño 24 y una ventana de etiquetas de tamaño 24.
  6. Agrupar las ventanas en lotes de tamaño fijo.
- **Ventanas secuenciales**
  1. Crear un Dataset en `TensorFlow` a partir de una matriz de datos.
  2. Crear ventanas de datos de tamaño 24+1 con un desplazamiento de 24.
  3. Aplanar las ventanas para convertirlas en forma de "vector".
  4. Dividir cada ventana en una ventana de entrada de tamaño 30 y una ventana de etiquetas de tamaño 24.
  5. Agrupar las ventanas en lotes de una única ventana.

3.3.4. Proceso de entrenamiento

El proceso de entrenamiento empleado puede resumirse en los siguientes puntos:

1. **Compilación del Modelo:** El modelo de la red neuronal se compila especificando la función de pérdida (loss), el optimizador y las métricas para el entrenamiento. En todos los modelos, se ha utilizado la función de pérdida de MAE (2.10) y el optimizador Adam con la tasa de aprendizaje (`lr=1e-4` en el modelo Dense y `lr=1e-3` en el resto de modelos). Se registra la métrica de Error Absoluto Medio (Mean Absolute Error) para supervisar el rendimiento.
2. **Callbacks:** Se definen *callbacks* específicos para el entrenamiento en esta etapa.

- ***Early Stopping*:** Detiene el entrenamiento si la métrica de validación deja de mejorar, lo que ayuda a prevenir el sobreajuste. El valor se estableció en 10 épocas.
- ***Model Checkpoint*:** Guarda el modelo con los mejores resultados (menor MAE en los datos de validación).

3. **Entrenamiento del Modelo:** Se inicia el entrenamiento del modelo. El entrenamiento continúa durante un número máximo de épocas (`MAX_EPOCHS=500`) y se evalúa el rendimiento en cada época utilizando un conjunto de datos de validación (`valid_set`).

3.3.5. Modelos empleados

En la siguiente sección se muestra una breve descripción de cada uno de los distintos modelos implementados. Donde únicamente se han empleado métodos de predicción iterativos de ventanas secuenciales, es decir que por cada ventana de 24 pasos de tiempo, se realizan 24 predicciones desplazadas una hora hacia el futuro.

- **Modelo Dense (Red Neuronal Densa):**
  - Capa de entrada (Input): Esta capa define la forma de entrada de los datos con una ventana de tamaño `window_size` y 4 características.
  - Capa densa (Dense): Con 4 neuronas, función de activación $Relu(z)$ (ver ecuación (2.7)).
  - Capa densa (Dense): Con 2 neuronas, función de activación $Relu(z)$.
  - Capa de salida (Dense): Con 1 neurona, produce la salida final del modelo.
- **Modelo RNN (Red Neuronal Recurrente):**
  - Capa de entrada (Input): Define la forma de entrada de los datos con una ventana de tamaño `window_size` y 4 características.
  - Capa SimpleRNN (SimpleRNN): Con 4 neuronas y la opción `return_sequences=True`, procesa secuencias de datos y devuelve secuencias.
  - Capa SimpleRNN (SimpleRNN): Con 2 neuronas y `return_sequences=True`, procesa las secuencias intermedias.
  - Capa de salida (Dense): Con 1 neurona, produce la salida final.
- **Modelo stateful LSTM (Red Neuronal LSTM):**
  - Capa de entrada (Input): Define la forma de entrada de los datos con una ventana de tamaño `window_size`, 4 características y el modo stateful.

  - Capa LSTM (LSTM): Con 4 neuronas, `return_sequences=True` y estado conservado entre lotes.
  - Capa LSTM (LSTM): Con 2 neuronas, `return_sequences=True` y estado conservado entre lotes.
  - Capa de salida (Dense): Con 1 neurona, produce la salida final.

- **Modelo CNN (Red Neuronal Convolucional):**
  - Capa de entrada (Input): Define la forma de entrada de los datos con una ventana de tamaño `window_size`, 4 características.
  - Capa de convolución 1D (Conv1D): Con 4 filtros, kernel de tamaño 3, stride 1, función de activación $Relu(z)$ y relleno 'causal'.
  - Capa de convolución 1D (Conv1D): Con 2 filtros, kernel de tamaño 3, stride 1, función de activación $Relu(z)$ y relleno 'causal'.
  - Capa de salida (Dense): Con 1 neurona, produce la salida final.

3.4. Etiquetado de datos

El aspecto más crítico de este proyecto es precisamente el etiquetado de los saltos que se producen en el consumo eléctrico, y que en definitiva, contribuirán a predecir, con más precisión, el consumo futuro. La etiqueta, además de los datos ambientales, que posiblemente ayuden a la inteligencia artificial a predecir el futuro, debe indicar el aparato responsable. En este último caso, se proponen cuatro tipos de aparatos:

- Aparatos de consumo basal. Incluye routers, repetidores, el modo stand-by de numerosos aparatos, etc. Todos ellos se agruparan como un único dispositivo por cada fase, y no se tendrán en cuenta para el diseño del sistema. En caso necesario, su valor se restaría antes de hacer los cálculos, y se suma al resultado de la predicción.

- Periódicos. Incluye, principalmente, a los frigoríficos y congeladores de la vivienda. Su etiquetado se puede realizar masivamente con un sencillo script de Python.

- Aparatos de media y alta potencia y consumo bien definido. La identificación se realizará manualmente, con la asistencia de una aplicación Android, que facilitará el etiquetado masivo de datos similares.

- Aparatos de consumo irregular o difícil de determinar. Como ejemplos, tenemos la lavadora y el lavavajillas, e incluso el propio ordenador, pues su consumo varía significativamente entre su uso como editor de texto, o durante la ejecución de TensorFlow.

- Autoetiquetado: Hay dispositivos inteligentes capaces de comunicar su encendido o apagado con MQTT. A pesar de que es el que resulta más fácil de identificar, no se incluyen en este proyecto, ya que los disponibles en la vivienda son precisamente aparatos de muy bajo consumo.

3.4.1. Aplicación Android

La función principal es calibrar la potencia de los aparatos para facilitar el etiquetado de los datos eléctricos. Mediante la aplicación, que se subscribirá al servicio MQTT, se pretende asociar cada salto de potencia a un determinado aparato. Algunas de las características de la aplicación son las siguientes:

- La aplicación debe tener una entrada de texto en la que se indique el nombre del aparato (nuevo, o ya existente), y botones para iniciar y/o finalizar la medición.
- La aplicación debe tener selectores de fase, así como del tipo de medida que se va a realizar (cálculo del encendido o del apagado del aparato).
- La aplicación tendrá una función principal, la medición, que detectará el valor de la potencia antes de cambiar el estado del aparato, y el valor que tendrá después. En ambos casos, los valores de potencia de las fases deben estabilizarse antes y después de la actuación y será necesario establecer umbrales tanto para la estabilización del aparato, como para el propio salto de potencia.
- El valor medido se podrá almacenar, opcionalmente, en la base de datos (en la colección *aparatos*.

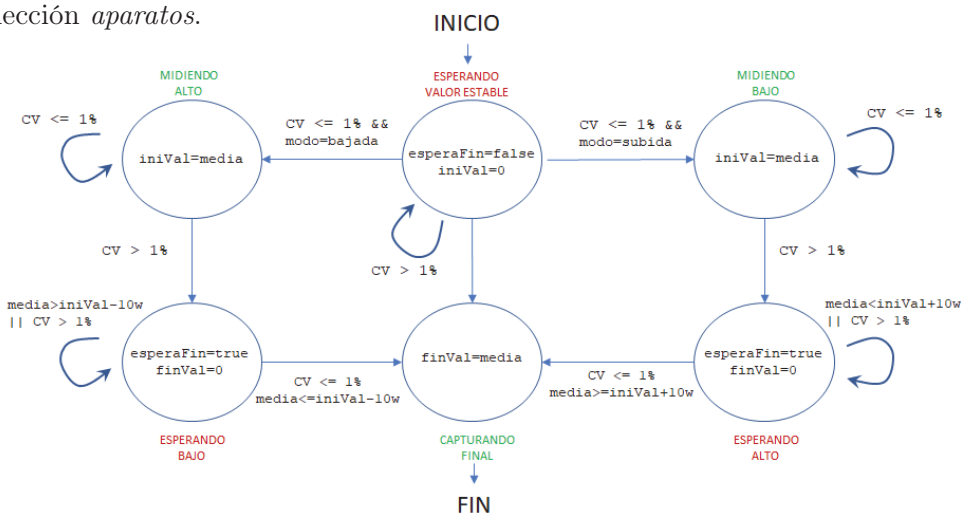


Figura 3.7: Diagrama de la máquina de estados Mealy de la aplicación de medición.

En la figura 3.7 se muestra un esquema (en la forma de una máquina de estados Mealy) de cómo se realizará una medición. En el esquema se ha considerado que el salto mínimo

de potencia del aparato debe ser de 10W, y para la estabilización de las mediciones se ha utilizado un umbral del 1 % para el coeficiente de variación (CV) de las últimas x medidas, siendo CV = desviación estándar/media aritmética.

Tras la calibración del salto de potencia asociado al aparato, la aplicación deberá almacenar esta información en la base de datos, directa o indirectamente. Por motivos de seguridad de la plataforma de almacenamiento utilizada [38], la aplicación Android le encargará al demonioMQTT (véase la sección 3.2.2) que realice la inserción, mediante la publicación de un mensaje MQTT.

Finalmente cabe destacar que la aplicación Android no es estrictamente necesaria para este proyecto, ya que las mediciones se pueden realizar desde un ordenador de sobremesa con un script de Python. Sin embargo, la necesidad de estar junto al interruptor del aparato durante la medición la hacía altamente recomendable.

3.5. Modelo de aprendizaje profundo

En esta sección, se presenta la metodología empleada para el montaje de una red neuronal pensada para la predicción de series temporales. El uso de algoritmos de aprendizaje profundo para realizar este tipo de estimaciones en el tiempo requieren de un enorme volumen de datos correctamente tratados. Como fue mencionado anteriormente, la ambición de este proyecto de construir un modelo solido capaz de predecir el consumo de potencia de la vivienda con un amplio horizonte de tiempo (predecir el consumo en el día o semana siguiente) es incompatible con los tiempos académicos disponibles para la realización de este proyecto. No obstante, se ha desarrollado la infraestructura necesaria para que ha medida que pase el tiempo y se vayan recogiendo más datos, el sistema sea capaz de aumentar su horizonte temporal.

En el momento de redacción de este párrafo, a finales de octubre de 2023, el sistema ha sido capaz de recoger datos (de potencia, ambientales y de otra índole) minuto a minuto desde mediados de septiembre. Se han obtenido del orden de 60000 pasos de tiempo registrados desde ese momento, para cada una de las características del modelo. A priori puede parecer una buena cantidad de datos para poder entrenar los modelos, sin embargo, nada más lejos de la realidad, pues para construir un modelo plenamente funcional, se requerirían varios años para que el modelo internamente aprenda la relación entre cada una de las variables. Únicamente se han recogido datos del inicio otoñal por lo que es prácticamente imposible que el modelo sea capaz de predecir la influencia en el consumo domestico de unas vacaciones, o de un día extremadamente caluroso.

Siendo conscientes de las limitaciones del sistema, a continuación se muestra la metodología empleada para el desarrollo de un modelo de predicción de series temporales que sea capaz de predecir el consumo de la vivienda con un horizonte temporal pequeño, es

decir, construir un sistema capaz de predecir el consumo en la vivienda en los siguientes 15 minutos.

3.5.1. Consulta de datos en MongoDB

Para la consulta de datos de potencia y ambientales en nuestra base de datos MongoDB, se siguió el siguiente proceso:

1. Conexión a MongoDB Atlas utilizando la URI de conexión.
2. Acceso a la base de datos y a las colecciones de interés.
3. Realización de la consulta de los datos y almacenamiento de los mismos.
4. Conversión de los datos a un DataFrame de Pandas para su posterior manipulación y análisis. Los datos quedan almacenados en los DataFrames "`df`" y "`df_amb`"[1].
5. Cierre de la conexión a la base de datos.

Tabla 3.3: Estructura de la DataFrame `df` de los datos de potencia.

id	tiempo	fase0	fase1	fase2	ambiente
6504459b84ff2c7680042049	2023-09-15 11:53:00	217.0	68.0	328.0	NaN
650445d784ff2c768004204c	2023-09-15 11:54:00	212.0	70.0	341.0	NaN
...	...	...	...	...	...
653b72377ddb31e3c20e942c	2023-10-27 08:21:00	358.0	106.0	234.0	653b71c07ddb31e3c20e9424
653b73277ddb31e3c20e945e	2023-10-27 08:22:00	359.0	105.0	232.0	653b71c07ddb31e3c20e9424

Tabla 3.4: Estructura de la DataFrame `df_amb` de los datos ambientales.

id	fechaUTC	festivo	alturaSol	T (ºC)	humedad	personas
6501c8ca8d405bf6e40c7499	2023-09-13 14:35	0	43.9	28.0	66.0	<NA>
6501cb238d405bf6e40c74fb	2023-09-13 14:45	0	42.2	28.0	65.0	<NA>
...	...	...	...	...	...	...
653b6f687ddb31e3c20e93e7	2023-10-27 08:05	0	10.3	19.0	48.0	11
653b71c07ddb31e3c20e9424	2023-10-27 08:15	0	17.4	19.0	49.0	11

Observando las Tablas 3.3 y 3.4 se observa que la dataFrame `df` en la variable **ambiente** muestra el código identificador de la fila correspondiente en la dataFrame `df_amb`, por lo que se dispone a combinar ambas dataFrames en una única en función de las variables "**ambiente**" en `df` y "**id**" en `df_amb`. Se utiliza una combinación de tipo "left join", asegurando que todas las filas de `df` se conserven y solo se añadan las filas coincidentes de `df_amb`. Si no hay una coincidencia para una fila específica en `df`, los valores de `df_amb` para esa fila serán `NaN`.

[1]Los datos ambientales contemplan un código AEMET para identificar el estado del cielo. Esta variable ha sido descartada al no considerarse influyente en el consumo de la vivienda, al no disponer de placas fotovoltaicas o termosolares

3.5.2. Gestión de los datos defectuosos

En la Figura 3.8 se muestra una representación de todos los datos de potencia recopilados desde el 15 de septiembre de 2023. En la Figura se puede observar existen periodos en los que las publicaciones MQTT del sensor electrónico Shelly presentan anomalías. Estas pueden manifestarse como registros de consumo nulo (0 W) o debido a que el dispositivo experimenta fallos y se çongela". Todos estos datos defectuosos serán eliminados y posteriormente tratados.

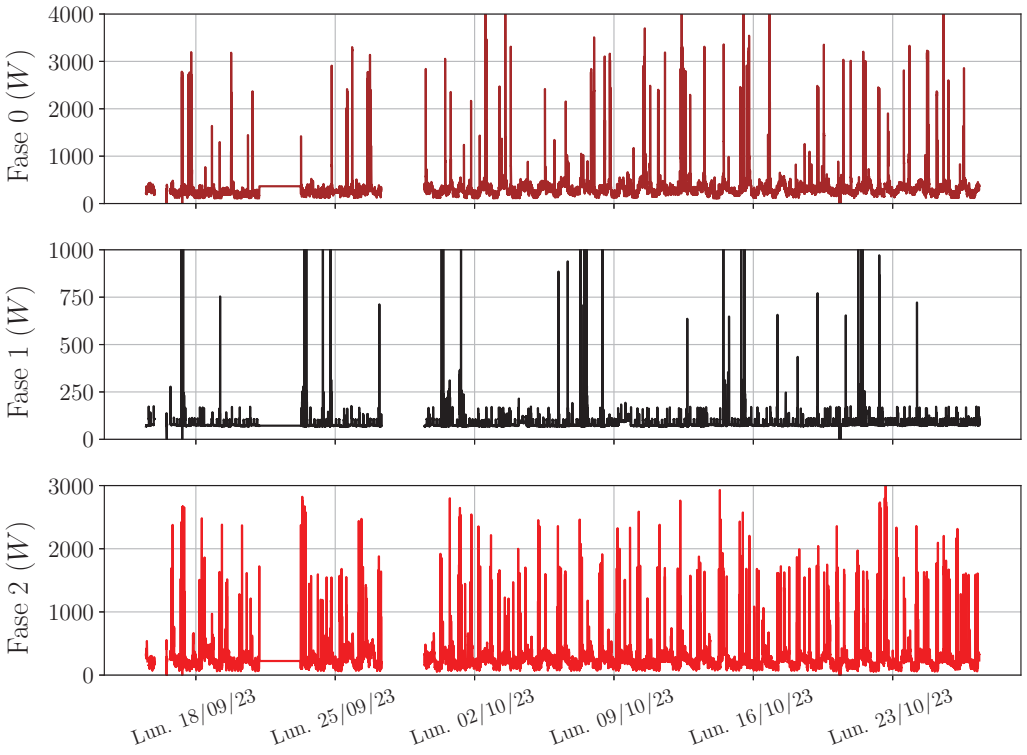


Figura 3.8: Representación de los datos originales de potencia captados por el sistema.

Tabla 3.5: Número de datos ausentes de cada característica en la base de datos.

Característica	Valor	Característica	Valor
tiempo	0	unix	0
fase0	4531	dia	0
fase1	4531	mes	0
fase2	4531	hora	0
festivo	5207	minuto	0
alturaSol	5207	temperatura	5287
humedad	5427	personas	13039
Pasos de tiempo totales: 60384			

De la misma manera, existen numerosos datos ambientales que no han sido recopilados, debido a publicaciones defectuosas, o bien debidas a que, en las primeras estancias de las recogida de datos, no se habían considerado en un inicio. De esta manera el total de datos resultantes se puede resumir en la Tabla 3.5.

El problema de las redes neuronales es que no se pueden alimentar los modelos con datos vacíos por lo que se presentan dos alternativas.

1. Eliminar las filas con datos de defectuosos.
2. Estimar los valores ausentes.

La primera estrategia asegura que todos los valores introducidos son buenos, pero a cambio se sacrifican muchos valores que pueden ser valiosos, a su vez, pueden afectar a la detección de patrones periódicos estacionales. Este enfoque más simplista es el más fácil de abordar pero supone un mayor riesgo.

La segunda estrategia consiste en estimar el valor ausente mediante distintos métodos como podría ser valores medios, interpolaciones o valores más probables entre otros métodos. Con esta estrategia no se sacrifican datos, no obstante se introducen valores falsos que podrían afectar al rendimiento del modelo. Una correcta gestión de estos datos puede ser crucial para el éxito de la red neurona.

Ante la escasez de suficientes datos, se ha optado por la segunda estrategia y a continuación se muestra la estrategia seguida ante los dos grandes grupos de datos de este proyecto. Datos numéricos (potencias, temperatura, humedad...) y datos categóricos (nº de personas, estado del cielo, festivos).

- **Datos numéricos:** Se agrupan los datos del DataFrame `df` según las columnas 'hora' y 'minuto' y luego calcula el valor promedio de cada columna numérica para cada combinación única de 'hora' y 'minuto'. En la Figura 3.9 se muestra cual sería el valor un día promedio de las variables numéricas. Si alguna celda en las columnas especificadas (como 'fase0', 'fase1', etc.) tiene un valor ausente (NaN), este valor será reemplazado por el valor medio correspondiente.
- **Datos categóricos:** Para los datos categóricos se han realizado distintas gestiones
  - `df[personas]`: Representa un número binario de 4 bits. por ejemplo el número 13 (1101) simboliza que todas las personas menos la *persona3* se encuentran en la vivienda. Para tratar mejor estos datos, se establece una nueva característica por cada miembro que indique si se encuentra o no en casa. Ante la ausencia de datos, se supondrá que la persona se encontraba en casa en ese momento.
  - `df[festivos]`: Esta característica ante la ausencia de datos registrados, se supondrá que el instante de tiempo es no festivo.

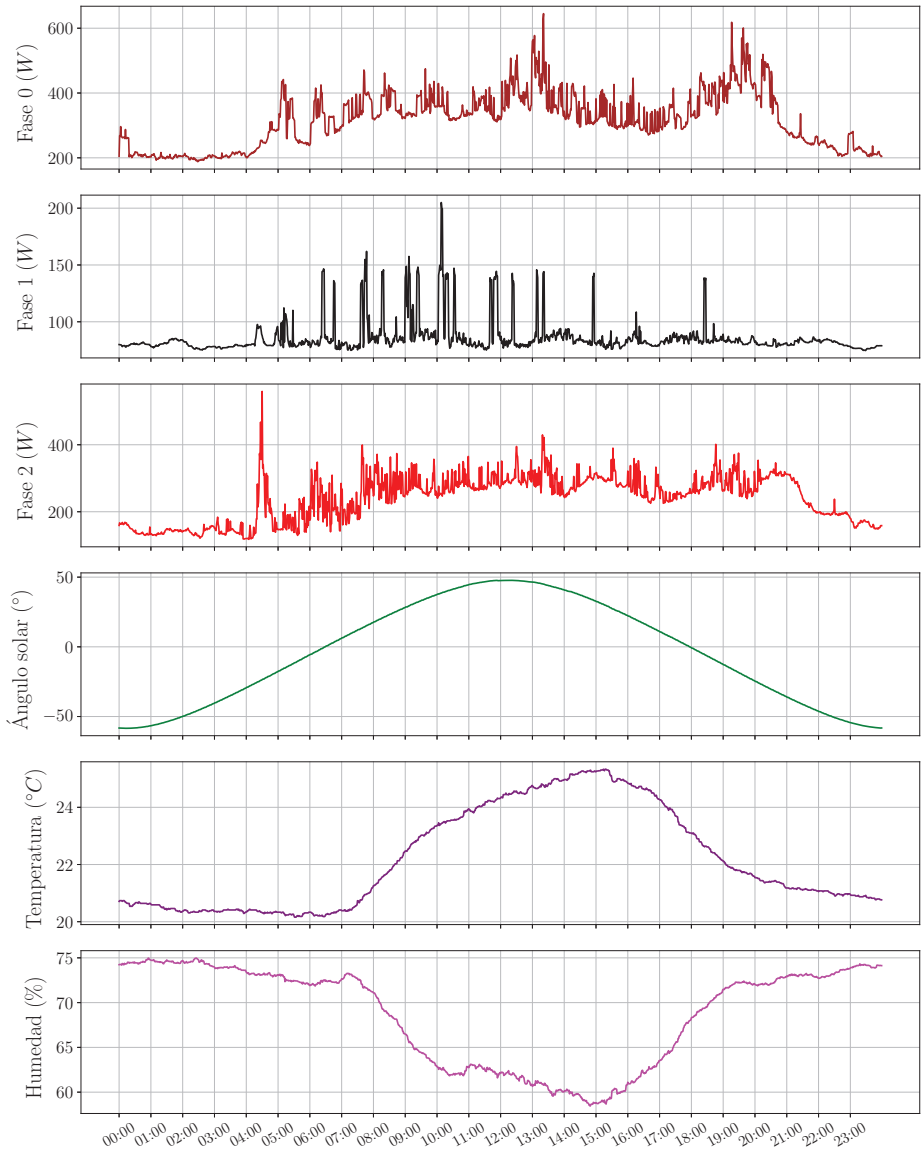


Figura 3.9: Valores diarios promedios de cada variable numérica.

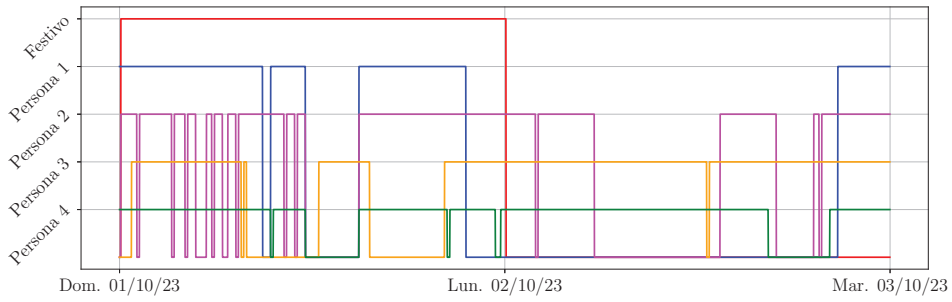


Figura 3.10: Representación temporal de los datos categóricos.

3.5.3. Creación del conjunto de datos definitivos

Una vez realizada la gestión de los datos perdidos, se incorporan a los datos señales temporales que tienen el objetivo de detectar las frecuencias correspondientes a los minutos, horas, días, semanas y años. Estas señales se pueden obtener con las siguientes expresiones:

$$\begin{aligned}
\texttt{df[hour sen]} &= \text{sen}(2\pi \cdot \texttt{df[unix]}/60),, \\
\texttt{df[hour cos]} &= \cos(2\pi \cdot \texttt{df[unix]}/60 \cdot 60), \\
\texttt{df[week sen]} &= \text{sen}(2\pi \cdot \texttt{df[unix]}/60 \cdot 60), \\
\texttt{df[week cos]} &= \cos(2\pi \cdot \texttt{df[unix]}/60 \cdot 60 \cdot 24), \\
\texttt{df[year sen]} &= \text{sen}(2\pi \cdot \texttt{df[unix]}/60 \cdot 60 \cdot 24 \cdot 365), \\
\texttt{df[year cos]} &= \cos(2\pi \cdot \texttt{df[unix]}/60 \cdot 60 \cdot 24 \cdot 365).
\end{aligned} \tag{3.2}$$

Donde `df[unix]` representa el instante de tiempo en segundos registrado desde el 1 de enero de 1970.

Al examinar las Figuras 3.8 y 3.9, es evidente que los datos muestran una variabilidad significativa o "ruido". Esta variabilidad es particularmente prominente en los datos relacionados con la potencia, lo cual se puede atribuir a la naturaleza intrínsecamente aleatoria de ciertos consumos de energía. Sin embargo, es crucial destacar que este "ruido" no debe ser descartado como irrelevantes o errores; en realidad, es información valiosa. La razón es que, dentro de estas fluctuaciones, podemos identificar patrones de consumo específicos, como picos de consumo durante la hora de cocina o patrones cíclicos relacionados con el funcionamiento de electrodomésticos como los frigoríficos. No obstante, para la predicción de series temporales con tan pocos datos registrados (un mes únicamente) este ruido impide lograr un resultado mínimamente satisfactorio, por lo que se procede a el establecimiento de un filtro Gaussiano [39] con un valor de $\sigma = 20$ que se encargue de atenuar las señales el cual viene dado por la siguiente expresión:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}, \tag{3.3}$$

donde:

- $G(x)$ es el valor del filtro Gaussiano en el punto x.
- σ es la desviación estándar del filtro, y en este caso es 20.

Un filtro Gaussiano es una herramienta que permite suavizar o desenfocar una señal (o imagen). Lo hace asignando mayor peso a los datos cercanos al punto central y menos peso a los datos más lejanos, siguiendo una distribución normal o gaussiana. La fórmula dada anteriormente describe matemáticamente este filtro.

En la fórmula, x es la distancia desde el centro del filtro. La desviación estándar, σ, controla el ancho del filtro. Un valor mayor de σ resultará en un suavizado más pronunciado. En este caso, se ha seleccionado $\sigma = 20$ para lograr un grado específico de suavizado.

El exponente negativo asegura que los valores disminuyan a medida que nos alejamos del centro, y el factor multiplicativo al principio normaliza el filtro para que su área total sea 1. En práctica, esto significa que el filtro no alterará el "nivel general" de la señal, solo su detalle. En la Figura 3.11 se muestran las señales de potencias una vez se han aplicado el filtro.

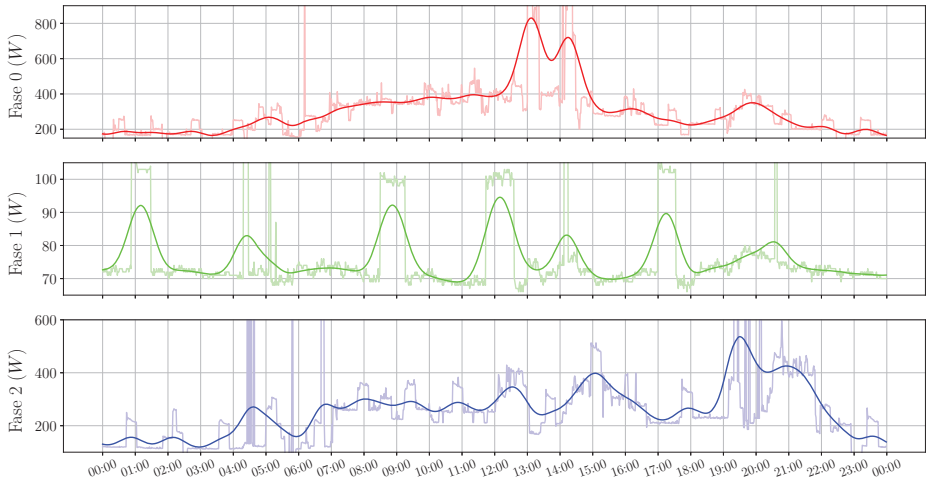


Figura 3.11: Representación de las señales de potencia filtradas el día lunes 2 de octubre de 2023.

3.5.4. Segmentación y normalización

Durante el proceso de preparación de datos, se dividió el conjunto de datos en tres subconjuntos: 70 % para entrenamiento, 25 % para validación y 5 % para pruebas.

Posteriormente, se llevó a cabo una normalización utilizando el método *Min-Max*. Esta técnica ajusta todos los valores numéricos en el conjunto de datos al intervalo de [0, 1]. La fórmula para la normalización *Min-Max* es:

$$\texttt{df\_norm} = \frac{\texttt{df} - \texttt{min(train\_df)}}{\texttt{max(train\_df)} - \texttt{min(train\_df)}}, \tag{3.4}$$

donde `train_df`, `max(train_df)` y `min(train_df)` es el conjunto de datos de entrenamiento y sus valores máximos y mínimos, de forma que todas las variables se normalizan en todos los pasos de tiempo en función de sus propios valores extremos .

3.5.5. Creación de ventanas temporales

Para que el modelo de aprendizaje profundo anticipe la potencia que se consumirá en el futuro, requiere datos recientes. Por ejemplo, si queremos predecir el consumo de los próximos 20 minutos, debemos proporcionarle al modelo la información de los últimos 40 minutos. Para que se realice esta tarea con la precisión deseada, es necesario preparar los datos para que el entrenamiento se realice de acuerdo a este formato, por lo que será necesario una creación de ventanas temporales.

Un buen modelo de predicción será aquel en el que se logre prever un extenso horizonte temporal, tal como las próximas 24 horas o incluso una semana, tal como se mencionó anteriormente. Sin embargo, debe destacarse que, en cualquier modelo, una mayor precisión tiende a ser observada cuando la predicción se encuentra más próxima al momento actual.

Los modelos de predicción meteorológica que se utilizan a diario son ejemplos claros de este principio. Gracias a años de datos acumulados y a tecnologías avanzadas, se consigue prever el clima con hasta una semana de anticipación o más. No obstante, si solo se contara con registros del clima de unas pocas semanas, se vería limitada la capacidad de proyección a períodos de tiempo mucho más cortos.

De manera similar, en el caso de contar solamente con 60,000 minutos de datos registrados, una cifra significativamente menor en comparación con los vastos registros de modelos meteorológicos, se observa que el horizonte temporal de predicción se reduce considerablemente, limitándose a apenas 20 minutos.

Para la preparación de los datos de entrenamiento, se han creado ventanas temporales de 60 minutos (`window_size=60`), de los cuales 40 minutos serán tomados como entradas (`input_width=40`) y 20 minutos como etiquetas (`label_width=20`) que serán comparadas con las predicciones que realice el modelo, las etiquetas están desfasadas al futuro respecto a los datos de entrada unos 20 pasos de tiempo (`shift=20`), este es el funcionamiento de una ventana, veamos como se agrupan para todos los datos disponibles para la validación y el entrenamiento:

1. Preparación de Datos: Convertir los datos del DataFrame de Panda en un formato adecuado (En una matriz Numpy de float32).

2. Creación de Series Temporales: Se toman secuencias de `window_size` datos, la siguiente secuencia es consecutiva al tener un `stride=1`, por lo que si la primera secuencia va desde $t = 0$ a $t = 59$, la segunda va desde $t = 1$ a $t = 60$.

3. Shuffling y Batch Size: Baraja el conjunto de ventanas aleatoriamente y establece un lote de ventanas de tamaño `batch_size=64`.

4. División de la Ventana: Dividir cada ventana en datos de entrada y etiquetas.

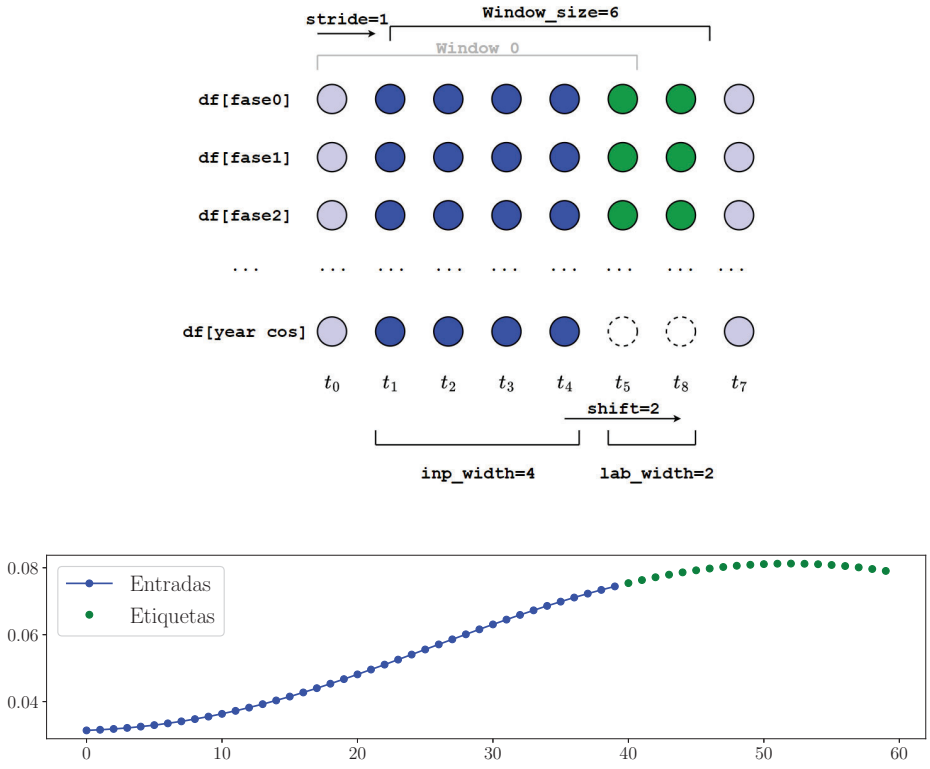


Figura 3.12: Representación del funcionamiento de las ventanas temporales.

3.5.6. Diseño de la red neuronal

La arquitectura de la red neuronal propuesta se construye con el objetivo principal de que las capas LSTM puedan recordar y aprender de los pasos de tiempo anteriores, mientras que las capas Dense se encargan de construir las señales de salida.

El diseño específico de la red es el siguiente:

- **Capa de Entrada**: La red comienza con una capa de entrada diseñada para recibir datos con una forma específica, que corresponde a un total de 40 pasos de tiempo y un número determinado de características (19 en total).
- **Capas LSTM**: A continuación, se incorporan dos capas LSTM consecutivas, ambas con 1024 neuronas. La primera capa LSTM devuelve secuencias, es decir, produce una salida para cada paso de tiempo en la entrada, lo cual es esencial para alimentar la siguiente capa LSTM. La segunda capa LSTM no devuelve secuencias, proporcionando una salida final basada en todos los pasos de tiempo de entrada.
- **Capa Dense**: Tras las capas LSTM, se introduce una capa Dense (totalmente conectada) con `20*3` neuronas y una función de activación ReLU. Esta capa tiene la

responsabilidad de procesar la información proporcionada por las capas LSTM y comenzar la construcción de las señales de salida.

- **Capa de Reconfiguración**: Por último, se aplica una capa de reconfiguración para ajustar las dimensiones de la salida a la forma deseada, que es de `OUT_STEPS` pasos de tiempo con 3 características por paso.

```
1
inputs = keras.Input(shape=(IN_STEPS, ventana.example[0].shape[2]), name='
    Input')
3 x = keras.layers.LSTM(1024, return_sequences=True)(inputs)
x = keras.layers.LSTM(1024, return_sequences=False)(x)
5 x = keras.layers.Dense(OUT_STEPS*3, activation='relu')(x)
outputs = keras.layers.Reshape([OUT_STEPS, 3])(x)
7
model = keras.Model(inputs=inputs, outputs=outputs, name='model')
```

Código 3.4: Configuración del modelo en TensorFlow

3.5.7. Establecimiento del compilador, función de pérdida y algoritmo optimizador

Una vez definida la arquitectura de la red neuronal, es fundamental establecer el proceso de compilación del modelo, especificando la función de pérdida y el algoritmo optimizador que guiarán el proceso de entrenamiento.

- **Función de Pérdida**: Se utiliza la función de pérdida de Error Absoluto Medio (*Mean Absolute Error*). Esta función cuantifica la magnitud de los errores entre las predicciones del modelo y las verdaderas etiquetas.
- **Optimizador**: Se elige el algoritmo de optimización Adam.
- **Métricas de Evaluación**: Al igual que la función de pérdida, se utiliza el Error Absoluto Medio como métrica de evaluación para monitorear el desempeño del modelo durante el entrenamiento y la validación.
- **Callbacks**: Se introduce un mecanismo de detención temprana (*Early Stopping*) con una paciencia de 10 épocas. Esto implica que si durante 10 épocas consecutivas no se observa mejora en la métrica de validación, el entrenamiento se detendrá, evitando posibles sobreajustes y ahorrando tiempo computacional.

Con todo configurado, se procede al entrenamiento del modelo utilizando el conjunto de datos de entrenamiento y validación. Se establece un número máximo de 500 épocas para el entrenamiento.

```
2
model.compile(loss=keras.losses.MeanAbsoluteError(),
4                  optimizer=tf.optimizers.Adam(),
                   metrics=[tf.metrics.MeanAbsoluteError()])
6
early_stopping = keras.callbacks.EarlyStopping(patience=10)
8
history = model.fit(
10     ventana.train,
       epochs=500,
12     validation_data=ventana.val,
       callbacks=[early_stopping],
```

Código 3.5: Configuración de los ajustes de entrenamiento

3.6. Validación

La validación del sistema consistirá en evaluar su capacidad para predecir adecuadamente el consumo de electricidad en la vivienda de los próximos días, semana o meses. Sin embargo, debido a la falta de tiempo, no será posible realizar una validación exhaustiva y con largos periodos de tiempo.

Por lo tanto, la validación se centrará en evaluar el rendimiento del sistema en un período de tiempo limitado, como un día o una semana. Para ello, se utilizarán una serie de indicadores, como curvas de consumo, correlación entre los datos históricos y las predicciones, y acumulados por minuto, hora, etc. También se utilizará la propia información que ofrezca `TensorFlow` respecto a la calidad de su entrenamiento.

Estos últimos indicadores permitirán evaluar la precisión de las predicciones del sistema, así como su capacidad para detectar patrones en el consumo de electricidad.

Capítulo 4

Desarrollo y resultados

4.1. Recogida de datos

Desafortunadamente, las limitaciones temporales de un proyecto de estas características impiden que el proceso de recogida de datos tenga una duración suficiente como para recopilar la mínima información que pueda llevarlo al mejor término. Sería necesario, por ejemplo, que el sistema fuera capaz de reconocer todas las estaciones, por lo que, al menos, necesitaría todo un año de captura de datos. Sin embargo, y gracias a que se empezó el proceso de captura de datos en la fase más temprana del proyecto, tenemos información suficiente para extraer interesantes conclusiones que serán de gran utilidad para el posterior entrenamiento del sistema.

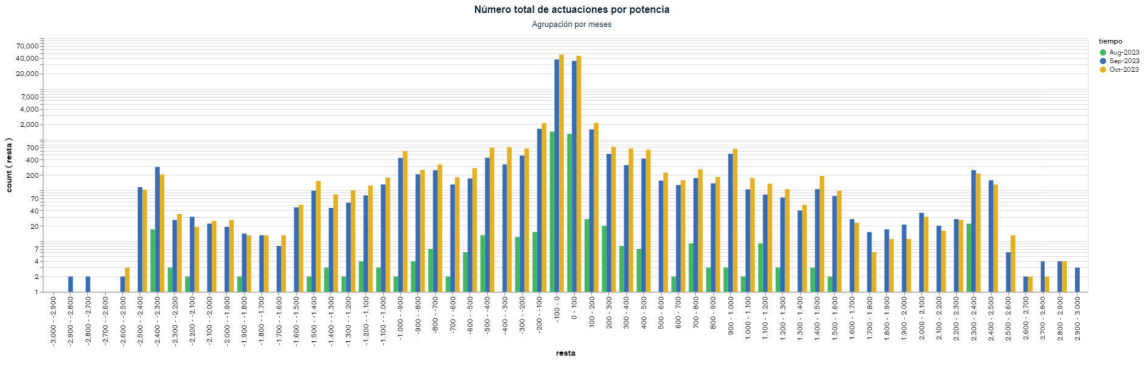


Figura 4.1: Número de saltos de potencias registrados.

En el momento en el que se escriben estas líneas, la aplicación lleva apenas dos meses recogiendo información de saltos de potencias (con alrededor de 195,000 mediciones, lo que supone, de media, una actuación cada 22 segundos). De ellas, más de la mitad de las actuaciones se corresponden con saltos de potencia inferiores a 20W que en su inmensa mayoría no se deben al encendido o apagado de un aparato (obviamente), sino a fluctuaciones en el

funcionamiento de algunos de ellos. Como ejemplo, el consumo de los ordenadores personales varía significativamente según la operación que realiza, o una pantalla de televisión consume más o menos según el brillo de la escena que emite.

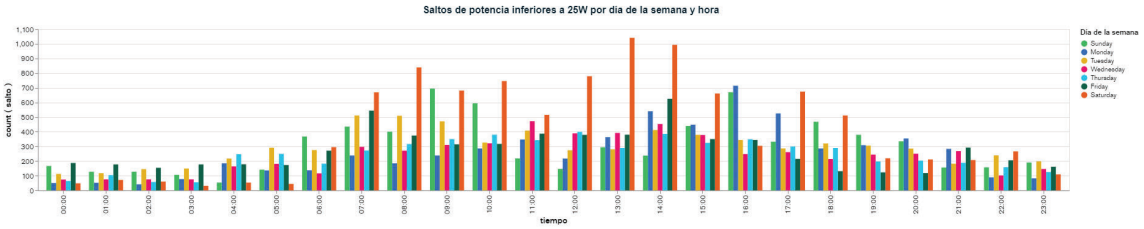


Figura 4.2: Número de saltos de potencias inferiores a 25W registrados por el sistema.

Precisamente fueron los primeros análisis de los saltos de potencias lo que determinó que una parte importante del consumo del hogar no podría claramente etiquetarse, sino que debía ser la propia Inteligencia Artificial la que descifrara el comportamiento. Por ello, se empezaron a recoger datos periódicos que facilitaran el análisis de series temporales. La captura de estos datos periódicos se realizó almacenando los valores en el segundo cero de cada minuto, y en el momento de redacción de este párrafo se han recogido 55,000 muestras.

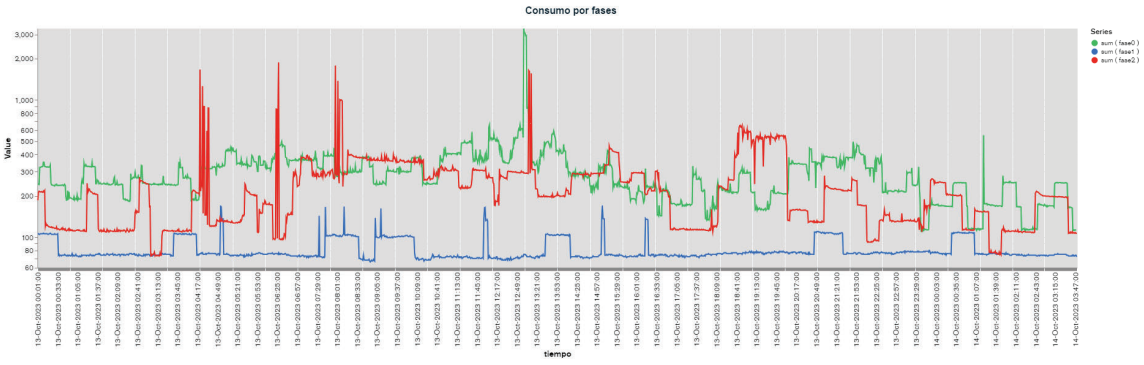
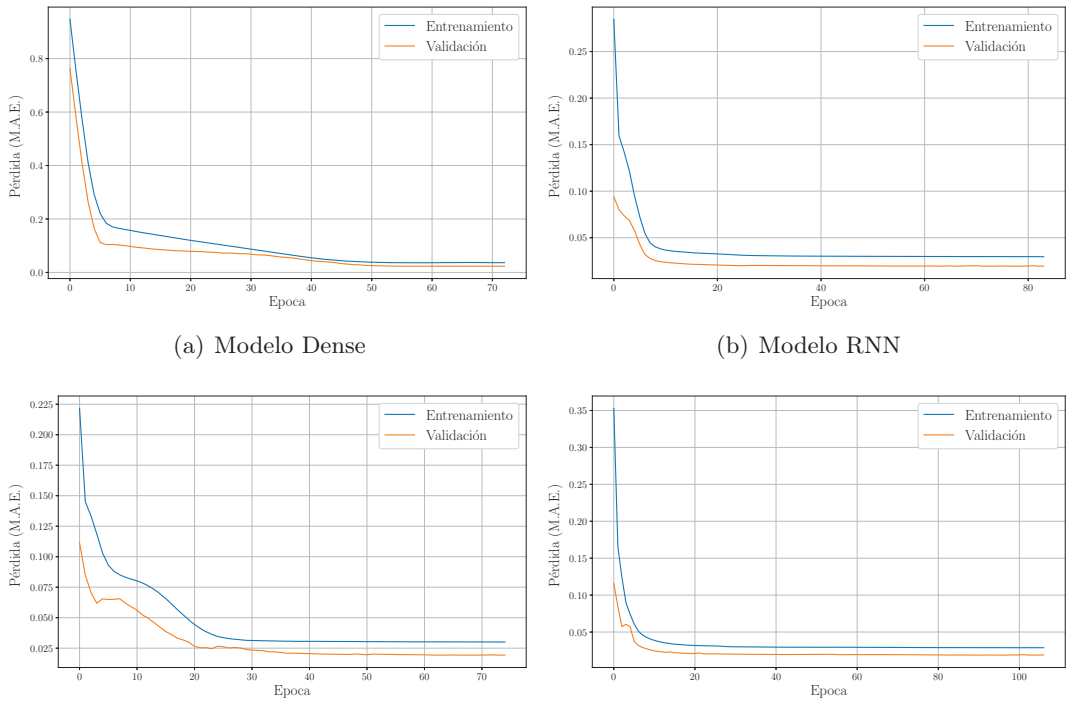


Figura 4.3: Potencia de cada fase entre los días 13 a 15 de octubre de 2023 (viernes a domingo).

Finalmente, la recogida de datos ambientales, cada 10 minutos, ha permitido recoger alrededor de 5,300 muestras. Cada una de estas muestras lleva un identificador (asignado automáticamente por MongoDB) y todos los datos de potencia (tanto periódicos como de salto) almacenan el identificador del último dato ambiental recogido, siempre que el intervalo de tiempo transcurrido sea inferior a una hora.

4.2. Previsión del precio de la electricidad

Una vez implementados los modelos descritos en el Apartado 3.3 se han sometido a el proceso de entrenamiento. Durante dicho proceso se ha anotado el valor de la función de pérdida de las predicciones de entrenamiento y validación con respecto a las etiquetas durante cada iteración para poder estudiar la convergencia de los modelos y estudiar el proceso del descenso del gradiente. En la Figura 4.4 se puede observar cada uno de los procesos en los cuatro modelos empleados.



(a) Modelo Dense

(b) Modelo RNN

(c) Modelo LSTM

(d) Modelo CNN

Figura 4.4: Proceso de entrenamiento de los diferentes modelos para la predicción del precio marginal de la Electricidad en España.

En la Figura se observa como el valor del error absoluto medio se estabiliza aproximadamente en el valor normalizado de 0.02 en todos los modelos (aproximadamente un error medio de 6€/MWh con respecto a los valores reales) en aproximadamente 100 épocas o iteraciones al conjunto de datos de entrenamiento. la mayor parte de estas épocas muestran el valor del error absoluto medio ya estabilizado, por lo que es un claro indicio de que los datos se están sobreajustando, por lo que sería conveniente reducir el *early stopping* del modelo. No obstante, estos modelos fueron diseñados únicamente por el autor para familiarizarse con los métodos de aprendizaje profundo y estudiar el comportamiento de los diferentes modelos, por lo que, al tratarse de modelos no relacionados con la temática central de este proyecto de predecir el consumo de una vivienda, dichos modelos se

propondrán ser mejorados en lineas futuras de trabajo.

Una vez entrenado todos los modelos y realizado predicciones en los datos de validación se calcula el error absoluto medio de las predicciones frente a las etiquetas, obteniendo los siguientes resultados:

Tabla 4.1: Valores de MAE en los datos de validación para diferentes modelos.

Modelo	MAE
Dense	0.0232
RNN	0.0194
LSTM	0.0193
CNN	0.0187

Se observa que los modelos los modelos **RNN**, **LSTM** y **CNN** presentan los mejores resultados, especialmente el modelo **CNN**. Cabe recalcar que estos modelos presentan a priori tan buenos resultados puesto que son solo capaces de analizar una hora hacia el futuro, en los datos de validación para realizar cada predicción usa datos reales como entrada. Si se utilizasen las predicciones a futuro empleando sus propias predicciones como entradas, el error cometido se iría acumulando. En la Figura 4.5 se muestran gráficamente las predicciones en las primeras 24 horas de validación, donde para predecir cada hora, se introducen los valores de precio reales normalizado de las 24 horas anteriores como entradas.

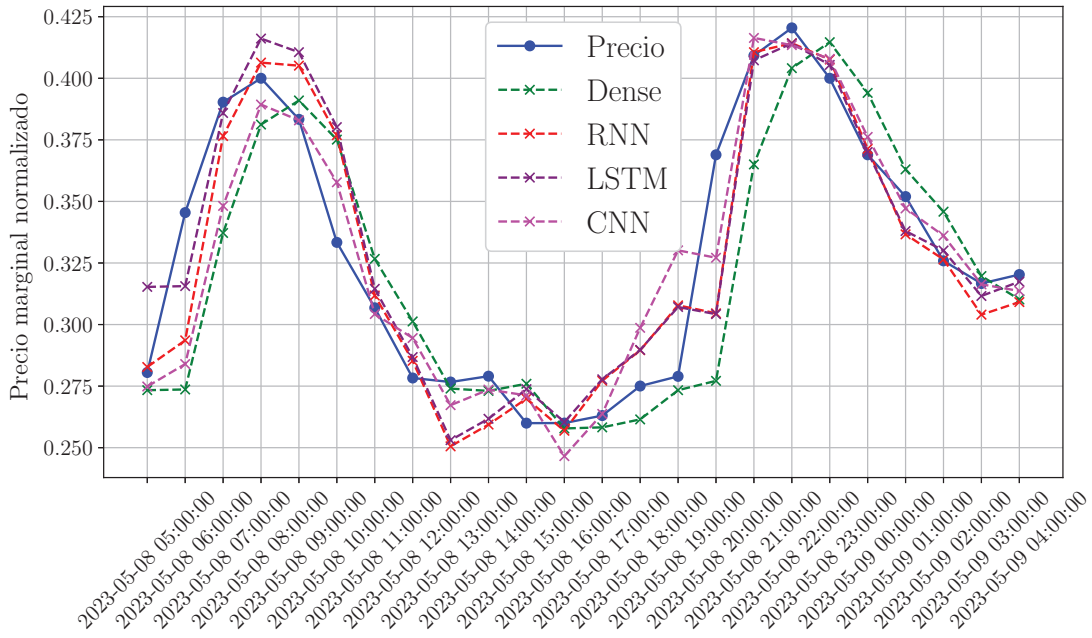


Figura 4.5: Representación de las predicciones las 24 primeras horas de los datos de validación frente a los valores reales del precio de la electricidad normalizado..

Observando la Figura 4.5, se puede apreciar que el modelo **Dense** prácticamente imita los valores del precio normalizado de una hora anterior, mientras que el resto de modelos realizan predicciones similares.

Estos resultados no son del todo representativos por lo que a continuación se muestran los resultados, esta vez en los datos de prueba, donde se alimenta al modelo con sus propias predicciones.

Tabla 4.2: Valores de MAE en el conjunto de prueba para diferentes modelos.

Modelo	MAE
Dense	0.0900
RNN	0.0806
LSTM	0.0414
CNN	0.0488

En la Tabla 4.2 se puede observar como el error aumenta considerablemente con respecto a los errores cometidos en la validación, esto se debe exclusivamente a la acumulación del error al introducir sus propias predicciones como entradas. En la Figura 4.6 se muestran gráficamente las predicciones en el día de prueba frente al valor real del precio normalizado ese día.

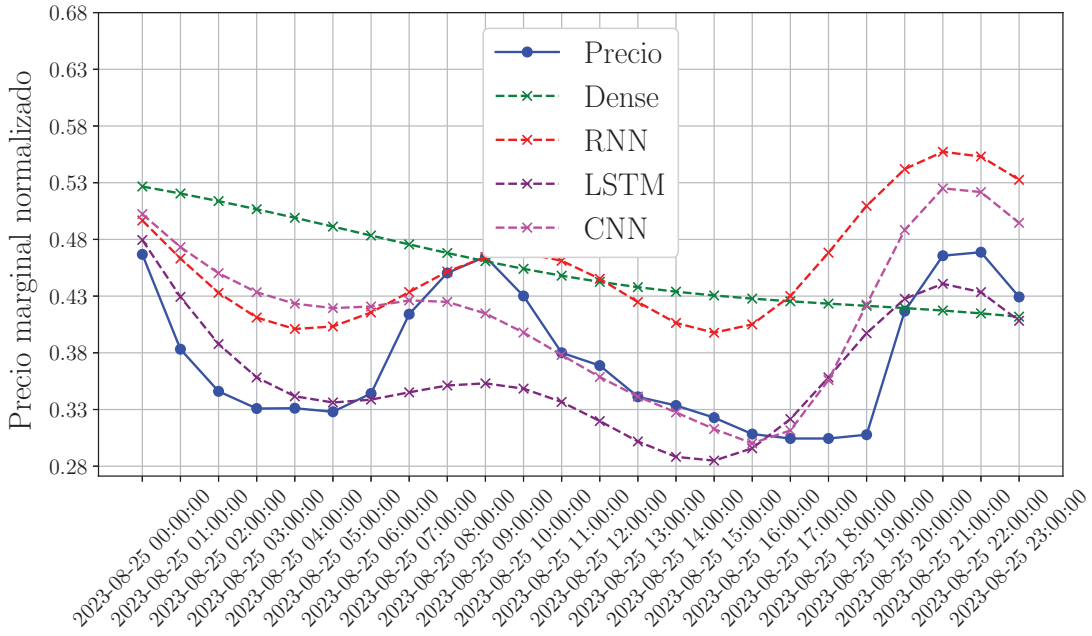


Figura 4.6: Representación de las predicciones de los datos de prueba frente a los valores reales del precio de la electricidad normalizado.

Se observa en la Figura 4.6 varios hechos a destacar. En primer lugar, el modelo **Dense**

sometido a un proceso iterativo para poder predecir 24 horas desconocidas para el modelo, es incapaz de predecir valores para un horizonte temporal mayor a una hora. El resto de modelo de modelos si son capaces de predecir la existencia de horas valles y punta, destacando principalmente el modelo **CNN** el cual ha conseguido predecir las horas correspondientes al medio día con excelente precisión, y el modelo **LSTM** el cual ha conseguido predecir de manera más general el comportamiento de la curva con mayor precisión.

4.3. Etiquetado de los datos

Una de las propuestas iniciales de este proyecto consistía en etiquetar exhaustivamente los datos de potencia no sólo con la información ambiental presente, sino también mediante la identificación del aparato o los aparatos responsables del consumo. Sin embargo, como ya se indicó en la sección 4.1, desde las primeras fases del proyecto ya sabíamos que una gran parte del consumo se debía a algunos aparatos con un elevado número de oscilaciones imprevisibles. Y aunque estas oscilaciones difícilmente se pueden etiquetar, también se sabe que hay, por otro lado, una cierta cantidad de valores de potencias registrados que, sin ser muy elevados en número, sí que son los responsables de una parte elevada del consumo.

Sirvan como ejemplo los datos del último mes. En la vivienda se consumieron, entre el 24 de septiembre y el 23 de octubre, un total de 465kWh de los cuales, prácticamente la mitad (220kWh) se debe a los aparatos de consumo constante (160kWh) o periódicos (60kWh), y que son fácilmente etiquetables. Hay aproximadamente un 20-25 % del consumo (esta es sólo una estimación previa *a grosso modo*) que se corresponde con aparatos que también son fácilmente identificables y etiquetables. Entre ellos, por ejemplo, hay una tostadora (755W), un microondas (1270W), y también se incluirían la plancha, el horno, o los quemadores de la cocina de inducción, que alternan encendidos y apagados de periodicidad variable, y a priori desconocida, pero de potencias bien definidas.

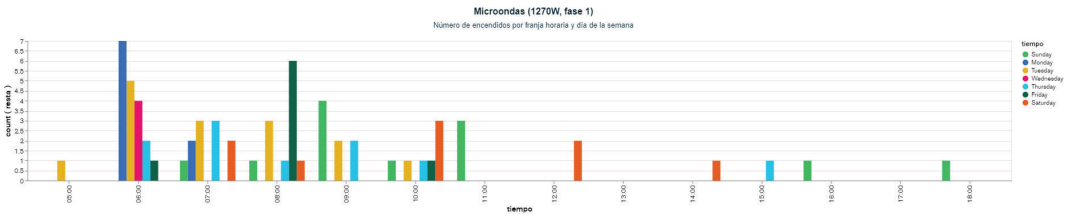


Figura 4.7: Número de saltos de potencia de $1270W \pm 1\,\%$ (presumiblemente debidos al microondas) registrados en un mes, y agrupados por día de la semana y hora UTC.

En definitiva, quedaría una especie de "rumor de fondo" en el consumo, que puede llegar a ser el responsable de alrededor de en torno al 25 % del consumo del hogar, y que se debe a ciertos aparatos que podríamos denominar *irregulares*. Entre ellos se incluirían la lavadora y el lavavajillas, los distintos ordenadores personales, y también los televisores.

La consecuencia es que el etiquetado de los aparatos debería suponer tan solo una cierta ayuda para la Inteligencia Artificial (a la hora de predecir el consumo), pero la mayor parte del sistema de aprendizaje será autónomo. En cualquier caso, el etiquetado de los aparatos no deja de ser extremadamente útil, pues una posible (e inmediata) ampliación de este proyecto incluirá un mecanismo de retroalimentación mediante el cual el sistema informará al usuario sobre los aparatos que hayan sido responsables del consumo en un cierto periodo de tiempo.

4.3.1. Desarrollo de la aplicación Android

Para el desarrollo de la aplicación Android se ha utilizado el lenguaje de programación Kotlin y el *framework* Compose para el desarrollo de la interfaz de usuario. Se ha programado en Android Studio. Tanto el lenguaje de programación como el de diseño de la interfaz de usuario han sido elegidos (frente a los tradicionales lenguajes Java y los Views Xml de diseño) considerando las más actuales tendencias en programación Android. El diseño de la aplicación es simple y minimalista, pues su único objetivo es implementar, en una herramienta móvil, el algoritmo de la figura 3.7.

4.3.2. Proceso de etiquetado

El etiquetado se ha llevado a cabo mediante el siguiente procedimiento:

1. Las potencias de los aparatos periódicos, como frigoríficos, congeladores, etc. (se han detectado 5 de ellos), y que se identifican claramente en las curvas de potencia (como en la Figura 4.3), se han identificado con la ayuda de un script de python (en el notebook anexo buscaPeriodicos.ipynb) dentro del cual hay una función que busca un salto de subida al que le corresponde una bajada. Con ayuda de esta función y las correspondientes curvas de potencia se ha determinado que, por ejemplo, el frigorífico de la cocina consume alrededor de 70W y está encendido durante alrededor de 15 minutos (en el último mes). Con esos valores se han podido emparejar encendidos con sus correspondientes apagados. Los valores se han almacenado manualmente en MongoDB.

2. Cada una de la potencias basales de cada fase se ha calculado utilizando los datos de una madrugada en lo que no se detecte otro consumo, y evitando los tramos de encendidos de los frigoríficos. Se ha calculado la media y se ha almacenado manualmente en MongoDB.

3. Con la ayuda de la aplicación Android (Figura 4.8) se ha identificado la potencia de varios aparatos, tanto en el proceso de encendido como en el de apagado (que no tiene porqué ser la misma). La aplicación publica, con el tópico prefijo/demonio/inserta,

los valores de potencia, fase y nombre, en formato *json*, mientras que el demonio le asigna el identificador numérico, y un tipo de aparato (4) que se corresponde con aparatos de potencia bien definida. Por motivos de privacidad no se muestra el listado, pero sirva el microondas, de potencia 1270W en la fase 1 como ejemplo (Figura 4.7). Un sencillo script de Python busca entonces todos los saltos de potencias en la base de datos MongoDB con tres filtros: que su potencia y fase sean las del aparato (con un margen de error), y que no tengan etiqueta previa.

Cabe destacar que la aplicación Android se ha utilizado también para corroborar las medidas de los aparatos periódicos del primer párrafo de esta sección.

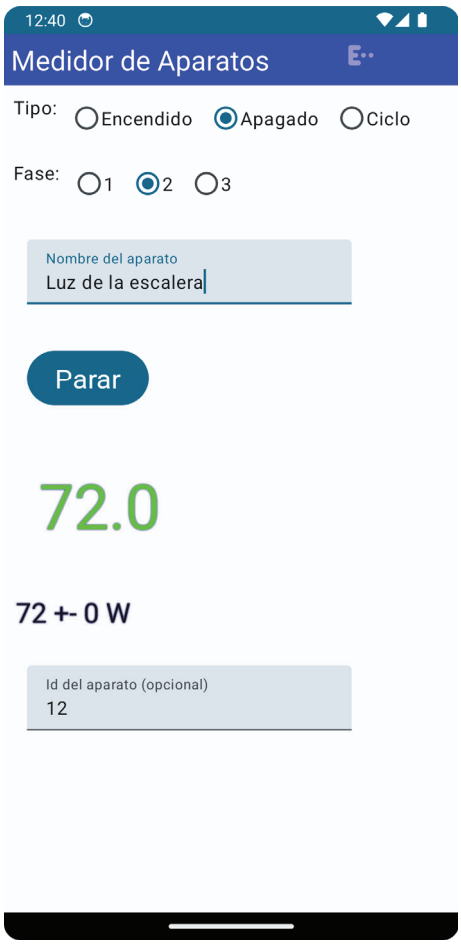


Figura 4.8: Actividad principal de la aplicación Android.

4.4. Modelo de aprendizaje profundo

4.4.1. Estimación del consumo mediante el uso de métodos directos de predicción de series temporales

El entrenamiento del modelo descrito en la Sección 3.5 se ha basado en un método de predicción directa donde, ante una ventana de datos del pasado introducida como entrada (40 minutos), se obtiene una ventana de predicciones de salida (20 minutos) (Véanse los Apartados 2.3.1 y 3.5.3). Como resultado, se tienen las métricas del proceso de entrenamiento mostradas en la Figura 4.9. Se aprecia que el modelo converge tras 23 épocas, durante un tiempo de computación de 5 minutos y 45 segundos.

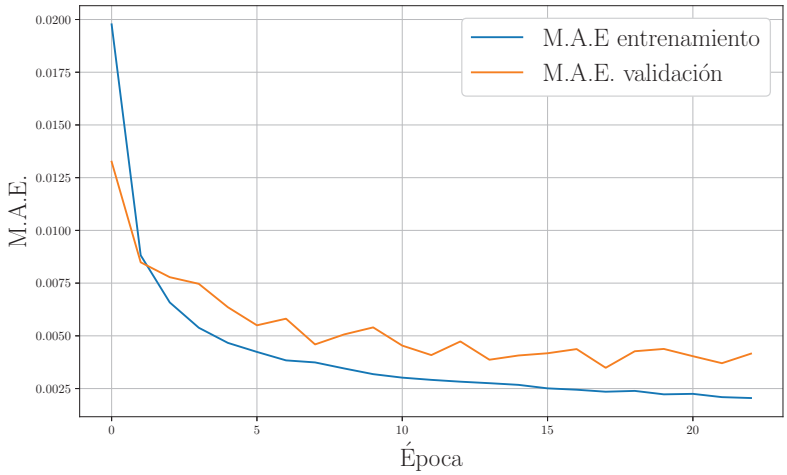


Figura 4.9: Métricas del error medio absoluto (MAE) durante el proceso de entrenamiento.

Tabla 4.3: Error absoluto medio normalizado de cada conjunto de datos.

Conjunto	MAE
Entrenamiento	0.0018
Validación	0.0042
Prueba	0.0072

Realizando una evaluación de los 3 conjuntos de datos de entrenamiento, validación y test, en la Tabla 4.3 se muestran los errores absolutos medios obtenidos tras realizar las predicciones de cada uno de los conjuntos. Observando los resultados, se aprecia que el error medio absoluto en los datos de validación y prueba son aproximadamente el doble y el triple mayores respectivamente en comparación con los datos de entrenamiento.

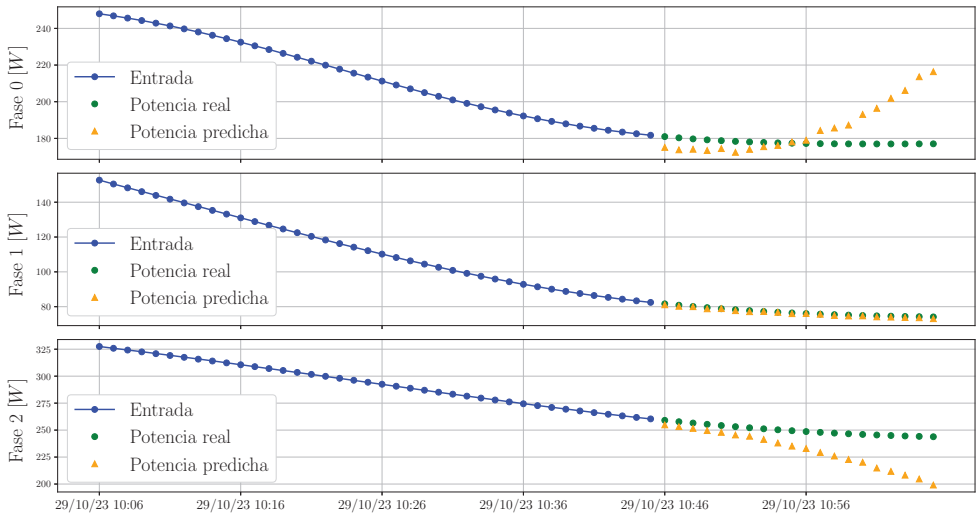


Figura 4.10: Predicción del consumo de la vivienda en los próximos 20 minutos calculado el domingo 29 de octubre a las 10:46 de la mañana.

Este modelo solamente es capaz de predecir los próximos 20 minutos futuros, por tanto, a continuación, se muestra en la Figura 4.10 la predicción del consumo en la vivienda para los próximos 20 minutos desde el momento de la redacción de este mismo párrafo, domingo 29 de octubre a las 10:46 de la mañana. En la Figura se observa como el modelo predice con relativa precisión cual va a ser la potencia consumida en los primeros 10 minutos futuros siendo, a medida que se aleja de los datos de entrada, cada vez menos precisa.

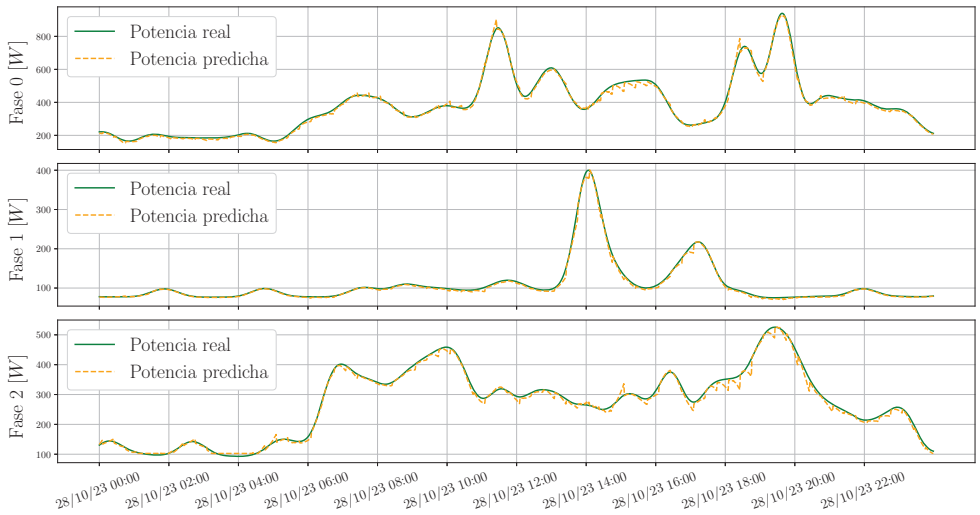


Figura 4.11: Recopilación de las predicciones realizadas cada hora durante el sábado 28 de octubre de 2023.

Para una mejor interpretación de los resultados, se mostrarán las predicciones en el

conjunto de datos de prueba. En la Figura 4.11 se pueden ver las predicciones por ventanas realizadas a cada hora del día 28 de octubre. En la Figura se aprecia que en las ventanas de tiempo que suceden cambios bruscos de pendiente, el modelo tiende a errar en las predicciones.

4.4.2. Estimación del consumo mediante un modelo discontinuo

Al abordar la tarea de predecir el consumo futuro de energía en una vivienda, una primera aproximación lógica sería adoptar un enfoque basado en series temporales, es decir, teniendo en cuenta la continuidad de los datos. De hecho, la mayoría de los esfuerzos iniciales de este proyecto se enfocaron en la construcción de un modelo de esta naturaleza que lograra un desempeño adecuado. No obstante, y dada la limitada cantidad de datos con los que se contaban, pronto se evidenció que obtener predicciones precisas con este enfoque era prácticamente imposible. Es precisamente la naturaleza caótica del consumo eléctrico, dado a los múltiples factores que influyen en él, lo que hace que pierda el sentido de continuidad de la serie temporal.

En vista de estos desafíos, se tomó la decisión de desviar el curso de la investigación y optar por un enfoque alternativo: la construcción de un modelo discontinuo. A diferencia del modelo basado en series temporales, el modelo discontinuo no se basa en ventanas de tiempo históricas para realizar predicciones sobre el futuro. En lugar de alimentar al modelo con valores previos de consumo, se opta por proporcionarle otras características relevantes presentes en el DataFrame `df`, excluyendo específicamente los valores de potencia, que se consideran como las etiquetas o salidas a predecir.

La Función F del modelo discontinuo que se desea encontrar durante el entrenamiento, puede ser descrita matemáticamente de la siguiente manera:

$$(P_0, P_1, P_2) = F(\text{Representación temporal}, \text{Altura solar}, \text{C. ambientales}, \text{nº Personas}) \tag{4.1}$$

Usando una notación más convencional, típicamente asociada con redes neuronales, la relación se define como:

$$(\hat{y_1}, \hat{y_2}, \hat{y_3}) = F(x_4, x_5, x_6, \ldots, x_{14}) \tag{4.2}$$

Finalmente, se decidió prescindir de las características relacionadas con las condiciones ambientales y el número de personas presentes en el hogar. La decisión de eliminar las características ambientales se basó en que, tras múltiples pruebas, no se identificó una relación clara entre estas condiciones y la potencia consumida, al menos durante los meses de septiembre y octubre, que no presentaron condiciones climáticas extremas en una ciudad como Málaga. Por otro lado, aunque el modelo identificó cierta correlación entre la

presencia de personas en el hogar y la potencia consumida, utilizar esta característica en predicciones sin tener certeza de cuándo las personas estarán en casa resultó en una mayor tasa de errores. Además, predecir el consumo basado en la presencia de personas puede ser imprevisible y podría representar un riesgo en términos de privacidad y de cumplimiento con la legislación sobre protección de datos. Por lo tanto, se consideró más prudente eliminar esta característica del modelo.

Este modelo se entrenará utilizando una selección del conjunto total de datos disponibles, que consiste en 66,277 registros hasta el momento del entrenamiento. Para preparar los datos, el 80 % de estos registros (es decir, 53,021 registros) se seleccionarán aleatoriamente para constituir el conjunto de entrenamiento (`train_df`). A diferencia de los modelos de series temporales por ventanas, algunas muestras de los últimos días recogidos son susceptibles de llegar a ser utilizadas para el entrenamiento. Los datos restantes, que corresponden al 20 % (13,256 registros), se separarán para formar el conjunto de pruebas (`test_df`), asegurando que estos datos no se encuentren solapados con los datos de entrenamiento, ya que se eliminan los índices ya incluidos en `train_df` del DataFrame original. Esta división garantiza que el modelo se evalúe en datos completamente nuevos y no vistos durante el entrenamiento, lo cual es crucial para obtener una medida precisa de su rendimiento general. La segmentación se puede visualizar en la Figura 2.5, que ilustra claramente la proporción de datos asignada para el entrenamiento y las pruebas.

- Datos de entrenamiento: 80 %
  - Datos de entrenamiento: 80 %
  - Datos de validación: 20 %
- Datos de Prueba: 20 %

Para abordar el problema de predicción del consumo de energía, se implementó una red neuronal profunda en `Keras`. A continuación, se presenta una descripción detallada de la arquitectura del modelo:

- **Capa de Entrada:**
  - Número de neuronas: Igual al número de características en `train_df`.
- **Capas Ocultas:**
  - Número de capas ocultas: 6.
  - Neuronas por capa: 2048 densamente conectadas.
  - Función de activación: `ReLU`.
- **Capa de Salida:**

- Número de neuronas: 3 densamente conectadas.
- Sin función de activación

Durante la fase de desarrollo y experimentación, se observó que el rendimiento del modelo mejoraba significativamente al aumentar el número de neuronas en las capas ocultas. Esta observación está respaldada por el hecho de que las redes neuronales con mayor capacidad, es decir, con más neuronas, pueden modelar funciones más complejas y capturar relaciones no lineales con mayor facilidad. Sin embargo, también es importante tener en cuenta que aumentar excesivamente la capacidad de la red puede llevar a problemas de sobreajuste, donde el modelo se adapta demasiado a los datos de entrenamiento y pierde capacidad de generalización en datos no vistos.

En nuestro caso, después de varias iteraciones y pruebas con diferentes arquitecturas, se encontró que una configuración con 2048 neuronas en las capas ocultas producía los mejores resultados en términos de precisión y capacidad de generalización. Por lo tanto, se decidió adoptar esta arquitectura para el modelo final.

El modelo se entrenó utilizando la función de pérdida Huber (2.11) y el optimizador Adam. Se configuró para un máximo de 2000 épocas. Además, se implementó un ajuste del ritmo de aprendizaje: durante las primeras 250 épocas se mantuvo constante a un valor de `lr=1e-3`, y a partir de esa época se redujo multiplicándolo por 0.99 en cada época subsiguiente. Se utilizó también un *early stop* tras 15 épocas sin mejoras. El proceso de entrenamiento puede observarse en la Figura 4.12.

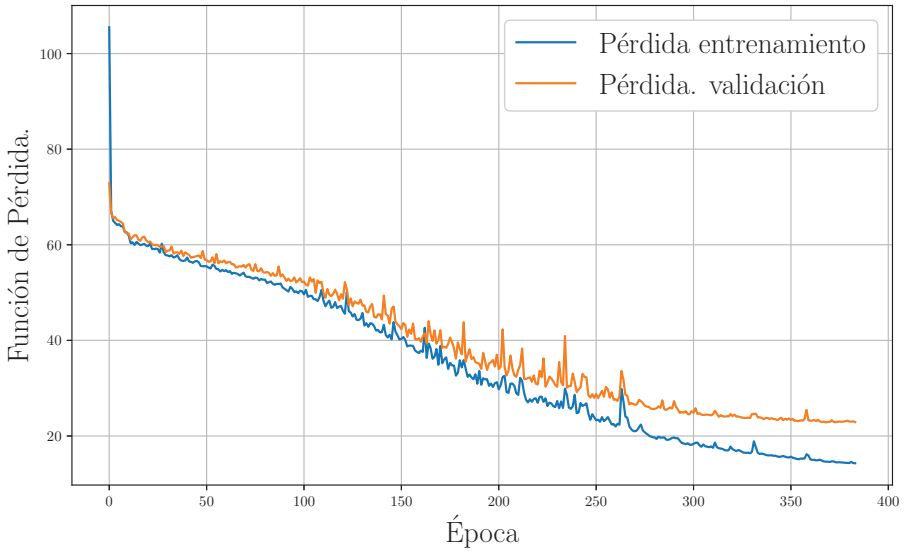


Figura 4.12: Métricas de la función de pérdida de Huber durante el proceso de entrenamiento.

La gráfica de la Figura 4.12, al inicio, ambas curvas de pérdida muestran un valor elevado, cerca de 100. A medida que avanzan las épocas, ambas funciones de pérdida disminuyen significativamente, con una mayor tasa de descenso en las primeras 50 épocas aproximadamente. Posterior a las 50 épocas, la pérdida de entrenamiento sigue una tendencia decreciente más moderada, y la pérdida de validación también sigue esta tendencia pero con una ligera variabilidad. Aproximadamente después de la época 250, donde se aplicó el ajuste en la tasa de aprendizaje (`lr*0.99`), las pérdidas parecen estabilizarse. Al final, las pérdidas de entrenamiento y validación convergen a valores cercanos entre sí, con la pérdida de entrenamiento ligeramente por debajo de la de validación.

Una vez entrenado el modelo y realizadas las predicciones en los datos de validación, se calcula el error absoluto medio de las predicciones frente a las etiquetas, obteniendo los resultados mostrados en la Tabla 4.4. Cabe destacar que el error cometido en los datos de prueba, los cuales desconoce completamente el modelo, únicamente es de apenas 22.11 W, bastante cercano a el error cometido durante el entrenamiento.

Con el objetivo de probar el funcionamiento del modelo en una situación realista, se le ha preguntado que realice predicciones de consumo en el día en el que se esta redactando este apartado, día 31 de octubre de 2023.

La Figura 4.13 muestra los resultados de la previsión de potencia para el día 31 de octubre de 2023. Se pueden hacer las siguientes observaciones:

En general, se observa que la predicción de la Inteligencia Artificial (línea roja punteada) sigue de cerca a la potencia real (línea azul), indicando una buena precisión en la predicción a corto plazo.

Hay ciertos momentos, especialmente en las horas pico, donde las predicciones del sistema se alinean casi perfectamente con la potencia real. Esto evidencia que algunos de esos valores han sido seleccionados durante el proceso de entrenamiento, lo que explica la precisión extremadamente alta en esos puntos. Pero aunque estos datos pueden ocultar la capacidad real del sistema para predecir “a ciegas”, en realidad son una excelente prueba de que todo el proceso de incorporación de datos, preparación de la red, y su entrenamiento, es correcto.

Se pueden identificar además, claramente, las periodicidades en los datos reales, asociadas a consumos periódicos en la vivienda. La Inteligencia Artificial parece detectar bien estas periodicidades, especialmente en la segunda gráfica, donde las predicciones se alinean muy bien con los patrones observados en la potencia real.

En la figura 4.14, sin embargo, se ha utilizado un modelo diferente, entrenado con datos anteriores. Ninguna de las dos predicciones de dicha figura difiere excesivamente de los datos reales, y además, es congruente con el error medio de 70 vatios de dicho modelo.

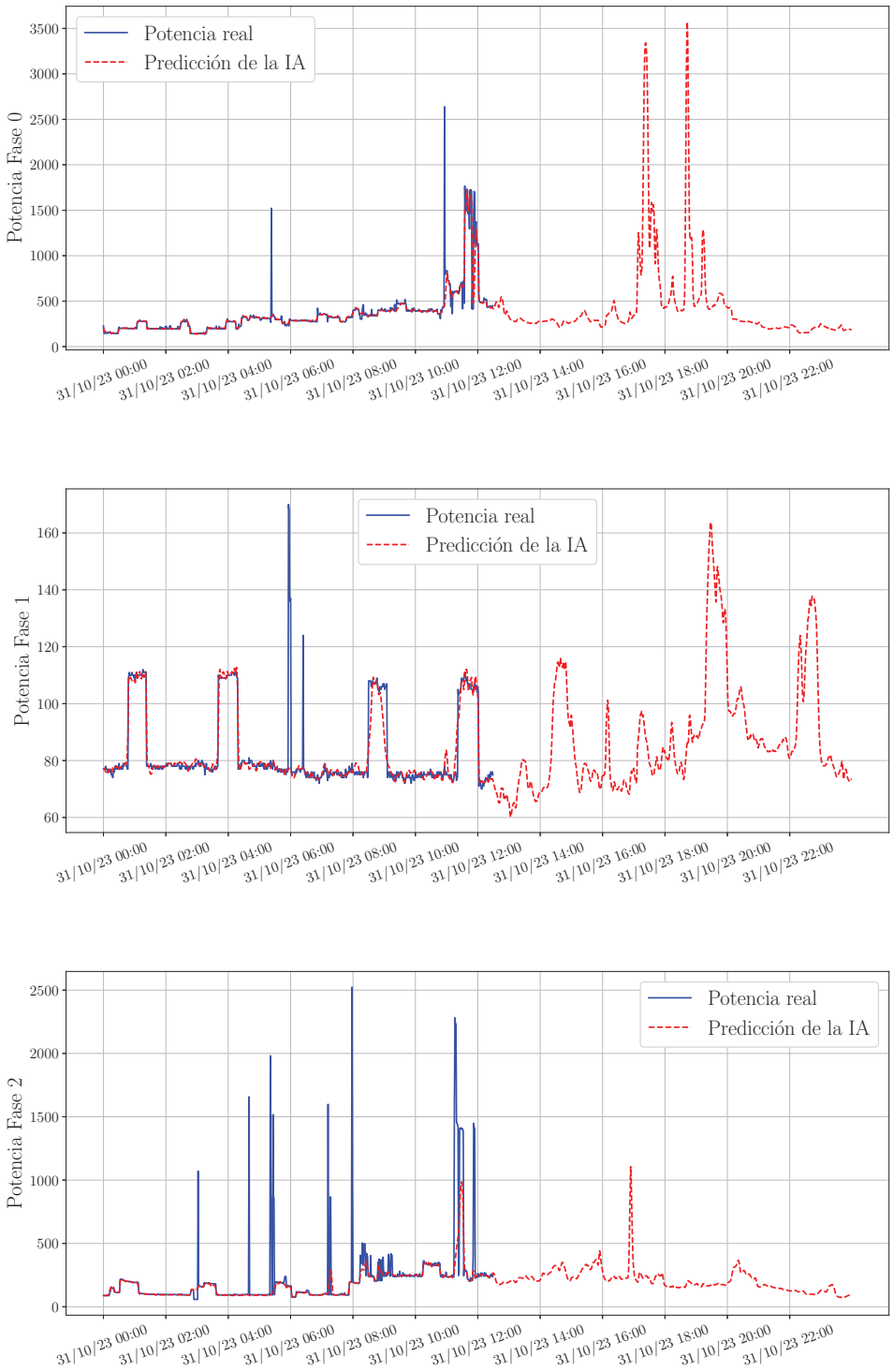


Figura 4.13: Resultados de la previsión de potencia el día 31 de octubre de 2023.

Tabla 4.4: Valores del error medio absoluto en vatios cometido en los distintos datos.

Tipo de datos	**MAE**
Entrenamiento	16.35 W
Prueba	22.11 W

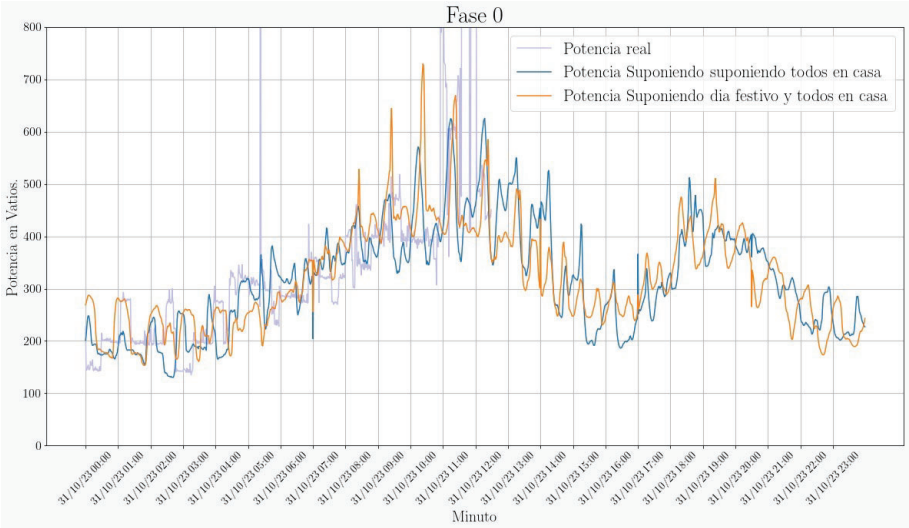


Figura 4.14: Dos previsiones de la fase 0 para el 31 de octubre, usando un modelo entrenado con datos de semanas anteriores, presuponiendo que el 31 es festivo, y que no lo es.

A modo de curiosidad, se muestra en la Figura 4.15 la previsión de potencia según el modelo el 24/12/2023, indicándole que se trata de un día Festivo, donde se prevé una intensa actividad en la vivienda horas previas a la cena. Aunque, es imposible validar estas predicciones, se anticipan que no serán del todo correctas, puesto que se trata de fechas las cuales el modelo aun no ha sido entrenado.

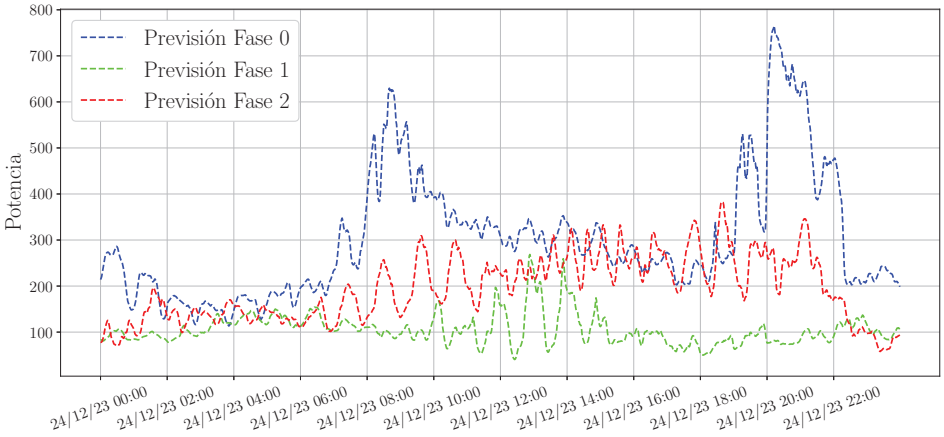


Figura 4.15: Resultados de la previsión de potencia el día 24 de diciembre de 2023.

Capítulo 5

Conclusiones

5.1. Recogida y almacenamiento de datos

En la ejecución de este proyecto, la fase de toma de datos ha sido fundamental para el desarrollo exitoso del sistema predictivo basado en inteligencia artificial. La arquitectura de recopilación de datos, meticulosamente diseñada, combinó tecnologías IoT de alto rendimiento con un reducido ancho de banda, asegurando una recolección continua y fiable de datos. La implementación de un protocolo de comunicaciones basado en MQTT permitió una captura de datos continua, y en tiempo real, una característica esencial para el análisis efectivo de series temporales.

La calidad y diversidad de los datos recopilados han sido pilares en la precisión de los modelos predictivos. Se han considerado múltiples factores, como la hora del día, el día de la semana, festividades, condiciones meteorológicas y la presencia o ausencia de personas, proporcionando un conjunto de datos variado. Esto fue crucial para entrenar modelos de aprendizaje profundo que resultaron en predicciones altamente precisas para una cantidad tan reducida de muestras recogidas.

El almacenamiento y manejo de los datos también desempeñaron un papel crucial. La elección de MongoDB, una base de datos noSQL, facilitó la gestión de grandes volúmenes de datos y ofreció la escalabilidad necesaria para futuras expansiones. El uso de JSON para el almacenamiento e intercambio de información ha mostrado ser simple y eficiente, pero sobre todo, compatible con las tecnologías y lenguajes de programación elegidos.

Además, la interfaz de usuario proporcionada por la aplicación Android permitió a los usuarios calibrar el consumo eléctrico de los diferentes aparatos en sus viviendas. Esta característica no solo es útil para enriquecer el conjunto de datos con información relevante para modelos que contemplen el etiquetado de los electrodomésticos y la iluminación, sino que también permitirá a los usuarios comprender y controlar mejor su consumo de energía.

Aunque la fase de predicción no era el objetivo principal, los experimentos realizados con los datos recopilados durante dos meses resultaron en predicciones de relativamente alta precisión. Esto no solo valida la calidad de la toma de datos sino que también demuestra el potencial del sistema desarrollado para realizar predicciones precisas, incluso con conjuntos de datos de tamaño relativamente reducido.

5.2. Previsión del precio marginal de la energía

El principal objetivo de utilizar diversos modelos de redes neuronales para predecir series temporales en la estimación del precio marginal de la energía es analizar el comportamiento de diferentes modelos predictivos (como modelos densos, recurrentes y convolucionales) frente a problemas asociados con la energía. Esto se debe a que el precio de la energía está intrínsecamente vinculado a la demanda de potencia, determinada durante las tasaciones intradiarias en el mercado.

Al analizar los procesos de entrenamiento presentados en la Figura 4.4, es evidente que los cuatro modelos exhiben comportamientos análogos: una pronunciada disminución de la métrica de la función de pérdida durante las etapas iniciales, seguido de una estabilización en un valor casi constante. Este patrón evidencia un sobreentrenamiento en los modelos, lo que justifica la marcada disparidad entre las predicciones hechas a datos conocidos (como los datos de validación), como se ve en la Figura 4.5, y aquellas realizadas a datos desconocidos, como se muestra en la Figura 4.6.

Es importante señalar un aspecto distintivo de las predicciones mostradas en el día de prueba (Figura 4.6). Los modelos iterativos, como se muestra en la Figura 2.9, necesitan de las 24 horas previas para calcular cada hora del día a calcular. Así, al predecir la primera hora del 25/08/2023, se utiliza el precio histórico de las horas del día anterior. Sin embargo, para la siguiente hora y las subsiguientes, se requiere que la predicción anterior sirva como entrada, provocando que el error de una predicción influya en la siguiente. Este efecto acumulativo del error, como se observa claramente en el modelo Dense, lo vuelve ineficaz, haciendo que las predicciones a largo plazo sean prácticamente inviables. No obstante, al revisar la Figura, se aprecia que modelos como RNN, CNN y LSTM logran identificar, aunque con cierta variabilidad, las horas pico y valle.

Este hecho es notable, especialmente considerando la escasez de datos de entrenamiento: solamente un año, cuando idealmente se necesitarían varios. En conclusión, a pesar de las limitaciones observadas, los resultados no son del todo desfavorables. El hecho de que los modelos puedan predecir, con cierto grado de precisión, utilizando tan solo señales temporales diarias, semanales, anuales y el precio histórico de la energía, sugiere un camino prometedor. Al explorar nuevas características, ampliando la cantidad de datos y, posiblemente, probando enfoques atemporales, hay un potencial significativo para mejorar

la predicción del precio de la energía.

5.3. Predicción del consumo empleando series temporales

A la vista de los resultados, no del todo satisfactorio, obtenidos en la previsión del precio de la energía con el uso de modelos iterativos, se decidió optar por un método directo para calcular la potencia consumida en la vivienda en una ventana de tiempo futura, de manera que, alimentando al modelo con una ventana temporal de 40 minutos de datos conocidos, realizase de un solo disparo, una predicción de los próximos 20 minutos.

A lo largo de la elaboración del proyecto se exploraron una gran variedad de arquitecturas y modelos para la predicción de series temporales para predecir la potencia consumida en la vivienda, incluyendo modelos densos, recurrentes y convolucionales. Desafortunadamente, ninguno de estos modelos ofreció resultados satisfactorios. Finalmente se apostó por el empleo de un modelo basado en redes neuronales LSTM para predecir la potencia de las fases 0, 1 y 2. La arquitectura del modelo se compuso de dos capas LSTM, destinadas a capturar y "recordar" patrones temporales de una ventana de 40 minutos, y posteriormente una capa densa para predecir los siguientes 20 minutos.

El conjunto de datos que se han utilizado para entrenar y evaluar el modelo ha sido recopilado minuto a minuto durante un mes. A pesar de que la calidad de los datos recogidos es alta, dada la amplia variedad de características de cada paso temporal (potencia, humedad, temperatura, número de personas en la casa, entre otros), el periodo de un mes es claramente insuficiente si se busca que el modelo sea eficaz en predecir cualquier día del año.

La gestión de datos ausentes supuso un reto, donde se asumieron ciertas suposiciones, considerando variables numéricas de entrada como la potencia histórica de cada fase o los datos ambientales basados en un día promedio, calculando el valor medio de cada variable para cada minuto del día. Si bien estas suposiciones pueden ser útiles a corto plazo, la falta de diversidad en los datos puede limitar la generalización del modelo a largo plazo.

Uno de los retos encontrados en el proceso fue la eliminación de ruido a través del filtrado gaussiano. Si bien este proceso mejoró la claridad de la señal, tuvo el efecto adverso de suavizar y perder detalles valiosos, como los picos de potencia resultantes de acciones cotidianas, como el encendido de electrodomésticos de alta potencia.

Añadiendo a la complejidad, el sistema de recopilación de datos presenta un retraso de dos horas para subir los datos completos a la base de datos noSQL de MongoDB. Esta retraso, sumada al hecho de que el modelo solo predice 20 minutos en el futuro y que solo los primeros 10 minutos son medianamente confiables, pone en duda la utilidad práctica de tal enfoque de predicción con la cantidad actual de datos.

También es de destacar, que para realizar cada predicción, se necesita hacer una consulta a la base de datos sobre los datos inmediatamente anteriores. Este hecho resulta bastante engorroso si se quiere extrapolar el modelo a otras viviendas, sumado a la necesidad de que tendrían de estar meses, o incluso años recopilando datos para poder obtener unos resultados medianamente decentes con este modelo.

En conclusión, a pesar de los grandes esfuerzos que se han puesto y la alta calidad de los datos que se han recopilado, el corto tiempo de recolección y otros desafíos técnicos hacen que se limite la utilidad de los modelos de series temporales en este contexto particular. Se sugiere que se recolecten datos durante un período más largo para abordar de manera más adecuada la predicción de potencia en el futuro. Así, al menos a corto plazo, esta línea de investigación utilizando modelos de series temporales se ve descartada.

5.4. Predicción Atemporal del Consumo

El empleo de un modelo denso de redes neuronales no orientado específicamente a series temporales ha demostrado ser notablemente superior a los modelos basados en series temporales que se probaron previamente. Además de su superioridad en términos de precisión, este modelo denso presenta la ventaja de ser más sencillo de implementar. Esta simplicidad hace que sea una opción accesible y fácil de usar para cualquier usuario, especialmente si se considera su implementación en más viviendas. Con la posibilidad de desarrollar una aplicación para Android utilizando TensorFlow Lite, se vuelve aún más factible su adopción generalizada. Sin embargo, a pesar de sus fortalezas, el modelo actualmente enfrenta desafíos al prever consumos en días para los cuales no ha sido entrenado. Aunque presenta dificultades en escenarios completamente nuevos, muestra un rendimiento impresionante en días con los que está familiarizado.

Analizando el proceso de entrenamiento de la Figura 4.12 se pueden sacar varias conclusiones. El modelo muestra una convergencia rápida en las primeras épocas de entrenamiento, evidenciando que la arquitectura del modelo empleada (Un modelo simple pero con gran cantidad de neuronas) es apropiada para este conjunto de datos.

Tras la implementación de un algoritmo de decaimiento de la tasa de aprendizaje (*Learning Rate Schedule)*, se observa una notable estabilización en la función de pérdida después de la época 250, evitando grandes oscilaciones que el entrenamiento estaba empezando a sufrir al encontrar dificultades para encontrar el mínimo de la función de pérdida, lo que sugiere que el ajuste en la tasa de aprendizaje resultó ser una táctica efectiva. Además, la semejanza entre las pérdidas de entrenamiento y validación insinúa que el modelo no está sufriendo un sobreajuste significativo.

De la misma manera observando los resultados de la Figura 4.13 se aprecia que el modelo muestra una precisión destacada en los días que es bastante probable que contengan datos

de entrenamiento, ya que las predicciones de la IA se alinean estrechamente con los valores reales en varios intervalos.

La precisión extremadamente alta en ciertos puntos podría indicar que algunos de esos valores estuvieron presentes durante el proceso de entrenamiento del modelo, lo que podría explicar su precisión en esos momentos específicos.

Aunque el MAE de entrenamiento y de test son relativamente bajos, lo que indica una buena generalización, es crucial ser cauteloso al extrapolar estos resultados a situaciones completamente nuevas para el modelo.

La capacidad del modelo para detectar periodicidades en los datos sugiere que tiene el potencial de identificar y prever consumos periódicos en la vivienda con cierta precisión.

Para mejorar la precisión del modelo en escenarios desconocidos, sería beneficioso considerar un conjunto de datos de entrenamiento más extenso, posiblemente abarcando un rango de tiempo más amplio, como un año completo. Esto permitiría al modelo adaptarse mejor a las variaciones y patrones a lo largo del tiempo.

5.5. Trabajos futuros

Tras el análisis exhaustivo de los resultados presentados, surge la casi imperiosa necesidad de continuar este trabajo para explotar su enorme potencial. Y en primer lugar, por supuesto, está la mejora del sistema para que llegue a ser capaz de modelar, casi a la perfección, el consumo eléctrico del hogar. Algo que actualmente se suele denominar como "gemelo digital".

Pero, para llevar a cabo una evolución exitosa de este proyecto hacia un gemelo digital más completo, se requerirían varios pasos adicionales. En primer lugar, es crucial expandir la representación del sistema real en el modelo digital, incorporando una mayor cantidad de variables y factores que influyen en el consumo de energía, como la eficiencia energética de los dispositivos y de la propia vivienda, la energía de retorno, las condiciones térmicas concretas de cada habitación de la vivienda y otros parámetros relevantes. Además, se podría considerar la implementación de un sistema de simulación que permita la interacción en tiempo real con el gemelo digital para probar escenarios hipotéticos y optimizar el consumo de energía. La capacidad de tomar decisiones basadas en el gemelo digital, como la gestión de la demanda de energía y la optimización de recursos, sería un avance adicional. En última instancia, la evolución de este proyecto hacia un gemelo digital completo contribuiría de manera significativa a la gestión eficiente de la energía en entornos residenciales y comunitarios, brindando beneficios tanto económicos como ambientales.

Otra de las posibles líneas futuras de investigación son, evidentemente, las que nos llevan a probar diferentes arquitecturas para la red neuronal e incluso diferentes formas

de abordar el entrenamiento. Ya se han estudiado algunas de estas líneas, como parte de la investigación previa a la elección de las propuestas de la sección 3.5. Hay que tener en cuenta que estas diferentes alternativas no se han descartado, sino que se han considerado fuera del ámbito de este Trabajo de Fin de Máster por su dimensión. Una de ellas, como es el entrenamiento mediante imágenes, se ha explorado inicialmente y se describe en el Anexo A.

También es importante aprovechar la arquitectura escalable del sistema, diseñada para tener una gestión robusta de los datos y un sistema de comunicaciones escalable, para extenderlo a comunidades de vecinos u otras entidades urbanísticas superiores. Precisamente, el hecho de que la instalación de la vivienda utilizada en este proyecto sea trifásica no ha sido especialmente relevante por su "carácter trifásico", sino por su capacidad para incorporar tres instalaciones independientes en un modelo único, como si de tres viviendas independientes se tratara.

La extrapolación a un mayor número de hogares resulta pues, casi inmediata. Sería necesario, no obstante, ampliar la capacidad de los servidores en los que se han instalado las aplicaciones de este proyecto (como el broker MQTT, el demonio Linux o el servicio REST), aunque también sería necesaria una cuidadosa revisión de los problemas éticos y de seguridad propios del uso del big data y la Inteligencia Artificial, así como el ajuste del sistema a la Ley Orgánica de Protección de Datos. Por otra parte, esta ampliación se beneficiaría de un aumento significativo de los datos con los que se podría alimentar la Inteligencia Artificial del sistema, mejorando aún más la precisión y la capacidad para predecir el consumo de forma global. Y no es iluso suponer que el sistema aprendería por sí solo, por ejemplo, a reconocer modelos concretos de aparatos eléctricos, a identificar ciertos comportamientos comunitarios, e incluso a reconocer a las personas por sus hábitos de consumo eléctrico.

Málaga, 2 de noviembre de 2023

Referencias

[1] E. Meira de Oliveira and F. Luiz Cyrino. Forecasting mid-long term electric energy consumption through bagging arima and exponential smoothing methods. *Energy*, 144:776–788, 2018.

[2] S. Ozturk and F. Ozturk. Forecasting energy consumption of turkey by arima model. *Journal of Asian Scientific Research*, 8(2):52, 2018.

[3] S. Barak and S. Sadegh Saeedeh. Forecasting energy consumption using ensemble arima–anfis hybrid algorithm. *International Journal of Electrical Power & Energy Systems*, 82:92–104, 2016.

[4] G. Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.

[5] X. Wang and M. Meng. A hybrid neural network and arima model for energy consumption forcasting. *J. Comput.*, 7(5):1184–1190, 2012.

[6] J.S. Chou and D.S. Tran. Forecasting energy consumption time series using machine learning techniques based on usage patterns of residential householders. *Energy*, 165:709–726, 2018.

[7] I. Koprinska, D. Wu, and Z. Wang. Convolutional neural networks for energy time series forecasting. In *2018 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2018.

[8] R. Rick and L. Berton. Energy forecasting model based on cnn-lstm-ae for many time series with unequal lengths. *Engineering Applications of Artificial Intelligence*, 113:104998, 2022.

[9] B. Vega Márquez, C. Rubio Escudero, I Nepomuceno-Chamorro, and Á. Arcos-Vargas. Use of deep learning architectures for day-ahead electricity price forecasting over different time periods in the spanish electricity market. *Applied Sciences*, 11(13):6097, 2021.

[10] P. Lara Benítez, M. Carranza García, J. M Luna Romera, and J.C. Riquelme. Temporal convolutional networks applied to energy-related time series forecasting. *applied sciences*, 10(7):2322, 2020.

[11] A. Manowska, A. Wycisk, A. Nowrot, and J. Pielot. The use of the mqtt protocol in measurement, monitoring and control systems as part of the implementation of energy management systems. *Electronics*, 12(1):17, 2022.

[12] I. Priyadarshini, A. Alkhayyat, A. Gehlot, and R. Kumar. Time series analysis and anomaly detection for trustworthy smart homes. *Computers and Electrical Engineering*, 102:108193, 2022.

[13] J.A. Mauricio. *Análisis de series temporales*. Universidad Complutense de Madrid, 2007.

[14] M.P. Viñals. *Series temporales*, volume 64. Univ. Politèc. de Catalunya, 2009.

[15] S.J. Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[16] J. Torres Viñals. *Deep learning : introducción práctica con Keras*. Kindle Direct Publishing, Barcelona, 2018.

[17] W. S McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[18] J. Torres Viñals. *Python deep learning : introducción práctica con Keras y TensorFlow 2*. Marcombo, Barcelona, 2020.

[19] Wikipedia. Error cuadrático medio (mse). `https://es.wikipedia.org/wiki/Error_cuadr%C3%A1tico_medio`, 2023.

[20] Wikipedia. Error absoluto medio (mae). `https://es.wikipedia.org/wiki/Error_absoluto_medio`, 2023.

[21] P.J. Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics: Methodology and distribution*, pages 492–518. Springer, 1992.

[22] E. Weisstein. Chain rule. `https://mathworld.wolfram.com/ChainRule.html`. From MathWorld–A Wolfram Web Resource.

[23] M. Abadi, A. Agarwal, P. Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems. `https://arxiv.org/abs/1603.04467`, 2015.

[24] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] F. Chollet. *Deep Learning with Python*. Simon and Schuster, 2021.

[26] B. Lim and S. Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, 2021.

[27] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[28] S. Bai, J.Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[29] A. Borovykh, S.r Bohte, and C.W. Oosterlee. Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*, 2017.

[30] MQTT Version 3.1.1 OASIS Standard. `https://mqtt.org`, Último acceso: 31 de octubre, 2023.

[31] J. Romero. Gráficas de consumo en vivienda. `https://javiromero.es/consumo`, Último acceso: 31 de octubre, 2023.

[32] MongoDB, Inc. *MongoDB*. MongoDB, Inc., 2009.

[33] Agencia Estatal de Meteorología. Servicio OpenData AEMET. `https://opendata.aemet.es`, Último acceso: 31 de octubre, 2023.

[34] J. Romero. Servicio REST de datos ambientales. `http://www.javiromero.es/consumo/servicio.php`, Último acceso: 31 de octubre, 2023.

[35] J. Romero. Documentación Doxygen del servicio REST de datos ambientales. `http://www.javiromero.es/consumo/html`, Último acceso: 31 de octubre, 2023.

[36] J. Romero and F. Romero. Documentación Doxygen de demonioMQTT. `https://casium2.uma.es/demonio`, Último acceso: 31 de octubre, 2023.

[37] Omie - Operador del mercado ibérico de energía. `https://www.omie.es`, Último acceso: 31 de octubre, 2023.

[38] MongoDB, Inc. *MongoDB Atlas Database on AWS*. MongoDB, Inc., 2016.

[39] Wikipedia. Función gaussiana. `https://es.wikipedia.org/wiki/Funci%C3%B3n_gaussiana`, 2023.

[40] F. Romero, L. F. Romero, and P. M. Ortigosa. Reconocimiento de fármacos mediante inteligencia artificial. In *XXXII Jornadas Sarteco*, Alicante, Spain, september 2022.

[41] A. Semenoglou, E. Spiliotis, and V. Assimakopoulos. Image-based time series forecasting: A deep convolutional neural network approach. *Neural Networks*, 157:39–53, 2023.

[42] J. Romero. Animación que representa el consumo eléctrico durante el mes de octubre de 2023. `https://youtu.be/YWfudbM0GCM`, Último acceso: 31 de octubre, 2023.

Anexos

Anexo A

Entrenamiento con imágenes

Casi todos los investigadores en Inteligencia Artificial comparten la opinión de que la frase "una imagen vale más que mil palabras" es perfectamente aplicable al propio entrenamiento de las redes neuronales, o lo que es lo mismo, que su aprendizaje puede ser más rápido cuando la información llega de forma gráfica. Y el por qué esto es así quizás queda oculto en la misteriosa naturaleza del funcionamiento de las redes neuronales, pero, sin embargo, funciona. También influye, por supuesto, que la inmensa mayoría de la investigación para el desarrollo de sistemas de Inteligencia Artificial, como el propio TensorFlow, se ha centrado especialmente en el reconocimiento de imágenes, por lo que son más rápidos y eficientes cuando trabajan sobre ellas. Así, por ejemplo, una Inteligencia Artificial puede reconocer fácilmente una proteína o un fármaco por su aspecto, más que por una descripción textual de su disposición atómica [40]. O también es capaz de predecir una serie temporal a partir de una secuencia de las imágenes que representan los fragmentos de la curva [41].

Con esta motivación, se ha desarrollado una herramienta para convertir las información con la que se alimenta el sistema en imágenes, y que, sin formar parte de los objetivos y resultados propios de este Trabajo de Fin de Máster, si se ha considero mencionar como anexo, ya que permite visualizar gráficamente sus datos, y sobre todo, porque abre una línea futura de investigación para la mejora del reconocimiento.

Así, siguiendo la línea de la reciente publicación [41], así como la de la numerosa bibliografía en ella mencionada, se decidió trasladar las diferentes ventanas de tiempo de la curva que representa el consumo en una fase, a imágenes. En particular, se ha trasladado la curva a una secuencia de imágenes en formato PNG con 128×128 píxeles de resolución, y se ha aprovechado el ancho de 128 píxeles de cada imagen para que se representen en ella aproximadamente dos horas (128 minutos) de datos.

Por otra parte, en la mayoría de las secuencias temporales de imágenes de la literatura, se utiliza una representación con escalas de grises o monocromo, ya que para una simple

curva de una serie temporal, utilizar una gama cromática superior es redundante. Sin embargo, la propuesta que se hace en este trabajo es la siguiente. ¿Por qué no aprovechar toda la escala RGB para agregar la información ambiental más relevante?

Además, teniendo en cuenta que, con mucha diferencia, la hora del día y el que sea festivo o no, son los factores más importante en el consumo eléctrico, decidimos incorporar esta información como fondo de imagen, según se muestra en la figura de la derecha, como ejemplo. Por ello, se proponen las escalas de colores para el fondo de la figura A.2, en las que, de izquierda a derecha se representan las horas del día.

Estos mapas de colores tienen unas características especiales:

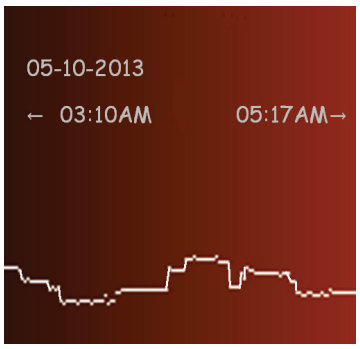


Figura A.1: Imagen de la curva durante las primeras horas de la madrugada de un día laborable.

- Son mapas cíclicos. Las 23:59 de un día comparten el color (negro) con las 00:00 del día siguiente.
- Los festivos y los días laborables utilizan pares de colores clave diferentes.
- Las horas centrales del día usan un color diferente a los extremos, para que la red neuronal pueda reconocer la hora según el degradado del fondo.
- Festivos y días laborables comparten el fondo negro, para que la transición sea continua, coincidiendo además con las horas nocturnas (de 00:00 a 02:00) en las que es menos relevante la situación festiva.
- Las tonalidades son oscuras para que la curva de consumo, que será dibujada en blanco, no se pueda confundir.

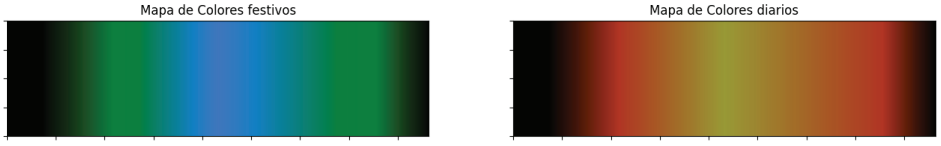


Figura A.2: Mapas de colores del fondo de las imágenes.

Una vez elegido el color de fondo, se dibuja sobre ella la curva de consumo, con cada fase en imágenes diferentes, y en una escala de 20W/píxel, ya que es extraordinariamente raro que se alcancen los 2560W (128x20W) en una misma fase de la vivienda (al menos provisionalmente).

Por otra parte, para agregar más información en cada imagen, se han incorporado los saltos de potencias registrados en la base de datos en forma de puntos, al que se le asigna la columna por la hora, y la fila por su potencia (en colores diferentes para encendido y apagado).

En la siguiente figura A.3 se muestra el resultado para las tres fases durante el desayuno de un día laborable. Mientras que la curva de consumo se visualiza, en blanco, en la parte inferior de cada imagen, en la parte superior aparece una nube de puntos claros que representan saltos de potencias. Los de mayor potencia son más gruesos. Las figuras están etiquetadas con los aparatos que se encienden y apagan durante la misma (por ejemplo, se ha marcado un punto rojo, que por su fase y rango de potencia, se ha etiquetado como tostadora):

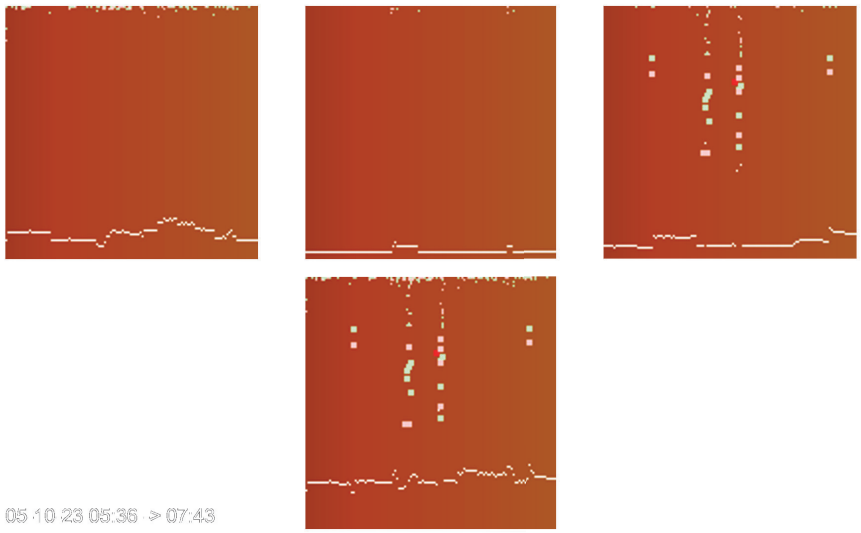


Figura A.3: Figuras de las tres fases (arriba) y del conjunto de la instalación (abajo) durante las horas del desayuno de un día laborable

Un simple vistazo a esta figura, y su comparación con figuras similares, como se muestran en la animación [42], nos permiten identificar la hora del desayuno de dos personas sobre las 6 y cuarto, y otras dos personas unos 20 minutos más tarde. Tras analizar toda la secuencia de la animación, hemos reconocido patrones que se repiten en aparatos con un comportamiento irregular. Por ejemplo, una "lavadora de delicado" tiene un patrón específico, como también lo tienen el "desayuno en familia" del domingo, o el "ciclo rápido del lavavajillas". Eso nos lleva a hacernos la siguiente propuesta: Si nosotros somos capaces de distinguirlos, la Inteligencia Artificial también sabrá hacerlo, abriendo de esa forma una nueva vía de investigación realmente apasionante e ilusionante. Por supuesto, la aplicación Android jugaría en este caso, un papel fundamental para el etiquetado de dichos patrones.

Anexo B

Código de Entrenamiento del modelo

```
1
import numpy as np
3 import pandas as pd
import matplotlib.pyplot as plt
5 import time

7 import tensorflow as tf
keras = tf.keras
9
# Configuracion de fuentes
11 plt.rcParams.update({
    "text.usetex": True,
13     "font.family": "serif",          # Fuente serif para texto principal
    "font.serif": ["CM Roman"],
15     "font.sans-serif": ["CM Sans Serif"],  # Fuente sans-serif para texto
    secundario
    "font.size": 16,
17 })

19 print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

21 # Cargar los DataFrames de entrenamiento, validacion y prueba
df_original = pd.read_csv("D:\\PREDICCION_POTENCIA\\datos\\df.csv", sep=';')
23 df = pd.read_csv("D:\\PREDICCION_POTENCIA\\datos\\dataset.csv", sep=';')

25
df.drop(columns=['year_sin', 'year_cos','humedad','temperatura','Papa','Mama
    ','Ipe','Javi'], inplace=True) # Sin_ambiente_ni_personas.h5
27
# Crear datos de entrenamiento y test
29 train_df=df.sample(frac=0.8,random_state=0)
test_df=df.drop(train_df.index)
31
```

```
# Extraer las columnas y juntarlas en un nuevo DataFrame
train_labels = train_df[['fase0', 'fase1', 'fase2']]

# Eliminar las columnas de train_df
train_df.drop(columns=['fase0', 'fase1', 'fase2'], inplace=True)

# Extraer las columnas y juntarlas en un nuevo DataFrame
test_labels = test_df[['fase0', 'fase1', 'fase2']]

# Eliminar las columnas de train_df
test_df.drop(columns=['fase0', 'fase1', 'fase2'], inplace=True)


train_stats=train_df.describe()
train_stats=train_stats.transpose()
train_stats

# NORMALIZACION
def norm(x):
    return  (x-train_stats['min'])/(train_stats['max']-train_stats['min'])

norm_train_df=norm(train_df)
norm_test_df=norm(test_df)

#MODELO
inputs = keras.Input(shape=(len(train_df.keys())), name='Input')
x = keras.layers.Dense(2048,activation='relu')(inputs)
x = keras.layers.Dense(2048,activation='relu')(x)
x = keras.layers.Dense(2048,activation='relu')(x)
x = keras.layers.Dense(2048,activation='relu')(x)
x = keras.layers.Dense(2048,activation='relu')(x)
x = keras.layers.Dense(2048,activation='relu')(x)
outputs = keras.layers.Dense(3)(x)

model = keras.Model(inputs=inputs, outputs=outputs, name='model')

model.summary()

# COMPILAR

keras.backend.clear_session()
tf.random.set_seed(42)
np.random.seed(42)

# Definicion del modelo (asumo que ya lo has definido anteriormente)

model.compile(loss=keras.losses.Huber(),
              optimizer=tf.optimizers.Adam(),  # En la primera fase, el LR por defecto de Adam se utilizara
              metrics=[tf.metrics.MeanAbsoluteError()])
```

```
83 early_stopping = keras.callbacks.EarlyStopping(patience=15)

85 # Definir una funcion para el LearningRateScheduler
   def scheduler(epoch, lr):
87     if epoch < 250:
           return lr  # Mantener la tasa de aprendizaje para las primeras 300
       epocas
89     else:
           return lr*0.99  # Ajustar la tasa de aprendizaje a 1e-4 despues de
       la epoca 300
91
   lr_schedule = keras.callbacks.LearningRateScheduler(scheduler)
93
   history = model.fit(
95     norm_train_df,
       train_labels,
97     batch_size=2048,
       epochs=2000,
99     validation_split=0.2,
       callbacks=[early_stopping, lr_schedule],  # Agregar lr_schedule a la
       lista de callbacks
101    verbose=1)
```

codigos/entrenamiento.py

Anexo C

Códigos auxiliares

Se adjuntan, a continuación, algunos códigos que se consideran de interés. Se ha procurado, a lo largo de todo este Trabajo de Fin de Máster, que todos los códigos vayan acompañados de una documentación exhaustiva, aprovechando la facilidad ofrecida por los entornos de desarrollo, como VSCode y Android Studio, con herramientas como Doxygen [35, 36]. En el caso de los códigos Python, se han utilizado en muchos casos los *notebooks* de Jupyter para enriquecer la documentación.

Se muestran como ejemplo los *notebooks* correspondientes al etiquetado de los aparatos de comportamiento periódico (frigoríficos, congeladores, etc.) así como los utilizados para la generación de imágenes en el Anexo A.

insertaAparatos

November 2, 2023

1 Inserción/reemplazo en base de datos MongoDB

1.1 Inserción de datos de aparatos en aparatos.json

Nota: No olvidar meter la contraseña antes de utilizar, y borrarla antes de subirlo a Github o publicar

[1]:
```
#pip install pymongo
from pymongo import MongoClient
import json
from pymongo import UpdateOne, InsertOne
```

[4]:
```
# Ruta del archivo JSON
jsonFilePath = 'aparatos.json'

# URL de conexión a MongoDB Atlas
uri = 'mongodb+srv://noctiluca:xxx@cluster0.o22eilj.mongodb.net/?
 ↪retryWrites=true&w=majority'

# Nombre de la colección en la que quieres insertar los documentos
collectionName = 'aparatos'
```

[5]:
```
# Leer el contenido del archivo JSON
with open(jsonFilePath,'r', encoding='utf-8') as file:
    jsonData = json.load(file)

# Conectar a la base de datos
client = MongoClient(uri)
db = client['tfmJavi']
collection = db[collectionName]

bulk_operations = []

for data in jsonData:
    filter = {"_id": data["_id"]}
    update = UpdateOne(filter, {"$set": data}, upsert=True)
    bulk_operations.append(update)
```

```
result=collection.bulk_write(bulk_operations)


# Cerrar la conexión a la base de datos
client.close()
```

Para depurar, descomentar dump

[5]:
```
def dump(obj):
  for attr in dir(obj):
    print("obj.%s = %r" % (attr, getattr(obj, attr)))
# dump(result)
```

Ejemplo de archivo:

```
[
  {
    "_id": 1,
    "nombre": "Basal Fase R",
    "Potencia": 30,
    "fase":1,
    "tipo":1
  },
  {
    "_id": 2,
    "nombre": "Basal Fase S",
    "Potencia": 30,
    "fase":2,
    "tipo":1
  },
  {
    "_id": 3,
    "nombre": "Basal Fase T",
    "Potencia": 59,
    "fase":3,
    "tipo":1
  }
  ]
```

[ ]:

etiquetaValoresPeriodicos

November 2, 2023

1 Etiquetado de datos periódicos

Nota: No olvidar meter la contraseña, y borrarla antes de subirlo a Github.

Comienza instalando pandas y pymongo...

[3]:
```
# pip install nbconvert[qtpdf]
# pip install pymongo
```

[1]:
```
from pymongo import MongoClient
import json
from pymongo import UpdateOne, InsertOne
from datetime import datetime, timedelta
import pandas as pd
```

La URI conecta con la base de datos...

[2]:
```
# Ruta del archivo JSON
jsonFilePath = 'aparatos.json'

# URL de conexión a MongoDB Atlas
uri = 'mongodb+srv://noctiluca:***@cluster0.o22eilj.mongodb.net/?
 ↪retryWrites=true&w=majority'

# Nombre de la colección en la que quieres insertar los documentos
collectionName1 = 'potencias'
collectionName2 = 'aparatos'
```

[3]:
```
# Configura la conexión a la base de datos MongoDB
client = MongoClient(uri)
db = client["tfmJavi"]
collection1 = db[collectionName1]
collection2 = db[collectionName2]
# Descargar los documentos de la colección
```

Introducimos algunos parámetros por defecto: Fase Valor medio (vmedia) Margen de error en el valor (verror) Duración media (tmedia) Margen de error en la duración

[4]:
```
fase=2
vmedia=26
verror=9
tmedia=35
terror=3
```

Creamos la función que calcula los pares, usando pandas

[11]:
```
def calcula(vmedia,verror,tmedia,terror,fase,verbose=False):
    global collection1
    # print("hola")
    documents = collection1.find({"fase": fase})
    #print(document)
    # Convertir los documentos a un DataFrame de Pandas
    df = pd.DataFrame(documents)
    # print(df.tail(10))
    # Convertir el timestamp a un objeto datetime
    df["tiempo"] = pd.to_datetime(df["tiempo"])
    df = df.sort_values("tiempo")
    df["salto"] = df["hasta"]-df["desde"]
    #print(df)
    vmin=vmedia-verror
    vmax=vmedia+verror
    tmin=tmedia-terror
    tmax=tmedia+terror

    positivos = df[df["salto"].between(vmin, vmax)]
    negativos = df[df["salto"].between(-vmax, -vmin)]
    # Obtener el timestamp del último documento filtrado


    ts = positivos["tiempo"].iloc[0]
    ss = positivos["salto"].iloc[0]
    #print(str(ts)+" "+str(ss))
    #negativos = negativos[negativos["tiempo"] >= (ts + timedelta(minutes=30))]
    fin=negativos.empty==True
    cnt=0
    while fin!=True:
        # Filtrar los documentos que se encuentren en un rango de 30 a 35␣
↪minutos después del último documento filtrado
        #print(df2["tiempo"].iloc[0])
        ts = positivos["tiempo"].iloc[0]
        ss = positivos["salto"].iloc[0]
        ds = positivos["desde"].iloc[0]
        hs = positivos["hasta"].iloc[0]
        candidatos = negativos[negativos["tiempo"] >= (ts +␣
↪timedelta(minutes=tmin))]
```

```
        candidatos = candidatos[candidatos["tiempo"] <= (ts +␣
↪timedelta(minutes=tmax))]
        # print(candidatos)
        encontradoEnRato=candidatos.empty==False
        if encontradoEnRato:
            td = candidatos["tiempo"].iloc[0]
            sd = candidatos["salto"].iloc[0]
            dd = candidatos["desde"].iloc[0]
            hd = candidatos["hasta"].iloc[0]
            if verbose:
                print(str(ts)+"\t "+str(ss)+"\t "+str(ds)+"\t "+str(hs))
                print(str(td)+"\t "+str(sd)+"\t "+str(dd)+"\t "+str(hd))
            ts=td
            cnt=cnt+1
        else:
            ts=ts + timedelta(minutes=35)
        positivos = positivos[positivos["tiempo"] >=ts]
        negativos = negativos[negativos["tiempo"] >=ts]
        fin=positivos.empty==True
    print("Encontrados "+str(cnt)+" pares")
```

[8]:
```
verbose=False
calcula(vmedia,verror,tmedia,terror,fase)
```

```
                      tiempo  fase  hasta  desde                       _id  \
37084 2023-10-31 19:13:04.005     2    135    154  654151c07ddb31e3c20f6b08
37085 2023-10-31 19:13:04.459     2    124    135  654151c07ddb31e3c20f6b0a
37086 2023-10-31 19:13:05.077     2    143    124  654151c17ddb31e3c20f6b0c
37087 2023-10-31 19:13:25.071     2    173    143  654151d57ddb31e3c20f6b0e
37088 2023-10-31 19:13:41.244     2    114    173  654151e57ddb31e3c20f6b10
37089 2023-10-31 19:25:29.992     2    168    110  654154a97ddb31e3c20f6b53
37090 2023-10-31 19:25:34.989     2    157    168  654154ae7ddb31e3c20f6b55
37091 2023-10-31 19:25:35.453     2    174    157  654154af7ddb31e3c20f6b57
37092 2023-10-31 19:28:00.118     2    139    172  654155407ddb31e3c20f6b67
37093 2023-10-31 19:28:25.662     2     84    139  654155597ddb31e3c20f6b69

                      ambiente
37084  654150187ddb31e3c20f6ab9
37085  654150187ddb31e3c20f6ab9
37086  654150187ddb31e3c20f6ab9
37087  654150187ddb31e3c20f6ab9
37088  654150187ddb31e3c20f6ab9
37089  654152707ddb31e3c20f6b14
37090  654152707ddb31e3c20f6b14
37091  654152707ddb31e3c20f6b14
37092  654154c97ddb31e3c20f6b5b
37093  654154c97ddb31e3c20f6b5b
Encontrados 170 pares
```

Vamos a buscar los aparatos...

[9]:
```
    #aparatos = collection2.find({"fase": fase})
    aparatos = collection2.find(sort=[('_id', 1)])
    aparatosdf = pd.DataFrame(aparatos)
    print(aparatosdf)
```

```
    _id  fase              nombre  potencia  tipo  tiempo
0     1     1        Basal Fase 1     115.0     1     NaN
1     2     2        Basal Fase 2      75.0     1     NaN
2     3     3        Basal Fase 3      75.0     1     NaN
3     4     1    Frigorífico ppal      70.0     2    15.0
4     5     1     Congelador ppal      55.0     2    53.0
5     6     3   Frigorífico abajo      91.0     2     NaN
6     7     3    Congelador abajo      35.0     2     NaN
7     8     2  Desconocido fase 2      32.0     2    35.0
8     9     3     TostadoraBlanca     758.0     3     2.0
9    10     3             Freidora    2000.0     3    10.0
10   11     1              Prueba       NaN     4     NaN
11   12     3         microondas     1270.0     4     NaN
12   13     1    Luz de la cocina      48.0     4     NaN
```

[12]:
```
# 8 desconocido fase 2
calcula(29,5,35,5,2)
```

```
Encontrados 194 pares
```

[13]:
```
# 4 nevera o coge arriba
calcula(70,20,15,3,1)
```

```
Encontrados 367 pares
```

[14]:
```
# 5 nevera o congelador arriba
calcula(55,5,53,10,1)
```

```
Encontrados 210 pares
```

[16]:
```
# 9 tostadora
calcula(759,30,2,2,3)
```

```
Encontrados 152 pares
```

Ejemplo actualizar bd:

[17]:
```
updateResult = collection2.replace_one({"_id": 11},{"nombre": "Prueba","fase":
 ↪1,"tipo":4})
aparatos = collection2.find(sort=[('_id', 1)])
aparatosdf = pd.DataFrame(aparatos)
print(aparatosdf)
```

	_id	fase	nombre	potencia	tipo	tiempo
0	1	1	Basal Fase 1	115.0	1	NaN
1	2	2	Basal Fase 2	75.0	1	NaN
2	3	3	Basal Fase 3	75.0	1	NaN
3	4	1	Frigorífico ppal	70.0	2	15.0
4	5	1	Congelador ppal	55.0	2	53.0
5	6	3	Frigorífico abajo	91.0	2	NaN
6	7	3	Congelador abajo	35.0	2	NaN
7	8	2	Desconocido fase 2	32.0	2	35.0
8	9	3	TostadoraBlanca	758.0	3	2.0
9	10	3	Freidora	2000.0	3	10.0
10	11	1	Prueba	NaN	4	NaN
11	12	3	microondas	1270.0	4	NaN
12	13	1	Luz de la cocina	48.0	4	NaN

[18]:
```
test1=aparatosdf.iloc[10]
print(test1)
type(test1)
test2=aparatosdf.loc[aparatosdf["_id"] == 11]
print(test2.iloc[0])
type(test2)
# Son la misma fila pero dan diferentes resultados. ¿porqué?
documento = {
    "_id": test1["_id"],
    "nombre": test1["nombre"],
    "tipo": 4
}
documento
```

```
_id             11
fase             1
nombre      Prueba
potencia       NaN
tipo             4
tiempo         NaN
Name: 10, dtype: object
_id             11
fase             1
nombre      Prueba
potencia       NaN
tipo             4
tiempo         NaN
Name: 10, dtype: object
```

[18]:
```
{'_id': 11, 'nombre': 'Prueba', 'tipo': 4}
```

[19]:
```
aparatosdf.index=aparatosdf["_id"]
print(aparatosdf.iloc[11])
```

```
_id                  12
fase                  3
nombre      microondas
potencia         1270.0
tipo                  4
tiempo              NaN
Name: 12, dtype: object
```

[20]:
```
test2["nombre"].item()
```

[20]:
```
'Prueba'
```

[21]:
```
test2.iloc[0]
```

[21]:
```
_id             11
fase             1
nombre      Prueba
potencia       NaN
tipo             4
tiempo         NaN
Name: 10, dtype: object
```

database2imagenes

November 2, 2023

1 Generador de imágenes para entrenamiento

Este documento tiene dos partes. La primera en la descarga de datos de mongodb; la segunda es la conversión a imágenes

1.1 Recuperación (relacional) de datos con pyMongo

El objetivo es descargar las tablas de saltos (potencias) y valores periódicos de potencias, en dataframes, pero agregando la información de ambiente de la otra tabla

[1]:
```
from pymongo import MongoClient
import json
from pymongo import UpdateOne, InsertOne
from datetime import datetime, timedelta
import pandas as pd
import numpy as np
import PIL
```

[3]:
```
import numpy as np
from PIL import Image
# URL de conexión a MongoDB Atlas
uri = 'mongodb+srv://noctiluca:***@cluster0.o22eilj.mongodb.net/?
 ↪retryWrites=true&w=majority'
# Nombre de la colección en la que quieres insertar los documentos
collectionPotencias = 'potencias'
collectionPeriodico = 'periodic'
collectionAmbientes = 'ambientes'
collectionAparatos= 'aparatos'
fase=1 # Ya no hace falta
```

[4]:
```
client = MongoClient(uri)
db = client["tfmJavi"]
col1 = db[collectionPotencias]
col2 = db[collectionPeriodico]
col3 = db[collectionAmbientes]
col4 = db[collectionAparatos]
# docs1 = col1.find({"fase": fase}) #no usado
# docs2 = col2.find()
```

```
# docs3 = col3.find()
docs4 = col4.find()
```

[5]:
```
from bson import ObjectId  # Importa la clase ObjectId de PyMongo
pipeline = [

   {
        "$addFields": {
            "ambiente_id_objectId": {
                "$cond": {
                    "if": {
                        "$eq": ["$ambiente", ""]
                    },
                    "then": "$ambiente",
                    "else": { "$toObjectId": "$ambiente" }
                }
            }
        }
   }




    ,
    {
        "$lookup": {
            "from": "ambientes",     # Nombre de la segunda colección
            "localField": "ambiente_id_objectId",  # Campo de la colección␣
 ↪'orders'
            "foreignField": "_id",        # Campo de la colección 'customers'
            "as": "ambientes"          # Alias para el resultado de la unión
        }
    }
]
saltos_df = pd.DataFrame(col1.aggregate(pipeline))
#result=list(col1.aggregate(pipeline))
```

[6]:
```
potencias_df = pd.DataFrame(col2.aggregate(pipeline))
#result=list(col1.aggregate(pipeline))
```

1.2 Visualizamos algunos datos para comprobar

[7]:
```
pd.DataFrame(docs4)
```

[7]:

	_id	fase	nombre	potencia	tipo	tiempo
0	1	1	Basal Fase 1	115.0	1	NaN
1	2	2	Basal Fase 2	75.0	1	NaN

```
2     3     3        Basal Fase 3      75.0     1     NaN
3     4     1    Frigorífico ppal      70.0     2    15.0
4     5     1     Congelador ppal      55.0     2    53.0
5     6     3   Frigorífico abajo      91.0     2     NaN
6     7     3    Congelador abajo      35.0     2     NaN
7     8     2  Desconocido fase 2      32.0     2    35.0
8     9     3     TostadoraBlanca     758.0     3     2.0
9    10     3            Freidora    2000.0     3    10.0
10   11     1              Prueba       NaN     4     NaN
11   13     1    Luz de la cocina      48.0     4     NaN
12   12     3         microondas     1270.0     4     NaN
```

[8]:
```
print(col3.find()[4135])
saltos_df.iloc[150000]
```

```
{'_id': ObjectId('652c52d000211de7c20ed636'), 'fechaUTC':
'2023-10-15T20:59:00+00:00', 'festivo': True, 'alturaSol': -39.6, 'cielo':
'12n', 'temperatura': 21, 'humedad': 73, 'personas': 14}
```

[8]:
```
tiempo                                          2023-10-15 21:04:39.031000
fase                                                                     1
hasta                                                                  324
desde                                                                  336
_id                                                 652c53e700211de7c20ed660
ambiente_id_objectId                                652c52d000211de7c20ed636
ambientes               [{'_id': 652c52d000211de7c20ed636, 'fechaUTC':…
ambiente                                            652c52d000211de7c20ed636
Name: 150000, dtype: object
```

[13]:
```
saltos_df.tail(5)
```

[13]:
```
                      tiempo  fase  hasta  desde                       _id  \
231172 2023-10-31 21:06:55.158     1    228    244  65416c6f7ddb31e3c20f6da7
231173 2023-10-31 21:07:25.158     1    256    228  65416c8d7ddb31e3c20f6daa
231174 2023-10-31 21:07:42.262     1    229    258  65416c9e7ddb31e3c20f6dac
231175 2023-10-31 21:08:51.462     3    164    153  65416ce37ddb31e3c20f6daf
231176 2023-10-31 21:10:22.406     3    163    151  65416d3e7ddb31e3c20f6db3

           ambiente_id_objectId  \
231172  65416c387ddb31e3c20f6d9b
231173  65416c387ddb31e3c20f6d9b
231174  65416c387ddb31e3c20f6d9b
231175  65416c387ddb31e3c20f6d9b
231176  65416c387ddb31e3c20f6d9b

                                                ambientes  \
231172  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
231173  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
```

```
231174  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
231175  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
231176  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…

                        ambiente
231172  65416c387ddb31e3c20f6d9b
231173  65416c387ddb31e3c20f6d9b
231174  65416c387ddb31e3c20f6d9b
231175  65416c387ddb31e3c20f6d9b
231176  65416c387ddb31e3c20f6d9b
```

[14]:
```
col1.find()[39280]
potencias_df.iloc[39280]
potencias_df.tail(5)
```

[14]:
```
                           _id                  tiempo  fase0  fase1  fase2  \
62268  65416d9f7ddb31e3c20f6db5 2023-10-31 21:12:00.000    228     79    160
62269  65416ddb7ddb31e3c20f6db6 2023-10-31 21:13:00.000    228     79    160
62270  65416e177ddb31e3c20f6db7 2023-10-31 21:14:00.001    226     79    160
62271  65416e537ddb31e3c20f6db8 2023-10-31 21:15:00.001    228     78    158
62272  65416e8f7ddb31e3c20f6db9 2023-10-31 21:16:00.001    231     78    158

                        ambiente      ambiente_id_objectId  \
62268  65416c387ddb31e3c20f6d9b  65416c387ddb31e3c20f6d9b
62269  65416c387ddb31e3c20f6d9b  65416c387ddb31e3c20f6d9b
62270  65416c387ddb31e3c20f6d9b  65416c387ddb31e3c20f6d9b
62271  65416c387ddb31e3c20f6d9b  65416c387ddb31e3c20f6d9b
62272  65416c387ddb31e3c20f6d9b  65416c387ddb31e3c20f6d9b

                                              ambientes
62268  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
62269  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
62270  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
62271  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
62272  [{'_id': 65416c387ddb31e3c20f6d9b, 'fechaUTC':…
```

2 Segunda parte. Crear una imagen

2.0.1 Función que convierte datos a imagen

[244]:
```
# simple ejemplo fondo = [53, 724, 3]
# datos = np.full((128, 128, 3), fondo, dtype=np.uint8)
```

[268]:
```
from PIL import ImageDraw,ImageFont
def data_img(array,nombre,txt):
    array=array.astype("uint8")
    image = Image.fromarray(array)
```

```
    if txt != "":
        font = ImageFont.truetype("arial.ttf", 10)
        draw = ImageDraw.Draw(image)
        draw.text((0, 0), txt, font=font, fill=(0, 0, 0))
    image.save("d:/output/tfm/"+nombre+".png")

def data_4img(array1,array2,array3,array4,nombre,txt,todas=False):
    array1=array1.astype("uint8")
    array2=array2.astype("uint8")
    array3=array3.astype("uint8")
    array4=array4.astype("uint8")
    image1 = Image.fromarray(array1).resize((512,512))
    image2 = Image.fromarray(array2).resize((512,512))
    image3 = Image.fromarray(array3).resize((512,512))
    image4 = Image.fromarray(array4).resize((512,512))
    new_image = Image.new("RGB", (1920, 1080))
    new_image.paste(image1, (96, 10))
    new_image.paste(image2, (96*2+512, 10))
    new_image.paste(image3, (96*3+2*512, 10))
    new_image.paste(image4, (96*2+512, 10+512+36))


    if txt != "":
        font = ImageFont.truetype("arial.ttf", 40)
        draw = ImageDraw.Draw(new_image)
        draw.text((96, 1020), txt, font=font, fill=(255, 255, 255))
    new_image.save("d:/output/tfm/F_"+nombre+".png")
    if todas:
        image1.save("d:/output/tfm/1_"+nombre+".png")
        image2.save("d:/output/tfm/2_"+nombre+".png")
        image3.save("d:/output/tfm/3_"+nombre+".png")
        image4.save("d:/output/tfm/A_"+nombre+".png")
```

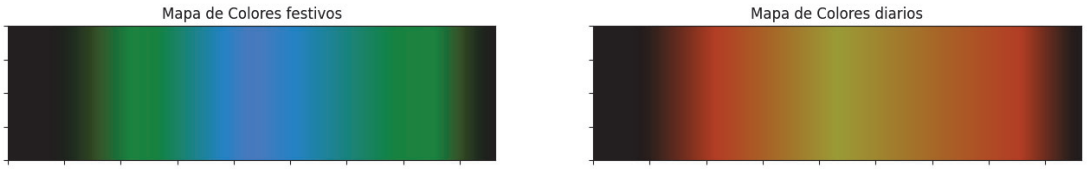
2.0.2 Preparación del mapa de colores

[246]:
```
import numpy as np
import matplotlib.pyplot as plt
import pytz
from matplotlib.colors import LinearSegmentedColormap
```

[264]:
```
def mapaColores(festivo,n_colores=86400):
    color_noche=[0,0,0]
    if festivo:
        # Definir colores de inicio (azul claro) y final (verde oscuro)
        color_medio = [0, 0.5, 1]  # Azul claro (RGB: 0, 128, 255)
```

```
        color_inicio = [0, 0.5, 0]  # Verde oscuro (RGB: 0, 128, 0)
        color_final = color_inicio
    else:
        color_inicio = [0.7, 0.2, 0]
        color_medio = [0.6, 0.6, 0]
        color_final = [0.7, 0.2, 0]

    colores = []

    # Negro a las 12:00 -> 3:00
    # color inicio a las 03:00 -> 6:00
    # color medio a las 12 -> 15
    # color inicio a las 21 -> 24
    # Y cuando termine hago un shift de tres horas
    v=0.125
    for i in range(n_colores):
        ratio = i / (n_colores - 1)

        if ratio < 0.08333:
            r=0
            g=0
            b=0
        if ratio >= 0.08333 and ratio < 0.25:
            coef=(ratio-(2/24))*(1/(0.25-0.08333))
            r = np.interp(coef, [0, 1], [color_noche[0], color_inicio[0]])
            g = np.interp(coef, [0, 1], [color_noche[1], color_inicio[1]])
            b = np.interp(coef, [0, 1], [color_noche[2], color_inicio[2]])
        if ratio >= 0.25 and ratio< 0.5:
            coef=(ratio-0.25)*1/(0.25)
            r = np.interp(coef, [0, 1], [color_inicio[0], color_medio[0]])
            g = np.interp(coef, [0, 1], [color_inicio[1], color_medio[1]])
            b = np.interp(coef, [0, 1], [color_inicio[2], color_medio[2]])
        if ratio >= 0.5 and ratio < 0.875:
            coef=(ratio-0.5)*1/(0.375)
            r = np.interp(coef, [0, 1], [color_medio[0], color_final[0]])
            g = np.interp(coef, [0, 1], [color_medio[1], color_final[1]])
            b = np.interp(coef, [0, 1], [color_medio[2], color_final[2]])
        if ratio >= 0.875:
            coef=(ratio-0.875)*(1/0.125)
            r = np.interp(coef, [0, 1], [color_final[0], color_noche[0]])
            g = np.interp(coef, [0, 1], [color_final[1], color_noche[1]])
            b = np.interp(coef, [0, 1], [color_final[2], color_noche[2]])


        colores.append([r, g, b])
    # Mejor no, para que el negro coincida con las 12 nochecolores =␣
↪colores[3600*21:] + colores[:3600*21]
```

```
    return colores

n_colores=86400
colores_festivos=mapaColores(True,n_colores)
colores_diarios=mapaColores(False,n_colores)

def showMapa():
    global colores_diarios,colores_festivos,n_colores
    # Crear un mapa de colores personalizado
    cyclic_cmap_f = LinearSegmentedColormap.from_list("cyclic_cmap_f",␣
 ↪colores_festivos, N=n_colores)
    cyclic_cmap_d = LinearSegmentedColormap.from_list("cyclic_cmap_d",␣
 ↪colores_diarios, N=n_colores)

    # Crear una muestra de gradiente de colores
    gradient = np.linspace(0, 1, n_colores).reshape(1, -1)
    gradient = np.vstack((gradient, gradient))

    # Crear una figura para visualizar el mapa de colores
    # Crea una figura para visualizar el mapa de colores con dos subplots␣
 ↪idénticos
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 2))
    ax1.set_title('Mapa de Colores festivos')
    ax1.set_yticklabels([])  # Oculta etiquetas en el eje Y
    ax1.set_xticklabels([])  # Oculta etiquetas en el eje X
    ax1.imshow(gradient, aspect='auto', cmap=cyclic_cmap_f)

    ax2.set_title('Mapa de Colores diarios')
    ax2.set_yticklabels([])  # Oculta etiquetas en el eje Y
    ax2.set_xticklabels([])  # Oculta etiquetas en el eje X
    ax2.imshow(gradient, aspect='auto', cmap=cyclic_cmap_d)

    plt.show()
showMapa()
```



[250]:
```
def circulo(datos,h,t,r=1,col=[255,200,200]):
    if r==1:
        datos[h,t]=col
```

```
    else:
        datos[max(0,h-1),  max(0,t-1)]=col
        datos[h,           max(0,t-1)]=col
        datos[min(127,h+1),max(0,t-1)]=col
        datos[max(0,h-1),t]=col
        datos[h,t]=col
        datos[min(127,h+1),t]=col
        datos[max(0,h-1),  min(127,t+1)]=col
        datos[h,           min(127,t+1)]=col
        datos[min(127,h+1),min(127,t+1)]=col


def datos(n=39280,fase=1):
    global colores_festivos,fondo,potencias_df,saltos_df
    filas = potencias_df.loc[max(n - 127, 0):n]

    sfase='fase'+str(fase-1)
    try:
        festivo=filas.iloc[127]['ambientes'][0]['festivo']
    except IndexError:
        festivo=False
    tbase=int(filas.iloc[0]['tiempo'].value/1e9 )
    tfin=int(filas.iloc[127]['tiempo'].value/1e9 )
    timezone = pytz.timezone("Europe/Madrid")
    timestamp1 = datetime.utcfromtimestamp(tbase)
    timestamp2 = datetime.utcfromtimestamp(tfin)
    local_time1 = timezone.localize(timestamp1)
    local_time2 = timezone.localize(timestamp2)
    local_time1 =timestamp1.replace(tzinfo=pytz.utc).astimezone(timezone)
    local_time2 =timestamp2.replace(tzinfo=pytz.utc).astimezone(timezone)
    time_str = local_time1.strftime("%d-%m-%y %H:%M")+" -> "+local_time2.
 ↪strftime("%H:%M")
    # Imprime la hora local
    print(time_str, end='\r')
    mapa=colores_diarios
    if(festivo):
        mapa=colores_festivos
    fondo=[]




    for i in range(128):
        t=int(filas.iloc[i]['tiempo'].value/1e9 ) # hora unix
        t=(t+3600*2)%86400 #segundo del dia
        rgb=mapa[t]
        rgb = [(int)(c * 256) for c in rgb]
```

```
        fondo.append(rgb)

    #
    datos = np.full((128, 128, 3), fondo, dtype=np.uint8)
    for i in range(128):
        if fase != -1:
            h=filas.iloc[i][sfase]
        else:
            h=filas.iloc[i]["fase0"]+filas.iloc[i]["fase1"]+filas.
↪iloc[i]["fase2"]

        h=min(127,int(h/20))
        datos[127-h,i]=[255,255,255]

    if fase != -1:
        candidatos=saltos_df.loc[(saltos_df['tiempo'] >=filas.
↪iloc[0]['tiempo'])&(saltos_df['tiempo']<=filas.
↪iloc[127]['tiempo'])&(saltos_df['fase']==fase)]
    else:
        candidatos=saltos_df.loc[(saltos_df['tiempo'] >=filas.
↪iloc[0]['tiempo'])&(saltos_df['tiempo']<=filas.iloc[127]['tiempo'])]

    # print(fase)
    tlength=int(filas.iloc[127]['tiempo'].value/1e9 )-int(filas.
↪iloc[0]['tiempo'].value/1e9 )
    for i,val in candidatos.iterrows():
        salta=int(val['hasta'])-int(val['desde'])
        h2=abs(salta)
        h=min(127,int(h2/20))
        t=int(val['tiempo'].value/1e9 )-tbase

        t=(int)(min(128*(t/tlength),127))
        # print(salta)
        if salta> 0:
            #datos[h,t]=[255,200,200]
            circulo(datos,h,t,2 if h2>500 else 1,[255,200,200])
        else:
            #datos[h,t]=[200,255,200]
            circulo(datos,h,t,2 if h2>500 else 1,[200,255,200])
        if 750 <= abs(salta) <= 780: circulo(datos,h,t,2,[255,0,0])

    return [datos,time_str]
```

[269]:
```
# ini=17806 1 de octubre
ini=55000
ini=17806
fin=potencias_df.shape[0]
```

```
fin=55607
# fin=18000
for i in range(ini,fin):
    [dato1,time_str1]=datos(i,1)
    [dato2,time_str2]=datos(i,2)
    [dato3,time_str3]=datos(i,3)
    [datoA,time_str4]=datos(i,-1)
    # data_img(dato1,"1_"+str(i).zfill(5),"")
    # data_img(dato2,"2_"+str(i).zfill(5),"")
    # data_img(dato3,"3_"+str(i).zfill(5),"")
    # data_img(datoA,"A_"+str(i).zfill(5),"")
    data_4img(dato1,dato2,dato3,datoA,str(i).zfill(5),time_str4,False)
```

27-10-23 06:03 -> 08:10

Generación de un video a partir de secuencia de imágenes (con ayuda de chatGPT)

[ ]:
```
!ffmpeg -y -framerate 100 -start_number 17806 -i d:/output/tfm/%05d.png -vf␣
↪scale=512:512,pad=1280:720:(1280-512)/2:(720-512)/2 -c:v libx264 -pix_fmt␣
↪yuv420p d:/output/tfm/fase1.mp4
```

Aplicación Android: MainActivity.kt

```kotlin
package com.helioserver.myapplication

import android.app.Activity
import android.app.AlertDialog
import android.content.Context
import android.content.DialogInterface
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.util.Log
import android.widget.Toast
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.selection.selectable
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material3.Button
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.RadioButton
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.MutableState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.graphics.RectangleShape
import androidx.compose.ui.graphics.Shadow
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.semantics.Role
import androidx.compose.ui.text.TextStyle
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.compose.ui.unit.sp
import com.helioserver.myapplication.ui.theme.MyApplicationTheme

import com.mongodb.ConnectionString
import com.mongodb.MongoClientSettings
import com.mongodb.ServerApi
import com.mongodb.ServerApiVersion
import com.mongodb.client.model.Filters.eq
import com.mongodb.kotlin.client.MongoClient

import kotlinx.coroutines.runBlocking
```

```
import org.json.JSONObject
import java.lang.Thread.sleep
import java.net.URL
import kotlin.math.sqrt

import info.mqtt.android.service.MqttAndroidClient
import org.eclipse.paho.client.mqttv3.IMqttActionListener
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken
import org.eclipse.paho.client.mqttv3.IMqttToken
import org.eclipse.paho.client.mqttv3.MqttAsyncClient
import org.eclipse.paho.client.mqttv3.MqttCallback
import org.eclipse.paho.client.mqttv3.MqttConnectOptions
import org.eclipse.paho.client.mqttv3.MqttException
import org.eclipse.paho.client.mqttv3.MqttMessage
import org.json.JSONArray


/*********************************************************************************************
 *                            Math
 *********************************************************************************************/
val valores = arrayOf(0.0, 0.0, 0.0, 0.0, 0.0)


fun calcularMedia(valores: Array<Double>): Double {
    val suma = valores.sum()
    return suma / valores.size
}

var mqttClient: MqttAndroidClient? =null


fun calcularDesviacionEstandar(valores: Array<Double>): Double {
    val media = calcularMedia(valores)
    val sumaDeCuadrados = valores.map { (it - media) * (it - media) }.sum()
    return sqrt(sumaDeCuadrados / valores.size)
}

fun formatearNumero(numero: Double, decimales: Int=0): String {
    return String.format("%."+decimales+"f", numero)
}


var iniVal=0.0
var finVal=0.0
var esperandoFin=false //
var sentido="Encendido"

fun calcula():Boolean {
    val media = calcularMedia(valores)
    val desviacionEstandar = calcularDesviacionEstandar(valores)
    val lim=10
    val mediaFormateada = formatearNumero(media)
    val desviacionFormateada = formatearNumero(desviacionEstandar)

    infoMath.value=mediaFormateada+" +- "+desviacionFormateada+" W"

    val apagando=sentido=="Apagado"
    val estable=desviacionEstandar/media<0.01
    colorCnt.value=if(estable)Color.Green else Color.Red


```

```
120
121     //Salto a estado superior esquina. Espera allí mientras sea estable
122     if( estable&&!esperandoFin)
123         iniVal=media
124
125     //Salto a estado inferior esquina. iniVal consolidado. Está así para depurar
126     if(!esperandoFin)
127         if(iniVal!=0.0)
128             if(!estable)
129                 esperandoFin=true
130
131
132     var condicion1=if(apagando)media<=iniVal-lim else media >= iniVal+lim
133
134     if(esperandoFin && condicion1 && estable) {finVal = media;return true}
135     return false
136 }
137
138
139 lateinit var mContext:Context
140
141 class MainActivity : ComponentActivity() {
142
143
144
145     /**
146      * No usado, porque MongoDb no funciona en kotlin sin realm
147      */
148     data class Aparatos(val _id: String, val nombre: String, val fase: Int, val potencia:
    Int, val tipo: Int)
149
150
151     fun mongo()
152 {
153     // Replace the placeholder with your Atlas connection strin
154     var connectionString = "mongodb://noctiluca:**********@ac-70ujafr-shard-00-
    00.o22eilj.mongodb.net:27017,ac-70ujafr-shard-00-01.o22eilj.mongodb.net:27017,ac-70ujafr-
    shard-00-02.o22eilj.mongodb.net:27017/?ssl=true&replicaSet=atlas-srdkbk-shard-0&authSource=
    admin&retryWrites=true&w=majority"
155     connectionString = "mongodb://noctiluca:**********@ac-70ujafr-shard-00-
    00.o22eilj.mongodb.net:27017,ac-70ujafr-shard-00-01.o22eilj.mongodb.net:27017,ac-70ujafr-
    shard-00-02.o22eilj.mongodb.net:27017/tfmJavi?replicaSet=atlas-srdkbk-shard-0&retryWrites=
    true&w=majority"
156     //connectionString = "
    mongodb+srv://noctiluca:**********@cluster0.o22eilj.mongodb.net/?retryWrites=true&w=
    majority"
157
158     val serverApi = ServerApi.builder()
159         .version(ServerApiVersion.V1)
160         .build()
161     val mongoClientSettings = MongoClientSettings.builder()
162         .applyConnectionString(ConnectionString(connectionString))
163         .serverApi(serverApi)
164         .build()
165         // Create a new client and connect to the server
166     MongoClient.create(mongoClientSettings).use { mongoClient ->
167             val database = mongoClient.getDatabase("tfmJavi")
168             runBlocking {
169                 //database.runCommand(Document("ping", 1))
170                 val collection = database.getCollection<Aparatos>("aparatos")
171                 println(collection)
172                 val doc = collection.find(eq("nombre", "Freidora")).firstOrNull()
173                 if (doc != null) {
```

```
                println(doc)
            } else {
                println("No matching documents found.")
            }
        }
        println("Pinged your deployment. You successfully connected to MongoDB!")
    }
    //val client = MongoClient.create(settings)


}

/*

val TAGMQTT="MqttLog"
val MQTTTOPIC="/tfmJavi/android/prueba"
fun conectar() {
    try {
        if(true) {
            Log.d(TAGMQTT,"Iniciando intento de conexión")
            if(mqttClient  ==null)return
            Log.d(TAGMQTT,"Hay ya un objeto")
            //if(mqttClient!!.isConnected)                    return
            //Log.d(TAGMQTT,"...y no está conectado")
            mqttClient?.setCallback(MyMqttCallback(this,applicationContext))
            mqttOptions.userName="noctiluca"
            mqttOptions.password="NoctilucaESDN".toCharArray()
            mqttOptions.isAutomaticReconnect=true
            var token = mqttClient?.connect(mqttOptions)
            token?.actionCallback=conexionCB
        }
        else
            mqttClient?.connect(mqttOptions, null, conexionCB)
    } catch (e: MqttException) {
        e.printStackTrace()
    }

}

*/

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            MyApplicationTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    pantalla()
                }
            }
        }
        textoCnt.value=""
        mContext= this@MainActivity

        if(mqttClient==null)mqtt(this)
        conectar()

    }
```

```
235
236     override fun onPause() {
237         super.onPause()
238         dejademedir()
239         fase="2"
240     }
241     //*************************************************************************
242     //                                MQTT
243     //*************************************************************************
244
245
246     // https://www.emqx.com/en/blog/android-connects-mqtt-using-kotlin
247
248     companion object {  //Creo que es equivalente a static
249         const val TAGMQTT = "Regatas_MqttClient"
250         const val TAGVIDA = "Regatas_Ciclo"
251     }
252
253     fun unsubscribe(topic: String) {
254         try {
255             mqttClient?.unsubscribe(topic, null, object : IMqttActionListener {
256                 override fun onSuccess(asyncActionToken: IMqttToken?) {
257                     Log.d(TAGMQTT, "Unsubscribed to $topic")
258                 }
259
260                 override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?)
    {
261                     Log.d(TAGMQTT, "Failed to unsubscribe $topic")
262                 }
263             })
264         } catch (e: MqttException) {
265             e.printStackTrace()
266         }
267     }
268
269     fun disconnect() {
270         try {
271             mqttClient?.disconnect(null, object : IMqttActionListener {
272                 override fun onSuccess(asyncActionToken: IMqttToken?) {
273                     Log.d(TAGMQTT, "Disconnected")
274                 }
275
276                 override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?)
    {
277                     Log.d(TAGMQTT, "Failed to disconnect")
278                 }
279             })
280         } catch (e: MqttException) {
281             e.printStackTrace()
282         }
283     }
284
285     fun subscribe(topic: String, qos: Int = 0) {
286         try {
287             mqttClient?.subscribe(topic, qos, null, object : IMqttActionListener {
288                 override fun onSuccess(asyncActionToken: IMqttToken?) {
289                     Log.d(TAGMQTT, "Subscribed to $topic")
290                 }
291
292                 override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?)
    {
293                     Log.d(TAGMQTT, "Failed to subscribe $topic")
```

```
                }
            })
        } catch (e: MqttException) {
            e.printStackTrace()
        }
    }

    fun publish(topic: String, msg: String, qos: Int = 0, retained: Boolean = false) {
        try {
            val message = MqttMessage()
            message.payload = msg.toByteArray()
            message.qos = qos
            message.isRetained = retained
            mqttClient?.publish(topic, message, null, object : IMqttActionListener {
                override fun onSuccess(asyncActionToken: IMqttToken?) {
                    Log.d(TAGMQTT, "$msg published to $topic")
                }

                override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?)
{
                    Log.d(TAGMQTT, "Failed to publish $msg to $topic")
                }
            })
        } catch (e: MqttException) {
            e.printStackTrace()
        }
    }

    val mqttOptions = MqttConnectOptions()

    class MyMqttCallback( private val context: Context) : MqttCallback {

        override fun messageArrived(topic: String?, message: MqttMessage?) {
            Log.d(TAGMQTT, "Receive message: ${message.toString()} from topic: $topic")
            // {boya:1,coordenadas:{lat:36.717583,lon:-4.36260}}
            // topic propuesta android/regata/campo
            val json = JSONArray(message.toString())
        }

        override fun connectionLost(cause: Throwable?) {
            Log.d(TAGMQTT, "Connection lost ${cause.toString()}")
            //activity.disconnect()
            //connect(applicationContext)
            //connect()

        }

        override fun deliveryComplete(token: IMqttDeliveryToken?) {

        }

    }

    fun mqtt(context:Context)    {
        mqttClient= MqttAndroidClient(context, serverURI, MqttAsyncClient.generateClientId()
).apply{
            //setForegroundService(notis)
        }
        return
    }

    val serverURI = "tcp://broker.feliperomero.es:1883"
```

```
354
355     val MQTT_TOPIC="tfmJavi/android/prueba"
356
357     val conexionCB: IMqttActionListener = object : IMqttActionListener {
358         override fun onSuccess(asyncActionToken: IMqttToken?) {
359             Log.d(TAGMQTT, "Connection success")
360             subscribe(MQTT_TOPIC, 0)
361             enviaPrueba(mfase =fase)
362
363         }
364         override fun onFailure(asyncActionToken: IMqttToken?, exception: Throwable?) {
365             Log.d(TAGMQTT, "Connection failure")
366             if (mqttClient!!.isConnected) subscribe(MQTT_TOPIC, 0)
367         }
368     }
369
370     fun conectar() {
371         try {
372             if(true) {
373                 Log.d(TAGMQTT,"Iniciando intento de conexión")
374                 if(mqttClient==null)return
375                 Log.d(TAGMQTT,"Hay ya un objeto")
376                 //if(mqttClient!!.isConnected)                    return
377                 //Log.d(TAGMQTT,"...y no está conectado")
378                 mqttClient?.setCallback(MyMqttCallback(this))
379                 mqttOptions.userName="noctiluca"
380                 mqttOptions.password="NoctilucaESDN".toCharArray()
381                 mqttOptions.isAutomaticReconnect=true
382                 var token = mqttClient?.connect(mqttOptions)
383                 token?.actionCallback=conexionCB
384             }
385             else
386                 mqttClient?.connect(mqttOptions, null, conexionCB)
387         } catch (e: MqttException) {
388             e.printStackTrace()
389         }
390
391     }
392
393
394     fun enviaPrueba(mfase:String=fase, nombre:String= nombreAparato, potencia:String="1024")
395     {
396         val body= getJson(mfase,nombre,potencia)
397         publish(MQTT_TOPIC,body)
398     }
399
400 }
401
402 val textoCnt = mutableStateOf("0")
403 var cnt=0
404 val infoMath = mutableStateOf("0 ± 0W")
405 val colorCnt = mutableStateOf(Color.Red)
406 val textoBtn = mutableStateOf("Iniciar medida")
407 var preguntando=false
408 var fase="2"
409 var grabando=false
410 var nombreAparato : String="Luz de la cocina"
411 var salto=0
412
413
414
```

```
fun getJson(fase:String,nombre:String,potencia:String):String
{
    return "{\n" +
        "\t\"fase\": "+fase+",\n" +
        "\t\"nombre\": \""+nombre+"\",\n" +
        "\t\"potencia\": "+potencia+"\n" +
        "}\n"
}

var candidatoJson=""
fun finThread(contexto:Context)
{
    //if(preguntando)return
    //preguntando=true
    val alert=AlertDialog.Builder(contexto)
    salto=(finVal- iniVal).toInt()
    //val sfase=(fase.toInt()+1).toString()
    candidatoJson=getJson(fase, nombreAparato,salto.toString())
    alert.setMessage(candidatoJson)
    alert.setTitle("¿Desea almacenar esta medida?")
    alert.setPositiveButton("Si"){ dialog, which ->diceSi(contexto)}
    alert.setNegativeButton("No"){ dialog, which ->diceNo(contexto)}
    alert.setNeutralButton("Inténtalo otra vez"){ dialog,which -> noDicena() }
    alert.setCancelable(false)
    alert.show()
    }


fun diceSi(contexto:Context){
    //thread!!.interrupt()
    dejademedir()
    var mensaje="Enviado "+ candidatoJson
    Toast.makeText(contexto,mensaje, Toast.LENGTH_SHORT).show()
    MainActivity().publish("tfmJavi/demonio/inserta", candidatoJson)
}
fun diceNo(contexto:Context){
    dejademedir()
}

fun resetea(){
    iniVal=0.0
    finVal=0.0
    esperandoFin=false
    preguntando=false
}
fun continua(){
    resetea()
    cnt=0
}
fun dejademedir()
{
    resetea()
    cnt=0
    textoBtn.value="Iniciar"
    grabando=false
    textoCnt.value=""
    infoMath.value=""
}

fun noDicena(){
resetea()
```

```
476 }
477
478
479 /**
480  * Trabajador que se pone en marcha con start y finaliza con stop
481  * Básicamente es un bucle indefinido que cada X tiempo, mide la potencia
482  *
483  * @author JRC
484  */
485
486 //var thread: Thread? =null
487 fun nuevoThread(){
488
489     if(!grabando)return
490     val thread = Thread(Runnable {
491
492         while (grabando) {
493             sleep(1000)
494             var valor=""
495             if(grabando)
496                 try {
497                     val fasem=(fase.toInt()-1).toString()
498                     val apiResponse = URL("http://casa.vereda.es:60227/emeter/"+fasem)
    .readText()
499                     //val stringBuilder: StringBuilder = StringBuilder(apiResponse)
500                     val jsonObject = JSONObject(apiResponse)
501                     valor = jsonObject.getString("power")
502                     textoCnt.value =valor+" w"
503                     valores[cnt%5]=valor.toDouble()
504                     cnt=cnt+1
505                     val condicion=calcula()
506
507                     if(condicion&&!preguntando&&grabando){
508                         preguntando=true
509                         Handler(Looper.getMainLooper()).post { finThread(mContext)}
510                     }
511
512
513                 } catch (e: Exception) {
514                     // handle the exception here
515                     textoCnt.value="error"
516                 }
517
518         }
519     })
520     thread!!.start()
521
522 }
523
524 fun test(): (DialogInterface, Int) -> Unit {
525     return TODO("Provide the return value")
526 }
527
528 @Composable
529 fun radioSentido() {
530     val selectedValue = remember { mutableStateOf("Encendido") }
531     val isSelectedItem: (String) -> Boolean = {
532         selectedValue.value == it
533
534     }
535     val items = listOf("Encendido", "Apagado", "Ciclo")
```

```
    val onChangeState: (String) -> Unit = { selectedValue.value = it; sentido=it}

    Row(Modifier.padding(8.dp)) {
        Text(text = "Tipo: ")
        items.forEach { item ->
            Row(
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier
                    .selectable(
                        selected = isSelectedItem(item),
                        onClick = {
                            onChangeState(item);
                        },
                        role = Role.RadioButton
                    )
                    .padding(8.dp)
            ) {
                RadioButton(
                    selected = isSelectedItem(item),
                    onClick = null
                )
                Text(
                    text = item,
                    modifier = Modifier
                )
            }
        }
    }
}


@Composable
fun radioFases() {
    val selectedValue = remember { mutableStateOf("2") }

    val isSelectedItem: (String) -> Boolean = { selectedValue.value == it }
    val onChangeState: (String) -> Unit = { selectedValue.value = it;  fase=it}

    val items = listOf("1", "2", "3")
    Row(Modifier.padding(8.dp)) {
        Text(text = "Fase: ")
        items.forEach { item ->
            Row(
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier
                    .selectable(
                        selected = isSelectedItem(item),
                        onClick = { onChangeState(item) },
                        role = Role.RadioButton
                    )
                    .padding(8.dp)
            ) {
                RadioButton(
                    selected = isSelectedItem(item),
                    onClick = null
                )
                Text(
                    text = item,
                    modifier = Modifier
                )
            }
```

```
        }
    }
}


@Composable
fun infoPotencia(name: MutableState<String>, modifier: Modifier = Modifier) {
    //var tttB by remember { mutableStateOf("Prueba1") }
    Text(
        //text = name.value
        text= textoCnt.value,
        color= colorCnt.value,
        style = TextStyle(
            fontSize = 54.sp,
            shadow = Shadow(
                color = Color.Blue,
                blurRadius = 3f
            )),
        //modifier = modifier,
        modifier = modifier.padding(15.dp)

    )
}


@Composable
fun infoEstadistica() {
    Text(
        //text = name.value
        text=infoMath.value,
        style = TextStyle(
            fontSize = 24.sp,
            shadow = Shadow(
                color = Color.Blue,
                blurRadius = 3f
            )),
        //modifier = modifier,
        modifier = Modifier.padding(20.dp)
    )
}
/*****************************************************************************/

/*****************************************************************************/

@Composable
fun barraArriba(name: MutableState<String>, modifier: Modifier = Modifier) {
    var tttB by remember { mutableStateOf("Prueba1") }
    val actividad = (LocalContext.current as? Activity)
    Row(
        modifier = Modifier
            .background(Color.Blue, RectangleShape)
            .padding(6.dp)
            .fillMaxWidth()

    ){
        Text(
            text = name.value,
            //text=textoCnt.value,
            color=Color.White,
            style = TextStyle(
                fontSize = 24.sp,
```

```
                shadow = Shadow(
                    color = Color.Blue,
                    blurRadius = 3f
                ))
            //modifier = modifier,
        )
        Icon(
            //imageVector=ImageVector.vectorResource(R.drawable.eii_background),
            //Icons.Rounded.Build,
            painter= painterResource(id =R.drawable.eii_icon ),
            contentDescription = "eo",
            tint= Color.White,
            modifier= Modifier
                .fillMaxWidth()
                .clickable { actividad?.finish() }

        )
    }
}



fun clickBoton()
{
    if(grabando) {
        dejademedir()
        //textoBtn.value="Iniciar"
        //tttTexto.value=prueba2
        //grabando=false

    }else {
        textoBtn.value="Parar"
        grabando=true
        nuevoThread()
    }
}


@Composable
fun idAparatoBox() {
    var value by remember { mutableStateOf("12") }
    TextField(
        value = value,
        onValueChange = { value = it },
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number),
        label = { Text("Id del aparato (opcional)") },
        maxLines = 2,
        modifier = Modifier.padding(20.dp)
    )
}

@Composable
fun botonPpal() {
    //var textoB by remember { mutableStateOf("Iniciar medición") }
    val contexto=LocalContext.current
    Button(onClick = {
        Toast.makeText(contexto,"Iniciando medida", Toast.LENGTH_SHORT).show()
        /*
        if(grabando) {
            //textoB="Iniciar med"
```

```
            grabando=false

        }else {
            //textoB="Parar med"
            grabando=true
            nuevoThread()
        }
        */

        clickBoton() // No usada
        //your onclick code here
    },
            modifier=Modifier.padding(20.dp)
    ) {
        Text(
            //text = textoB,
            text = textoBtn.value,
            style = TextStyle(
            fontSize = 24.sp
            ))
    }
}

@Composable
fun etiquetaAparatoBox() {
    var text by remember { mutableStateOf(nombreAparato) }

    TextField(
        value = text,
        onValueChange = { text=it; nombreAparato=it },
        label = { Text("Nombre del aparato") },
        modifier=Modifier.padding(20.dp)
    )
}


@Composable
fun pantalla(){
    var etiqueta=remember { mutableStateOf("Medidor de Aparatos") }
    var prueba2=remember { mutableStateOf("Texto global") }
    Column {
        barraArriba(name=etiqueta)
        radioSentido()
        radioFases()
        etiquetaAparatoBox()
        botonPpal()
        infoPotencia(name=etiqueta)
        infoEstadistica()
        idAparatoBox()
    }

}


@Preview(showBackground = true, showSystemUi = true)
@Composable
fun MiPreview() {
    MyApplicationTheme {
        pantalla()
    }
}
```

Anexo D

Presupuesto de instalación

Para la instalación del sistema de medida y almacenamiento básico, que incluya los sensores y un mini-PC en el que se instalen el servidor web, el broker MQTT Mosquitto, el servicio demonioMQTT y las bases de datos, se calcula un presupuesto aproximado de 275,17€:

- Sensor Shelly 3EM, 120,88€
- Medidor de energía Gifort, 10,99€
- Raspberry Pi model 4, 69,85€
- Alimentador para Raspberry, 12,46€
- Memoria Sandisk High Endurance 256GB, 20,99€
- SAI Tecnoware ERA PLUS 750VA, 40€

Para un acceso constante, e independiente de la red, de mediciones de temperatura y sensación térmica, especialmente en viviendas en las que no hay una estación AEMET en las proximidades, se propone la adquisición de material adicional por un coste de 134,12€:

- Estación PRO SIGNAL PSG04174, 69,12€
- Licencia software Meteobridge, 65€