Trabajo de Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Predicción de demanda y generación renovable con Deep Learning: Aplicación a la optimización de estaciones de carga de vehículos eléctricos

Autor: Francisco Ramos Pérez

Tutor: Francisco Rodríguez Rubio y Carlos Vivas Venegas

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo de Fin de Grado
Grado en Ingeniería de Tecnologías Industriales

Predicción de demanda y generación renovable con Deep Learning: Aplicación a la optimización de estaciones de carga de vehículos eléctricos

Autor:
Francisco Ramos Pérez

Tutor:
Francisco Rodríguez Rubio y Carlos Vivas Venegas
Catedrático y Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo de Fin de Grado: Predicción de demanda y generación renovable con Deep Learning: Aplicación a la optimización de estaciones de carga de vehículos eléctricos

Autor: Francisco Ramos Pérez
Tutor: Francisco Rodríguez Rubio y Carlos Vivas Venegas

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

Agradecimientos a mis padres, hermanos, familiares, novia, amigos y a todas las personas que me han ayudado a llegar aquí.

A todos los profesores que se han preocupado por nuestra formación y por que seamos buenas personas. Especialmente a D. Pedro Álvarez y a D. José María Gallardo, que siempre estarán en nuestro recuerdo.

Francisco Ramos Pérez
Sevilla, 2022

Resumen

Los vehículos eléctricos se están popularizando y son claves para el transporte del futuro debido a su contribución en la reducción de las emisiones de carbono. Uno de los desafíos clave es el soporte que tendrá que dar la infraestructura de la red a todas las estaciones de carga de vehículos eléctricos que se están implantando a gran escala. La solución pasa por la utilización de algoritmos de planificación inteligentes para gestionar la creciente demanda de carga. El uso de técnicas basadas en datos y de *machine learning* para aprender el comportamiento de la carga de vehículos eléctricos y de la generación fotovoltaica pueden servir para mejorar estos algoritmos de planificación. Por tanto, en este Trabajo de Fin de Grado se propone un simulador de una estación y un algoritmo de planificación dinámica de la demanda de carga de la estación. Este algoritmo hace uso de predicciones de la demanda y de la producción fotovoltaica generadas por modelos de redes neuronales. Estos modelos secuenciales han sido entrenados con datos obtenidos de una base de datos pública en el caso de la demanda y de una planta fotovoltaica real en el caso de la producción. En predicción de la demanda, el modelo que ha obtenido mejores resultados ha sido el modelo con redes LSTM con una ventana temporal de 4 días, obteniendo un MAE de 4.41 kW y un RMSE de 4.10 kW sobre los datos de testeo. En la predicción de la generación, el mejor modelo ha resultado ser el modelo de redes CNN+LSTM con una ventana temporal de 1 día, obteniendo un MAE de 55.60 kW y un RMSE de 104.61 kW sobre los datos de testeo.

Abstract

Electric vehicles are becoming increasingly popular and are key to the transport of the future due to their contribution to reducing carbon emissions. One of the key challenges, however, is how the grid infrastructure could provide support to all the electric vehicle charging stations that comes with large-scale EV deployment. The solution to this lies in the utilization of smart scheduling algorithms to manage the growing public charging demand. The use of data-driven techniques and machine learning to learn the behaviour of electric vehicle charging and photovoltaic generation can be used to improve these scheduling algorithms. Therefore, in this Final Degree Project, a station simulator and a smart scheduling algorithm to manage charging demand are proposed. This algorithm makes use of demand and photovoltaic production forecasting generated by neural network models. These sequential models have been trained with data obtained from a public database in the case of demand and from a real PV plant in the case of production. In demand forecasting, the model that obtained the best results was the model with LSTM networks with window size of 4 days, obtaining an MAE of 4.41 kW and an RMSE of 4.10 kW on the test data. In generation forecasting, the best model was the CNN+LSTM networks model with a window size of 1 day, obtaining an MAE of 55.60 kW and an RMSE of 104.61 kW on the test data.

Índice Abreviado

Índice

Notación

EV	*Electric Vehicle*, Vehículo Eléctrico
AC	*Alternating Current*, Corriente Alterna
DC	*Direct Current*, Corriente Continua
V2G - VG2	*Vehicle to Grid - Grid to Vehicle*, Red a Vehículo y Vehículo a Red
PV	*Photovoltaic*, Fotovoltaico
MPPT	*Maximum Power Point Tracking*, Seguimiento de punto de máxima potencia
BMS	*Battery Management System*, Sistema de gestión de baterías
LCL	Filtro de bobinas y condensadores en serie y en paralelo
w	*Weights*, pesos
b	*Bias*, sesgos
σ	Función de activación sigmoide
$\mathscr{J}$	Función de coste
$\mathscr{L}$	Función de similitud
$\hat{y}^{[i]}$	Salida predicha muestra i
$y^{[i]}$	Salida real muestra
W	Matriz de pesos
$x^{\langle t \rangle}$	Entrada instante temporal t
$y^{\langle t \rangle}$	Salida instante temporal t
$a^{\langle t \rangle}$	Activación instante temporal t
T_x	Tamaño secuencia de entrada
T_y	Tamaño secuencia de salida
Γ_u	Puerta de actualización
Γ_f	Puerta de olvido
Γ_o	Puerta de salida
$c^{\langle t \rangle}$	Estado de la célula
$\tilde{c}^{\langle t \rangle}$	Nuevo candidato para el estado de la célula
ANN	Redes neuronales perceptrón multicapa
CNN	Redes neuronales convolucionales
RNN	Redes neuronales recurrentes
RNN-st	Modelo de redes neuronales recurrentes con capas apiladas
LSTM	Redes neuronales recurrentes *Long Short Term Memory*

Bi-RNN	Redes neuronales recurrentes bidireccionales
LSTM-st	Modelo de redes neuronales recurrentes LSTM con capas apiladas
CNN + LSTM	Modelo con capas de redes convolucionales y redes neuronales recurrentes LSTM

1 Introduccion y objetivos

El auge de los vehículos eléctricos en las carreteras eleva la necesidad y la importancia de las instalaciones de carga que abastecen de energía a estos vehículos. El insuficiente número de estaciones de carga es una de las principales problemáticas para el desarrollo de la tecnología de vehículos eléctricos. Uno de los desafíos clave es el soporte que tendrá que dar la infraestructura de la red a todas las estaciones de carga de vehículos eléctricos que se están implantando a gran escala.

Por otro lado, en la actualidad, las energías renovables están ganando importancia debido al desarrollo de nuevas formas óptimas de retener y almacenar la energía eólica y solar. A medida que el uso de las energías renovables siga creciendo, un objetivo clave será modernizar la red eléctrica.

El objetivo global de este Trabajo de Fin de Grado es proponer un simulador de una micro-red inteligente con integración de energías renovables y con almacenamiento en el sistema de energía capaz de gestionar la demanda de carga de los vehículos eléctricos y planificar la energía de una manera eficiente. La solución a este problema requiere de la utilización de algoritmos de planificación inteligentes para gestionar la creciente demanda de carga. El uso de técnicas basadas en datos y de *machine learning* para aprender el comportamiento de la carga de vehículos eléctricos y de la generación fotovoltaica pueden servir para mejorar estos algoritmos de planificación. El algoritmo planteado hace uso de predicciones de la demanda y de la producción fotovoltaica generadas por modelos de redes neuronales. Estos modelos secuenciales han sido entrenados con datos obtenidos de una base de datos pública en el caso de la demanda y de una planta fotovoltaica real en el caso de la producción.

Para alcanzar este objetivo se realiza, en primer lugar, un análisis del estado del arte de las actuales tecnologías de carga de vehículos eléctricos existentes, así como un estudio teórico sobre los componentes de una estación de vehículos eléctricos con soporte renovable. Este estudio se recoge en el Capítulo 2 del presente Trabajo de Fin de Grado.

Seguidamente, en el Capítulo 3 se realiza el modelado y la implementación de un simulador del sistema unificado de una instalación de carga inteligente de vehículos eléctricos con soporte renovable. Así, en el contexto de una estación eléctrica con soporte renovable, se visualiza cómo las fluctuaciones de la potencia disponible pueden ser compensadas y reguladas usando la tecnología V2G-G2V y un banco de baterías como almacén de energía.

A continuación, se desarrolla el principal reto de este trabajo, que es la optimización y planificación dinámica en una estación de carga de vehículos eléctricos. Este bloque temático se desarrolla en los Capítulos 4 a 7 de esta memoria. Así, previamente al planteamiento del problema de optimización y de las predicciones que serán necesarias, es necesario conocer el estado del arte de los modelos de deep learning, así como las bases teóricas de las redes neuronales. Esta información servirá como punto de partida para el desarrollo de los capítulos posteriores (Capítulo 4).

En el Capítulo 5 se aborda el problema de la optimización y planificación dinámica de la estación de carga propiamente dicha. Ello requiere realizar una predicción de la futura demanda de carga

que se va a abastecer en la estación, así como de la producción de energía fotovoltaica.

Se pretende, dentro de los objetivos de este trabajo, aplicar modernas técnicas de *deep learning* a un problema de optimización real de una micro-red de carga y, ligado a ello, analizar los resultados obtenidos con la intención de obtener el mejor modelo posible que permita predecir dichas series temporales de la demanda y de la producción.

En los Capítulos 6 y 7 se plantean los distintos modelos para la predicción de la demanda de carga y la predicción de la generación fotovoltaica. Se hará uso de datos reales previamente manipulados y procesados, obtenidos de bases de datos públicas y de una planta fotovoltaica experimental ubicada en la Escuela Técnica Superior de Ingeniería en Sevilla. En estos capítulos se incluye, además, un estudio y comparación de distintas arquitecturas de redes neuronales variando distintos hiperparámetros con la finalidad de encontrar los modelos e hiperparámetros óptimos.

2 Documentación y estudio de tecnologías de carga de vehículos eléctricos.

Los vehículos de motor tradicionales, que utilizan diesel o gasolina, emiten sustancias como el monoxido de carbono, oxidos de nitrógeno, hidrocarburos y partículas. Estas sustancias causan la contaminación del aire. La contaminación del aire representa un riesgo medioambiental para la salud de los seres vivos e impulsa el cambio climático causado por el efecto invernadero y el daño de la capa de ozono.

Hoy en día, los vehículos eléctricos forman parte de una tecnología que está creciendo exponencialmente y se espera que en un futuro cercano sustituyan completamente a los vehículos de motor tradicionales. Los vehículos eléctricos reducen las emisiones de dióxido de carbono, lo que significa una reducción considerable de la contaminación del aire. Otra ventaja de los vehículos eléctricos es que son energéticamente más eficientes, es decir, necesitan menos energía para realizar el mismo trabajo. Esto resulta en un consumo más bajo y un ahorro económico en combustible.

En este primer capítulo se realiza un estudio de las tecnologías de carga de los EV, así como de los propios puntos de carga. Posteriormente, se explican los posibles componentes del soporte de energía renovable de una estación, así como el sistema de almacenamiento de baterías. Finalmente se explica el concepto de micro-red de gestión de flujos energéticos.

2.1 Puntos de carga

Aunque las baterías se cargan con energía de corriente continua, la mayoría de los vehículos eléctricos tienen un convertidor de alterna (AC) a continua (DC) a bordo que les permite conectarse a una toma de corriente eléctrica doméstica estándar.

Para facilitar la carga de mayor potencia, que requiere convertidores de AC a DC mucho más grandes, el convertidor se incorpora a la estación de carga en lugar de al vehículo y la estación suministra energía de DC ya convertida directamente al vehículo evitando el convertidor de a bordo del vehículo. Son las llamadas “estaciones de carga de corriente continua”. La mayoría de los modelos de coches totalmente eléctricos pueden aceptar tanto AC como DC.

Existen tres niveles estándar de cargadores de vehículos eléctricos según la velocidad y energía de carga que tengan.

El nivel 1 es la instalación de carga más lenta. Los cargadores del nivel 1 se enchufan directamente a un enchufe estándar de 120 V de AC proporcionando una media de 1.3 kW a 2.4 kW de energía

de AC. Una carga total para una batería vacía de un vehículo eléctrico puede durar más de 24 horas. Suelen ser los cargadores utilizados en zonas residenciales o domésticas.

Los cargadores del nivel 2 operan a 208-240 V y proporcionan desde 3 kW hasta 19kW de potencia. Un vehículo eléctrico medio puede ser cargado completamente en menos de 8 horas. Algunos cargadores de nivel 2 pueden proporcionar más energía de la que algunos vehículos eléctricos puede recibir. Por tanto, los resultados variarán dependiendo de la combinación del punto de carga y del vehículo. Estos cargadores son muy comunes y pueden ser encontrados en muchos lugares públicos. También existe la posibilidad de realizar la instalación de un cargador de este tipo para uso doméstico, de manera que se asegure la carga completa del vehículo durante la noche.

Las cargadores de carga rápida DC (nivel 3) son los más rápidos disponibles con una potencia máxima de 350 kW a la salida. Están diseñados para llenar un 80 % de la batería de un vehículo eléctrico en 20-40 minutos, y al 100 % en 60-90 minutos. El ratio máximo de carga a menudo está limitado por la capacidad del vehículo eléctrico para recibir la potencia. La mayoría de vehículos eléctricos actuales cargan a un máximo de 50 kW, mientras que hay nuevos modelos que son capaces de cargar a 200 kW. Estos puntos de carga se encuentran normalmente en estaciones en lugares comerciales o industriales más que en zonas residenciales. Suelen estar situados en lugares de paso por carreteras, de manera que así permitan realizar largos viajes a los vehículos eléctricos. [1].

En la Tabla 2.1 se presenta un resumen comparativo de los niveles de carga en los vehículos eléctricos.

Tabla 2.1 Comparativa niveles de carga de EV.

Características	Nivel 1	Nivel 2	Nivel 3
Precio de carga	Bajo	Medio	Alto
Tipo de energía	AC	AC	DC
Velocidad	Lenta	Media	Rápida
Potencia	1.3-3.4 kW	3-19 kW	50-350 kW
Localización	Residencial	Residencial, pública, trabajo	Pública

2.2 Aparamenta e instrumentación de conexión a la red eléctrica

En este apartado se recoge una breve información sobre los distintos tipos de conectores que se utilizan actualmente en distintos países. Se diferencian entre conectores de carga lenta (AC) y rápida (DC) [2] [3] [4].

En la Figura 2.1 se muestra un resumen de los siguientes apartados.

Conectores AC

2.2.1 SAE J1772 - Tipo 1

Este conector es el estándar de la industria para todos los vehículos eléctricos que realizan la carga de nivel 1 o nivel 2 en Norte América. Todos los fabricantes utilizan el SAE J1772 excepto Tesla, que incluye en cada coche que vende un adaptador que permite a sus coches utilizar estaciones de carga que tienen conectores J1772 además de las estaciones de carga propias de Tesla que usan su propio conector.

De cualquier modo, no suele haber problema para cargar los vehículos en puntos de carga AC, ya que los vehículos suelen llevar su propio cable. La principal desventaja de este enchufe es que sólo permite el uso de una fase y no tiene incorporado un sistema de bloqueo automático.

2.2.2 Mennekes - Tipo 2

Los EV europeos utilizaban el conector SAJ1772 (tipo 1) hasta que los principales fabricantes de automóviles empezaron a buscar una nueva solución en la que se pudiera aprovechar las tres fases. En 2003 se establecieron las nuevas especificaciones IEC 62196, en base a las cuales se fabricó el enchufe tipo 2 “mennekes”, que se convirtió rápidamente en la nueva norma europea. Gracias a que ambos tipos de enchufes (tipo 1 y 2) utilizan el mismo protocolo J1772 de señalización para la comunicación, los fabricantes de automóviles pueden fabricar los vehículos de la misma manera y sólo al final instalar el tipo de enchufe que corresponda al mercado en el que se venderá el coche. También existen adaptadores pasivos entre estos tipos. Otra ventaja importante del enchufe de tipo 2 es que admite un sistema de bloqueo automático incorporado.

2.2.3 GB/T

En China, bajo la supervisión de la Comisión de Normalización de Guobiao, se desarrolló un conector GB / T y, actualmente, es el único que se utiliza. El hecho de que no existan otros tipos de conectores en todo el país que puedan competir facilita el desarrollo de la infraestructura de recarga. Hay que tener en cuenta que China es el país con la red más densa de estaciones de carga y tiene la mayor cuota de coches eléctricos del mundo.

A primera vista, el conector parece ser el mismo que el del tipo 2, pero los cables del interior están dispuestos en orden inverso, por lo que son incompatibles.

Conectores DC

2.2.4 CHAdeMO

Este es el primer conector de carga rápida (DC) que se introdujo. Originalmente se implantó para ser el estándar de la industria, desarrollado a través de la colaboración de cinco fabricantes de automóviles japoneses diferentes.

Es el estándar oficial en Japón y la mayoría de cargadores rápidos DC japoneses utilizan un conector CHAdeMO. Esto difiere en Norte América, donde Nissan y Mitsubishi son los únicos fabricantes que usan este tipo de conector, en concreto los modelos Nissan LEAF y Mitsubishi Outlander PHEV.

Kia cambió CHAdeMO en 2018 y ahora utiliza CCS, que se explica a continuación. Los conectores CHAdeMO no comparten parte del conector con la entrada J1772 (carga de nivel 1 o 2), al contrario que el sistema CCS. Por tanto, requieren una entrada adicional CHAdeMO en el coche que significa un puerto de carga más grande.

2.2.5 CCS - Tipo 1 y Tipo 2

Poco después de que se introdujera el CHAdeMO, se implementó un segundo conector llamado Sistema de Carga Combinada (CCS) como un estándar de carga adicional.

Lo que diferencia a los conectores CCS de los CHAdeMO es que permiten la carga de AC y DC en un mismo puerto, eliminando la necesidad de tener que utilizar dos conectores distintos para tener posibilidad de realizar la carga rápida además de la carga de nivel 1 o 2. Se trata de conectores originales del tipo 1 o del tipo 2 a los que se añaden dos pines más en la parte inferior. En el caso de la carga de corriente continua, estos dos pines inferiores participan en la propia carga y de la parte superior sólo se utilizan los pines de comunicación y el conductor de tierra, que proporciona el punto de referencia para los sistemas de protección. Estos conectores pueden soportar una potencia de hasta 350 kW.

Actualmente es el tipo de conector de corriente continua más popular. El tipo 1 es común en Estados Unidos, mientras que el tipo 2 de CCS se utiliza en Europa. Este conector es el preferido por los fabricantes de automóviles como BMW, Ford, Jaguar, GM, Polestar, Volkswagen, etc.

2.2.6 GB/T

Al igual que con la carga de AC, China tiene sus propias normas para la carga de DC. GB/T está trabajando actualmente con CHAdeMO para desarrollar una tercera generación de conectores que debería ser capaz de transmitir 900 kW.

2.2.7 Supercargador de Tesla

Tesla utiliza conectores diferentes a los de cualquier otra marca, lo que permite a los clientes de Tesla cargar en sus propias estaciones de carga que no pueden ser utilizadas por ningún otro vehículo.

Sin embargo, al mismo tiempo, Tesla también ofrece adaptadores para otros tipos de enchufes, por lo que para sus vehículos no es un problema utilizar las estaciones de carga con un enchufe de tipo 1 o CHAdeMO.

En la guerra por el enchufe DC predominante en Europa, Tesla se inclinó por el enchufe CCS Tipo 2 en su Modelo 3.

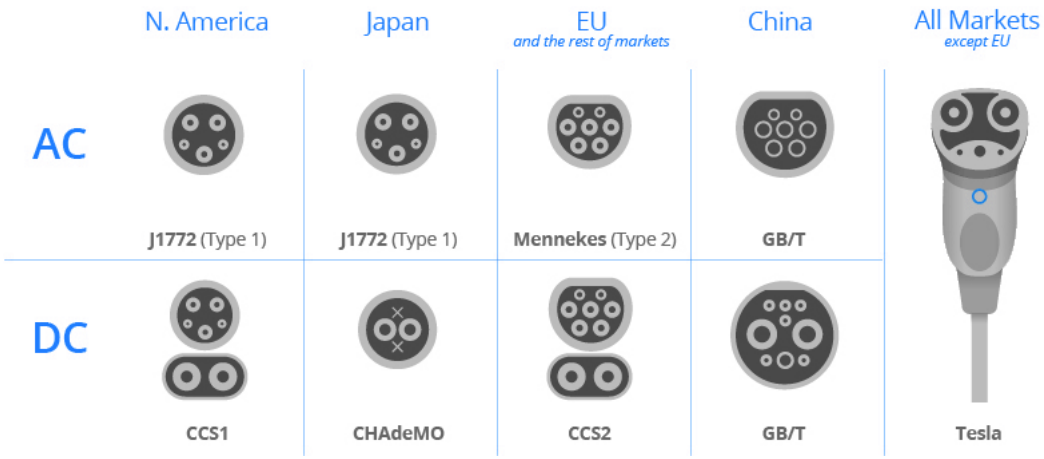


Figura 2.1 Tipos de conectores de carga de vehículos eléctricos actuales [2].

2.3 Normativa ISO 15118

Es importante conocer la norma ISO 15118 y las utilidades de las técnicas de "*Plug and Charge*" (enchufar y cargar) que, actualmente, ofrecen cada vez más fabricantes de vehículos eléctricos.

La norma ISO 15118, transcrita a norma española como UNE-EN ISO 15118-1:2015 (la última actualización es de 2019 aún no traducida a Español), se denomina como "Interfaz de comunicación entre el vehículo y la red eléctrica. Parte 1: Información general y definición de casos de uso". Esta normativa tiene por objeto mejorar las comunicaciones entre los puntos de recarga, coches eléctricos y la red eléctrica [5].

Es una interfaz de comunicación entre el vehículo y la red eléctrica (V2G) que permite la carga/descarga bidireccional de los vehículos eléctricos. Una de las principales características que utiliza esta interfaz estándar es *Plug and Charge*.

Esencialmente, el protocolo *Plug and Charge* se comunica con cualquier estación de carga en la que se enchufe y transmite el tipo de vehículo eléctrico que se está enchufando, facturando sin problemas al conductor. En este sentido, la única acción que necesitan los conductores es enchufar el cable de carga a su vehículo eléctrico. Como resultado, el sistema *Plug and Charge* elimina la

necesidad de utilizar varias tarjetas, llaveros o aplicaciones móviles requeridas por las diferentes redes de recarga que se puede encontrar en una ruta determinada.

Tesla lleva ofreciendo esta tecnología *Plug and Charge* desde hace aproximadamente una década, pero otros EV como el Mustang Mach-E, el Lucid Air y el Porsche Taycan admiten esta tecnología de forma estándar [4].

2.4 Sistemas de generación de energía renovable

La energía renovable, a menudo denominada energía limpia, procede de fuentes o procesos naturales que se reponen constantemente. Por ejemplo, la luz del sol y el viento siguen brillando y soplando, aunque su disponibilidad dependa del tiempo y la meteorología. Incluyen la bioenergía, la energía hidroeléctrica, la energía geotérmica, la energía solar, la energía eólica y la energía oceánica (de las mareas y las olas).

Ahora que se disponen de formas innovadoras y menos costosas de captar y retener la energía eólica y solar, las energías renovables están ganando importancia. En España, han llegado a constituir el 46,7 % del total de la producción eléctrica en 2021 según Red Eléctrica de España [6]. La expansión de las energías renovables se está produciendo a escalas grandes y pequeñas, desde la construcción de gigantescos parques eólicos en alta mar hasta paneles solares en los tejados de las casas que pueden vender energía a la red.

A medida que el uso de las renovables siga creciendo, un objetivo clave será modernizar la red eléctrica, haciéndola más inteligente, más segura y mejor integrada en todas las regiones.

2.4.1 Energía fotovoltaica

La energía solar fotovoltaica es una fuente de energía renovable y limpia que utiliza la radiación solar para producir electricidad. Se basa en el llamado efecto fotoeléctrico, por el cual determinados materiales son capaces de absorber fotones (partículas lumínicas) y liberar electrones, generando una corriente eléctrica.

Para ello, se emplea un dispositivo semiconductor denominado celda o célula fotovoltaica, que puede ser de silicio monocristalino, policristalino o amorfo, o bien otros materiales semiconductores de capa fina. Las de silicio monocristalino se obtienen a partir de un único cristal de silicio puro y alcanzan la máxima eficiencia, entre un 18 % y un 20 % de media. Las de silicio policristalino se elaboran en bloque a partir de varios cristales, por lo que resultan más baratas y poseen una eficiencia media de entre el 16 % y el 17,5 %. Por último, las de silicio amorfo presentan una red cristalina desordenada, lo que conlleva peores prestaciones (eficiencia media de entre un 8 % y un 9 %) pero también un precio menor [7].

2.4.2 Energía eólica

La energía eólica es aquella que se obtiene a partir de la fuerza del viento a través de un aerogenerador que transforma la energía cinética de las corrientes de aire en energía eléctrica. El proceso de extracción se realiza principalmente gracias al rotor, que transforma la energía cinética en energía mecánica y al generador, que transforma dicha energía mecánica en eléctrica [8].

2.5 Micro red de gestión de flujos energéticos

A pesar de la reducción de la contaminación por parte de las energías renovables y su ayuda a la construcción de un mundo más sostenible, las energías renovables tienen algunos puntos débiles. La proporción de electricidad de la red generada por las energías renovables depende casi totalmente de los recursos naturales que muchas veces no son constantes, como el sol o el viento. Esto puede

dar lugar a una gran fluctuación en la potencia disponible en la red, lo que puede ser un problema para las compañías eléctricas.

Sin embargo, si los hogares y las empresas tuvieran la flexibilidad de poder elegir el momento en que cargan sus vehículos eléctricos, podrían ayudar a equilibrar la red y apoyar la generación de energía renovable. Esto sería posible mediante puntos de carga inteligentes que pudieran comunicarse con la red y así ayudar a equilibrar la demanda con la oferta.

Un cargador inteligente permitirá cargar un EV cuando las condiciones meteorológicas lo permitan y haya disponible un exceso de energía no contaminante y de bajo coste. Del mismo modo, como un cargador inteligente de EV puede adaptarse a la velocidad y el tiempo de carga, también puede ayudar a minimizar la demanda en la red cuando la energía generada por las renovables sea escasa. Esto justifica la necesidad de una micro red inteligente que pueda gestionar los flujos de carga de una manera óptima.

3 Simulador de una instalación de carga de vehículos eléctricos con soporte renovable.

En este capítulo se realiza el estudio, modelado e implementación de un simulador del sistema unificado de una instalación de carga inteligente de vehículos eléctricos con soporte renovable.

Esta estación cuenta con cargadores de carga lenta (AC) y rápida (DC), un banco de baterías como sistema de almacenamiento y un panel de control para la monitorización de la estación. Además, posee soporte renovable basado en un sistema de generación con una planta fotovoltaica y con generación micro eólica mediante una turbina de viento.

Se implementará tecnología "*vehicle to grid- grid to vehicle*" (V2G-G2V) en los cargadores de corriente continua. La tecnología V2G permite que los EV, además de cargar, puedan ceder electricidad a la red, de forma que se mejore la estabilidad de ésta durante las horas de mayor demanda. Se plantean los vehículos como elementos de almacenamiento de energía. Además, cuenta con un banco de baterías que se puede cargar y descargar cuando haya excedentes de energía o cuando sea necesario por demanda. Esta es la razón por la que se requiere el diseño de una micro-red que gestione los flujos energéticos procedentes de los EV y del banco de almacenamiento.

Se explica, en primer lugar, la arquitectura general de la estación. Posteriormente, se desarrollan los diferentes componentes del sistema y los sistemas de control que han sido modelados en Matlab-Simulink. Finalmente, se realizan distintas simulaciones en las que se visualizan los distintos modos de funcionamiento de la estación y, por último, se analizan los resultados.

3.1 Arquitectura general

La instalación está alimentada principalmente por tres fuentes: la red eléctrica, los paneles fotovoltaicos y la turbina eólica. La carga de los vehículos se lleva a cabo mediante el uso de cargadores de continua y de alterna, que están conectados a un bus DC y a un bus AC respectivamente. Los paneles fotovoltaicos y el banco de baterías se conectan en paralelo al bus DC. La turbina eólica está conectada directamente a la red, de manera que la potencia llegue al sistema en caso de demanda o sea directamente cedida a la red en caso de exceso de energía. Otros componentes importantes de la micro-red son los transformadores. Estos reciben la energía de la red y proveen de baja tensión al resto de la instalación. Se añaden elementos de filtrado posteriores a la transformación [9].

El sistema cuenta con distintos convertidores que se utilizan para elevar/reducir la tensión o para transformar la naturaleza de la corriente (alterna o continua). Los convertidores que se utilizan son los siguientes:

- convertidor bidireccional AC/DC previo al bus DC.
- convertidor elevador para la conexión del panel PV al bus DC.
- convertidores bidireccionales en los puestos de carga DC, así como en el banco de baterías.

Por último, la estación cuenta con un centro de control desde el cual es posible gestionar los vehículos conectados que cargan o descargan, así como las órdenes para direccionar los flujos de potencia entre los bloques principales de la micro red. También se representa en el centro de control un balance de potencias general que puede servir para monitorizar la estación.

3.2 Componentes del sistema

3.2.1 Red

El punto de conexión con la red de distribución se realiza mediante una subestación que transforma de 120kV a 25kV. Se modela la linea de distribución de media tensión con bloques que permiten definir la longitud de la línea, así como la resistencia y capacidad por unidad de longitud. Se conecta también un transformador de puesta a tierra.

En la Tabla 3.1 se presentan los valores de los parámetros de diseño de la red de acometida y en la Figura 3.1 se recoge gráficamente el modelado.

Tabla 3.1 Parámetros de diseño de la red de acometida.

Nombre	Símbolo	Valor	Unidad
Potencia nominal de la red	S_{red}	2500	MVA
Relación de transformación del transformador	r_t	120/25	kV/kV

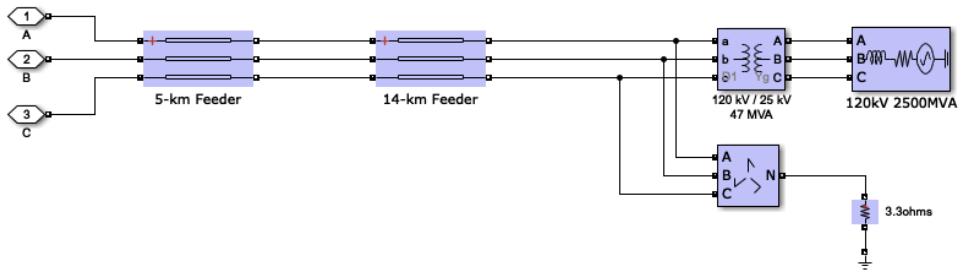


Figura 3.1 Modelado de la red de acometida.

3.2.2 Turbina eólica

Para la turbina eólica se toma un modelo de un aerogenerador de velocidad variable con control de paso mediante un generador de inducción doblemente alimentado. Ésta se conecta a un transformador elevador de 575 V a 25 kV para poder ser conectada a la red en un punto de acoplamiento común. Su implementación se ha hecho a través de un bloque precompilado de Matlab, denominado *DFIG Wind Turbine*.

En la Tabla 3.2 se presentan los valores de los parámetros de diseño de la turbina de viento y en la Figura 3.2 se recoge gráficamente el modelado.

Tabla 3.2 Parámetros de diseño de la turbina de viento.

Nombre	Símbolo	Valor	Unidad
Potencia nominal de turbina	P_w	100	kW
Factor de potencia	fdp	0.9	-
Tensión en el estátor	V_s	575	V_{rms}
Tensión en el rótor	V_r	1975	V_{rms}
Relación de transformación	r_t	25/575	kV/V

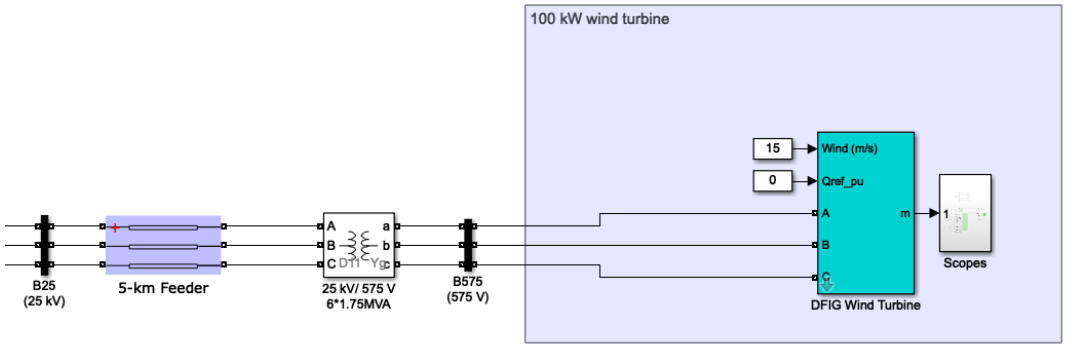


Figura 3.2 Modelado de la turbina de viento.

3.2.3 Paneles fotovoltaicos

El sistema de producción fotovoltaica está conformado por una conexión en serie y en paralelo de paneles fotovoltaicos. Estos están dispuestos en filas y columnas, que se calculan teniendo en cuenta la cantidad de potencia necesaria y la tensión deseada en los terminales de salida.

Para conseguir una tensión de salida de aproximadamente 400 V, se calcula el número de paneles en serie necesarios. Del mismo modo, para conseguir que la potencia nominal que entregue el array PV sea de 50 kW, se obtiene el número de series que están en paralelo.

Por tanto, se busca un modelo de panel cuya tensión de máxima potencia sea un submúltiplo entero de 300 y su potencia máxima, un submúltiplo de la potencia entregada por cada conexión en serie de paneles.

Se selecciona el modelo *SunPower SPR-305E-WHT-D* con $V_{mp} = 54.7V$ y $P_{max} = 305.23kW$.

Se tiene que:

$$400 = n_s * V_{mp} \tag{3.1}$$

$$n_s \approx 8 \tag{3.2}$$

$$50000 = n_p * n_s * P_{max} \tag{3.3}$$

$$n_p \approx 21 \tag{3.4}$$

Obteniendo un total a la salida:

$$V_{T,mp} = 437.6V \tag{3.5}$$

$$P_{T,max} = 51.28kW \tag{3.6}$$

En la Figura 3.3 se representan las distintas curvas tensión-corriente y tensión-potencia para distintas temperaturas con una irradiancia fija de $1000 W/m^2$.

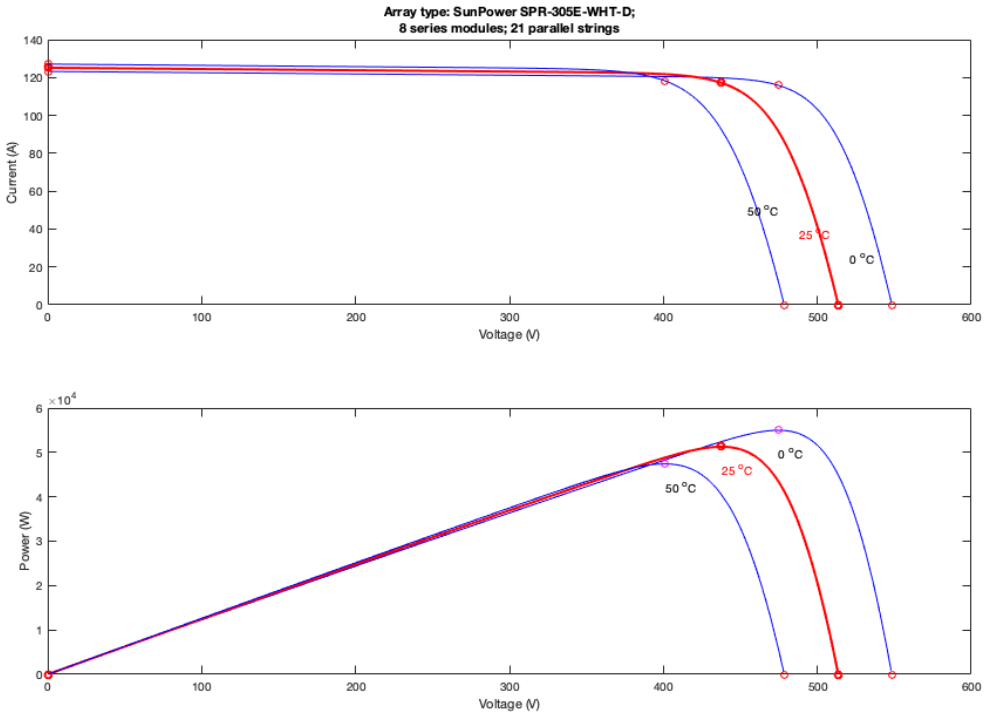


Figura 3.3 Representación curvas array PV.

Este array fotovoltaico está conectado al bus de continua mediante un convertidor elevador en una fase que eleva la tensión de la salida del mismo teniendo en cuenta el punto de máxima potencia del panel fotovoltaico, usando un controlador MPPT. No obstante, por simplicidad, se ha dejado una irradiancia y temperatura fijas. Para que los paneles sufran menos, se coloca en paralelo con el array PV un condensador de 0.05 mF.

En la Tabla 3.3 se presentan los valores de los parámetros de diseño del array fotovoltaico y en la Figura 3.4 se recoge gráficamente el modelado de la red de acometida.

Tabla 3.3 Parámetros de diseño del array PV.

Nombre	Símbolo	Valor	Unidad
Número de paneles en serie	n_s	8	-
Número de paneles en paralelo	n_p	21	-
Potencia máxima nominal de un panel	P_{max}	305.23	kW
Tensión de máxima potencia de un panel	V_{mp}	54.7	V
Condensador a la salida del array	C_{pv}	0.05	mF

3.2.4 Elementos de filtrado

Se conecta un filtro LCL a la salida de los terminales del inversor para reducir los harmónicos y obtener una tensión y corriente más senoidal.

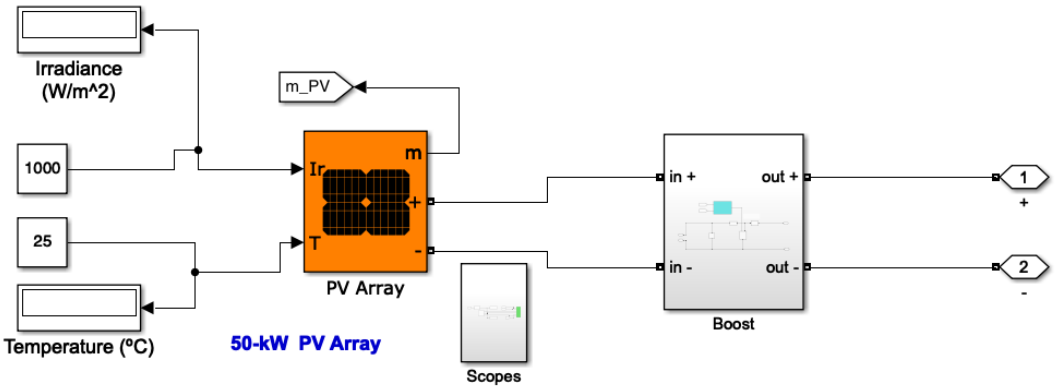


Figura 3.4 Modelado del array PV.

En la Tabla 3.4 se presentan los valores de los parámetros del filtro LCL y en la Figura 3.5 se recoge gráficamente el modelado.

Tabla 3.4 Parámetros de diseño del filtro LCL.

Nombre	Símbolo	Valor	Unidad
Inductancia serie	L_{inv}	0.5	mH
Resistencia serie	R_p	0.002	Ω
Potencia aparente disipada	Q_c	10	kVAR

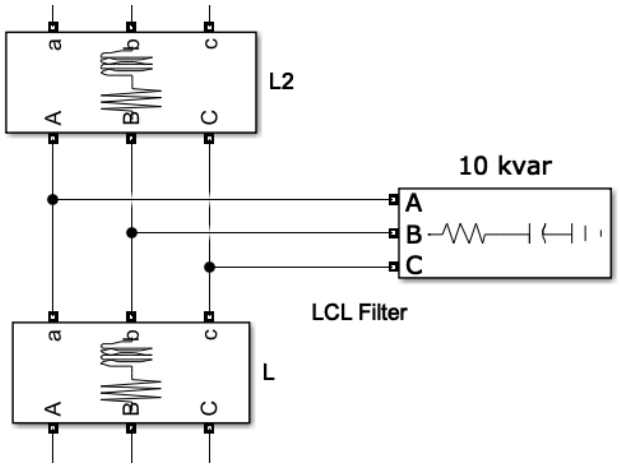


Figura 3.5 Modelado del filtro LCL.

3.2.5 Convertidor AC/DC bidireccional

El convertidor AC/DC bidireccional es un inversor conectado a la red que permite el flujo de potencia en ambos sentidos.

Durante la carga de los EV en los cargadores DC, si la potencia fotovoltaica generada no es suficiente para abastecer dicha demanda, el convertidor AC/DC bidireccional rectifica la corriente alterna a continua. En caso contrario, cuando los EV conectados a los cargadores DC se estén descargando, el convertidor bidireccional AC/DC invertirá la potencia continua a alterna.

En la Tabla 3.5 se presentan los valores de los parámetros del convertidor AC/DC y en la Figura 3.6 se recoge gráficamente el modelado.

Tabla 3.5 Parámetros de convertidor AC/DC.

Nombre	Símbolo	Valor	Unidad
Condensadores del bus DC	C_{DC}	1.7	mF
Tensión del bus DC	V_{DC}	1500	V

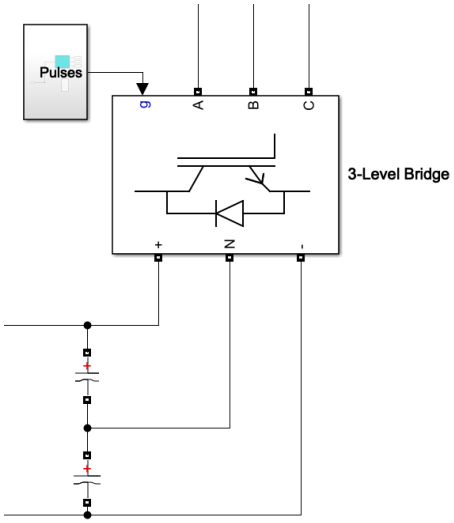


Figura 3.6 Modelado del AC/DC.

3.2.6 Cargadores DC

Los cargadores de continua se localizarán conectados al bus DC y estarán dentro de los EVSE (*Electric Vehicle Supply Equipment*), en los cuales se conectarán los vehículos.

Se muestra el modelado del bus DC en la Figura 3.7.

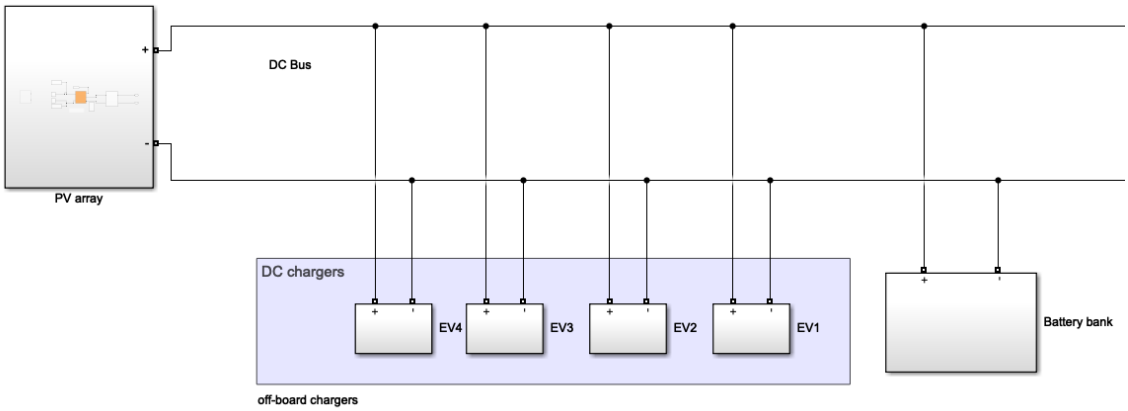


Figura 3.7 Modelado del bus DC.

Estos cargadores se modelan como un convertidor bidireccional formado por dos transistores IGBT que recibirán señales complementarias. Según sean estas señales, el convertidor funcionará como elevador o reductor, determinando el sentido del flujo de la potencia.

Los vehículos se modelan como baterías conectadas al otro extremo del convertidor bidireccional. Se muestra en la Figura 3.8 la configuración del cargador DC bidireccional.

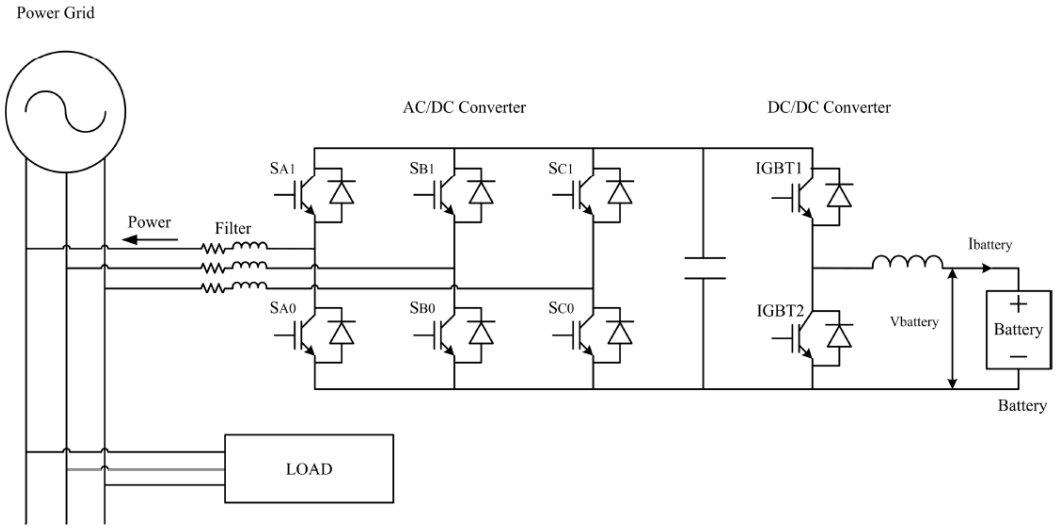


Figura 3.8 Configuración de convertidor DC bidireccional junto a convertidor AC/DC [10].

En la Tabla 3.6 se presentan los valores de los parámetros de los cargadores DC y en la Figura 3.9 se recoge gráficamente el modelado.

Tabla 3.6 Parámetros de los cargadores DC.

Nombre	Símbolo	Valor	Unidad
Condensadores del bus DC	C_{DC}	1.7	mF
Tensión del bus DC	V_{DC}	1500	V

3.2.7 Banco de baterías

El sistema de almacenamiento de energía se modela de manera similar que los cargadores DC. Se utiliza un convertidor bidireccional DC/DC conectado al bus DC y a una batería de almacenamiento.

3.2.8 Cargadores AC

Los cargadores de alterna se conectarán al bus AC usando el equipo de carga localizado dentro del propio vehículo.

Se muestra el modelado del bus AC en la Figura 3.10.

Este equipo equipo de carga se modela como un puente rectificador trifásico conectado a una BMS que, por simplicidad, se ha modelado con un convertidor bidireccional conectado a una batería, al igual que en los cargadores DC.

En la Figura 3.11 se recoge gráficamente el modelado de los cargadores AC.

3.2.9 Panel de control

El panel de control cuenta con una interfaz desde la cual se puede gestionar el número de vehículos que están conectados a los cargadores, así como decidir si estos se estarán cargando o descargando. También se podrá controlar si el sistema de almacenamiento con baterías debe alimentarse o ceder potencia. Esta interfaz se muestra en la Figura 3.12.

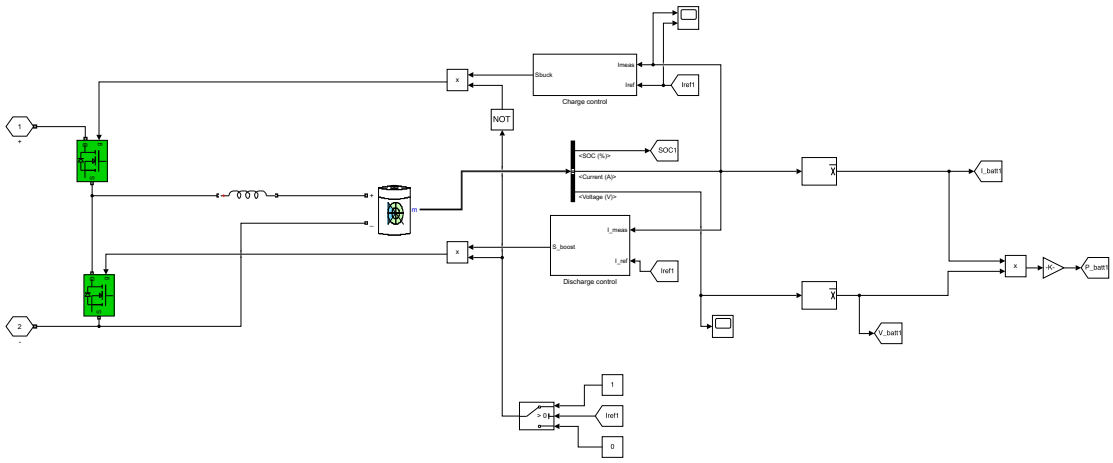


Figura 3.9 Modelado de cargador DC.

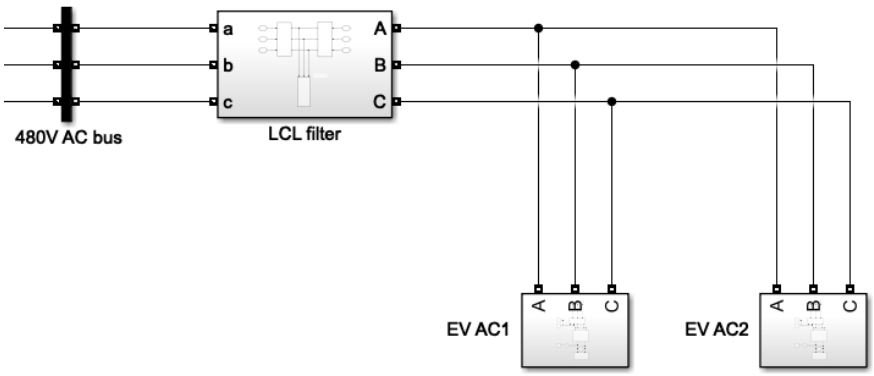


Figura 3.10 Modelado del bus AC.

Cuenta con una pantalla en la cual se representa el balance de potencias en tiempo real de simulación, viendo la cantidad de potencia que cede o absorbe cada elemento del sistema y comprobando que el sumatorio del intercambio de potencias es nulo.

Finalmente, cuenta con un sistema de control automático que decide de forma automática si se debe guardar potencia en el banco de baterías o si se debe usar la potencia de éstas para alimentar al sistema. Para realizar esta tarea se propondrá un algoritmo de control que tendrá como salida la consigna para el banco de baterías.

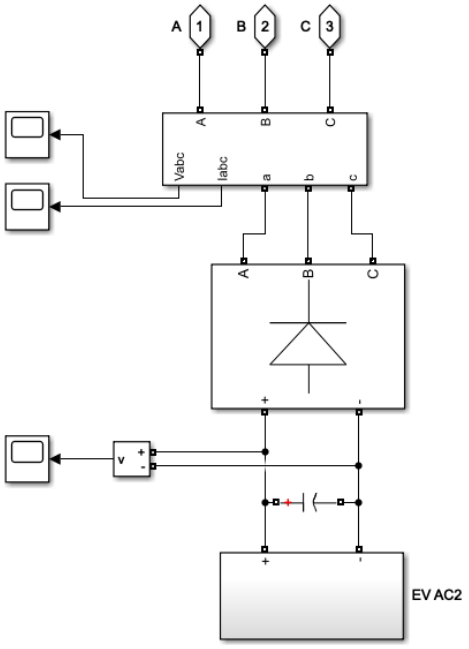


Figura 3.11 Modelado de cargador AC.

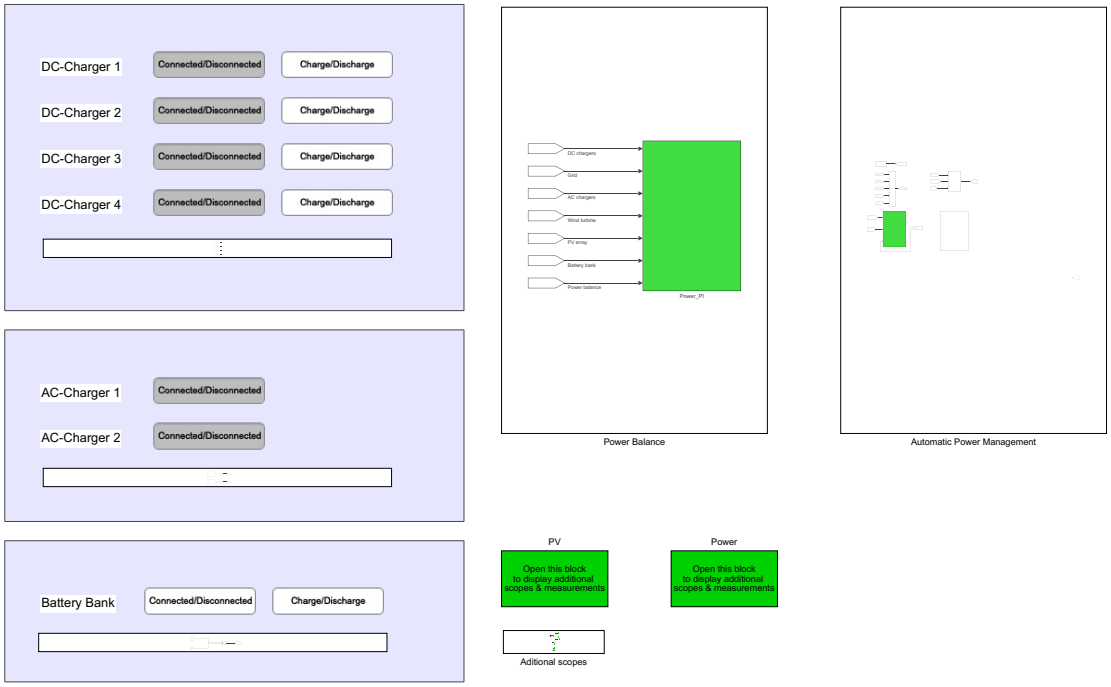


Figura 3.12 Modelado del panel de control.

3.3 Sistema de control

3.3.1 Control cargadores DC

Los cargadores DC servirán para cargar o descargar energía de la batería conectada.

Se muestra en la Figura 3.13 la implementación de una estrategia de control para una carga / descarga con corriente constante utilizando controladores PI [9].

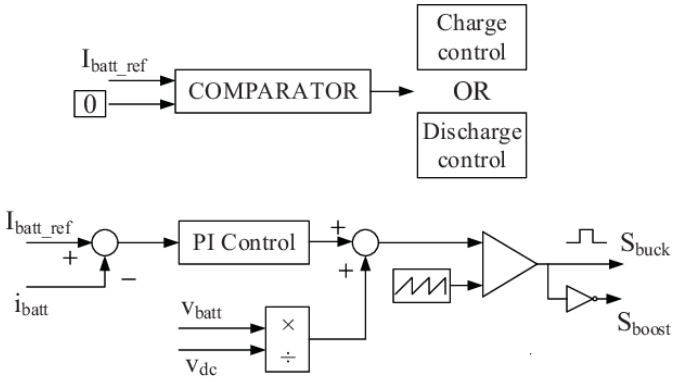


Figura 3.13 Estrategia de control para carga de batería [9].

El controlador, en primer lugar, compara la referencia de la corriente de la batería con 0, de manera que determina el signo de la señal para elegir entre el modo de operación de carga o de descarga.

Una vez se ha determinado el modo, la corriente de referencia se compara con la corriente medida, y el error se pasa por el controlador PI para generar los pulsos de conmutación para los interruptores S_{buck} y S_{boost}. S_{boost} se apagará durante el proceso de carga y S_{buck} se apagará durante el proceso de descarga.

3.3.2 Control MMPT convertidor elevador

El array fotovoltaico se conecta a un convertidor elevador DC/DC que se controla mediante un controlador con MPPT al que se alimenta con la corriente y tensión de salida del array. Este controlador busca que el panel trabaje en su punto de máxima potencia.

3.3.3 Control AC/DC bidireccional

Se propone un control en cascada en el marco de referencia síncrono propuesto para el controlador del inversor. El control vectorial convencional que utiliza 4 controladores PI en un bucle anidado se muestra en la Figura 3.14. [11].

La estructura de control consta de dos bucles de control de tensión y dos lazos de control de corriente internos. El lazo exterior del eje d controla la tensión del bus de corriente continua y el lazo interior controla la corriente alterna activa. Del mismo modo, el lazo exterior del eje q regula la magnitud de la tensión AC ajustando la corriente reactiva, que es controlada por el bucle de corriente interior del eje q. Además, los términos de desacoplamiento dq y señales de tensión de avance para mejorar el rendimiento durante los transitorios.

3.4 Modos de operación

Se denomina modo de operación a los distintos modos de funcionamiento del sistema según sea el direccionamiento de la potencia producida y demandada.

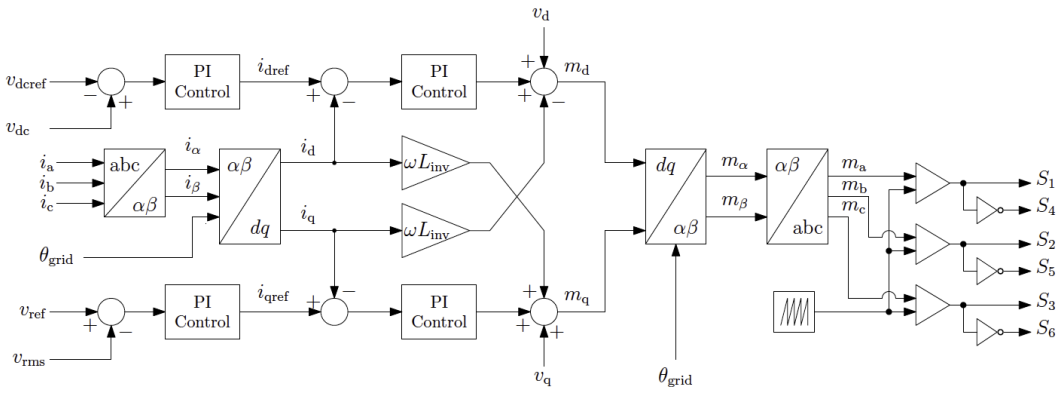


Figura 3.14 Control del convertidor AC/DC [11].

3.4.1 Modo I: trabajo en defecto de potencia

El primer modo de operación se utilizará, principalmente, en los momentos en los que exista un balance negativo entre la energía producida y demandada, es decir, cuando haya falta de potencia. En este caso será necesario buscar otra fuente de alimentación. Cuenta con dos submodos:

- **Modo I.a:** Consiste en utilizar la energía almacenada en el banco de baterías para abastecer la demanda de potencia del sistema.
- **Modo I.b:** En este caso, se utilizará energía procedente de la red, asumiendo el correspondiente coste económico.

3.4.2 Modo II: trabajo en exceso de potencia

El segundo modo de operación se utilizará cuando exista un balance positivo entre la energía producida y demandada, es decir, cuando haya exceso de potencia. En este caso será necesario seleccionar si almacenar o ceder dicho exceso de potencia. Cuenta con dos submodos:

- **Modo II.a:** Consiste en almacenar el exceso de energía producida en el banco de baterías, usado como sistema de almacenamiento.
- **Modo II.b:** En este caso, se cede el exceso de energía producida directamente a la red, teniendo en cuenta el supuesto beneficio económico.

Pueden existir otros modos de operación, como el caso en el que se ceda potencia del banco de baterías a la red o viceversa. Estos casos no se han tenido en cuenta ya que se han supuesto no óptimos.

La conmutación entre los distintos modos se realizará mediante un algoritmo de optimización que tendrá en cuenta otros factores para tomar decisiones óptimas. Se propone, en primer lugar, un algoritmo simplificado para realizar las primeras simulaciones, que se explica en el siguiente apartado.

3.4.3 Algoritmo de control simplificado

Para realizar este algoritmo se toma como nueva entrada el precio instantáneo de la electricidad, que se supondrá cambiante y vendrá definido en el modelo mediante un bloque generador de señal .

Este algoritmo es el más simple posible y funciona teniendo en cuenta si el precio de la electricidad de la red está por encima o por debajo de un umbral. Si el precio de la electricidad está por encima de dicho umbral, se considera que el precio es alto.

Si el precio es alto:

- Si hay exceso de potencia, se venderá el exceso a la red, obteniendo el beneficio de vender a un precio alto.
- Si hay defecto de potencia, se usará la energía almacenada en el banco de baterías, si la hay.

Si, por lo contrario, el precio es bajo:

- Si hay exceso de potencia, se almacenará el exceso en el banco de baterías (si no está lleno) evitando tener que vender a un precio bajo.
- Si hay defecto de potencia, se comprará la energía necesaria a la red, aprovechando que el precio de compra es bajo.

Este algoritmo es sencillo de implementar. Se utiliza un bloque función en el que la entrada es el balance de potencia que excluye la potencia de la red y el estado de carga del banco de baterías. La salida de dicho bloque función es la decisión de cargar o descargar el sistema de almacenamiento. El bloque función contiene el Código 3.1.

Código 3.1 Algoritmo simple de control.

```
function Iref_bank = fcn(P_excess,Elect_Price,SOC_bank)

if (P_excess>0)
    if(Elect_Price<50)
        if(SOC_bank>95)
            %Sell electricity to grid
            Iref_bank=0;
        else
            %Charge battery
            Iref_bank=-80;
        end
    else
        %Sell electricity to grid
        Iref_bank=0;
    end
else
    if(Elect_Price<50)
        %Buy electricity to grid
        Iref_bank=0;
    else
        if(SOC_bank<5)
            %Buy electricity to grid
            Iref_bank=0;
        else
            %Discharge battery
            Iref_bank=80;
        end
    end
end
```

En este algoritmo no se tienen en cuenta otros factores, como el coste de degradación de las baterías o la previsión de demanda y producción en el futuro. Este algoritmo se explica más detalladamente en la segunda parte del trabajo.

3.5 Simulación

En este apartado se realizan distintas simulaciones para comprobar el correcto funcionamiento del sistema. En estas simulaciones, por simplificar, se ha mantenido constante la producción solar y eólica. Las únicas variables de control serán el número de EV cargando o descargando y la consigna del banco de baterías .

En los ensayos 1 y 2, los sucesos y el control se realizarán manualmente. En los ensayos 2 y 3, el control de la acción del banco de baterías se realiza mediante el algoritmo simplificado propuesto en el apartado anterior.

Se representa el balance general de potencias, para los distintos casos.

3.5.1 Ensayo 1

En este primer ensayo se van a ilustrar los modos de funcionamiento I.a y I.b. Para ello se realiza el siguiente supuesto:

1. En primer lugar no habrá ningún coche cargando ni descargando.
2. Llegan dos vehículos DC [1] y comienzan a cargar (Modo I.b).
3. Llegan dos vehículos más DC a cargar.
4. Llegan dos vehículos AC a cargar.
5. El banco de baterías comienza a ceder potencia (Modo I.a).

3.5.2 Ensayo 2

En este ensayo se van a ilustrar los modos de funcionamiento II.a y II.b. Para ello se realiza el siguiente supuesto:

1. En primer lugar no habrá ningún coche cargando ni descargando (Modo II.b).
2. Llegan dos vehículos DC y comienzan a descargar.
3. Llegan dos vehículos más DC a descargar.
4. El banco de baterías comienza a almacenar potencia (Modo II.a).

En los ensayos 3 y 4 el control del banco de baterías pasa ser de manual a automático. Se utiliza el algoritmo simplificado explicado en el apartado anterior.

Para representar el funcionamiento del algoritmo de control del banco de baterías, además del balance de potencia, se muestra una gráfica que contiene el exceso de electricidad (positivo o negativo), la potencia del banco de baterías y el precio de la electricidad.

3.5.3 Ensayo 3

En este ensayo se va a ilustrar la conmutación automática de los modos de funcionamiento I.b a I.a Para ello se realiza el siguiente supuesto:

1. En primer lugar habrán 4 vehículos DC y vehículos AC cargando. El precio de la electricidad es bajo (Modo I.b).
2. El precio de la electricidad pasa a ser alto (Conmuta automáticamente a modo I.a).
3. El precio de la electricidad vuelve a ser bajo (Conmuta automáticamente a modo I.b).

[1] Por claridad, se denominan vehículos DC y vehículos AC a los vehículos que cargan en los cargadores rápidos (DC) y en los de corriente alterna (AC) respectivamente.

3.5.4 Ensayo 4

En este ensayo se va a ilustrar la conmutación automática de los modos de funcionamiento II.b a II.a Para ello se realiza el siguiente supuesto:

1. En primer lugar habrán 2 vehículos cargando en los cargadores DC. El precio de la electricidad es alto (Modo II.b).
2. El precio de la electricidad pasa a ser bajo (Conmuta automáticamente a modo II.a).
3. El precio de la electricidad vuelve a ser alto (Conmuta automáticamente a modo II.b).

Hablar del tiempo de simulación.

3.6 Resultados

En este apartado se comentan los resultados obtenidos en las simulaciones de los distintos ensayos[2].

3.6.1 Ensayo 1

Los resultados de la simulación del ensayo 1 se muestran en la Figura 3.15. La gráfica azul representa

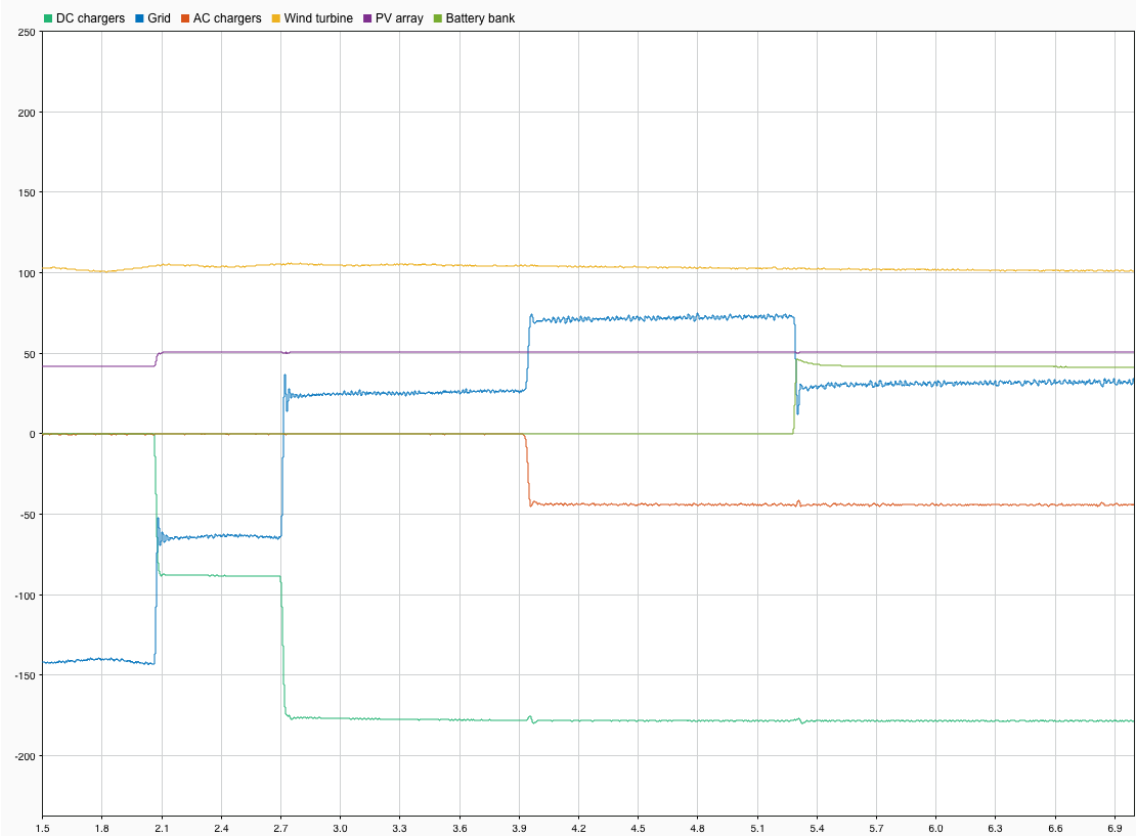


Figura 3.15 Ensayo 1.

la potencia de la red. En el primer intervalo se muestra cómo todo el exceso de potencia de la

[2] Se omiten los resultados de los primeros segundos de las simulaciones, ya que en estos sucede el transitorio inicial de la estación hasta que se estabilizan los distintos componentes. Dicho control inicial no se ha tenido en cuenta en este trabajo.

estación, que están produciendo los paneles solares y la planta fotovoltaica, se está vendiendo a la red.

En el segundo intervalo llegan dos vehículos DC a cargar. Se muestra con la gráfica verde claro la potencia relativa a los cargadores DC. En este caso, al llegar 2 vehículos a cargar, se consume potencia y, por tanto, la potencia cedida a la red es menor.

En el tercer intervalo llegan otros 2 vehículos DC a cargar. Se observa como aumenta la demanda tanto que se empieza a comprar energía de la red (pasa a tener un valor positivo). La red solventa toda la demanda de potencia (Modo I.b).

En el cuarto intervalo llegan dos vehículos a los cargadores AC. En este momento, la demanda pasa a ser máxima, teniéndose que comprar la energía necesaria a la red.

En el último intervalo se activa la descarga del banco de baterías de manera que ayuda a la red a alimentar la demanda de la estación. Se observa que al descargar el banco de baterías, la energía proveniente de la red disminuye (Modo I.a).

3.6.2 Ensayo 2

Los resultados de la simulación del ensayo 2 se muestran en la Figura 3.16.

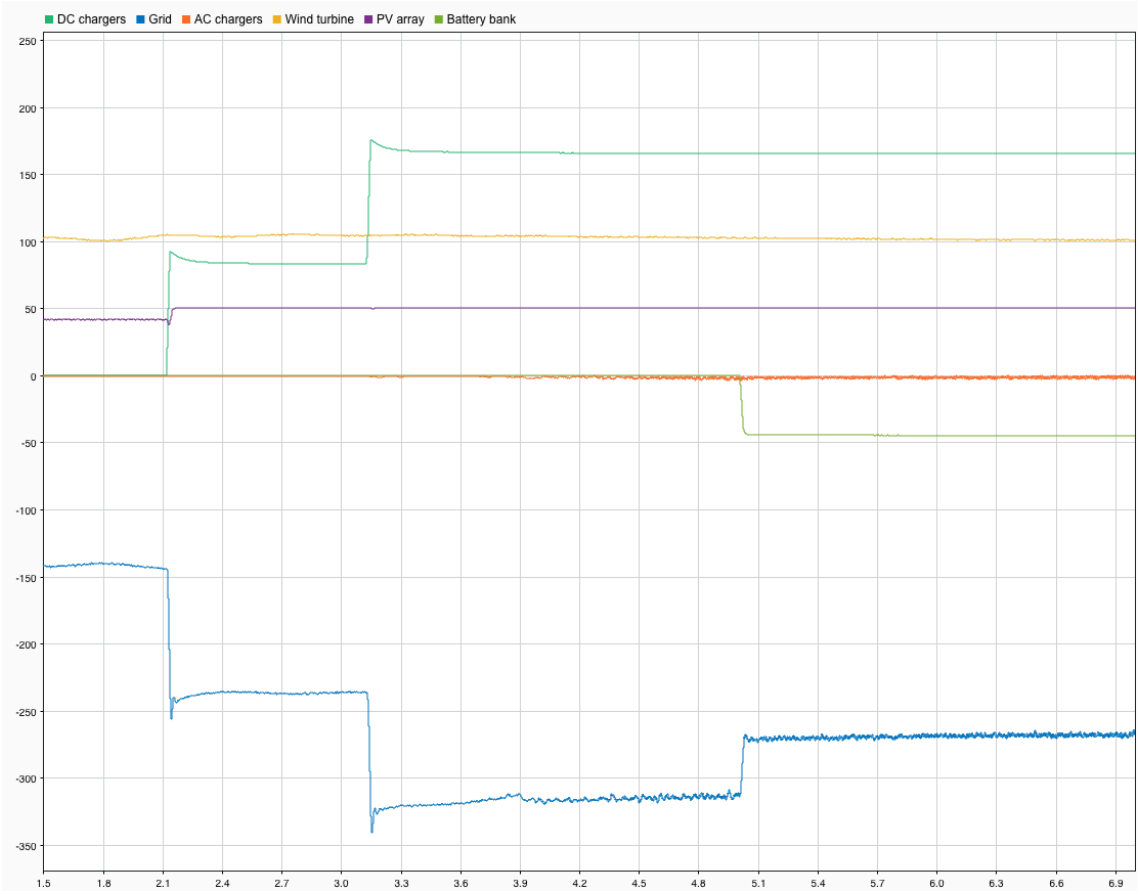


Figura 3.16 Ensayo 2.

En el primer intervalo se muestra cómo todo el exceso de potencia de la estación, que están produciendo los paneles solares y la planta fotovoltaica, se está vendiendo totalmente a la red (Modo II.b).

En el segundo intervalo llegan dos vehículos DC a descargar. Estos ceden potencia y, por tanto, el exceso de potencia en la estación es mayor.

En el tercer intervalo llegan otros 2 vehículos DC a descargar. Se observa como el exceso en la estación ahora es máximo.

En el último intervalo se activa la carga del banco de baterías manera que se almacena parte del exceso de la energía. Se observa que al cargar el banco de baterías, la energía cedida a la red disminuye (Modo II.a).

3.6.3 Ensayo 3

El balance de potencia resultante de la simulación del ensayo 3 se muestra en la Figura 3.17. La representación de la conmutación automática de modos en la simulación se representa en la Figura 3.18.

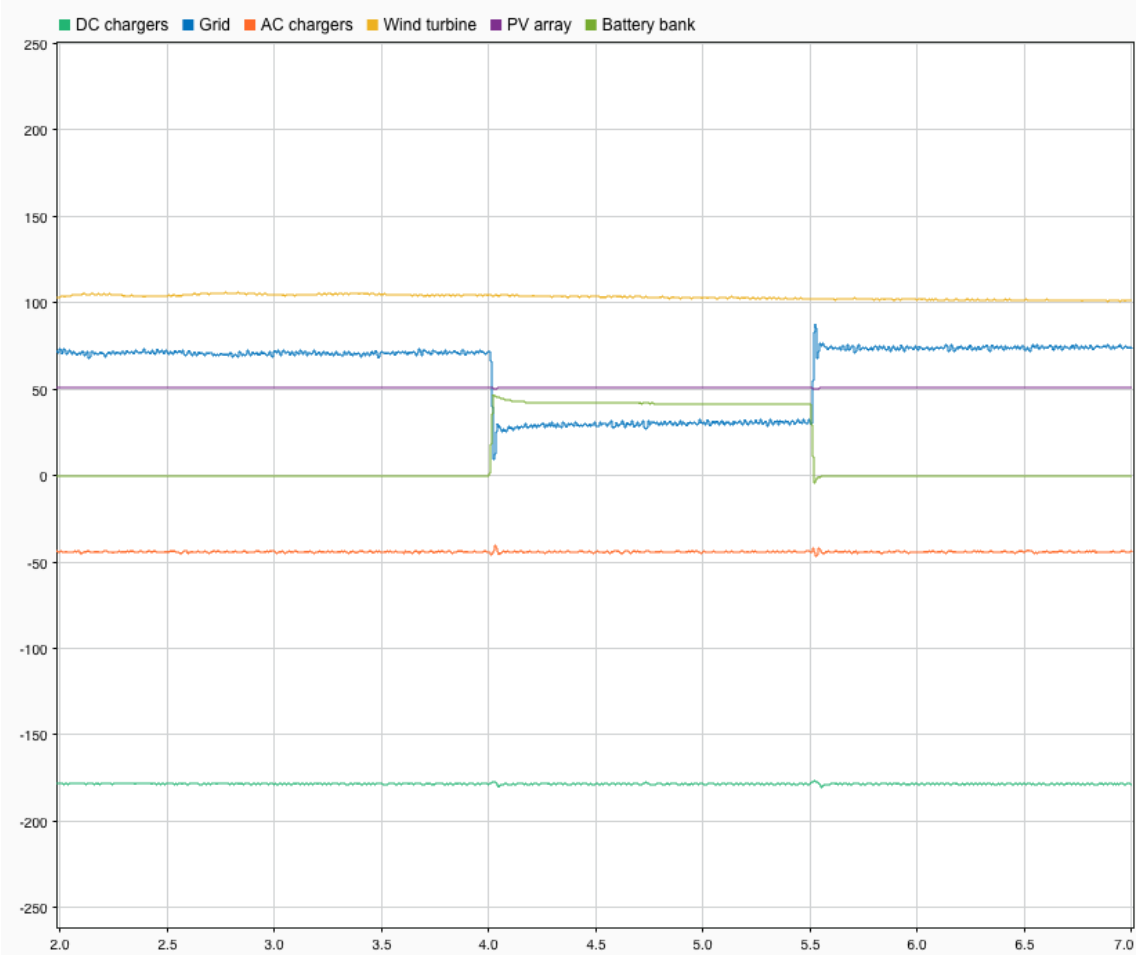


Figura 3.17 Ensayo 3. Balance de potencia.

En el primer intervalo se muestra cómo toda la demanda adicional de la estación provocada por la carga en los cargadores DC y AC, la está abasteciendo la red. Se está comprando totalmente a la red al ser el precio de la electricidad bajo y, por tanto, lo más rentable económicamente (Modo II.b).

En el segundo intervalo el precio de la electricidad pasa a ser alto. El algoritmo hace que el banco de baterías abastezca parte de la demanda, de manera que se evite tener que comprar a la red a un precio alto (Modo I.a).

En el último intervalo el precio de la electricidad vuelve a ser bajo, de este modo se muestra como el sistema de control automático hace que se vuelva a comprar toda la demanda adicional de la estación a la red.

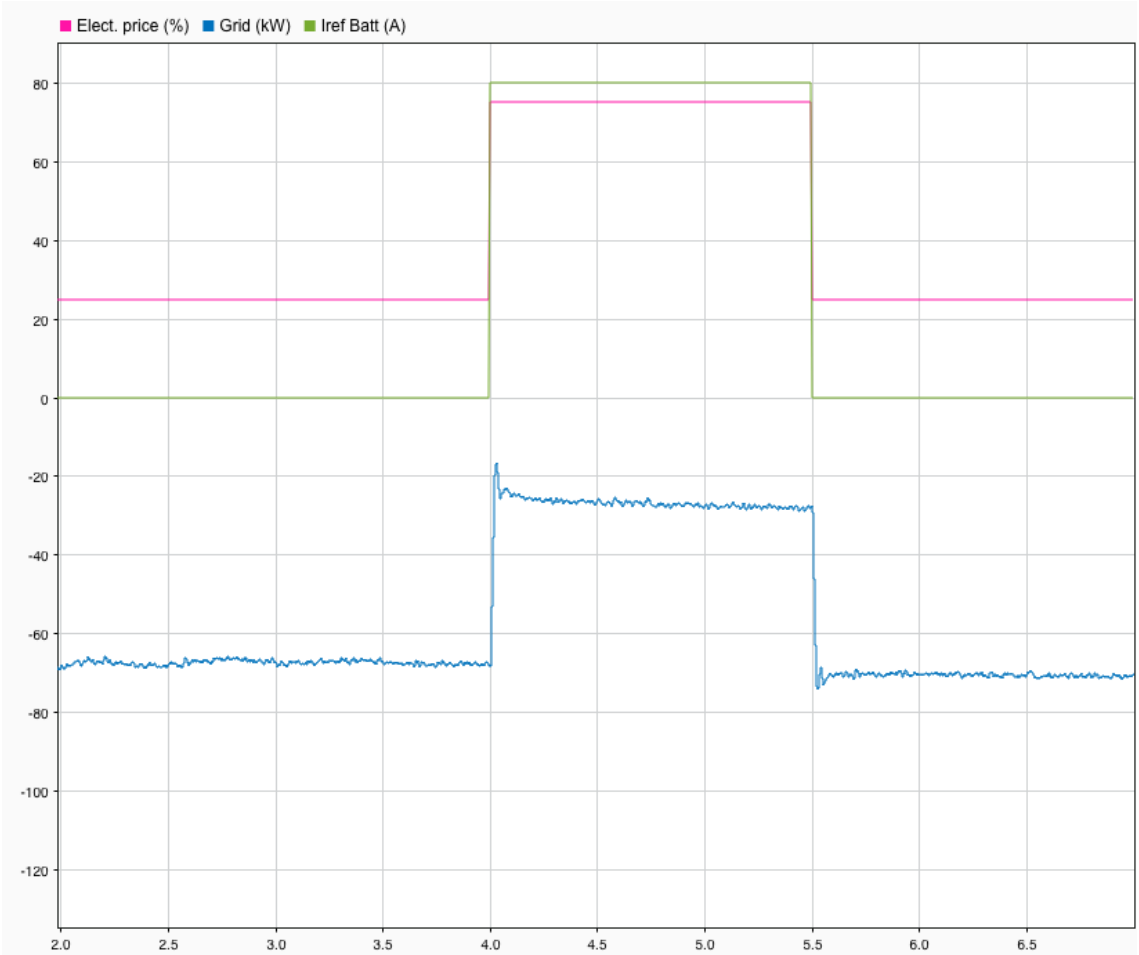


Figura 3.18 Ensayo 3. Conmutación automática.

3.6.4 Ensayo 4

El balance de potencia resultante de la simulación del ensayo 4 se muestra en la Figura 3.19. La representación de la conmutación automática de modos en la simulación se representa en la Figura 3.20. En el primer intervalo se muestra cómo todo el exceso de potencia de la estación que están produciendo los paneles solares, la planta fotovoltaica y la descarga en los cargadores DC, se está cediendo a la red. Se está comprando a la red al ser el precio de la electricidad alto y, por tanto, lo más rentable económicamente (Modo II.b).

En el segundo intervalo el precio de la electricidad pasa a ser bajo. El algoritmo hace que parte del excedente de la estación se almacene en el banco de baterías, de manera que se evite tener que vender a la red a un precio bajo (Modo II.a).

En el último intervalo el precio de la electricidad vuelve a ser alto. De este modo se muestra cómo el sistema de control automático hace que se vuelva a vender el exceso de la estación a la red en su totalidad.

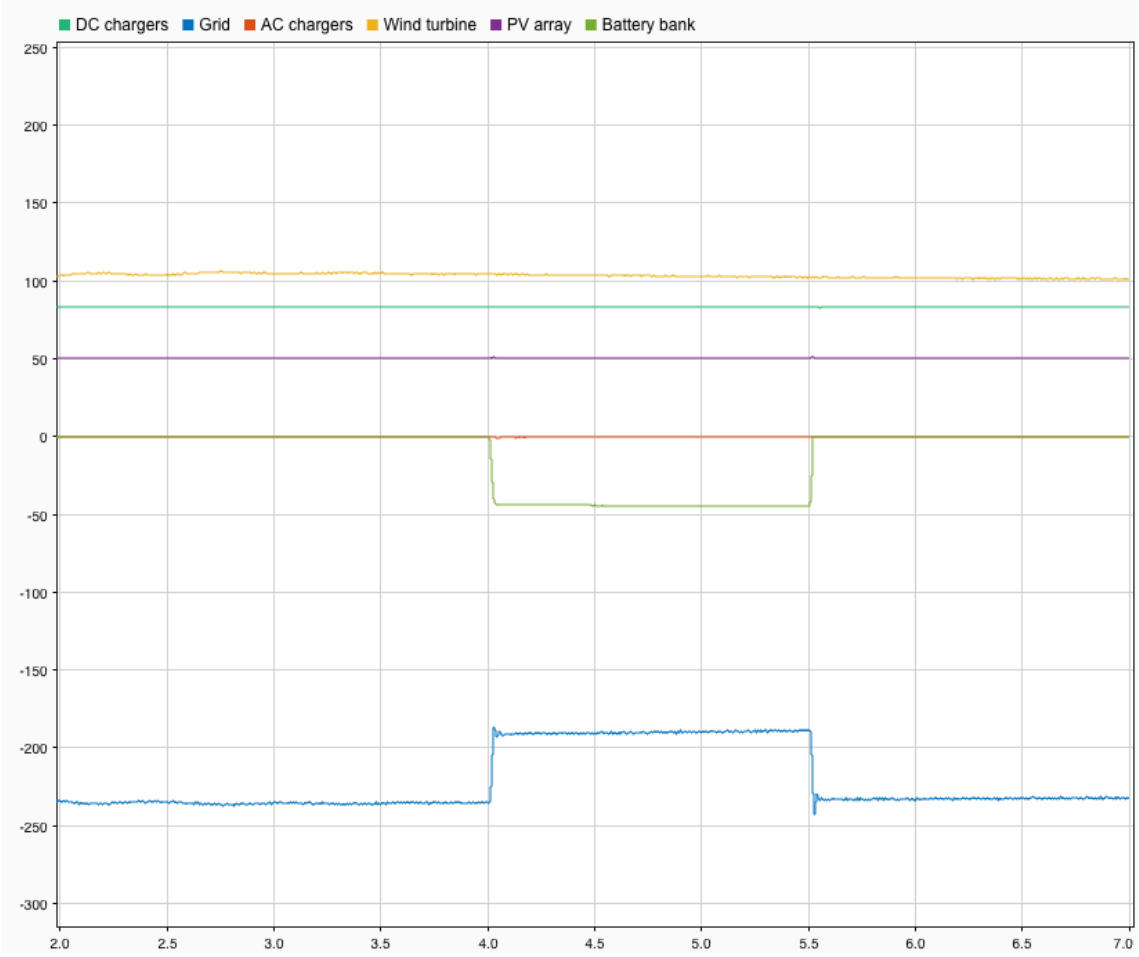


Figura 3.19 Ensayo 4. Balance de potencia.

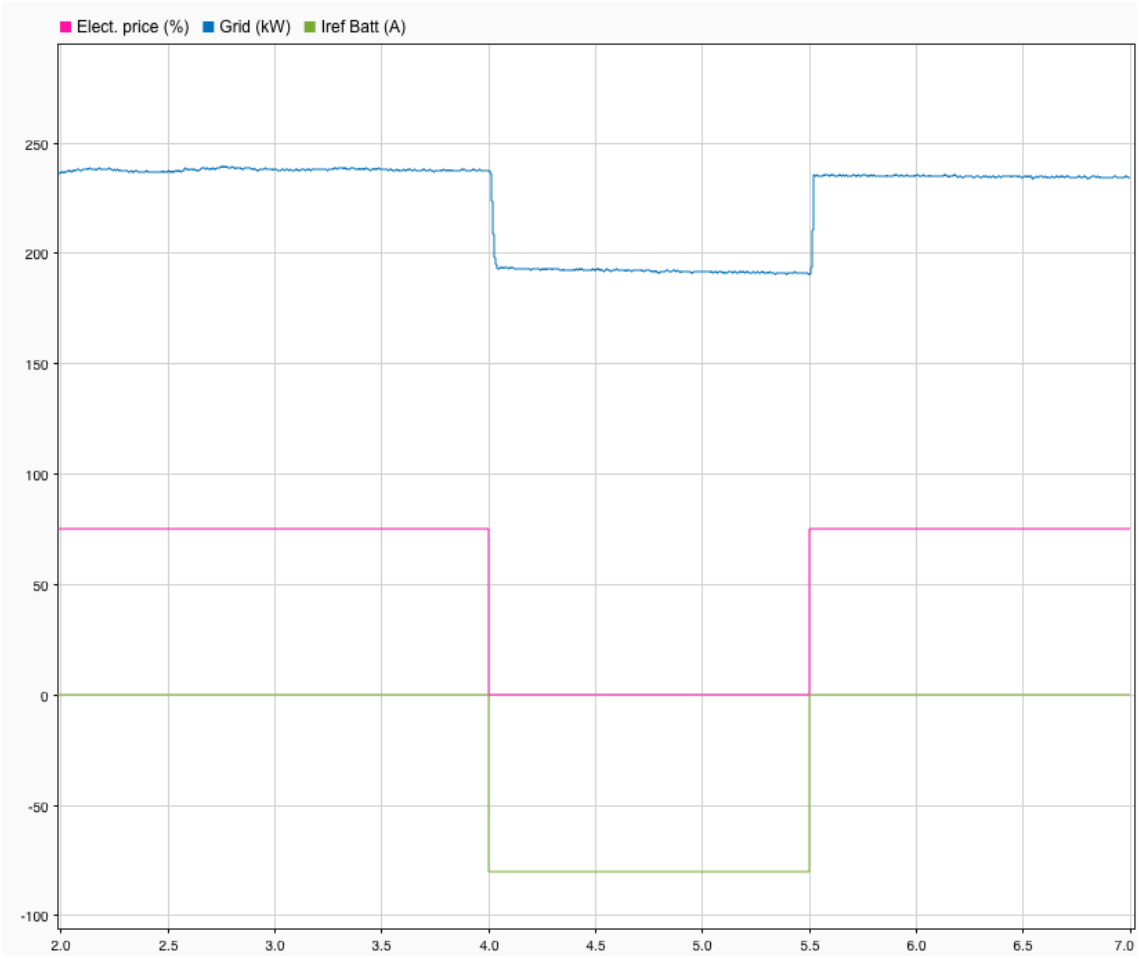


Figura 3.20 Ensayo 4. Conmutación automática.

3.7 Conclusión

En este capítulo se ha realizado un simulador de una estación de carga de vehículos con soporte renovable y almacenamiento de baterías.

Se ha demostrado la importancia de la gestión de los flujos de carga de la estación. Con el simulador se ha hecho una visualización del funcionamiento de la estación y se han controlado las principales variables de entrada y de salida. Se ha buscado la modularización del modelo de manera que éste sea fácilmente ampliable y adaptable.

Se ha implementado un primer algoritmo simple de gestión de los flujos de potencia del banco de baterías. Este algoritmo se mejorará con el uso de predicciones en la demanda y en la producción. En el próximo capítulo se explicarán estas técnicas.

Se proponen, como futuro trabajo sobre el simulador, las siguientes ideas: a) Abordar un estudio de optimización sobre el dimensionamiento de la estación (Número de cargadores, número de paneles solares, cantidad de baterías de almacenamiento, etc.). b) Mejorar el transitorio inicial mediante el uso disparadores u otras técnicas para conseguir un arranque del sistema más suave. c) Añadir funcionalidades a la interfaz (calcular el precio de la sesión de carga para el consumidor , realizar una programación dinámica de en qué tiempo llega un coche y cuándo se va, etc.). d) Añadir realismo al simulador (Los paneles solares produzcan energía según la hora del día, el precio de la electricidad venga dado por datos de un día real, redimensionar las intensidades de carga, capacidad de baterías, etc.). e) Añadir un selector que permita operar en modo manual o automático y otro que modifique el tipo de control. f) Implementar en el simulador distintos algoritmos de control, incluyendo el propuesto en los próximos capítulos, así como técnicas de control predictivo, aprendizaje reforzado, etc.

4 Fundamentos teóricos de las redes neuronales

El aprendizaje automático ha revolucionado el mundo tal y como lo conocemos en la última década. La explosión de la información ha dado lugar a la recopilación de cantidades masivas de datos. Esta cantidad de datos, unida al rápido desarrollo de la potencia de los procesadores y la paralelización de los ordenadores, ha hecho posible la obtención y el estudio enormes cantidades de datos con relativa facilidad.

En este capítulo se realizará inicialmente una breve introducción teórica a los modelos de aprendizaje que servirá para entender los modelos de predicción que se propondrán en el siguiente capítulo. Posteriormente, se presentará una visión general de las distintas aplicaciones. Luego, se profundizará en la teoría de las redes neuronales. Seguido de esto, se muestran los desafíos principales del Machine Learning y, por último, algunas herramientas para evaluar la robustez de los modelos.

4.1 Machine Learning

El aprendizaje automático o Machine Learning es un campo dentro de la inteligencia artificial. Es la práctica de usar algoritmos para analizar datos y aprender de ellos, para posteriormente poder hacer predicciones sobre datos no vistos. Destaca por el hecho de no requerir reglas programadas previamente para producir las salidas. El propio sistema es capaz de aprender por sí mismo para efectuar una tarea a través de una fase previa de entrenamiento.

El aprendizaje supervisado y el no supervisado son ejemplos de dos distintos enfoques de modelos de aprendizaje automático. Difieren en la forma en que se entrenan los modelos y el tipo de datos de entrenamiento que se requieren [12]. La principal diferencia entre ellos es que para el aprendizaje supervisado se necesitan datos etiquetados y en el aprendizaje no supervisado no es necesario que los datos estén etiquetados.

4.2 Perspectiva general del Machine Learning: Modelos y aplicaciones

Los modelos de aprendizaje dentro del aprendizaje automático se han ramificado y especializado en tareas específicas. Las aplicaciones son, por tanto, muy variadas.

En relación con el Procesamiento del Lenguaje Natural, las principales aplicaciones son: reconocimiento del habla [13], traducción del lenguaje [14] texto predictivo [15], y análisis de sentimientos [16]. Pueden utilizarse en asistentes inteligentes [17], ciberseguridad [18], generadores de contenido sintético [19] y en tareas de programación [20].

El uso de modelos de aprendizaje basados en imágenes ha incrementado exponencialmente el número de campos de aplicación de la Visión por Computador. Los modelos de aprendizaje pueden variar en función de la aplicación para la que se vayan a utilizar. Existen clasificadores de imágenes [21], detectores y segmentadores de objetos [22], modelos para el reconocimiento de actividades en vídeo [23], etc. Estos modelos pueden utilizarse para una amplia variedad de aplicaciones, como la asistencia sanitaria, el reconocimiento facial, los coches autónomos y la transferencia de estilos artísticos.

Los modelos de secuenciales, además de usarse en el procesamiento de lenguaje natural, se utilizan para la predicción y análisis de series temporales. Se usan en aplicaciones como modelado del clima, predicción del tráfico en la web, neurociencia, así como en economía y finanzas [24]. También puede ser utilizado para el control predictivo de sistemas.

Adicionalmente se encuentran los los modelos no supervisados. El aprendizaje no supervisado se refiere a los algoritmos utilizados para identificar patrones en conjuntos de datos que contienen puntos de datos que no están clasificados ni etiquetados. Así, los algoritmos pueden clasificar, etiquetar y agrupar los puntos de datos dentro de los conjuntos de datos sin tener ninguna guía externa para realizar esa tarea. En otras palabras, los usuarios no necesitan supervisar el modelo [25].

Entre las tareas no supervisadas podemos destacar, en primer lugar, el *clustering*. Éste sirve para encontrar estructuras en un conjunto de datos no etiquetados.

Le sigue la "asociación", que es otra tarea no supervisada cuyo objetivo principal es descubrir relaciones ocultas interesantes en grandes conjuntos de datos. Se utiliza para el análisis de cestas de mercado (qué artículos se compran juntos), la agrupación de clientes (qué tiendas tienden a visitar juntas), la agrupación de precios, venta cruzada, etc.

Del mismo modo cabe destacar la detección de anomalías. Tiene una gran variedad de aplicaciones. Se utiliza para la detección de intrusos, la detección de fraudes, la vigilancia militar, en los seguros, etc.

Por último, se destacan los *autoencoders*, que se utilizan principalmente como algoritmo de reducción de dimensionalidad (o de compresión)[26].

4.3 Redes neuronales y *Deep Learning*

Las redes neuronales son un conjunto de técnicas que se encuentran dentro del campo del Machine Learning y son el fundamento principal del aprendizaje profundo o *Deep Learning*.

Sus funciones son, después de ser entrenadas, simular el comportamiento del cerebro humano dotando a los programas informáticos de capacidad para detectar patrones y resolver problemas en los campos de la Inteligencia Artificial.

Las redes neuronales artificiales más comunmente utilizadas son las llamadas perceptrones multicapa. Estas son básicamente regresiones no lineales y modelos discriminantes que pueden ser implementadas mediante software convencional. Constan de un número frecuentemente elevado de "neuronas", es decir, de elementos de cálculo lineales o no lineales simples, interconectados de forma frecuentemente compleja y frecuentemente organizados en capas [27].

El aprendizaje profundo permite que los modelos computacionales compuestos por múltiples capas de procesamiento aprendan representaciones de datos con múltiples niveles de abstracción. El aprendizaje profundo descubre patrones intrínsecos en grandes conjuntos de datos utilizando el algoritmo de "propagación hacia detrás", que indica cómo la máquina debe cambiar sus parámetros internos que se utilizan para calcular la representación en cada capa a partir de la representación en la capa anterior [28].

Las redes convolucionales profundas han supuesto un gran avance en el procesamiento de imágenes, vídeo, voz y audio, mientras que las redes recurrentes se han desarrollado para su uso en datos secuenciales como el texto y el habla.

En los siguientes apartados se profundiza en los conceptos básicos de una red neuronal artifical común (ANN), de las redes convolucionales profundas (CNN) y de las redes recurrentes (RNN).

4.4 Red neuronal artificial (ANN)

A continuación, se explica la estructura general de una red neuronal.

Se denota x a las características de entrada que se alimentan a la red. Esta primera capa se llama capa de entrada. A la capa de entrada la siguen distintas capas denominadas capas ocultas, que se componen de varias unidades, también llamados perceptrones, nodos o neuronas. La red de la Figura 4.1 sólo tiene una capa.

La última capa, que en el caso de la red de la Figura 4.1 está formada por un único nodo, se llama capa de salida y es la responsable de calcular la salida predicha $\hat{y}$.

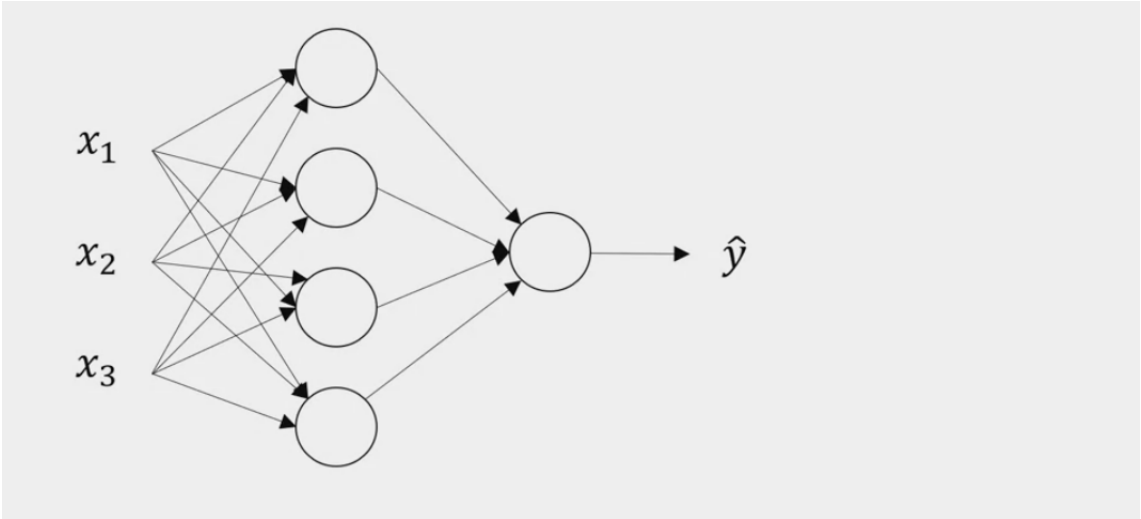


Figura 4.1 Ejemplo de red neuronal simple [29].

Para entender el concepto de red neuronal es útil conocer el funcionamiento de las neuronas o nodos de las que se componen las capas de la red.

A cada nodo le llegan unos valores de entrada x que, multiplicados por los pesos w (*weights*) y sumándoles un sesgo b (*bias*) mediante una función de activación σ permiten calcular las activaciones a que podrán ser alimentadas a capas posteriores. A esto se denomina propagación hacia delante.

$$z = w^T x + b \tag{4.1}$$

$$a = \sigma(z) \tag{4.2}$$

Una intuición que se puede tener es que si un peso tiene un valor alto, le estará dando importancia a una determinada característica de la entrada. De esta forma la red podrá buscar conexiones importantes entre los datos de entrada y de salida.

4.4.1 Funciones de activación

Cuando se construye una red neuronal, una de las selecciones que hay que hacer es qué función de activación utilizarán las unidades en las capas intermedias de la red neuronal, así como la función de activación de la unidad de salida.

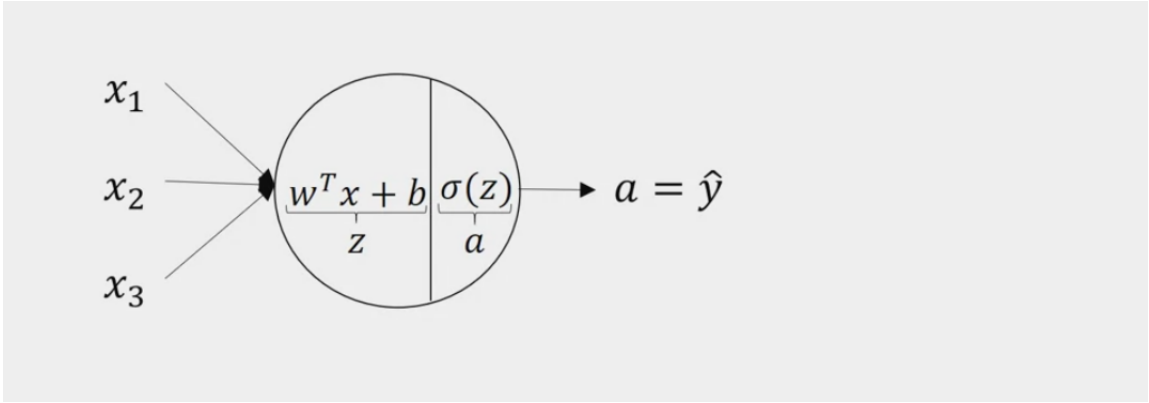


Figura 4.2 Representación gráfica [29].

En el apartado anterior se ha utilizado la función sigmoide como función de activación. Además de esta función, existen otras.

4.4.2 Función de coste

La función de coste ofrece información sobre el correcto funcionamiento de un algoritmo.

Para entrenar nuestra red, usaremos m muestras o ejemplos de datos etiquetados (datos de entrenamiento), es decir, datos de los cuales se conoce la salida verdadera (*ground truth*).

Estos ejemplos se pasarán como datos de entrada a la red y se calculará las salidas predichas. Se compararán estas salidas predichas con las salidas verdaderas, mediante una función de pérdida L. La función de coste J será la media de las funciones de pérdida de los datos de entrenamiento.

$$\mathscr{J}(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}(\hat{y}^{[i]}, y^{[i]}) \tag{4.3}$$

En concreto, existen algunas funciones de coste que son frecuentemente utilizadas:

1. **Error cuadrático medio (MSE):** consiste en elevar al cuadrado la diferencia entre la salida predicha y la salida verdadera y hacer la media.

$$\mathscr{J} = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{[i]} - y^{[i]})^2 \tag{4.4}$$

2. **Error absoluto medio (MAE):** en este caso se toma el valor absoluto de las diferencias en lugar de elevar al cuadrado. De este modo no se penalizan tanto los valores atípicos (*outliers*)

$$\mathscr{J} = \frac{1}{m}\sum_{i=1}^{m}|\hat{y}^{[i]} - y^{[i]}| \tag{4.5}$$

3. **Función de Huber:** Es una combinación de las dos anteriores funciones. Para valores de error inferiores a delta, usa el MSE, y para valores superiores usa el MSE. No ignora los valores atípicos ni los penaliza excesivamente.

$$\mathscr{L}_\delta = \begin{cases} \frac{1}{2}(\hat{y}^{[i]} - y^{[i]})^2 & \text{si } |\hat{y}^{[i]} - y^{[i]}| \leq \delta, \\ \delta|\hat{y}^{[i]} - y^{[i]}| - \frac{1}{2}\delta^2 & \text{si } |\hat{y}^{[i]} - y^{[i]}| > \delta, \end{cases} \tag{4.6}$$

4. **Valoración cruzada (Cross categorical entropy):** Se usa en tareas de clasificación multiclase. Mide la diferencia entre dos distribuciones de probabilidad, la verdadera y la predicha.

$$\mathscr{L} = \sum_{k=1}^{clases} (y_k \cdot \log \hat{y}_k) \tag{4.7}$$

4.4.3 Algoritmos de optimización

El proceso de entrenamiento tiene como fin obtener los parámetros entrenables (pesos w y sesgos b) que minimizan la función de coste. Se muestra un esquema del proceso de un entrenamiento general en la Figura 4.4

El descenso del gradiente es un algoritmo de optimización muy utilizado para esta tarea. Empezando desde un punto arbitrario, consiste en calcular la derivada (pendiente) de la función de coste J respecto a los parámetros w y b; y, conociendo la pendiente y su signo, se puede intuir en qué cantidad y hacia qué dirección deben actualizarse los parámetros para acercarse al mínimo global.

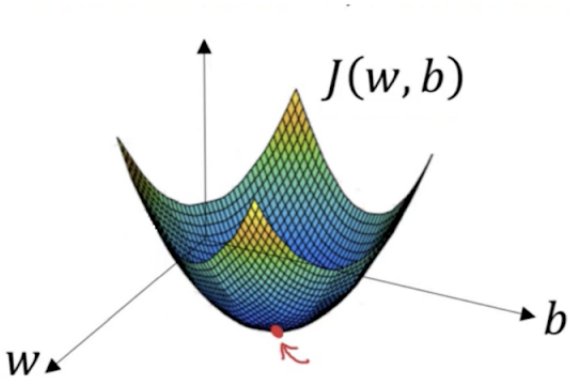


Figura 4.3 Representación simplificada de la función de coste y su punto mínimo [29].

El proceso del cálculo de las derivadas parciales del error con respecto a los distintos parámetros se denomina propagación hacia detrás.

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w} \tag{4.8}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b} \tag{4.9}$$

La cantidad en la que se actualizarán los parámetros es proporcional al módulo de la pendiente calculada y está ponderada por un valor α, llamado ratio de aprendizaje o *learning rate*.

Otros algoritmos de optimización que se derivan a partir del descenso del gradiente son:

1. Momentum: Tiene en cuenta los gradientes de las iteraciones anteriores a la hora de actualizar los parámetros. Para ello realiza una media exponencial de los gradientes anteriores. El momento ayuda a que la función coste converja al mínimo de una forma más rápida y consistente.
2. RMSprop: Se elevan al cuadrado las derivadas y se realiza una media exponencial para saber en qué dirección se está moviendo más rápido. Posteriormente, en la actualización de los parámetros, la raíz de estos términos dividen al ratio de aprendizaje, pudiendo así tomar una

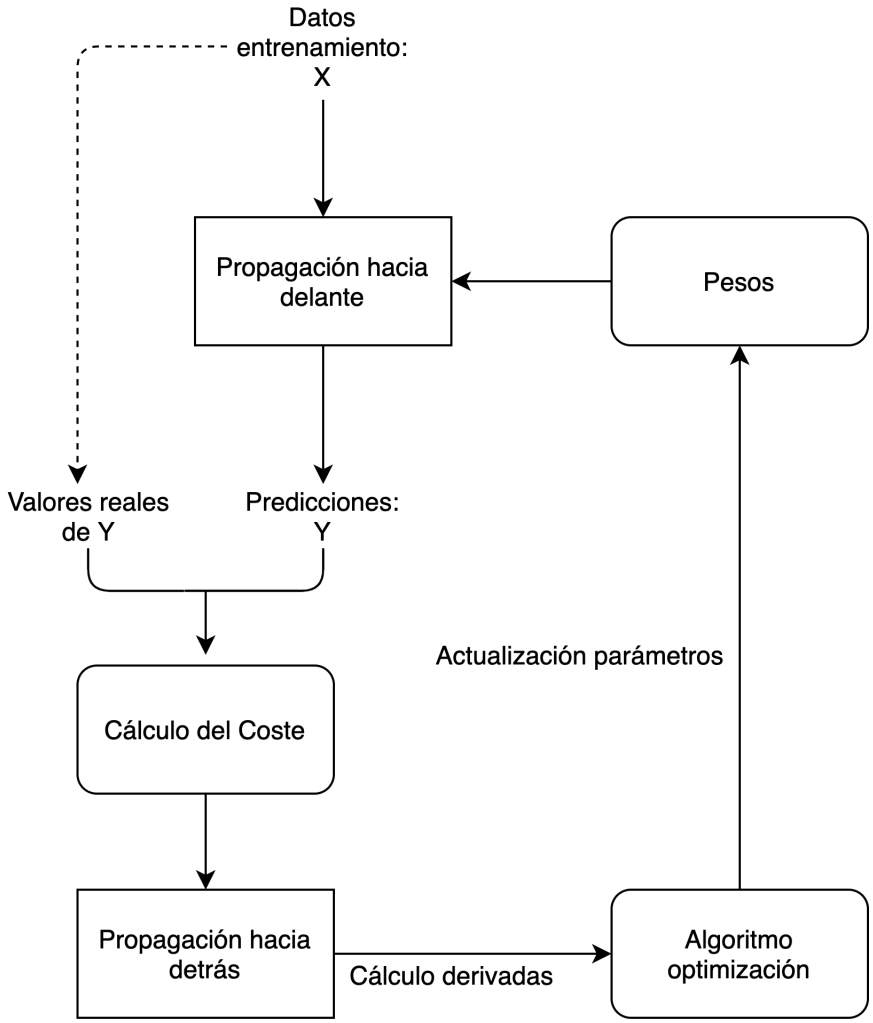
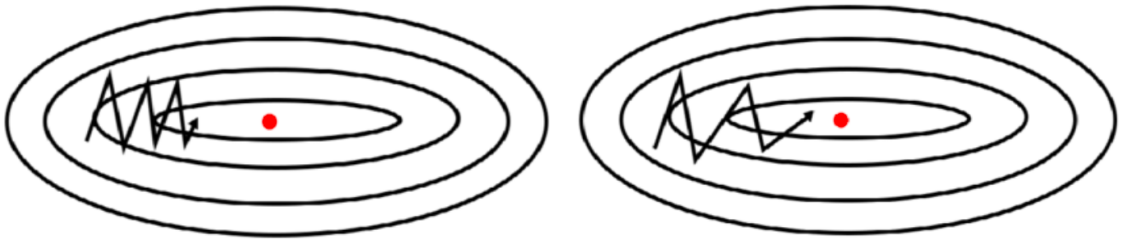


Figura 4.4 Esquema de un entrenamiento general.

dirección más directa hacia el mínimo global, obteniendo un aprendizaje más rápido que con el algoritmo anterior.

3. ADAM: Combina los efectos del descenso del gradiente con momento y del descenso del gradiente con RMSprop. Es un algoritmo de aprendizaje muy utilizado y se ha demostrado que es efectivo en muchas redes neuronales con amplia variedad de arquitecturas.



(a) Descenso del gradiente sin momentum. **(b)** Descenso del gradiente con momentum.

Figura 4.5 Descenso del gradiente con y sin momentum [30].

4.5 Redes neuronales convolucionales (CNN)

Las redes convolucionales, también conocidas como redes neuronales convolucionales o CNN, son un tipo especializado de red neuronal para procesar datos que tienen una topología conocida como de tipo reticular. Algunos ejemplos son los datos de series temporales, que pueden considerarse como una cuadrícula 1-D que toma muestras a intervalos de tiempo regulares, y los datos de imágenes, que pueden considerarse como una cuadrícula 2-D de píxeles. El nombre "red neuronal convolucional" indica que la red emplea una operación matemática llamada convolución. La convolución es un tipo especializado de operación lineal. Las redes convolucionales son simplemente redes neuronales que utilizan la convolución en lugar de la multiplicación matricial general en al menos una de sus capas [31].

4.5.1 Operación de convolución en capas convolucionales

La operación de convolución se realiza mediante el uso de un filtro o kernel sobre una entrada. La entrada sobre la que se realiza la convolución puede ser un array unidimensional o multidimensional de datos, también denominados tensores. Es común que la entrada sobre la que se realiza la convolución sea una imagen. Una imagen en escala de grises es un tensor 2-D, y en color un tensor 3-D ($altura \times ancho \times num\_canales$).

En el caso de una imagen RGB, el filtro o detector de características es una matriz tridimensional (3-D) de pesos, que puede ser considerado como un volumen. A continuación, dicho filtro se aplica a un área de la imagen y se calcula un producto escalar entre los píxeles de imagen y el filtro. Este producto escalar se guarda en una matriz de salida. A continuación, el filtro se desplaza un paso (o *stride*), repitiendo el proceso hasta que el filtro haya barrido toda la imagen. El resultado final de la serie de productos escalares de la entrada y el filtro se conoce como mapa de características, mapa de activación o característica convolucionada []. Se muestra un ejemplo de una convolución en la Figura 4.6.

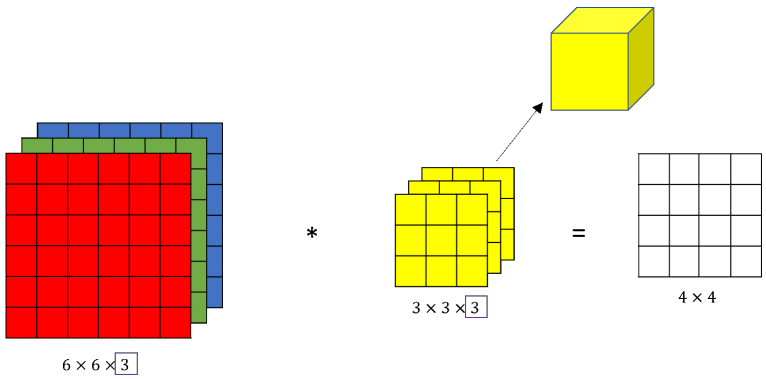


Figura 4.6 Ejemplo de operación de convolución realizada sobre una imagen 6x6 con un filtro 3x3 [32].

4.5.2 Capas convolucionales

En las capas convolucionales se suele utilizar más de un filtro. La salida se obtendrá apilando los distintos mapas de características obtenidos en las convoluciones realizadas con cada filtro.

El *stride* significa de cuánto en cuánto avanza el filtro mientras barre la imagen realizando la operación de convolución.

El *padding* sirve para ampliar los bordes de la entrada, de manera que la salida pueda tener las mismas dimensiones que la entrada (*same convolution*). Se denomina *zero-padding* cuando los valores que se añaden a los bordes de la imagen son ceros.

Se muestra un ejemplo de *stride* y *padding* en la Figura 4.7.

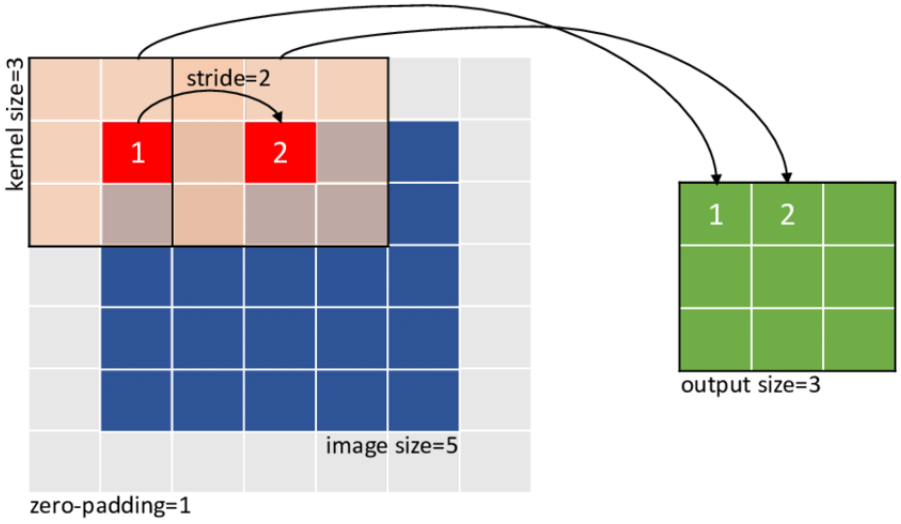


Figura 4.7 Ejemplo de una convolución con *stride* = 2 y *padding* = 1 [33].

Los hiperparámetros de una capa convolucional son, por tanto, el tamaño del filtro, el *stride*, el *padding*, y el número de filtros. Las dimensiones de la salida varían con el *stride* (alto y ancho) y con el número de filtros que se usan (profundidad).

En una red neuronal convolucional, además de existir las capas de convolución, están las capas de *pooling* y las capas completamente conectadas.

Las capas de *pooling* sirven para reducir el tamaño de las entradas, acelerar la computación y para hacer algunas de las características que detecta más robustas. Consiste en barrer la imagen del mismo modo que en la convolución pero realizando una operación de máximo (*max-pooling*) o de promediado de valores (*average-pooling*) en lugar de un producto escalar. Estas capas no tienen parámetros entrenables.

Detrás de las capas convolucionales y las capas de pooling se suelen colocar capas completamente conectadas (*fully connected layers*). Estas son capas con neuronas en la que los nodos están todos entreconectados. Después de pasar las capas completamente conectadas, es común que la capa final use la función de activación *softmax*, que sirve para obtener las probabilidades de que la entrada pertenezca a una clase particular (clasificación). La entrada a esta capa es la salida de la última capa convolucional o de la última capa de pooling, que se aplana (se desenrolla en un vector) en una previa capa de aplanamiento (*flatten layer*).

Un ejemplo de red convolucional clásica es el modelo AlexNet [34], que se esquematiza en la Figura 4.8

Los pesos que se entrenan en las capas convolucionales son los valores de los filtros. Una intuición que se puede generar es que cada filtro aprenderá a buscar una característica de la imagen de entrada de manera que, con cada uno de ellos se irá abstrayendo, a medida que avanza en capas más profundas, características de la imagen. Empezando por representaciones de bajo nivel como colores, ejes, etc. y en capas posteriores, representaciones más complejas. Se muestra un ejemplo en la Figura 4.9

4.5.3 Capa convolucional para entradas unidimensionales

Las capas convolucionales que se han explicado en los apartados anteriores realizaban convoluciones 2D, sacando parches de las imágenes y aplicando transformaciones idénticas a cada uno de ellos. Del mismo modo se pueden utilizar convoluciones 1D, extrayendo parches 1D locales (subsecuencias) de las secuencias, como se muestra en la Figura 4.10.

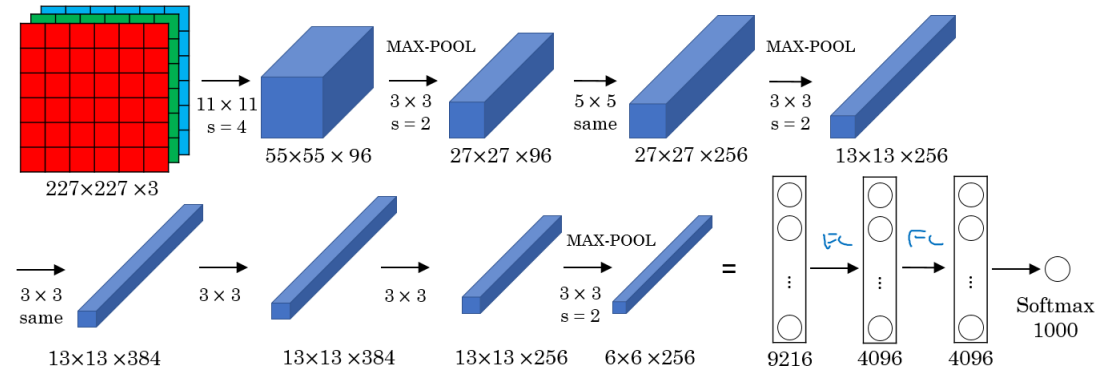


Figura 4.8 Modelo AlexNet [29].

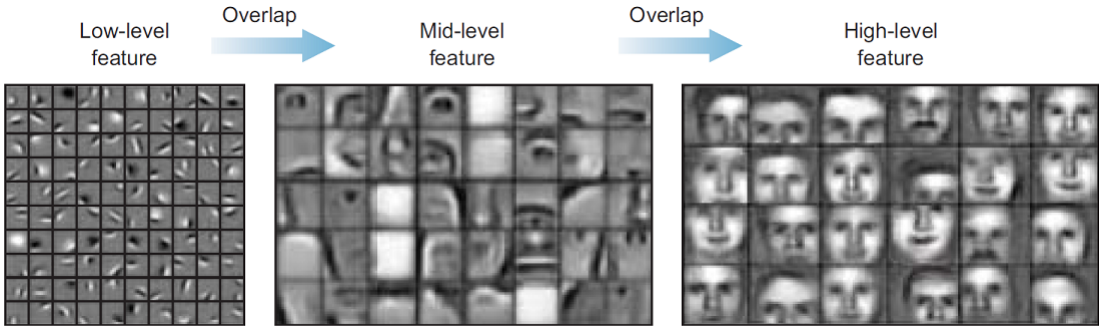


Figura 4.9 Ejemplo de extracción de características [35].

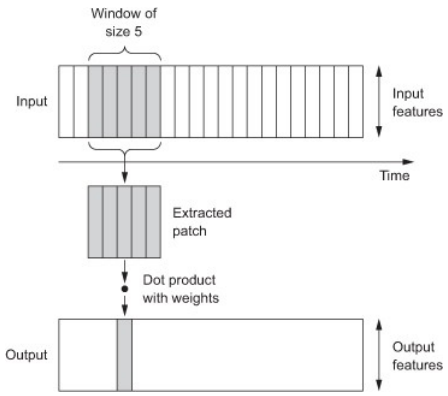


Figura 4.10 Funcionamiento de las convoluciones 1D: cada valor de la secuencia de salida se obtiene usando un parche temporal de la secuencia de entrada [36].

Por tanto, las capas convolucionales podrán ser utilizadas para procesar secuencias temporales. En el siguiente capítulo se construirán modelos en el que se usarán estas capas.

4.6 Redes neuronales recurrentes (RNN)

Las redes neuronales recurrentes, o RNN, son una familia de redes neuronales para procesar datos secuenciales. Así como una red convolucional es una red neuronal especializada en el procesamiento de una cuadrícula de valores X, como una imagen, una red neuronal recurrente es una red neuronal especializada en el procesamiento de una secuencia de valores $x(1),...,x(\tau)$. Las redes recurrentes están especializadas en procesar secuencias mucho más largas de lo que sería práctico mediante

redes convencionales. Además, la mayoría de las redes recurrentes pueden procesar secuencias de longitud variable [31].

4.6.1 Neurona recurrente

La neuronas recurrentes son como las neuronas simples explicadas en el apartado 4.4 pero, a diferencia de éstas, tienen conexiones hacia detrás. Esto implica que la salida de una neurona recurrente en un instante t también dependerá de resultados de instantes previos que se pasan como entrada. De esta manera, se puede decir que tienen una especie de "memoria".

Se muestra un ejemplo de una neurona recurrente en la Figura 4.11.

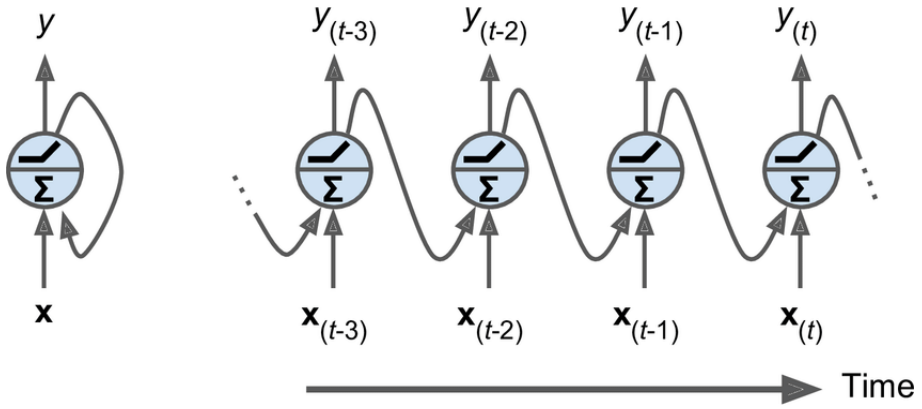


Figura 4.11 Neurona recurrente (izquierda), desarrollada en el tiempo (derecha) [37].

4.6.2 Capa neuronal recurrente

Una capa neuronal recurrente puede contener varias neuronas recurrentes. Este es el ejemplo que se explica a continuación y se muestra en la Figura 4.12

La entrada es x, que puede ser una secuencia de valores escalares (vector) o una secuencia de vectores con cada uno n_x características (secuencia multivariable).

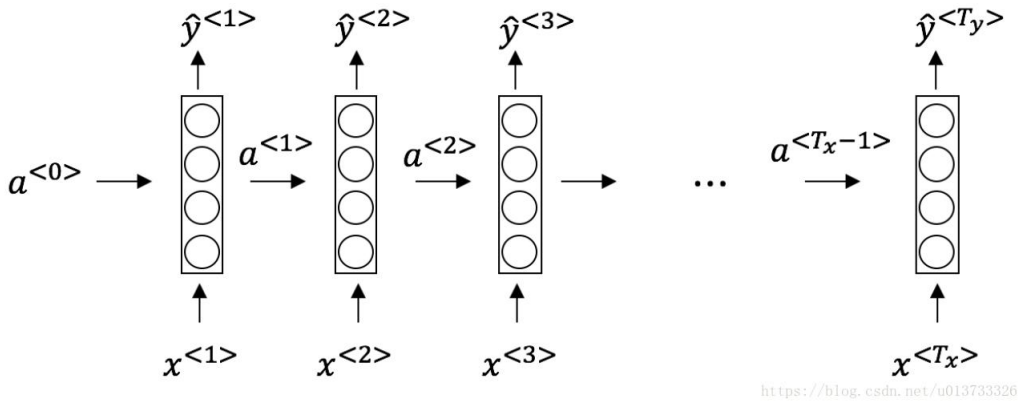


Figura 4.12 Capa neuronal recurrente [29].

En este caso T_x(dimensión temporal de la secuencia de entrada) es igual a T_y(dimensión de la secuencia de salida). En otros problemas, la arquitectura de la RNN puede ser diferente. Se explicarán algunos tipos de arquitecturas en el apartado siguiente.

Las salidas $\hat{y}$ comúnmente se calculan de manera diferente a las activaciones a, que son los valores que se alimentan a los instantes posteriores. El valor de la activación inicial $a^{\langle 0 \rangle}$ se suele inicializar con ceros, pero también se puede inicializar aleatoriamente.

Hay tres matrices de pesos: W_{ax}, W_{aa}y W_{ya} con dimensiones:

- W_{ax} $(n_{neuronas} \times n_x)$: matriz que relaciona la entrada $x^{\langle t \rangle}$ de un instante temporal con la activación de dicho instante $a^{\langle t \rangle}$.
- W_{aa} $(n_{neuronas} \times n_{neuronas})$: matriz que relaciona la activación del instante temporal anterior $a^{\langle t-1 \rangle}$ con la activación de dicho instante $a^{\langle t \rangle}$.
- W_{ya} $(n_y \times n_{neuronas})$: matriz que relaciona la activación del instante temporal $a^{\langle t-1 \rangle}$ con la salida de dicho instante $y^{\langle t \rangle}$.

La matriz de pesos W_{aa} es la memoria que la RNN intenta mantener de las capas anteriores. A continuación, se explican las ecuaciones de propagación.

Las salidas o activaciones de cada instante se calculan como:

$$a^{\langle t \rangle} = g(W_{aa}a^{\langle t-1 \rangle} + W_{ax}x^{\langle t \rangle} + b_a) \tag{4.10}$$

Y la salida de cada instante:

$$y^{\langle t \rangle} = g(W_{ya}a^{\langle t \rangle} + b_y) \tag{4.11}$$

Cabe recalcar que las matrices de pesos no varían de un paso temporal a otro, es decir W_{aa}, W_{ax}, W_{ya}, b_a y b_y son los mismos para todos los pasos temporales dentro de una capa recurrente neuronal.

4.6.3 Arquitecturas de redes recurrentes

En el ejemplo que se ha explicado en la Figura 4.12, el tamaño de la secuencia de entrada era el mismo que el tamaño de la secuencia de salida ($T_x = T_y$), estas arquitecturas de RNN se llaman de secuencia a secuencia. Sin embargo, existen otras arquitecturas en las que esto puede diferir.

Por ejemplo, si sólo se desea el último valor de la secuencia de salida, podemos ignorarlas todas menos ésta. Este tipo de arquitecturas se denominan de secuencia a vector. Un ejemplo de uso puede ser análisis de sentimiento de un texto, evaluando con una puntuación la valoración de un comentario.

Por lo contrario, se puede alimentar la red con una sóla entrada en el primer paso temporal y obtener de salida una secuencia. Éstas se denominan de vector a secuencia. Un ejemplo de uso es la generación sintética de texto, introduciendo una palabra y dejando que la red devuelva una secuencia de palabras.

Finalmente, se puede tener una red de secuencia a vector, llamada *encoder* seguida de una red de vector a secuencia, llamada *decoder*. Esto se puede utilizar para traducir un texto de un idioma a otro, codificando en primer lugar el texto de entrada en un vector y codificándolo posteriormente en otro idioma.

4.6.4 Variantes de RNN

Uno de los problemas de las redes recurrentes simples es que, aunque deberían retener en el tiempo t información sobre las entradas anteriores de la secuencia, muchas veces, estas secuencias son muy largas y, por tanto, es muy difícil la optimización.

En resumen, las RNNs no son buenas en largas dependencias temporales. Esto ocurre debido al desvanecimiento de los gradientes, que significa que los gradientes se hacen tan pequeños que los pesos apenas se actualizan.

Para solucionar la dificultad de memorizar secuencias temporales largas se proponen las redes neuronales recurentes con puertas, como son las LSTM [38] y las GRU.

4.6.5 Capas LSTM

La mayor diferencia entre las RNN y las LSTM es que el método LSTM añade un "procesador" al algoritmo para determinar si la información de entrada es importante o no. Este "procesador" se denomina "célula", y sustituye a las neuronas de las RNN. Como se muestra en la Figura 4.13, se diseñan tres puertas que se utilizan dentro de la célula: puerta de actualización (Γ_u), puerta de olvido (Γ_f) y puerta de salida (Γ_o); de manera que se pueda mantener y actualizar información importante de los datos previos al instante t.

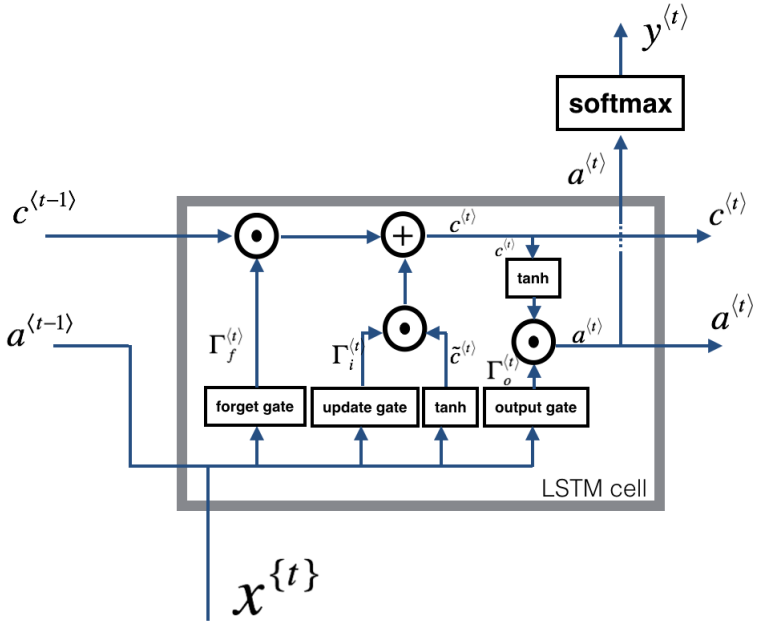


Figura 4.13 Diagrama de una célula LSTM [39].

Como intuición, se puede resumir que las LSTM son iguales que las RNN básicas pero añadiendo una especie de pasillo temporal que capta relaciones temporales largas mediante una célula que aprende a reconocer una entrada con información importante (función de la puerta de actualización), guardarla en la memoria de largo plazo, aprende a presevarla hasta que sea necesario (función de la puerta de olvido), y aprende a extraerla cuando sea necesario (función de la puerta de salida). Esto explica por qué las LSTM han sido tan exitosas en capturar patrones a largo plazo de series temporales, textos largos, audios, etc.

El esquema de los estados de la célula y actualización de parámetros es la siguiente:

$$\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c) \tag{4.12}$$

$$\Gamma_u = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u) \tag{4.13}$$

$$\Gamma_f = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f) \tag{4.14}$$

$$\Gamma_o = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o) \tag{4.15}$$

$$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle} \tag{4.16}$$

$$a^{\langle t \rangle} = \Gamma_o * \tanh c^{\langle t \rangle} \tag{4.17}$$

Donde Γ_u, Γ_f, Γ_o son las puertas de actualización, de olvido y de salida respectivamente. $a^{\langle t-1 \rangle}$ es la salida del instante anterior, $x^{\langle t \rangle}$ es la entrada en el instante actual y $c^{\langle t \rangle}$ es la memoria del bloque anterior. La puerta de olvido Γ_f lee la información $a^{\langle t-1 \rangle}$ y $x^{\langle t \rangle}$ y toma un valor entre 0 y 1 según sea importante preservar o descartar la información de $c^{\langle t-1 \rangle}$. La puerta de actualización Γ_u decide cuánta información se va a actualizar de la célula. Por otro lado, se genera un vector que es el nuevo candidato $\tilde{c}^{\langle t \rangle}$ mediante la función tanh. Usando las puertas de olvido y de actualización, se calcula el nuevo valor del estado de la célula $c^{\langle t \rangle}$, actualizando la información relevante y olvidando la obsoleta. Por último, para determinar la salida de la célula, se usa la puerta de salida Γ_o, que decide qué información del estado de la célula $c^{\langle t \rangle}$ es importante ser leída y entregada como salida de la célula en dicho paso temporal $a^{\langle t \rangle}$.

4.6.6 Capas RNN bidireccionales

Estas capas son una variación de una RNN común que pueden no sólo utilizar los valores pasados de las secuencia sino también valores posteriores para calcular el valor de la salida. Son útiles en las secuencias en las que el orden es muy importante, por ejemplo, para procesamiento de texto.

Al procesar las secuencias en ambas direcciones, pueden encontrar patrones que pueden no haber sido encontrados por las RNN comunes.

Un esquema de estas capas se muestra en la Figura 4.14, en la que se puede observar cómo cada neurona no sólo tiene una conexión con la neurona del paso temporal anterior, sino también una con la posterior.

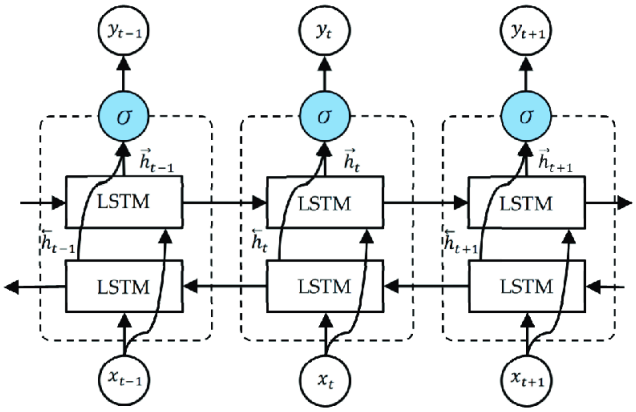


Figura 4.14 Diagrama de RNN bidireccional. En esta figura se denomina *h* a las activaciones, otra forma de denominar a las activaciones muy utilizadas en la literatura [40].

5 Optimización y planificación dinámica en la estación de carga de EV

Como se explica en capítulos anteriores, la producción de energía renovable y la demanda de carga de vehículos eléctricos pueden ser muy estocásticos. Esto puede provocar grandes fluctuaciones en la potencia disponible de la red.

En el ámbito de una estación eléctrica con soporte renovable, estas fluctuaciones de la potencia disponible pueden ser compensadas usando la tecnología V2G-G2V además de un banco de baterías como almacén de energía.

De este modo, surge un problema de optimización para gestionar y planificar los direccionamientos de los flujos de energía óptimos mediante la implementación de una micro-red inteligente. De esta forma se ayudará a estabilizar la red en las horas de mayor demanda, además de obtener un beneficio económico y un uso más eficiente de los recursos.

La novedad de este trabajo consiste en que, para realizar la planificacion dinámica de la carga, se utilizan modelos de deep learning que permiten estimar en un futuro las curvas de demanda y de producción. Usando estas curvas, un algoritmo clásico de optimización permitirá conocer las variables de decisión óptimas en un horizonte temporal.

En este capítulo se explica el problema de optimización y el algoritmo de optimización que se plantea. Posteriormente, se exponen los recursos que se utilizarán en posteriores capítulos para construir modelos de deep learning que sirvan para realizar las predicciones, tanto software como hardware, así como una introducción al flujo de trabajo de un proyecto de deep learning.

La información relativa al problema de optimización se ha realizado en el Trabajo de Fin de Grado *Optimización de carga de vehículos eléctricos con asistencia solar fotovoltaica y almacenamiento de energía* [41].

5.1 Problema de optimización

El problema de optimización se ha formulado teniendo como objetivo la minimización del coste monetario de la planta. La función objetivo debe incluir todas las variables que se pueden controlar, que son la potencia absorbida/cedida por las baterías de la planta y por la red eléctrica.

$$\min_{u} J(t) = \sum_{k=1}^{SH} (J_{GRID}(t_k|t) + J_{BES}(t_k|t)) \tag{5.1}$$

El término $J_{GRID}(t_k|t)$ es la penalización provocada por utilizar la potencia suministrada por la red eléctrica en el instante t_k, conocidos los datos del instante t. Este término podría reducir el coste total del sistema si la planta vende potencia a la red. El término $J_{BES}(t_k|t)$ debe tener dos cuestiones en

consideración, la depreciación que sufren las baterías después de cada kWh cargado o descargado y la penalización provocada por el deterioro de las baterías asociado a los ciclos de carga y descarga.

5.2 Condiciones de factibilidad de la solución

Para asegurar que el problema de optimización planteado tenga solución, deben verificarse ciertos límites y restricciones.

5.2.1 Restricciones operativas y técnicas

- **Potencia de la red eléctrica**

$$P^{GRID}(t) = P^{GRID}_{pur}(t) - P^{GRID}_{sale}(t) \tag{5.2}$$

Ambos términos de la expresión de la potencia de la red eléctrica se consideran positivos y no podrán ser diferentes de cero a la vez. Para indicar en el modo en el que se está trabajando (modo compra, modo venta) se definen dos variables binarias, δ^{GRID}_{pur} y δ^{GRID}_{sale}. Conociendo el valor de $P^{GRID}(t)$ (mayor o menor que 0) se puede saber en que modo se está trabajando. Finalmente, se indica un límite máximo y mínimo de potencia que puede ser suministrada por la red eléctrica.

- **Potencia de las baterías**

$$P^{BES}(t) = P^{BES}_{disc}(t) - P^{BES}_{ch}(t) \tag{5.3}$$

El mismo caso sucede con la potencia de las baterías. Para indicar en el modo en el que se está trabajando (modo carga, modo descarga) se definen dos variables binarias, δ^{BES}_{ch} y δ^{BES}_{disc}. Finalmente, se indica un límite máximo y mínimo de potencia que puede ser suministrada por las baterías y un límite al estado de carga mínimo y máximo que pueden alcanzar.

5.2.2 Restricciones físicas

Se debe cumplir que la energía consumida por los vehículos debe ser igual a la suma de la energía procedente de las diferentes fuentes, que en este caso son el sistema fotovoltaico, el sistema de almacenamiento y la red eléctrica.

5.3 Planteamiento del problema de optimización

A continuación, el problema de optimización se debe adaptar a un formato resoluble por las técnicas de optimización matemáticas.

El procedimiento de optimización que se va a utilizar es la *programación cuadrática de enteros mixtos* (MIQP), ya que la función objetivo es una función cuadrática y las restricciones están formadas por componentes enteras y continuas. Para poder utilizarlo, se debe expresar el problema con el siguiente formato.

Función objetivo:

$$\min_{u} J(t) = x^T Q x + c^T x + \alpha \tag{5.4}$$

Sujeta a:

- Restricciones lineales: $Ax = b$

- Restricciones limitantes: $l \leq x \leq u$

El procedimiento de optimización necesita que las restricciones sean lineales, por lo que se deben adaptar. Siguiendo las transformaciones MLD (Mixed logic dynamics) [42], se puede linealizar este tipo de ecuaciones mediante unas reglas sencillas, incluyendo en ciertos casos alguna variable auxiliar.

5.4 Herramientas para la obtención de las predicciones

La predicción de secuencias temporales ha sido clásicamente atacada por modelos estadísticos, físicos. Los modelos clásicos son generalmente directos y utilizan explicaciones representivas en la composición del modelo, mientras que los métodos de inteligencia artificial producen cajas negras o grises generando los resultados de las predicciones.

Como se ha contado en el capítulo anterior, los métodos de deep learning están teniendo mucho éxito en muchos ámbitos variados. El problema de predicción de demanda o generación de energía es un problema de series temporales.

Tradicionalmente, la previsión de las series temporales ha estado dominada por los métodos lineales porque están muy estudiados y son eficaces en muchos problemas de predicción sencillos.

La aplicación de métodos de deep learning a la predicción de series temporales es prometedora debido al aprendizaje automático de las dependencias temporales y el manejo automático de características temporales como las tendencias y la estacionalidad. Las redes neuronales del deep learning son capaces de aprender automáticamente mapeos complejos arbitrarios de las entradas a las salidas y admiten múltiples entradas y salidas.

Uno de los objetivos de este capítulo es aplicar técnicas transversales, que han estado llamando la atención y se están volviendo muy populares hoy en día, a un problema de optimización real y analizar sus resultados.

En este apartado se detallan algunas herramientas utilizadas de software, como los entornos de desarrollo utilizado y librerías importantes. Por otro lado, respecto al hardware, se introduce la importancia de una unidad de procesamiento gráfico (GPU) y la diferencia respecto a utilizar unidad central de procesamiento de un ordenador (CPU).

5.4.1 Entornos de desarrollo

Existen muchos entornos de desarrollo o IDE distintos, cada uno con puntos a favor y en contra. En este trabajo, para la realización de los scripts del procesamiento de los datos y del entrenamiento, se utilizan indistintamente Spyder y Vscode.

Por otro lado, para la visualización de los datos y de los tests realizados, se utilizan los llamados notebooks de Jupyther. Estos son los archivos que además de contener código pueden contener párrafos de texto, ecuaciones, figuras, etc. Son documentos legibles y visuales que pueden contener los resultados y su análisis (figuras, tablas, etc.), además de poder ser ejecutados para realizar un análisis de los datos.

Los notebooks de Jupyter se pueden editar y manipular desde distintos IDEs. Se destaca la herramienta Colab de google, que permita hacer uso de una especie de máquina virtual volátil en la nube con acceso a una GPU remota.

5.4.2 Gestor de librerías

Uno de los problemas a la hora del desarrollo de software es la instalación de las distintas dependencias para ejecutar un programa y la correcta versión de las mismas. Si se está trabajando en dos proyectos distintos al mismo tiempo, o más de una persona trabajan en un mismo ordenador, la instalación de nuevos paquetes con distintas versiones puede provocar problemas de incompatibilidad.

Para evitar esto, se puede utilizar un gestor de paquetes como Anaconda. Anaconda permite la creación de entornos virtuales de manera que los paquetes instalados sólo pertenezcan a ese entorno y no se tengan relación con los instalados en otro. De esta forma se evitan conflictos de compatibilidad con las versiones establecidas por las dependencias de un programa, que son las librerías que son necesarias instalar para el correcto funcionamiento.

En este trabajo se utiliza Keras con Tensorflow, debido a la mayor facilidad para implementar modelos propios y la experimentación sobre ellos.

5.4.3 Frameworks

Existen distintas librerías o *frameworks* para machine learning. Entre ellas, las tres más famosas actualmente son Keras, Tensorflow y Pytorch.

Keras es una API de alto nivel para el desarrollo de redes neuronales. Es ideal para la experimentación con modelos. Tiene mucho potencial por el fácil prototipado. Puede implementarse también por encima de Tensorflow.

Tensorflow es un framework de código abierto de aprendizaje automático desarrollada por Google. Es una librería matemática simbólica utilizada para las redes neuronales y es la más adecuada para la programación de flujo de datos en una serie de tareas. Ofrece múltiples niveles de abstracción para construir y entrenar modelos.

Pytorch es una framework de código abierto de aprendizaje automático basado en Torch, es relativamente nueva y fue desarrollada por el grupo de investigación de IA de Facebook.

Tensorflow y Pytorch son de más bajo nivel. Es por ello que tienen mayor flexibilidad y más funcionalidades. Ambos tienen rendimientos parecidos, son las más rápidas y eficientes.

5.4.4 GPU vs CPU

La función de las GPU y CPU es la de realizar las operaciones que requieren los algoritmos de los procesos. Por tanto, es importante conocer la diferencia entre ambas.

La GPU es mucho más rápida que una CPU debido a el gran ancho de banda, la reducción de la latencia bajo el paralelismo de los hilos y los registros fácilmente programables.

Debido a los factores anteriores, la CPU puede utilizarse para entrenar el modelo cuando los datos son relativamente pequeños. La GPU es adecuada para entrenar los modelos de deep learning a largo plazo para conjuntos de datos muy grandes debido a que acelera el entrenamiento. La CPU también se puede utilizar para entrenar un modelo de deep learning aunque irá con mayor lentitud.

Nvidia es la única empresa de computación gráfica que ha invertido fuertemente en deep learning hasta el momento. Es por eso que las librerías modernas de deep learning solo pueden correr en tarjetas de Nvidia. En este trabajo, se ha utilizado un ordenador AORUS17G WB con una RTX2070.

5.5 Flujo de trabajo en un proyecto de deep learning

En este apartado se explican las distintas fases de un proyecto de estas características.

1. Definición del problema: Consiste en analizar cuál es el problema que se desea resolver, analizar los datos hipotéticamente disponibles y fijar el objetivo final del proyecto.
2. Obtención de datos: Se ha de determinar qué datos se quieren recoger y se elige un intervalo de tiempo. Los datos se pueden obtener también mediante bases de datos que se puedan asimilar a las condiciones de nuestro proyecto. Es una fase muy importante, ya que muchas veces la calidad de los datos es más importante que los modelos en sí.
3. Procesamiento y análisis de datos: Los datos se visualizan para entenderlos y valorar la utilidad de los mismos. Se limpian, se filtran y posteriormente se exportan para utilizarlos en los entrenamientos.

4. Construcción de modelos: Los modelos se seleccionan según las características del problema que se quiere resolver. Valorando la tipología de los datos (texto, audio, imágenes, datos temporales, etc.) se pueden seleccionar las redes neuronales más adecuadas. También es necesario realizar el diseño de las propias arquitecturas (número de capas, funciones de activación, etc.) o utilizar otras ya existentes y probadas.
5. Entrenamiento de modelos: Es un procedimiento iterativo que consiste en entrenar, validar y reajustar el modelo hasta que funcione lo suficientemente bien. Para ello se debe seleccionar previamente las métricas de validación que se van a utilizar para medir la calidad del funcionamiento.
6. Desplegar en producción: Una vez se despliega el modelo, se pueden recoger más datos para volver a entrenar y ajustar los modelos. Se realizan mantenimientos y actualizaciones.

6 Predicción de la demanda en la estación de carga de EV

Como se explica en el capítulo anterior, para la implementación del algoritmo de optimización para la planificación dinámica de los flujos de carga de la estación será necesario realizar un modelo que haga predicciones sobre el consumo y la generación de energía de la estación. Este capítulo se centra en el primer problema de predicción: la demanda de carga en la estación.

La predicción precisa de la demanda tiene el potencial de traer significantes beneficios económicos y sociales. El modelo de optimización propuesto tiene como intención allanar el camino del crecimiento de las tecnologías de vehículos eléctricos y su integración en los sistemas de potencia. Además, proporciona un escaparate de la inteligencia artificial en los sistemas energéticos sostenibles.

Las técnicas de predicción se suelen clasificar según el tamaño de la secuencia de entrada y el horizonte temporal de predicción. En este caso, al utilizar un horizonte temporal de 24 horas (24 valores), se puede clasificar como un problema de predicción a medio-largo plazo.

En este capítulo se realiza, en primer lugar, la obtención y el procesamiento de datos de sesiones de carga de EVs. Posteriormente se diseñan modelos con distintas arquitecturas y tipos de redes neuronales, para luego ser entrenados y validados. Finalmente se realizan pruebas con los modelos entrenados y se analizan los resultados.

6.1 Datos

En este primer apartado se explica de dónde provienen los datos, cómo se filtran y se limpian y cómo se transforman para que tengan la estructura necesaria para poder entrenar a los modelos.

6.1.1 Obtención de los datos

Para realizar predicciones de la demanda de carga se utilizan datos reales de sesiones de carga en una estación de la Universidad de Caltech, en Estados Unidos. Los datos que se utilizan están públicamente disponibles en internet [43].

Los datos recogen distintos campos de las sesiones de carga. Se muestran en la Tabla 6.1 los mismos, con su significado.

Como se puede ver, los datos incluyen la hora de inicio y final de la sesión y la energía cedida en la sesión. Sin embargo, para el problema de predicción se necesita obtener los valores de la demanda en cada hora, teniendo en cuenta que pueden haber dos sesiones activas simultáneamente.

Tabla 6.1 Campos de los datos y su descripción [43].

Field	Type	Description
_id	string	Unique identifier of the session record.
chargingCurrent	timeseries	Time series of the measured current draw of the EVSE during the session.
clusterID	string	Unique identifier for a subset of EVSEs at a site, such as a single garage.
connectionTime	datetime	Time when the EV plugged in.
disconnectTime	datetime	Time when the EV unplugged.
doneChargingTime	datetime	Time when of the last non-zero current draw recorded.
kWhDelivered	float	Amount of energy delivered during the session.
pilotSignal	timeseries	Time series of the pilot signals passed to the EVSE during the session.
sessionID	string	Unique identifier for the session.
siteID	string	Unique identifier for the site.
spaceID	string	Unique identifier of the parking space.
stationID	string	Unique identifier of the EVSE.
timezone	string	Timezone of the site. Based on pytz format.
userID	string	Unique identifier of the user. Not provided for sessions which are not claimed using the mobile app.
userInputs	list(User Input)	Inputs provided by the user. Since inputs can be changed over time, there can be multiple user input objects in the list.

6.1.2 Preprocesamiento

El objetivo es obtener una serie temporal de la demanda de carga horaria en la estación. Para ello, se han de hacer una serie de operaciones de preprocesamiento usando los datos disponibles. Estas operaciones se realizan en el script *data_preprocessing.py*. Para la lectura, procesamiento y exportación de los datos se usa la librería de Python Pandas, que sirve para el análisis y manipulación de datos.

Inicialmente se leen y se cargan los datos. Usando la hora de inicio y fin de la sesión se calcula el tiempo de duración. Posteriormente, se calcula la potencia media de la sesión de carga dividiendo la energía de la sesión entre el intervalo de tiempo. Esto se realiza usando el Código 6.1.

Código 6.1 Lectura de datos y cálculo de potencia media de cada sesión.

```
import pandas as pd
import json

# Read the data
data_path = 'raw_data/acndata_sessions_2019.json'

data = json.load(open(data_path))
df = pd.DataFrame(data["_items"])

# Convert string column to a datetime64 type
df['connectionTime'] = pd.to_datetime(df['connectionTime'])
df['disconnectTime'] = pd.to_datetime(df['disconnectTime'])
```

```
# We take the columns needed
df=df[['connectionTime','disconnectTime','kWhDelivered']]

# Add a column named dummy
df['dummy'] = 1

# Data time interval
timedelta = df['connectionTime'].max() - df['connectionTime'].min()

# Add a column for session time interval
interval = df['disconnectTime']-df['connectionTime']

# Change time format and compute the power
interval=interval.dt.total_seconds()
interval=interval/3600
df['sessionTime']=interval
df['medPower']=df['kWhDelivered']/df['sessionTime']
```

Una vez se tiene la potencia media de cada sesión de carga, para calcular la demanda de carga horaria total de la estación se realiza una integración de las potencias de sesiones de carga concurrentes. Para ello, se dividen las sesiones en franjas horarias y las que se solapen, se sumarán. Se guardarán también, además de la suma horaria de las potencias, el número de sesiones concurrentes en cada franja de tiempo y la mediana de las potencias medias de dichas sesiones. Esto se realiza en el Código 6.2.

Código 6.2 Lectura de datos y cálculo de potencia media de cada sesión.

```
# Create a datetimeindex
date_series=pd.date_range(start=df['connectionTime'].min(),end=df['
    disconnectTime'].max(),freq="1H")

# Create a dataframe with the datetimeindex and a column named dummy
date_df = pd.DataFrame(dict(date=date_series, dummy=1))

# Create an union, join the column dummy to df dataframe
cross_join = date_df.merge(df, on='dummy')
cond_join = cross_join[(cross_join.connectionTime <= cross_join.date) &
    (cross_join.date <= cross_join.disconnectTime)]
grp_join = cond_join.groupby(['date'])
```

Finalmente, se guardan los datos en un nuevo *dataframe* y se exportan a un archivo comprimido del tipo .pkl. Esto se realiza en el Código 6.3

Código 6.3 Creación del nuevo *dataframe* y exportación de los datos.

```
# Create the new dataframe
final = (
    pd.DataFrame(dict(
        val_sessions=grp_join.size(),
        val_total_power=grp_join.instantPower.sum(),
        val_power_median=grp_join.instantPower.median()
```

```
    ), index=date_series)
    .fillna(0)
    .reset_index()
)

#%% Export 'final' dataframe into a pkl file
final.to_pickle("preprocessed_data/final_2018")
```

6.1.3 Visualización de los datos

Los datos se pueden abrir desde un notebook de Jupyter (*data_processing.ipynb*), de manera que se puedan manipular y representar de una forma más visual.

En dicho notebook, se importan los datos y se muestran las primeras filas:

Código 6.4 Unión de los datos y representación de las primeras filas.

```
# Import dataframes
df = pd.read_pickle('preprocessed_data/final_2018')
df1 = pd.read_pickle('preprocessed_data/final_2019')
df2 = pd.read_pickle('preprocessed_data/final_2020')

# Append preprocessed dataframes to create a new one
df=df.append(df1,ignore_index=True);
df=df.append(df2,ignore_index=True);

# Show first rows
df.head()
```

	index	val_sessions	val_total_power	val_power_median
0	2018-04-25 11:08:04+00:00	1.0	3.602725	3.602725
1	2018-04-25 12:08:04+00:00	1.0	3.602725	3.602725
2	2018-04-25 13:08:04+00:00	1.0	3.602725	3.602725
3	2018-04-25 14:08:04+00:00	2.0	1.459559	0.729779
4	2018-04-25 15:08:04+00:00	7.0	9.822627	0.895217

Después, se representa la serie temporal de la demanda horaria entre los años 2018 y 2019 en la Figura 6.1 usando el Código 6.5.

Código 6.5 Representación de la demanda de potencia horaria (kW) en 2018-2020.

```
# Plot hourly demanded power
df.plot(x='index',
     y='val_total_power',
     title='Total kW')
```

Se puede observar la falta de datos de sesiones en 2020. Esto se debe posiblemente a la época de pandemia. Se usan los datos de 2019 y se descartan el resto, ya que son los datos más regulares y de mayor calidad, lo que facilitará el entrenamiento. Los datos de 2019 se muestran en la Figura 6.2.

Código 6.6 Representación de la demanda de potencia horaria (kW) en 2018-2020.

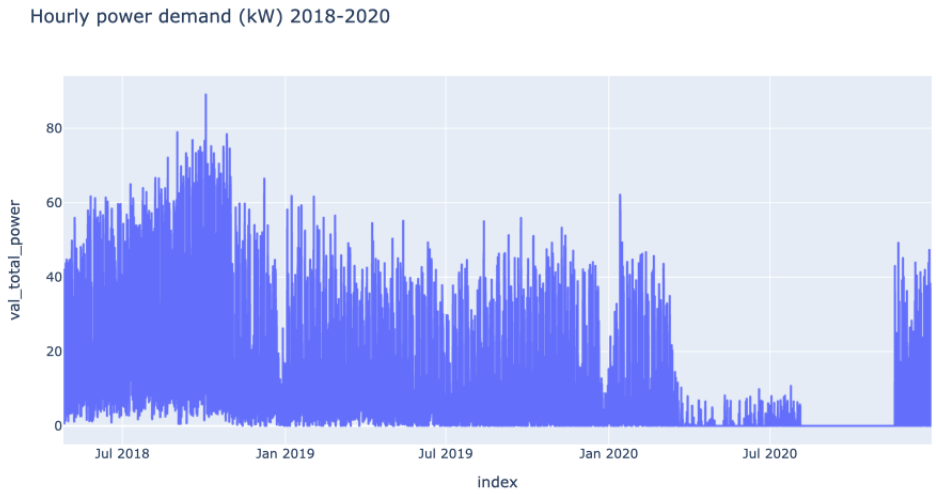


Figura 6.1 Representación de la demanda de potencia horaria (kW) en 2018-2020.

```
# Filter and select the interval to crop data
start='2019-01-01 00:01:00+00:00'
end='2019-12-10 23:59:00+00:00'

filt = (df['index'] >= pd.to_datetime(start)) & (df['index'] < pd.
    to_datetime(end))
df=df.loc[filt]

# Plot new-cropped data
df.plot(x='index',
        y='val_total_power',
        title='Total kWh consumed')
```

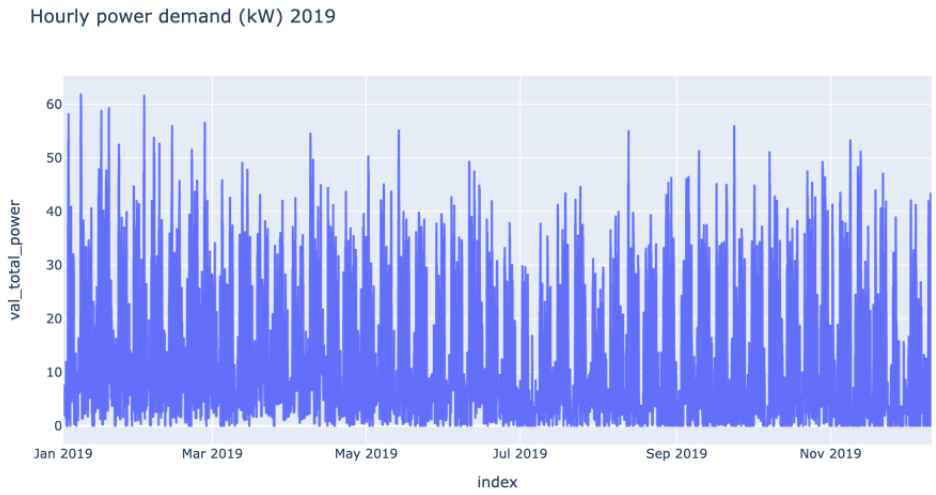


Figura 6.2 Representación de la demanda de potencia horaria (kW) en 2019.

Se representa también la mediana, para visualizar el valor de la potencia por sesión más representativo, en la Figura 6.3.

Código 6.7 Representación de la mediana de las potencias (kW) en 2019.

```
# Plot median of the power consumed by different concurrent sessions
df.plot(x='index',
        y='val_power_median',
        title='Median power (kW) per session')
```

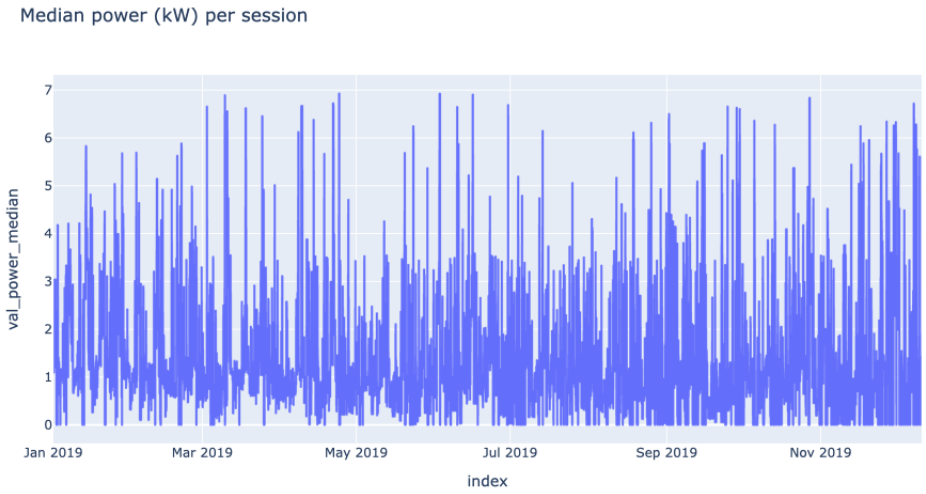


Figura 6.3 Representación de la mediana de las potencias (kW) en 2019.

Finalmente, se exportan y se guardan los datos resultantes, que son los que se usarán posteriormente en el entrenamiento de los modelos.

6.2 Modelos

La selección de una buena arquitectura es crucial para el desarrollo del proyecto. Es por eso que se prueban distintas modelos con arquitecturas y con distintos tipos de redes neuronales.

Se proponen modelos de redes neuronales simples, recurrentes, con LSTM, convolucionales, bidireccionales y combinaciones de ellas. A continuación se detallan las distintas arquitecturas y se explica por qué se han escogido.

Para la construcción del modelo se elabora la función denominada *build_model*. Ésta sirve para construir el modelo de manera automática introduciendo el nombre del modelo que se desea construir como una cadena de caracteres. Esta función llamará a la función que construya el modelo deseado. Se muestra en el Código 6.8.

Código 6.8 Función build_model, para elegir qué modelo construir según la entrada introducida.

```
def build_model(name, window_size, num_features):
    if name == 'CNN_LSTM':
        return get_CNN_LSTM(window_size, num_features)
    elif name == 'Simpler_RNN':
```

```
        return get_Simpler_RNN(window_size, num_features)
    elif name == 'Simple_RNN':
        return get_Simple_RNN(window_size, num_features)
    elif name == 'LSTM':
        return get_LSTM(window_size, num_features)
    elif name == 'LSTM_stacked':
        return get_LSTM_stacked(window_size, num_features)
    elif name == 'Bidirectional_LSTM':
        return get_Bidirectional_LSTM(window_size, num_features)
    elif name == 'Simple_ANN':
        return get_Simple_ANN(window_size, num_features)
```

6.2.1 Red neuronal simple, perceptrón multicapa (ANN)

En general, las redes neuronales como los Perceptrones Multicapa o MLP proporcionan distintas capacidades.

Una de sus características es la robustez al ruido en los datos de entrada y en la función de mapeo. Pueden incluso soportar el aprendizaje y la predicción en ausencia de algunos valores.

Por otro lado, las redes neuronales aprenden fácilmente relaciones lineales y no lineales.

También admiten entradas multidimensionales, es decir, se pueden utilizar un número arbitrario de características de entrada, que en este caso, pueden ser los valores de la secuencia de entrada. No obstante, en una red de perceptrón multicapa, se perdería la dimensión temporal de la secuencia de los datos.

Código 6.9 Función para construir el modelo de red neuronal simple.

```
def get_Simple_ANN(window_size, num_features):
    # ANN single
    reshape = tf.keras.layers.Lambda(lambda x: tf.squeeze(x, axis=-1),
                      input_shape=[window_size,num_features])
    l0 = tf.keras.layers.Dense(1)
    model = tf.keras.models.Sequential([reshape,l0])
    return model
```

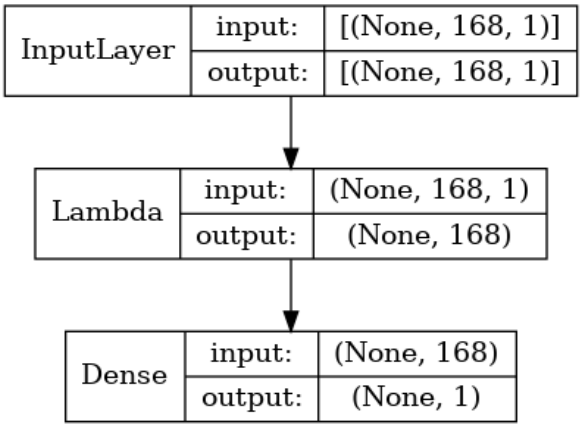


Figura 6.4 Resumen modelo ANN.

6.2.2 Red neuronal recurrente (RNN)

Como se vio en el Capítulo 4, las redes neuronales recurrentes tienen la capacidad de tener en cuenta la temporalidad de los datos. Es por ello que lo hace teóricamente un modelo más apropiado para un problema de predicción de series temporales.

Código 6.10 Función para construir el modelo de red neuronal recurrente simple.

```
def get_Simpler_RNN(window_size, num_features):
    # Simpler RNN
    model = tf.keras.models.Sequential([
      tf.keras.layers.SimpleRNN(64, activation='relu', input_shape = [
          window_size, num_features]),
      tf.keras.layers.Dense(1),
    ])
    return model
```

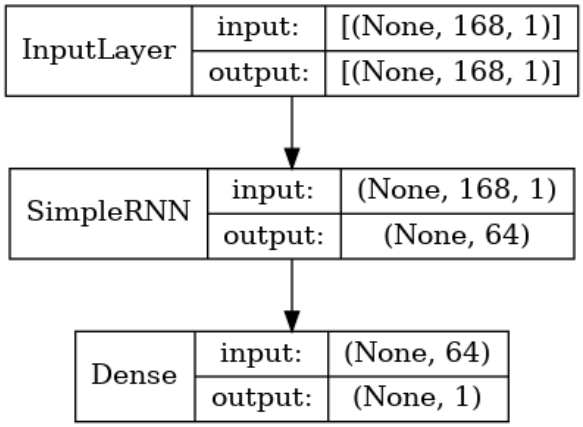


Figura 6.5 Resumen modelo RNN.

6.2.3 Red neuronal recurrente multicapa (st-RNN)

Este modelo cuenta con dos capas apiladas de neuronas recurrentes. La secuencia de entrada al modelo pasa por la primera capa, que genera de nuevo otra secuencia que al pasar por la segunda obtiene un valor a su salida, que tras pasar por una neurona común, será la salida del valor futuro predicho.

Código 6.11 Función para construir el modelo de red neuronal recurrente apilada.

```
def get_Simple_RNN(window_size, num_features):
    # Simple RNN
    model = tf.keras.models.Sequential([
      # tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                        # input_shape=[None]),
      tf.keras.layers.SimpleRNN(40, return_sequences=True, input_shape=[
          window_size,num_features]),
      tf.keras.layers.SimpleRNN(40),
      tf.keras.layers.Dense(1),
```

```
    ])
```

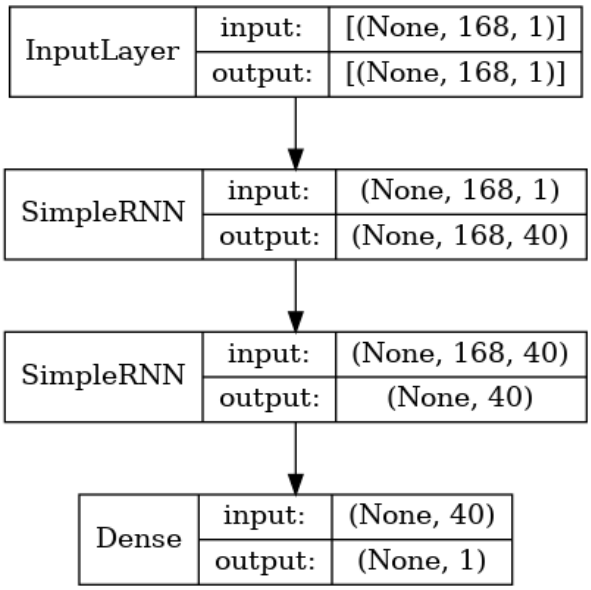


Figura 6.6 Resumen modelo st-RNN.

6.2.4 Red neuronal recurrente LSTM (LSTM)

Este modelo es el mismo que el modelo RNN pero utilizando neuronas del tipo LSTM, que en principio deberían mejorar el funcionamiento del modelo al tener estas neuronas la capacidad de memorizar información pasada en secuencias más largas.

Código 6.12 Función para construir el modelo de red neuronal LSTM.

```
def get_LSTM(window_size, num_features):
    # LSTM 1 * 64
    model = tf.keras.models.Sequential([
      tf.keras.layers.LSTM(64, return_sequences=False, input_shape = [
          window_size, num_features]),
      tf.keras.layers.Dense(1),
      ])
    return model
```

6.2.5 Red neuronal recurrente LSTM multicapa (st-LSTM)

Del mismo modo que se construyó la RNN-st, se construye una red neuronal recurrente multicapa con neuronas del tipo LSTM.

Código 6.13 Función para construir el modelo de red neuronal recurrente LSTM apilada.

```
def get_LSTM_stacked(window_size, num_features):
    # LSTM 2 * 32
    model = tf.keras.models.Sequential([
      tf.keras.layers.LSTM(32, return_sequences=True, input_shape = [
          window_size, num_features]),
```

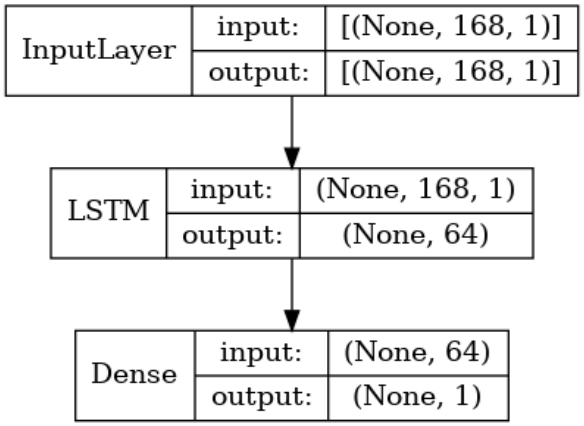


Figura 6.7 Resumen modelo LSTM.

```
      tf.keras.layers.LSTM(32),
      tf.keras.layers.Dense(1),
      ])
    return model
```

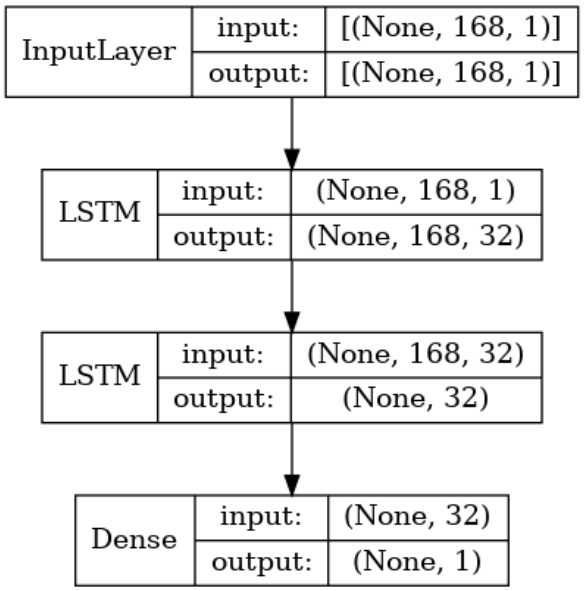


Figura 6.8 Resumen modelo st-LSTM.

6.2.6 Red neuronal recurrente bidireccional (Bi-LSTM)

Para este modelo se utilizan neuronas del tipo recurrentes bidireccionales que podrán tener en cuenta valores del pasado y del futuro a la hora de calcular las activaciones de las neuronas.

Código 6.14 Función para construir el modelo de red neuronal recurrente LSTM bidireccional.

```
def get_Bidirectional_LSTM(window_size, num_features):
    # Bidirectional LSTM
    model = tf.keras.models.Sequential([
```

```
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32,
       return_sequences=True), input_shape = [window_size,
       num_features]),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32)),
    tf.keras.layers.Dense(1),
    ])
  return model
```

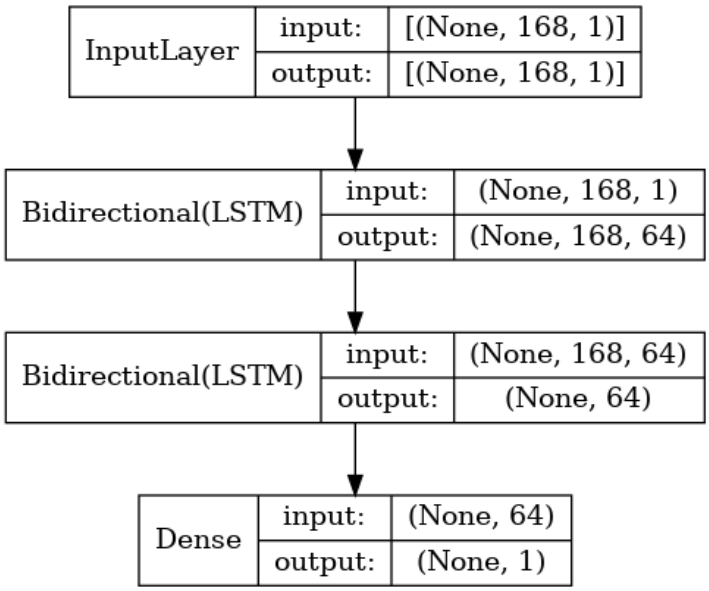


Figura 6.9 Resumen modelo Bi-LSTM.

6.2.7 Red neuronal convolucional y LSTM (CNN + LSTM)

La capacidad de las CNN para aprender y extraer automáticamente características de los datos de entrada puede aplicarse a los problemas de previsión de series temporales. En este modelo se utiliza una capa convolucional que genera una secuencia unidimensional que se pasa posteriormente por dos capas de neuronas recurrentes del tipo LSTM. Este modelo es el más complejo y es posible que sea más adecuado para problemas de predicción más difíciles, en el que la dimensión de las secuencias de entrada sean mayores.

Código 6.15 Función para construir el modelo CNN + LSTM.

```
def get_CNN_LSTM(window_size, num_features):
    # CNN + LSTM
    model = tf.keras.models.Sequential([
      # tf.keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
      #                   input_shape=[None,1]),
      tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                         strides=1, padding="causal",
                         activation="relu", input_shape=[window_size,
                            num_features]),
      tf.keras.layers.LSTM(64, return_sequences=True),
      tf.keras.layers.LSTM(64, return_sequences=False),
      tf.keras.layers.Dense(30, activation="relu"),
```

```
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(1),
  ])
  return model
```

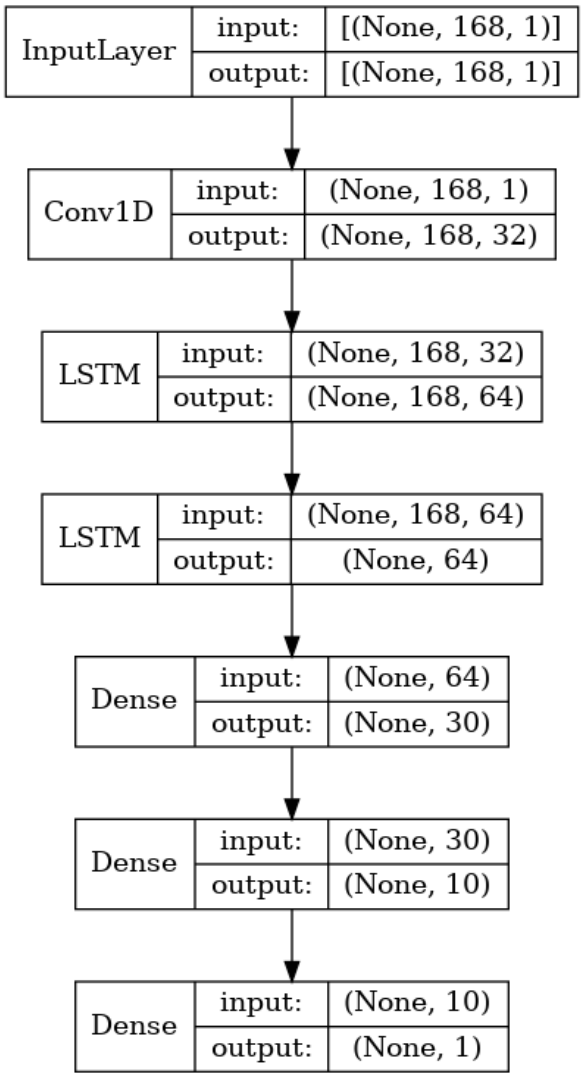


Figura 6.10 Resumen modelo CNN+LSTM.

6.3 Entrenamiento

Para el entrenamiento se ha desarrollado el script de python *train.py* y el notebook de jupyter *train_-demo.ipynb*. Éste último se ha realizado con el fin de mostrar de manera más visual como se realiza un entrenamiento, pero el script *train.py* es el que se utilizará a la hora de realizar entrenamientos y guardar las salidas de los mismos.

Entre los distintos pasos a tomar para realizar un entrenamiento se encuentran la carga y el normalizado de los datos. Posteriormente, se dividen los datos en un set de entrenamiento y otro de validación. Luego se construye el modelo y se compila y, finalmente, se realiza el entrenamiento.

6.3.1 Normalizado de los datos

Los datos deben ser normalizados antes de ser alimentados al modelo, de forma que los cálculos internos del entrenamiento se simplifican y la convergencia de la red se acelera. En este caso se normalizan los datos linealmente entre 0.1 y 1. Se evita el valor de 0 para evitar problema de desaparición del gradiente en la optimización de los pesos.

La fórmula es la siguiente:

$$y = \frac{x - x_{min}}{x_{max} - x} \tag{6.1}$$

Se implementa mediante la función *MinMaxScaler* de la librería *sklearn*, como se muestra en el Código 6.16.

Código 6.16 Normalización de los datos.

```
# Normalize the dataset
scaler = MinMaxScaler(feature_range=(0.1, 1))
series_scaled = scaler.fit_transform(series)
```

6.3.2 División de los datos para entrenamiento y validación

Se entrenarán los modelos usando una parte de los datos, estos datos son los datos de entrenamiento.

Se dejan reservados una parte de los datos que permitirán realizar validaciones, es decir, permitirán comprobar el funcionamiento del modelo sobre datos que no han sido usados para entrenar al modelo, por tanto, datos que el modelo no habrá visto previamente.

Se toma un 66 % de los datos para construir el set entrenamiento, y se reserva un 33 % para el set de validación.

La división de los datos se realiza en el Código 6.17.

Código 6.17 Normalización de los datos.

```
# Split training and validation sets
split_time = int(len(times) * 0.66)

time_train = times[:split_time]
x_train = series[:split_time] # Not scaled
x_train_scaled = series_scaled[:split_time]

time_test = times[split_time:]
x_test = series[split_time:] # Not scaled
x_test_scaled = series_scaled[split_time:]
```

6.3.3 Creación de generadores de entrenamiento y validación

Las series temporales de datos deben ser convertidas a estructuras de entradas y salidas antes de que puedan ser usados para entrenar a un modelo de aprendizaje supervisado.

Esto se puede realizar de una manera automática mediante el uso de generadores de series temporales. Para ello, se utiliza la función *preprocessing.sequence.TimeseriesGenerator* de Keras.

A la función hay que especificarle los datos de las entradas, los datos de las salidas, la longitud de las secuencias de entrada, y el número de pares entrada-salidas apilados (*batch size*), que servirá

para acelerar el entrenamiento, y esta nos devolverá los generadores que crearán las estructuras de entradas y salidas.

Se crea un generador para los datos de entrenamiento y otro para los datos de validación.

La creación de los generadores se realiza en el Código 6.18.

Código 6.18 Creación de los generadores de datos.

```
# Create the data generators for training
num_features = 1
batch_size=16
train_generator = TimeseriesGenerator(x_train_scaled,
                                      x_train_scaled,
                                      length=window_size,
                                      batch_size=batch_size)
validation_generator = TimeseriesGenerator(x_test_scaled,
                                           x_test_scaled,
                                           length=window_size,
                                           batch_size=batch_size)
```

6.3.4 Construcción del modelo

El modelo se construye llamando a la función explicada en el apartado 6.2. El argumento de esta función será el nombre del modelo, además del tamaño de la secuencia de entrada y la dimensión de los valores de la secuencia (unidimensionales en este caso).

La variable *model_name* podrá ser:

- Simple_ANN
- Simpler_RNN
- Simple_RNN
- LSTM
- LSTM_stacked
- Bidirectional_LSTM
- CNN_LSTM

Código 6.19 Llamada a la función para construir el modelo.

```
# Build the model:
model = build_model(model_name,
                    window_size,
                    num_features)
```

6.3.5 Compilación del modelo

La compilación es el paso final de la creación de un modelo. Una vez se ha realizado, se puede pasar a la fase de entrenamiento.

Mediante la compilación del modelo, se especifica la función de coste que utilizará, el optimizador y la métrica de evaluación. Se realiza un breve resumen a modo de recordatorio a lo que se explica en el capítulo 4.

- **Función de coste:** Sirve para calcular el error o la desviación en el proceso de entrenamiento. Se tratará de obtener los parámetros óptimos de la red que minimicen dicho coste.
- **Optimizador:** Proceso que optimiza los pesos de la red comparando las salidas predichas con las reales, utilizando la función de coste.
- **Métrica de evaluación:** Se utiliza para evaluar el funcionamiento del modelo a medida que va transcurriendo el entrenamiento.

En este caso, se utiliza la función de coste de *Huber* con el optimizador *Adam* y la precisión como métrica de evaluación. Se muestra en el Código 6.20.

Código 6.20 Llamada a la función para construir el modelo.

```
# Compilate the model
optimizer = 'adam'
metrics = ['acc']
model.compile(loss=tf.keras.losses.Huber(),
            optimizer=optimizer,
            metrics=metrics)
```

6.3.6 Configuraciones del entrenamiento

Para añadir configuraciones extras al entrenamiento se usan los denominados *callbacks*. Un *callback* es un objeto que puede llevar a cabo acciones en varias fases del entrenamiento (al principio, al final después de cada época, etc.).

En este caso se implementan el *early stopping* y el guardado de *checkpoints* de los pesos del entrenamiento en cada época. El *early stopping* sirve para que se detenga el entrenamiento cuando una métrica monitorizada deja de mejorar durante un número de épocas especificadas (paciencia).

Código 6.21 Configuración de los callbacks.

```
# Configure callbacks
callbacks = configure_callbacks(early_stopping = True,patience=10,
                            checkpoint= True, model_name= model_name )
```

6.3.7 Entrenamiento del modelo

Para realizar el entrenamiento del modelo, se usan los generadores de entrenamiento y validación creados anteriormente y los *callbacks*. Se introduce también el número de épocas, es decir, el número de pasos completos que se realizan sobre los datos de entrenamiento.

Código 6.22 Configuración de los callbacks.

```
# Train model
history = model.fit_generator(generator=train_generator,
                            verbose=1,
                            epochs=epochs,
                            validation_data=validation_generator,
                            callbacks=callbacks)
```

Una vez haya finalizado el entrenamiento del modelo, se crea un nuevo directorio en la carpeta *Resultados*, en la que se exportará y se guardará el archivo del modelo entrenado. También se guardará en este directorio una archivo de texto con las métricas de validación de los modelos, además de distintas gráficas relativas al proceso del entrenamiento y primeras gráficas con predicciones de prueba.

Código 6.23 Configuración de los callbacks.

```
# Create new dir to save results
try:
    os.mkdir(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
        _results')
except:
    pass

# Save the trained model
model.save(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/{model_name}_ws{window_size}_epochs{epochs}_model.h5')

# Plot the training and validation accuracy and loss at each epoch
plot_training(history)
plt.savefig(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/history.png')
```

La gráfica del proceso del entrenamiento es una representación del los costes de entrenamiento y validación frente a las épocas o iteraciones completas realizadas sobre el set de datos de entrenamiento. Con esta gráfica, se puede observar la manera en la que converge el error del modelo tras el progresivo ajuste de los pesos. Puede ser útil para analizar la bondad del entrenamiento y de los hiperparámetros escogidos, por ejemplo, si se observa que el error de entrenamiento se hace mucho menor que el error de validación, se puede detectar que hay sobreajuste a los datos de entrenamiento (*overfitting*), y se puede ver sobre qué época es mejor finalizar el entrenamiento. Estos análisis se realizarán posteriormente en el apartado de resultados.

6.4 Validación

Una vez se ha realizado el entrenamiento del modelo, se pueden realizar predicciones sobre datos de validación, es decir, datos que no han sido utilizados para entrenar al modelo, de manera que se pueda evaluar si el funcionamiento del modelo se "extrapola" a datos que no ha visto anteriormente. De esta manera, se puede asegurar que el modelo está generalizando el conocimiento aprendido en el entrenamiento.

La evaluación se puede realizar visualmente, realizando predicciones con el modelo y representando los datos predichos frente a los reales. No obstante, para evaluar correctamente el funcionamiento de manera cuantitativa se utilizan distintas métricas que servirán para medir el error de los datos predichos frente a los reales.

Las métricas utilizadas en este trabajo son la raíz del error cuadrático medio (RMSE) y el error absoluto medio (MAE). Estos errores se calcularán en las predicciones sobre los datos de entrenamiento y sobre los datos de validación respecto a los valores reales correspondientes. Esto se realiza en el Código 6.24.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{6.2}$$

$$RMSE = \sqrt{\sum_{j=1}^{n}\frac{(y_j - \hat{y}_j)^2}{n}} \tag{6.3}$$

Código 6.24 Fase de validación.

```
# Run predictions on training and validation set
train_predict = predict(model,
                        train_generator,
                        scaler)

test_predict = predict(model,
                       validation_generator,
                       scaler)

# Validate the model using RMSE and MAE
trainScore = math.sqrt(mean_squared_error(x_train[window_size:],
    train_predict[:,0]))
testScore = math.sqrt(mean_squared_error(x_test[window_size:],
    test_predict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
print('Test Score: %.2f RMSE' % (testScore))

train_mae = mean_absolute_error(x_train[window_size:], train_predict
    [:,0])
test_mae = mean_absolute_error(x_test[window_size:], test_predict[:,0])
print('Train Score: %.2f MAE' % (train_mae))
print('Test Score: %.2f MAE' % (test_mae))

# Write validation results on csv
csv_file = open(f'Results/train_{model_name}_ws{window_size}_epochs{
    epochs}_results/validation_metrics.txt', 'w')
csv_writer = csv.writer(csv_file, delimiter = '\n')
csv_writer.writerow(['Train Score: %.2f RMSE' % (trainScore), 'Test
    Score: %.2f RMSE' % (testScore)])
csv_writer.writerow(['Train Score: %.2f MAE' % (train_mae), 'Test Score:
     %.2f MAE' % (test_mae)])
csv_file.close()
```

En resumen, la validación es el proceso mediante el cual se evalúa el funcionamiento del modelo. Esta tarea es muy importante, ya que la validación y el análisis del entrenamiento se utilizarán para decidir qué cambios se deben hacer sobre el modelo para mejorarlo. Usualmente, los modelos tienen muchos hiperparámetros (arquitectura del modelo, número de épocas, optimizador,...). Es por ello que un correcto enfoque sobre el ajuste de los hiperparámetros puede servir para ahorrar mucho tiempo de entrenamiento de los modelos.

El flujo de trabajo en este tipo de proyectos muchas veces se basa en entrenar, validar, ajustar

hiperparámetros de los modelos y volver a realizar iteraciones hasta conseguir un modelo con un funcionamiento óptimo.

6.5 Test

La fase de test es igual que la fase de validación pero sin la finalidad de modificar el modelo para volver a entrenarlo y mejorarlo. Consiste en realizar predicciones sobre datos de una distribución separada de los datos de entrenamiento para evaluar finalmente el funcionamiento del modelo. Algunas veces, se usan los mismos datos de validación como set de datos de test.

Las pruebas que se realizan son una predicción sobre los datos de entrenamiento, sobre los datos de test y una predicción a más de un valor en el futuro.

6.5.1 Predicción sobre entrenamiento y sobre test

El modelo utiliza una secuencia de entrada del tamaño de la ventana temporal definida y obtiene como salida el valor del siguiente instante de tiempo a la secuencia de entrada. Se realizan dichas predicciones sobre el set de entrenamiento y sobre el set de test. Posteriormente se representan unas gráficas de todas las predicciones y unas gráficas ampliadas para poder observar las predicciones con más detalle. Para ello, se usa el Código 6.25.

Código 6.25 Representación de las predicciones sobre el set de entrenamiento y de test.

```
# Plot train and test predictions
plot_predictions(train_predict,
                 test_predict,
                 window_size,
                 times,
                 series,
                 x_train)

# Plot train and test predictions separately
plt.figure()
plt.plot(x_train[window_size:])
plt.plot(train_predict[:,0])
plt.title('Predictions on train')
plt.savefig(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/pred_train.png')

plt.figure()
plt.plot(x_test[window_size:])
plt.plot(test_predict[:,0])
plt.title('Predictions on test')
plt.savefig(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/pred_test.png')


# Plot train and test with zoom
plot_gtruth_and_predictions(x_train, train_predict, window_size, start =
     0, end = 100)
plt.title('Predictions on train (zoomed sample)')
```

```
plt.savefig(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/pred_train_zoom.png')

plot_gtruth_and_predictions(x_test, test_predict, window_size, start =
    0, end = 100)
plt.title('Predictions on test (zoomed sample)')
plt.savefig(f'Results/train_{model_name}_ws{window_size}_epochs{epochs}
    _results/pred_test_zoom.png')
```

6.5.2 Predicción a futuro

Los modelos que se han explicado anteriormente son capaces de predecir como salida el valor en el siguiente instante de tiempo utilizando una secuencia de entrada. Sin embargo, para el problema de optimización de la gestión de la estación, se necesitan más de un valor en el futuro, es decir, no un único valor de salida sino una secuencia de valores futuros. Existen varias maneras de realizar esto.

En este caso, se implementa mediante un bucle una forma de utilizar las propias predicciones para generar nuevas predicciones, de manera que sea posible generar más de un valor de salida, conociendo la misma secuencia de entrada.

Para ello se establece una ventana temporal que, inicialmente, incluirá los valores de la secuencia de entrada. Con estos valores, se genera la primera predicción. Ahora, se desliza la ventana temporal un instante de tiempo hacia delante, es decir, utilizando ahora la secuencia de entrada sin el primer valor y, añadiendo la predicción generada en el paso anterior, se monta una nueva secuencia de entrada y se vuelve a generar una nueva predicción. El número de veces que repitamos estos pasos será el número de valores que se predirán del futuro. La implementación de esto se realiza mediante el Código 6.26.

Código 6.26 Implementación de bucle para predecir más de un valor en el futuro.

```
prediction=[]
current_batch = x_test_scaled[start : start + window_size]
current_batch = current_batch.reshape(1, window_size, num_features) #
    Reshape

# Predict future
for i in range(future):
    current_pred = model.predict(current_batch)[0]
    prediction.append(current_pred)
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],
        axis=1)

# Inverse transform to before scaling so we get actual values
rescaled_prediction = scaler.inverse_transform(prediction)

# Plot results
plt.figure(figsize=(12,6), dpi=100)
plt.title('24h Horizon future predictions')
plt.plot(x_test[start + window_size: start + window_size + future])
plt.plot(rescaled_prediction)
plt.legend(['Ground truth', 'Predictions'])
plt.show()
```

Este método de predicción a futuro se implementa en el notebook de jupyter *test_demo.ipynb*, de manera que se muestra de manera visual cómo se pueden realizar inferencias.

Se realiza una función que usando la implementación anterior, vaya barriendo los datos y vaya generando predicciones. Es decir, se realizarán inferencias más de una vez utilizando el método anterior y se apilarán los resultados para luego representarlos mediante una gráfica. Esto se realiza mediante la función *test_forecast* con el Código 6.27.

Código 6.27 Implementación de bucle para predecir más de un valor en el futuro.

```
 # Run forecasting inferences
def test_forecast(x_test, x_test_scaled, model, scaler, window_size,
    num_features, future, num_inferences, start_all, verbose=0 ):
    start = start_all
    for i in range(num_inferences):
        prediction=[]
        current_batch = x_test_scaled[start : start + window_size]
        current_batch = current_batch.reshape(1, window_size,
            num_features)

        # Predict future
        for j in range(future):
            current_pred = model.predict(current_batch)[0]
            prediction.append(current_pred)
            current_batch = np.append(current_batch[:,1:,:],[[
                current_pred]],axis=1)

        # Inverse transform to before scaling so we get actual values
        rescaled_prediction = scaler.inverse_transform(prediction)

        if verbose == 1:
            plt.figure()
            plt.plot(x_test[start + window_size: start + window_size +
                future])
            plt.plot(rescaled_prediction)
            plt.show()

        if i == 0:
            all_predictions = rescaled_prediction
        else:
            all_predictions = np.concatenate((all_predictions,
                rescaled_prediction))


        start = start + future


    plt.figure()
    plt.plot(x_test[start_all + window_size: start_all + window_size +
        future*num_inferences])
    plt.plot(all_predictions)
```

```
    plt.show()
```

En el notebook *test_demo.ipynb* también se añade una prueba de concepto que hace referencia a una prueba en tiempo real. Esto consiste en realizar un bucle que realiza inferencias y las va representado, moviendo la ventana temporal y superponiendo la gráfica anterior, de manera que se ve cómo van variando las predicciones con el tiempo en tiempo real. Esto se realiza utilizando la función anterior y el Código 6.28.

Código 6.28 Prueba de concepto: simulación en tiempo real.

```
import cv2
from IPython.display import clear_output

start_all=0
num = 10

idx=0
while idx < num:
    num_inferences = 2
    clear_output(wait=True)
    test_forecast(x_test, x_test_scaled, model, scaler, window_size,
        num_features, future, num_inferences, start_all)
    cv2.waitKey(1)
    start_all += 1
    idx+=1
```

Por último, también se añade en el mismo notebook un método para comparar las predicciones de los distintos modelos entrenados de manera visual. Consiste en realizar un bucle que vaya recorriendo los distintos modelos, vaya realizando predicciones y las vaya representando, de manera que quede al final una gráfica con las distintas predicciones de los distintos modelos superpuestas. El código es el siguiente:

Código 6.29 Comparación de modelos.

```
model_names = ['Simple_ANN',
               'Simple_RNN',
               'Simpler_RNN',
               'LSTM',
               'LSTM_stacked',
               'Bidirectional_LSTM',
               'CNN_LSTM'
              ]

plt.figure(figsize=(16, 8), dpi=100)
for model_name in model_names:
    model = tf.keras.models.load_model(f"Results/train_{model_name}_ws{
        window_size}_epochs30_results/{model_name}_ws{window_size}
        _epochs30_model.h5")

    prediction=[]
    current_batch = x_test_scaled[start : start + window_size]
```

```
    current_batch = current_batch.reshape(1, window_size, num_features)
        #Reshape

    # Predict future
    for i in range(future):
        current_pred = model.predict(current_batch)[0]
        prediction.append(current_pred)
        current_batch = np.append(current_batch[:,1:,:],[[current_pred]],
            axis=1)

    # Inverse transform to before scaling so we get actual values
    rescaled_prediction = scaler.inverse_transform(prediction)

    plt.plot(rescaled_prediction)

plt.plot(x_test[start + window_size: start + window_size + future],'--'
    )
model_names.append('Ground truth')
plt.legend(model_names)
plt.title('Model comparison')
plt.show()
```

6.6 Resultados

Se entrenan los modelos para distintas longitudes de secuencias de entrada. Se compararán los resultados obtenidos para unas ventanas temporales de tamaño 96 y 168, es decir, ventanas temporales de 4 y 7 días respectivamente, recordando que los datos que se usan son horarios.

Por resumir, se muestran sólo los resultados para una secuencia de entrada de 96 valores, pero posteriormente se realizará una comparativa de los distintos modelos con los dos distintos tamaños de ventanas temporales.

6.6.1 Entrenamiento

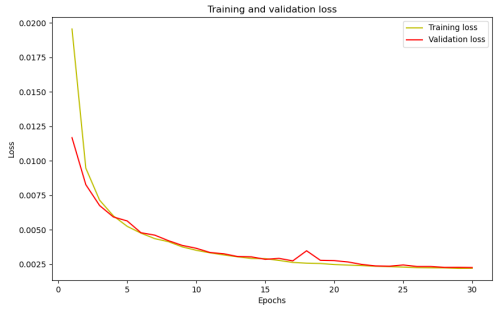
Se representan los historiales de los entrenamientos con los el valor del coste de entrenamiento y de validación frente al número de épocas. Los entrenamientos que se representan en la Figura 6.11 han sido realizados con 30 épocas y con un tamaño de secuencia de entrada de 96 (4 días).

Se observa que en todos ellos, el coste de entrenamiento ha ido disminuyendo de manera progresiva sin muchas oscilaciones, esto afirma que el optimizador elegido ha sido adecuado. Por otro lado, el coste de validación, aunque en algunos entrenamientos ha sido más inestable, se observa que ha ido disminuyendo conforme lo ha ido haciendo el coste de entrenamiento, esto significa que no está habiendo *overfitting* a los datos de entrenamiento y que, por tanto, el número de épocas elegido ha sido acertado.

6.6.2 Validación

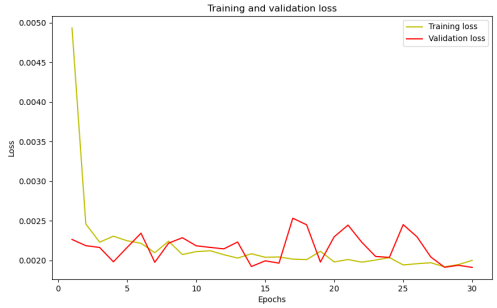
Como se comenta anteriormente, se realiza una comparación cuantitativa del funcionamiento de los distintos modelos con dos ventanas temporales distintas en la Tabla 6.2.

El modelo que ha obtenido los peores resultados ha sido el modelo clásico del perceptrón multicapa (ANN) con una ventana temporal de 168 valores. Esto demuestra la dificultad que tienen este tipo de redes para captar las dependencias temporales en largas secuencias.

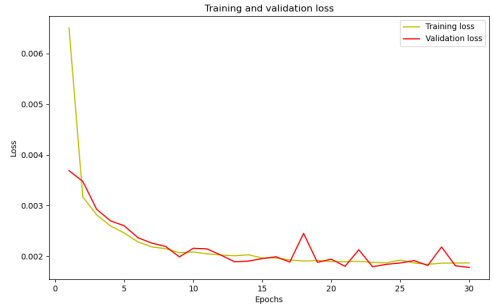


(a) Historial entrenamiento modelo ANN.

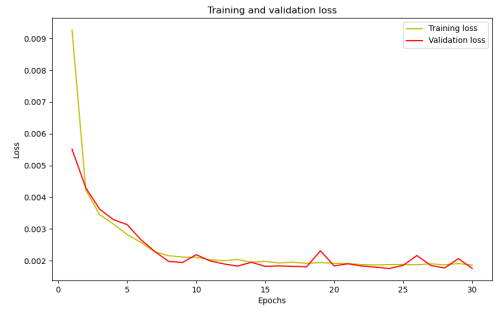
(b) Historial entrenamiento modelo RNN.



(c) Historial entrenamiento modelo st-RNN.

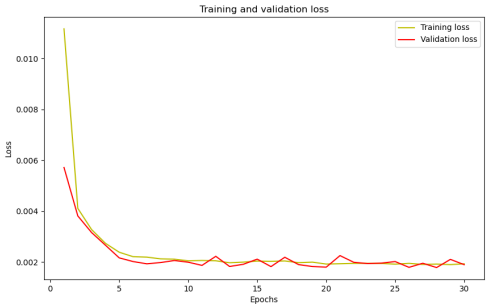


(d) Historial entrenamiento modelo LSTM.



(e) Historial entrenamiento modelo st-LSTM.

(f) Historial entrenamiento modelo Bi-LSTM.



(g) Historial entrenamiento modelo CNN+LSTM.

Figura 6.11 Historiales de los entrenamientos con los distintos modelos.

Tabla 6.2 Comparación de los modelos.

w_size	Métricas	ANN	RNN	st- RNN	LSTM	st- LSTM	Bi-LSTM	CNN + LSTM
96	RMSE train (kW)	4.46	4.20	4.27	4.09	4.03	4.15	4.17
	RMSE test (kW)	4.62	4.30	4.25	4.10	4.07	4.23	4.23
	MAE train (kW)	3.10	2.94	2.99	2.69	2.69	2.83	2.84
	MAE test (kW)	3.21	3.09	2.93	2.66	2.70	2.86	2.88
168	RMSE train (kW)	4.51	4.22	4.21	4.05	4.07	3.93	4.08
	RMSE test (kW)	4.72	4.39	4.35	4.12	4.13	3.96	4.08
	MAE train (kW)	3.40	2.85	2.95	2.72	2.70	2.72	2.76
	MAE test (kW)	3.59	2.86	3.06	2.73	2.68	2.68	2.75

Por lo contrario, la capacidad de obtener patrones temporales y de memorizar información a largo plazo de las redes LSTM han hecho que éste modelo sea el que ha obtenido los mejores resultados con una ventana temporal de 96 valores.

Respecto al tamaño la ventana temporal se puede afirmar que para modelos sencillos como las redes recurrentes simples y las LSTM un tamaño de secuencia de entrada menor ha resultado en un mejor funcionamiento. En cambio, para modelos más complejos como las redes bidireccionales o el modelo de convolución con LSTM es preferible el uso de ventanas temporales de mayor tamaño.

6.6.3 Test

En primer lugar se muestran las predicciones sobre todo el conjunto de datos de entrenamiento y los de validación con un horizonte temporal de 1 valor en el futuro. Los resultados se muestran en la Figura 6.11. Se ha mostrado sólo una franja de los datos para mejorar la visibilidad y mostrar más en detalle las predicciones. Las curvas verdes representan las predicciones de los modelos mientras que las azules representan los valores reales de la demanda.

Los resultados obtenidos han sido los esperados y muestran un buen funcionamiento cuando el horizonte temporal es de 1 valor.

El problema de predicción se complica cuando no es sólo 1 valor el que hay que predecir sino varios. En siguiente lugar, se realizan predicciones individuales a futuro con una secuencia de 24 valores de salida (1 día) cuyos resultados se muestran en la Figura 6.12. Para obtener estas gráficas se ha utilizado el modelo LSTM con una ventan temporal de 96 valores.

Se muestra que ahora los valores predichos difieren más de los valores reales que cuando el horizonte temporal era de 1 valor. Sin embargo, teniendo lo ambicioso que es el problema que se quiere resolver, los resultados son satisfactorios ya que, aunque no sean capaces de predecir las pequeñas fluctuaciones, son capaces de anticipar la tendencia de la serie temporal, que es lo que se necesita para el problema de optimización de la estación de vehículos eléctricos.

Siguiendo por la misma línea, se representan en la Figura 6.13 predicciones conjuntas con un horizonte temporal de 24 horas, para ver cómo sería la predicción recursiva de más de un día en adelante.

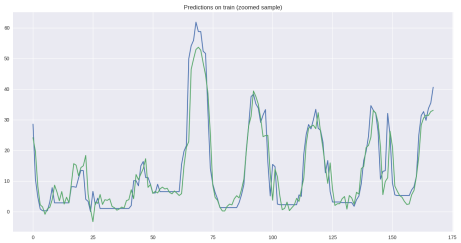
Finalmente, se representan los resultados de la comparativa de predicciones para varios modelos en la Figura 6.14.

6.7 Conclusiones

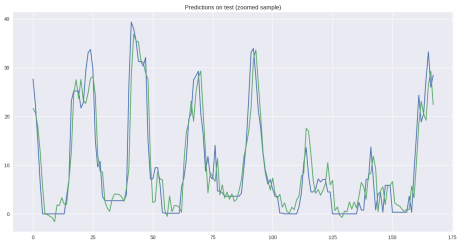
Se ha podido comprobar que para el uso de modelos basados en deep learning no es del todo necesario datos con muchas características para un obtener buenos resultados. Estos modelos son capaces de captar de manera efectiva la característica no lineal de los datos y las correlaciones temporales para predecir la potencial demanda de carga.

En este capítulo, se ha hecho uso de datos públicos para entrenar a los modelos tras un previo procesado de los mismos. Se ha realizado un estudio comparativo de distintos modelos, incluyendo los modelos ANN, RNN, LSTM, Bi-LSTM y CNN+LSTM implementándolos en dos escenarios distintos con distinto tamaño de ventana temporal, después de haber sido entrenados mediante un algoritmo de aprendizaje supervisado. Se han buscado los hiperparámetros óptimos de los modelos para conseguir el mejor funcionamiento. Métricas como MAE y RMSE han sido utilizadas para evaluar el funcionamiento de los modelos. Los resultados muestran que los modelos de deep learning se han ajustado adecuadamente y son efectivos a la hora de predecir la demanda de carga en el futuro de una manera precisa para la optimización de un sistema de gestión de potencia dinámica. Dentro de los modelos de deep learning utilizados, el modelo LSTM se ha mostrado superior al resto de modelos y se demuestra que cumple correctamente con la tarea de predecir la demanda en un plazo de un día.

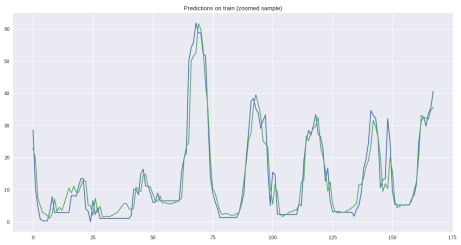
En trabajo futuro, se propone utilizar otras bases de datos distintas, incorporar otros métodos de predicción como el uso de series multivariables que incorporen más información, como el día de la semana, tiempo meteorológico, etc. También se propone utilizar modelos espacio-temporales, que tengan en cuenta la localización de la estación y el tráfico.



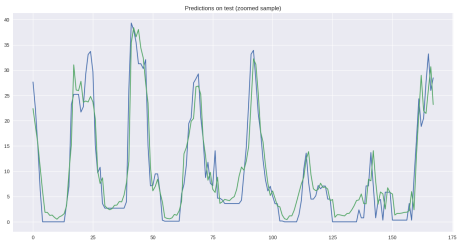
(a) Predicciones entrenamiento modelo ANN .



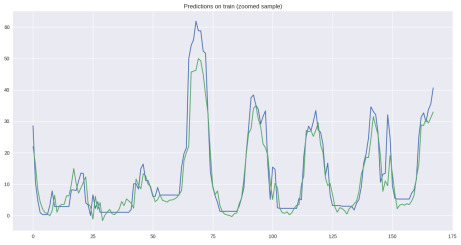
(b) Predicciones test modelo ANN.



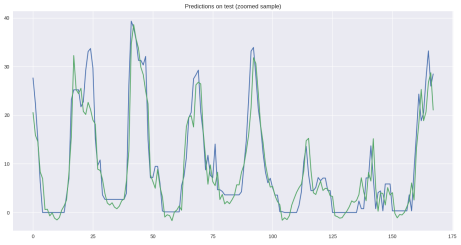
(c) Predicciones entrenamiento modelo RNN .



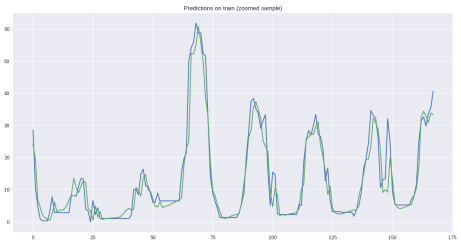
(d) Predicciones test modelo RNN.



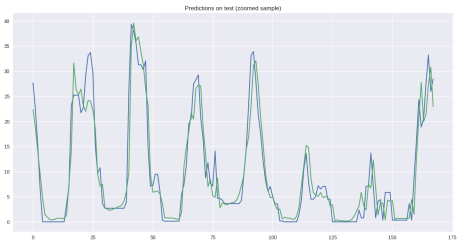
(e) Predicciones entrenamiento modelo RNN-st .



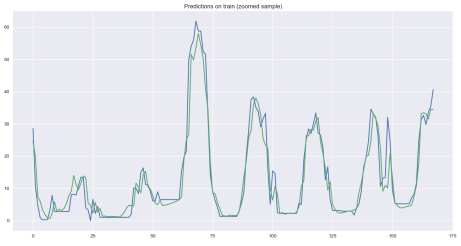
(f) Predicciones test modelo RNN.



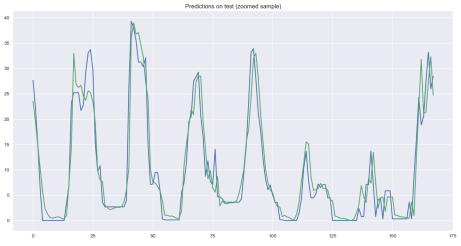
(g) Predicciones entrenamiento modelo LSTM .



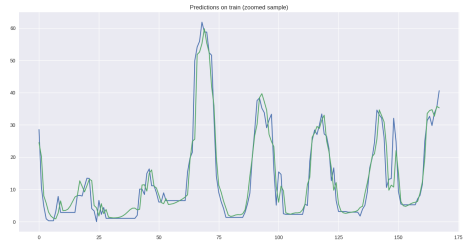
(h) Predicciones test modelo RNN.



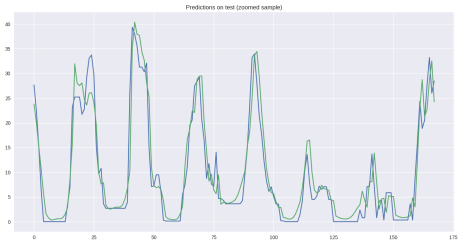
(i) Predicciones entrenamiento modelo LSTM-st .



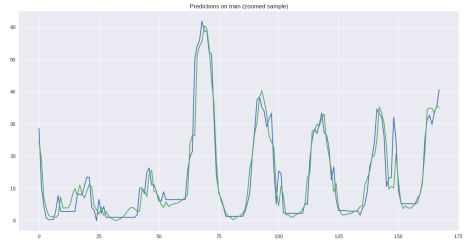
(j) Predicciones test modelo LSTM-st.



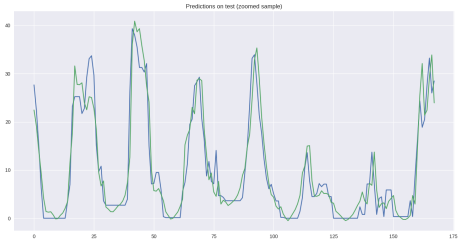
(k) Predicciones entrenamiento modelo Bi-LSTM .



(l) Predicciones test modelo CNN.



(m) Predicciones entrenamiento modelo CNN .



(n) Predicciones test modelo CNN.

Figura 6.11 Resultados predicciones (zoom). Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

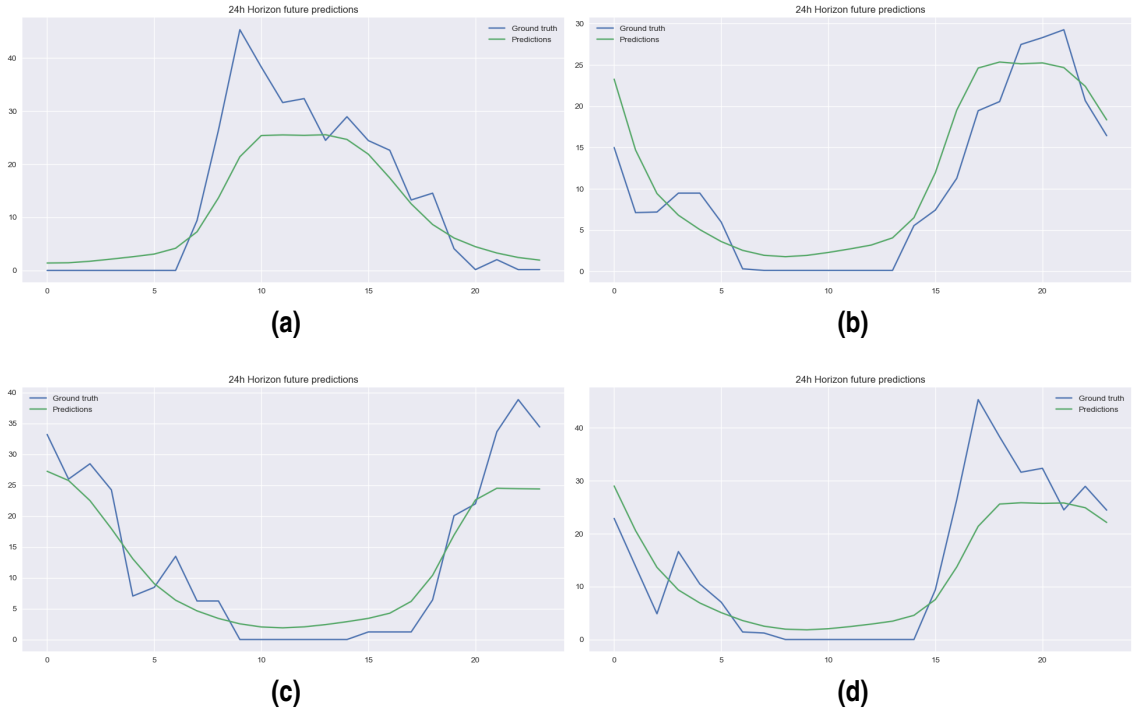


Figura 6.12 Resultados predicciones con horizonte de 24h. Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

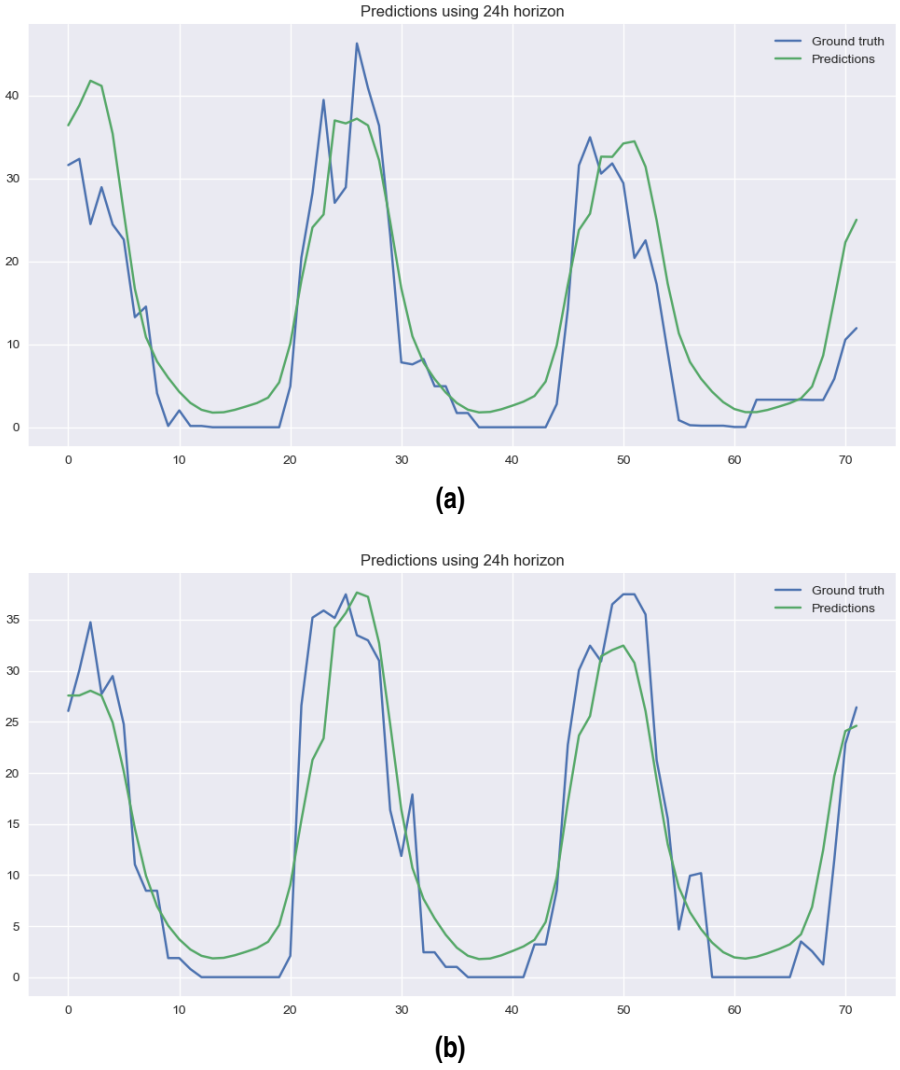
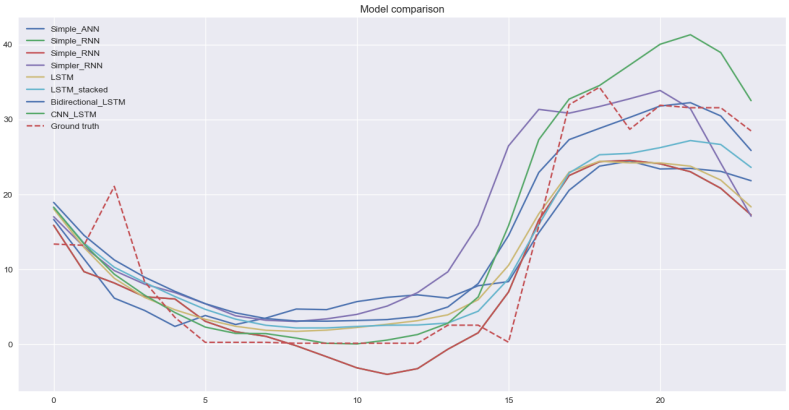
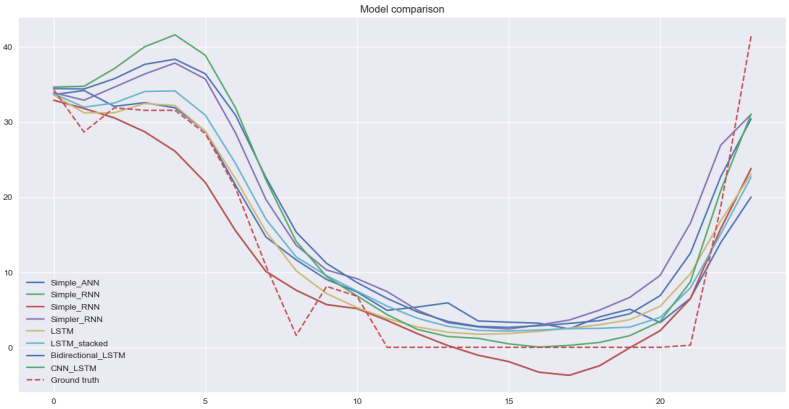


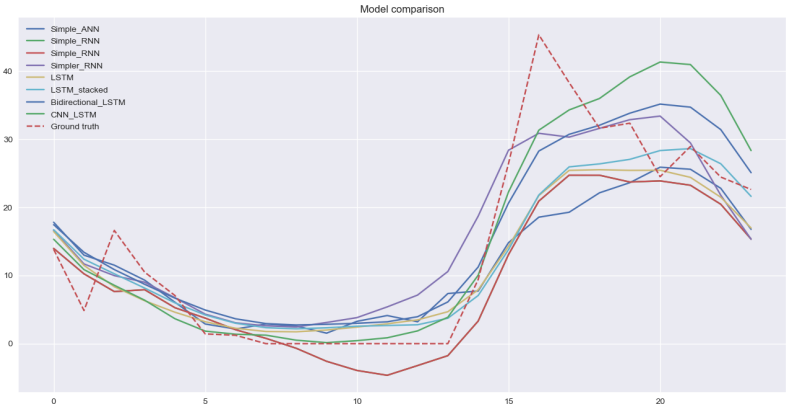
Figura 6.13 Resultados de varias predicciones consecutivas con horizonte de 24h. . Eje X: tiempo (h) - Eje Y: potencia demandada (kW).



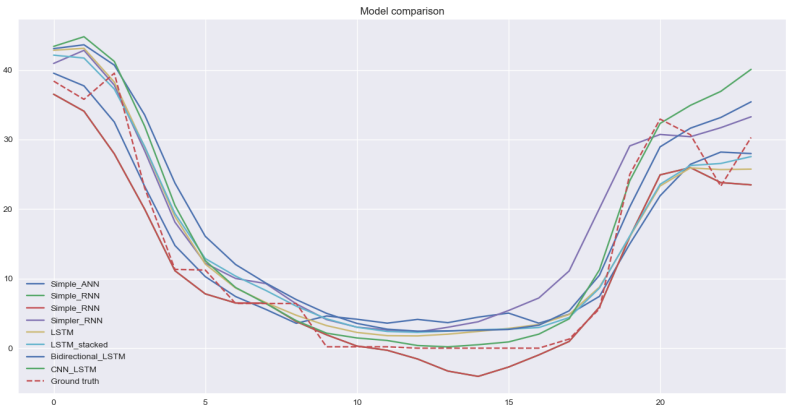
(a)



(b)



(c)



(d)

Figura 6.14 Resultados comparación de predicciones con horizonte de 24h. . Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

7 Predicción de la generación del sistema fotovoltaico en la estación de carga de EV

De entre las energías renovables, la tecnología fotovoltaica es una de las formas más importantes de convertir radiación solar en electricidad. Sin embargo, la cantidad de electricidad producida se ve gravemente influida por las condiciones meteorológicas (nubes, temperatura, etc.).

Debido al gran crecimiento que las plantas fotovoltaicas están experimentando en la actualidad, es de crucial importancia aprender a gestionar su alterabilidad inherente. Por tanto, la predicción de la energía solar fotovoltaica es una herramienta fundamental que ayuda a que el funcionamiento y control de la red sea fiable y económicamente rentable. De nuevo, se quiere proponer una solución a dicho problema de predicción mediante el uso de modelos de deep learning con redes neuronales.

En este caso, a diferencia de lo desarrollado en el capítulo anterior, se utiliza una entrada multidimensional. Esto significa que se entrenarán a los modelos con secuencias multivariables que contendrán información, además de los valores de producción pasados, de otras variables relacionadas con las condiciones meteorológicas.

El algoritmo de optimización para la planificación dinámica de la estación necesita, además de la predicción a futuro de la demanda de carga, la predicción de la producción solar. Para ello se proponen distintos modelos con distintas arquitecturas de redes neuronales, que permiten la predicción a largo plazo en un horizonte temporal de 24h. Al utilizar un horizonte temporal de 24 horas (24 valores), se puede clasificar como un problema de predicción a medio-largo plazo.

En este capítulo se realiza, en primer lugar, el procesamiento de los datos para el entrenamiento. Posteriormente se escogen los modelos que serán entrenados y validados. Finalmente se realizan pruebas con los modelos entrenados y se analizan los resultados.

7.1 Datos

Se utilizan datos históricos reales de una planta fotovoltaica experimental ubicada en la Escuela Técnica Superior de Ingeniería en Sevilla para entrenar a los modelos de predicción. Una vez los modelos son entrenados, se evalúa su funcionamiento de manera cualitativa (métodos gráficos) y cuantitativa (calculando métricas de validación).

Los datos se cargan utilizando Matlab. Para los días en los que no se han podido recoger datos debido a fallos en los sensores o mantenimientos, se ha copiado el día anterior para no dejar huecos grandes en la serie temporal. Tampoco se han eliminado directamente para no perder la temporalidad de los datos.

En este apartado se aborda el procesamiento de los datos y la transformación de los mismos para tener la estructura necesaria y así puedan ser alimentados a los modelos para ser entrenados.

7.1.1 Visualización y análisis inicial

Los datos originales tomados de la planta fotovoltaica tienen un tiempo de muestreo de 5 segundos. Tras el procesamiento inicial de Matlab el tiempo de muestreo pasa a ser de 5 minutos.

Posteriormente los datos se cargan en un script de Python. En la Figura 7.1 se enseñan unas primeras filas de datos de modo que se pueden observar las distintas columnas y el tiempo que hay entre cada muestra.

	Irradiancia solar. NIP pyrheliometer (W/m2)	Temperatura (°C)	Presión barométrica (MBar)	Humedad Relativa (%)	Irradiancia Teórica (W/m2)	Desviación Radiación directa (W/m2)
2014-01-01 00:00:00	0.0	14.200000	1021.216949	95.000000	0.0	0.0
2014-01-01 00:05:00	0.0	14.200000	1021.205085	95.440678	0.0	0.0
2014-01-01 00:10:00	0.0	14.200000	1021.200000	95.694915	0.0	0.0
2014-01-01 00:15:00	0.0	14.200000	1021.142373	95.915254	0.0	0.0
2014-01-01 00:20:00	0.0	14.154237	1021.000000	96.000000	0.0	0.0

Figura 7.1 Cabecera de los datos y primeras muestras.

Posteriormente, se representan los datos de las distintas columnas por separado, según se muestran en la Figura 7.2.

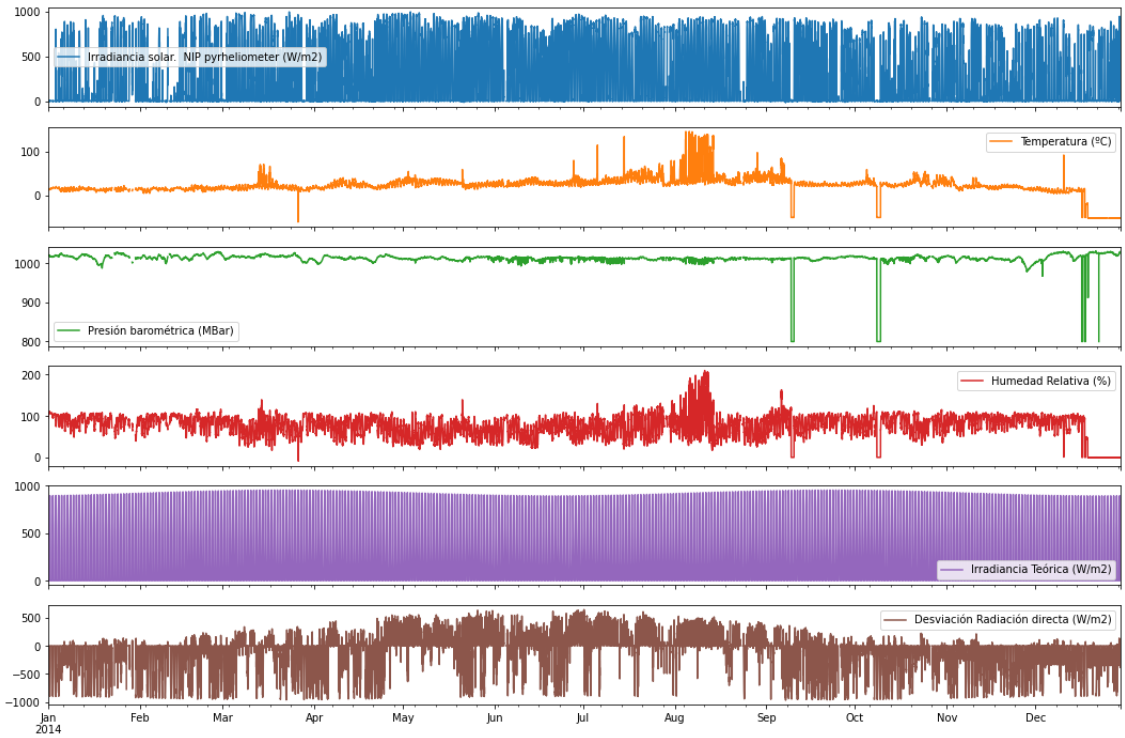


Figura 7.2 Representación de los datos sin procesar.

Para realizar un análisis previo de los datos, se representa un diagrama de cajas que permitirá mostrar de manera visual características de los datos, tales como la dispersión y simetría. Para su

realización se representan los tres cuartiles, así como los valores mínimo y máximo de los datos sin tener en cuenta valores puntuales que se separen mucho de la distribución (valores atípicos o *outliers*). En la Figura 7.3 se muestra un diagrama de cajas de los datos sin procesar.

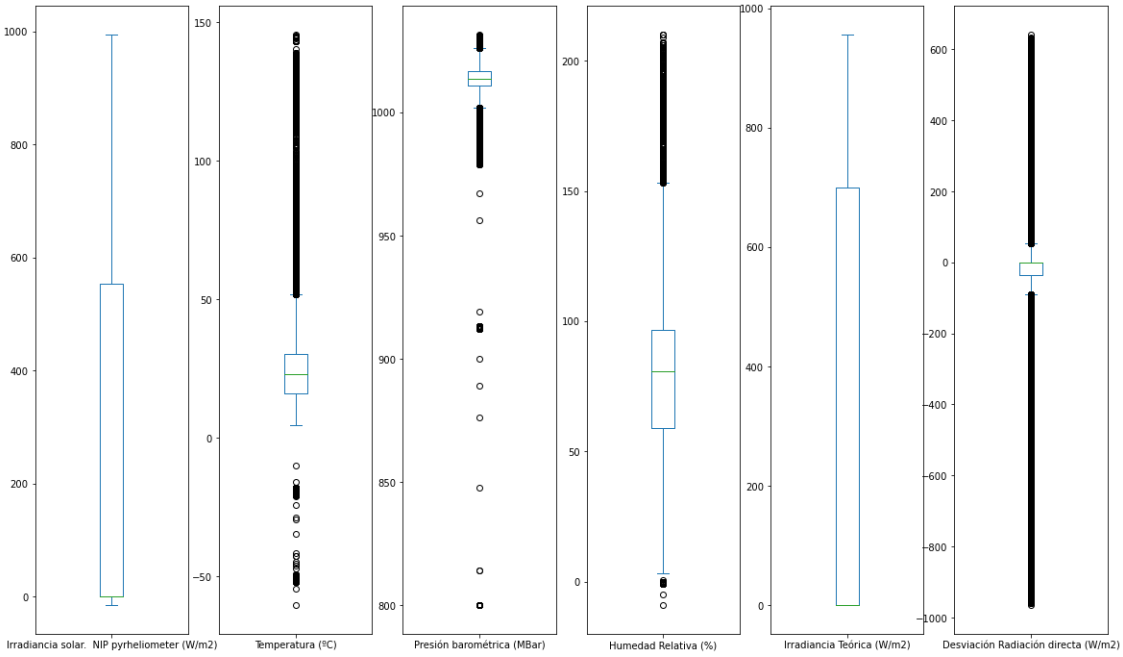


Figura 7.3 Diagrama de cajas de los datos sin procesar.

Se puede observar cómo las variables relativas a los datos meteorológicos tienen muchos valores atípicos. Esto puede ser debido a algunos fallos en instantes determinados de los sensores. Por ejemplo, en el caso de la temperatura, no es posible que haya alcanzado los extremos de 150ºC ó -50ºC. Para eliminar este ruido, se propondrá un filtrado de los datos en el siguiente apartado.

Para finalizar el análisis, se calculan las correlaciones de la salida que nos interesa predecir con las distintas variables disponibles. Esto sirve para ver qué tipo de la relación lineal hay entre las variables y la salida, es decir, ver si existe entre alguna relación proporcional positiva o negativa de la salida con las variables. En la Tabla 7.1 se recogen las variables y sus correlaciones con la irradiancia solar.

Tabla 7.1 Tabla de correlaciones.

Variables	Irradiancia solar. NIP pyrheliometer (W/m2)
Irradiancia solar. NIP pyrheliometer (W/m2)	1
Temperatura (ºC)	0.17
Presión barométrica (MBar)	0.07
Humedad Relativa (%)	-0.42
Irradiancia Teórica (W/m2)	0.71
Desviación Radiación directa (W/m2)	0.3

Se puede observar que la irradiancia solar está fuertemente correlacionada con la humedad relativa de una manera inversa. Esto puede explicarse atendiendo a la relación que tiene la humedad relativa con la nubosidad, es decir, considerando la proporción inversa que existe entre la cantidad de nubes que hay en el cielo en un cierto instante y la irradiancia solar que llega a la planta fotovoltaica.

7.1.2 Procesamiento

Inicialmente se realiza un filtrado de los valores atípicos para cada una de las variables. En este caso, se aplicará sólo a las variables de temperatura, presión y humedad. El filtrado consiste en eliminar los datos que estén por encima del cuantil 85 y los datos que estén por debajo del cuantil 15. La Figura 7.4 ofrece un diagrama de cajas de los datos una vez que han sido filtrados.

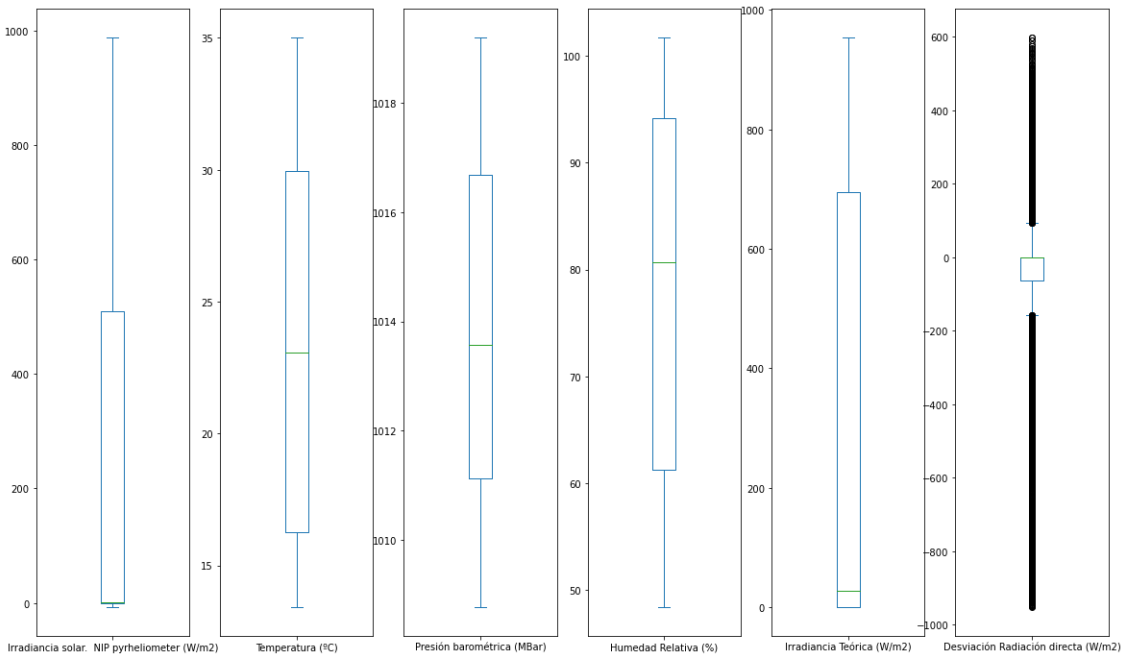


Figura 7.4 Diagrama de cajas de los datos tras el procesado.

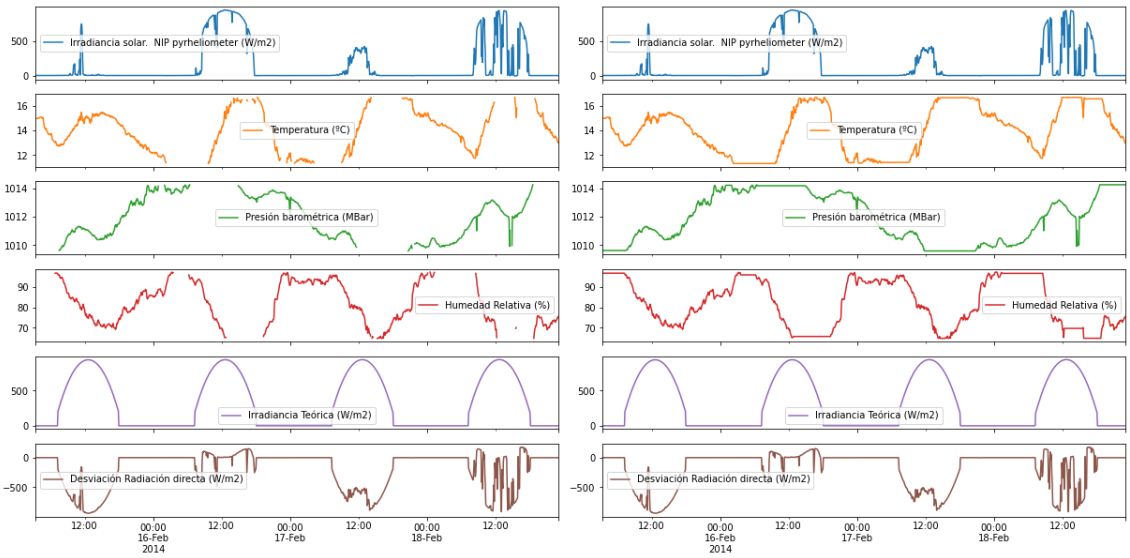
Por otro lado, además de los datos filtrado, aparecen algunos datos vacíos que pueden relacionarse con fallos en los sensores o mantenimientos. Para solucionar este problema, se reconstruyen los datos, copiando en los datos vacíos o nulos los valores vecinos previos inmediatamente disponibles. De esta manera se puede dar una solución a este problema de datos ausentes. Se muestra el resultado de la operación en la Figura 7.5.

La última operación de procesamiento que se realiza es el "remuestreo" de los datos. Para ello, se toma el valor medio de los datos consecutivos en un intervalo de tiempo, que será el nuevo tiempo de muestreo. En este caso, el nuevo tiempo de muestreo será de una hora. Se muestra una comparación del antes y después de esta operación en la Figura 7.6.

Finalmente, se calcula de nuevo la matriz de correlaciones entre las variables, obteniéndose los nuevos resultados para los datos procesados según se recoge en la Tabla 7.2.

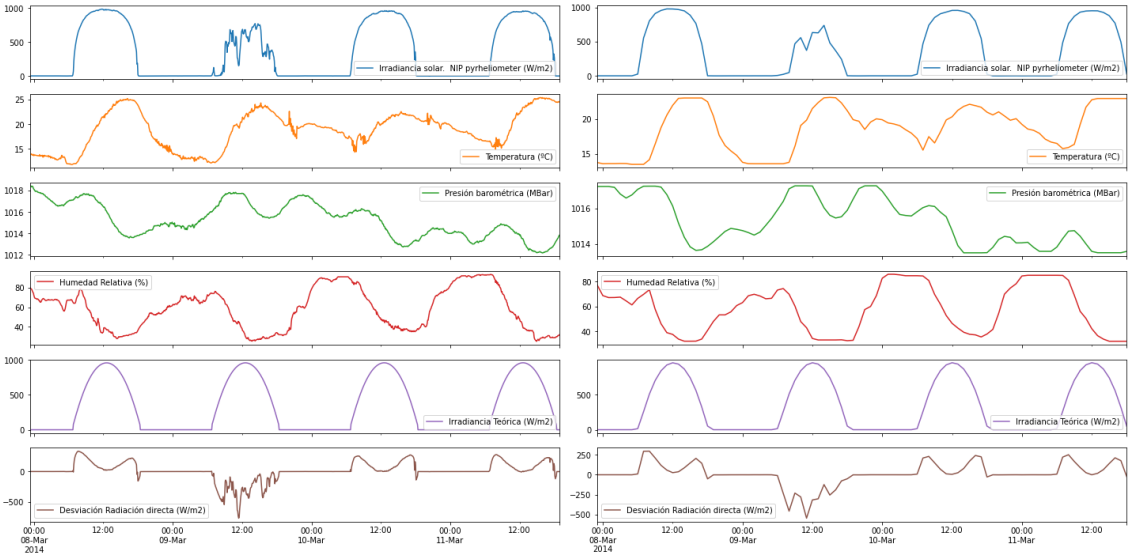
Tabla 7.2 Tabla de correlaciones después del procesado de los datos.

Variables	Irradiancia solar. NIP pyrheliometer (W/m2)
Irradiancia solar. NIP pyrheliometer (W/m2)	1
Temperatura (ºC)	0.32
Presión barométrica (MBar)	-0.01
Humedad Relativa (%)	-0.54
Irradiancia Teórica (W/m2)	0.72
Desviación Radiación directa (W/m2)	0.25



(a) Datos con tramos no disponibles. **(b)** Datos reconstruidos.

Figura 7.5 Comparación de los datos antes y después de ser reconstruidos.



(a) Datos muestreados cada 5 minutos. **(b)** Datos muestreados cada hora.

Figura 7.6 Comparación de los datos antes y después de ser remuestreados.

Se aprecia que después de la limpieza y "remuestreo" de los datos, la correlación de la irradiancia solar con la temperatura ha aumentado a casi el doble. La correlación con la humedad relativa también aumenta considerablemente. Aunque la correlación con la presión barométrica es muy baja, no se puede menospreciar la información de esta variable, pues los modelos que se proponen a continuación son capaces de encontrar relaciones no lineales entre las variables no consideradas en la correlación.

7.2 Modelos

Los modelos que se plantean para la solución al problema de predicción de la generación fotovoltaica son los mismos que se propusieron en el capítulo anterior, pero cambiando las dimensiones de la secuencia de entrada. Así, mientras que en el capítulo anterior se consideraban secuencias univariables, en este son multivariables.

Las entradas (x) vendrán conformadas por secuencias que contendrán los datos de las siguientes variables:

- Irradiancia solar. NIP pyrheliometer (W/m2)
- Temperatura (ºC)
- Humedad Relativa (%)
- Presión barométrica (MBar)
- Irradiancia Teórica ($W/m2$)

Y las salidas o etiquetas (y) a predecir serán los datos de la variable: "Irradiancia solar. NIP pyrheliometer (W/m2)"

Los hiperparámetros principales que hay que ajustar son la longitud de la secuencia multidimensional de entrada y la complejidad de los modelos en sí. Del mismo modo que en el capítulo anterior, se proponen distintas arquitecturas para ser entrenadas y validadas usando tamaños de ventana temporal diferentes. Posteriormente se realizará una comparación entre los distintos modelos con distintos tamaños de ventana temporal para encontrar la solución que genere las mejores predicciones.

7.3 Entrenamiento

Al igual que en el capítulo anterior, se desarrolla un script *train.py* en el que se realizarán de manera automática los entrenamientos según unos argumentos de entrada y guardará las salidas de dichos entrenamientos. Estas salidas son, entre otras, los resultados de las métricas de validación y los modelos exportados.

En el entrenamiento se siguen los mismos pasos que en el entrenamiento del capítulo anterior. Primero, se cargan y se normalizan de los datos. Luego, éstos se dividen en dos conjuntos de datos, uno para el entrenamiento y otro para la validación. Posterioremente, se construye el modelo y se compila para finalmente realizar el entrenamiento.

Los pasos de normalizar los datos, separar los datos del conjunto de entrenamiento y de validación, y crear los generadores varían ligeramente con respecto al capítulo anterior al ser secuencias de datos multidimensionales.

7.3.1 Normalizado de los datos

El normalizado de los datos se realiza variable a variable, es decir, columna a columna. Se utiliza el mismo escalador que en el capítulo anterior, es decir, el escalador mínimo-máximo. Este escalador considera el valor mínimo y máximo respectivo de una columna y reescala los datos de dicha columna de tal forma que el valor mínimo pase a valer 0.1 y el valor máximo pase a valer 1, interpolando el resto de valores intermedios.

El reescalado inverso consiste en devuelver los datos a su escala original una vez hayan sido procesados por los modelos. El objeto "escalador" necesita una entrada de la misma dimensión, por tanto, se copian las mismas predicciones en distintas columnas. De este modo se reescala inversamente la matriz generada, de la cual tan sólo interesarán los valores de la primera columna. Dichos valores de la primera columna corresponden a los valores de la irradiación.

En el Código 7.1 y Código 7.2 se presenta la normalización y el reescalado inverso de los datos.

Código 7.1 Normalización de los datos.

```
# Normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
df_scaled = scaler.fit_transform(df)
```

Código 7.2 Escalado inverso de los datos.

```
# Inverse transform to before scaling so we get actual values
predict_copies = np.repeat(prediction, df_scaled.shape[1], axis=-1)
rescaled_prediction = scaler.inverse_transform(predict_copies)[:,0]
```

7.3.2 División de los datos para entrenamiento y validación

En este caso es necesario separar los datos para las salida de las variables de entrada, que a diferencia del capítulo anterior, son diferentes. En el Código 7.3 se realiza la separación de los datos de donde se obtendrán las secuencias de entradas multidimensionales (x) de los datos de donde se obtendrán salidas o etiquetas (y).

Código 7.3 Separar las columnas de entradas y salida.

```
# Get the output and input columns
y_data = df_scaled[:,0]
x_data = df_scaled[:,:-1]
```

Una vez se ha realizado esto, en el Código 7.4 se separan los datos en dos conjuntos de distintos, uno para realizar los entrenamientos y otro para realizar las validaciones.

Código 7.4 Separar las columnas de entradas y salida.

```
# Split training and test set
split_time = int(len(df_scaled) * 0.66)

x_train = x_data[:split_time]
y_train = y_data[:split_time]

x_test = x_data[split_time:]
y_test = y_data[split_time:]
```

7.3.3 Creación de generadores de entrenamiento y validación

Los generadores de entrenamiento y validación, al igual que en el capítulo anterior, se definen introduciendo los datos que se utilizarán como entradas y los que se utilizarán como salida.

En el Código 7.6 se crean dos generadores, uno con el conjunto de datos de entrenamiento y otro con el conjunto de datos de validación.

Código 7.5 Separar las columnas de entradas y salida.

```
# Create the data generators for training and validation
num_features = x_data.shape[1]

train_generator = TimeseriesGenerator(x_train,
                                      y_train,
                                      length=window_size,
                                      batch_size=batch_size)
validation_generator = TimeseriesGenerator(x_test,
                                         y_test,
                                         length=window_size,
                                         batch_size=batch_size)
```

7.4 Validación

Una vez se ha realizado el entrenamiento del modelo, se realizan las validaciones para evaluar cualitativamente y cuantitativamente el funcionamiento y así poder reajustar hiperparámetros para volverlo a entrenar.

Las métricas utilizadas son las mismas que en el capítulo anterior, es decir, la raíz del error cuadrático medio (RMSE) y el error absoluto medio (MAE). Estos errores se calcularán en las predicciones sobre los datos de entrenamiento y sobre los datos de validación respecto a los valores reales correspondientes. Se realiza en el Código 7.6

Código 7.6 Separar las columnas de entradas y salida.

```
# Run predictions on training and validation set
train_predict = predict(model,
                        train_generator,
                        scaler,
                        num_features = df_scaled.shape[1])

test_predict = predict(model,
                       validation_generator,
                       scaler,
                       num_features = df_scaled.shape[1])

# Get ground truth values
y_gt = df.iloc[:,0]
# Split train and validation ground truth
train_Y_gt = y_gt[:split_time]
test_Y_gt = y_gt[split_time:]

# Validate the model using RMSE and MAE
trainScore = math.sqrt(mean_squared_error(df_train_predict , train_Y_gt[
    window_size:]))
testScore = math.sqrt(mean_squared_error(df_test_predict , test_Y_gt[
    window_size:]))

print('Train Score: %.2f RMSE' % (trainScore))
```

```
print('Test Score: %.2f RMSE' % (testScore))

train_mae = mean_absolute_error(df_train_predict , train_Y_gt[
    window_size:])
test_mae = mean_absolute_error(df_test_predict , test_Y_gt[window_size
    :])
print('Train Score: %.2f MAE' % (train_mae))
print('Test Score: %.2f MAE' % (test_mae))


# Write validation results on csv
csv_file = open(f'Results/train_{model_name}_ws{window_size}_epochs{
    epochs}_results/validation_metrics.txt', 'w')
csv_writer = csv.writer(csv_file, delimiter = '\n')
csv_writer.writerow(['Train Score: %.2f RMSE' % (trainScore), 'Test
    Score: %.2f RMSE' % (testScore)])
csv_writer.writerow(['Train Score: %.2f MAE' % (train_mae), 'Test Score:
     %.2f MAE' % (test_mae)])
csv_file.close()
```

7.5 Test

Finalmente se realizan predicciones sobre datos de una distribución separada de los datos de entrenamiento para evaluar el funcionamiento de los modelos. Se usan los mismos datos de validación como set de datos de test.

Las pruebas que se realizan son una predicción sobre los datos de entrenamiento, sobre los datos de test y una predicción a más de un valor en el futuro.

7.5.1 Prediccion a futuro

Los modelos que se han explicado anteriormente son capaces de predecir como salida el valor en el siguiente instante de tiempo utilizando una secuencia de entrada multivariable. Sin embargo, para el problema de optimización de la gestión de la estación de carga que estamos estudiando, se necesitan más de un valor en el futuro, es decir, no un único valor de salida sino una secuencia de valores futuros.

Se implementa, del mismo modo que en el capítulo anterior, mediante un bucle una forma de utilizar las propias predicciones para generar nuevas predicciones, de manera que sea posible generar más de un valor de salida conociendo la misma secuencia de entrada. La diferencia es que en este capítulo los datos son datos multivariables y se debería conocer, excepto los datos de la producción a predecir, el resto de datos (temperatura, presión...) en el futuro en ese horizonte temporal de predicción. Por tanto, se supone que el resto de datos son conocidos en el futuro.

La implementación de esto se realiza mediante el Código 7.7.

Código 7.7 Implementación de bucle para predecir más de un valor en el futuro.

```
# Forecasting in {future} horizon
prediction=[]
current_batch = x_test[start : start + window_size]
current_batch = current_batch.reshape(1, window_size, num_features) #
    Reshape
```

```
# Predict future
for i in range(future):
    current_pred = model.predict(current_batch)[0]
    prediction.append(current_pred)
    current_batch = np.append(current_batch[:,1:,:],[[x_test[start +
        window_size + i]]],axis=1)
    #current_batch = np.append(current_batch[:,1:,:],[[current_pred]],
        axis=1)
    current_batch[:,-1,0] = current_pred

# Inverse transform to before scaling so we get actual values
predict_copies = np.repeat(prediction, df_scaled.shape[1], axis=-1)
rescaled_prediction = scaler.inverse_transform(predict_copies)[:,0]

# Plot results
plt.figure(figsize=(12,6), dpi=100)
plt.plot(test_Y_gt[start + window_size: start + window_size + future].
    array)
plt.plot(rescaled_prediction)
plt.legend(['Ground truth','Predictions'])
plt.title('24h Horizon future predictions')
plt.show()
```

Este método de predicción a futuro se implementa en el notebook de Jupyter *test_demo.ipynb*, de manera que se muestra de manera visual cómo se pueden realizar inferencias.

También se realiza una función que, usando la implementación anterior, vaya barriendo los datos y vaya generando predicciones. Es decir, se realizarán inferencias más de una vez utilizando el método anterior y se apilarán los resultados para luego representarlos mediante una gráfica.

En el notebook *test_demo.ipynb* también se añade una prueba de concepto que hace referencia a una prueba en tiempo real. También se añade en el mismo notebook un método para comparar las predicciones de los distintos modelos entrenados de manera visual.

7.6 Resultados

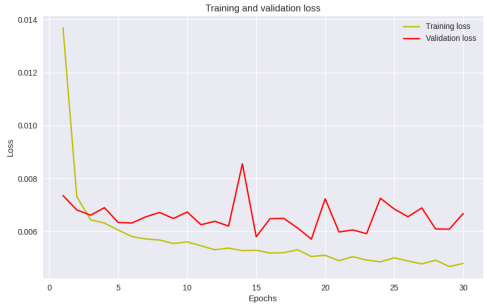
Se entrenan los modelos para distintas longitudes de secuencias de entrada. Se compararán los resultados obtenidos para unas ventanas temporales de tamaño 24 y 96, es decir, ventanas temporales de 1 y 4 días respectivamente. Se ha descartado la ventana temporal de 168 valores debido a que los modelos no convergían y su funcionamiento era peor.

Para resumir, se muestran sólo los resultados para una secuencia de entrada de 24 valores. No obstante, posteriormente se realizará una comparativa de los distintos modelos con los distintos tamaños de ventanas temporales.

7.6.1 Entrenamiento

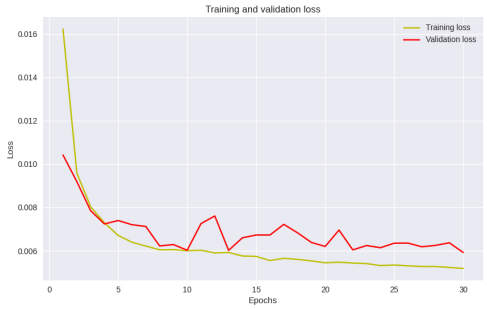
Se representan los historiales de los entrenamientos con el valor del coste de entrenamiento y de validación frente al número de épocas. Los entrenamientos que se representan en la Figura 7.7 han sido realizados con 30 épocas y con un tamaño de secuencia de entrada de 96 (4 días).

Se observa que, en todos ellos, el coste de entrenamiento ha ido disminuyendo de manera progresiva sin muchas oscilaciones. Esto afirma que el optimizador elegido ha sido adecuado.

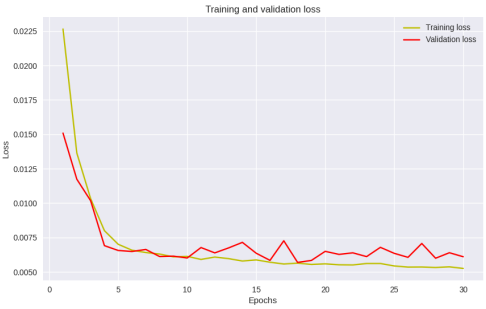


(a) Historial entrenamiento modelo RNN.

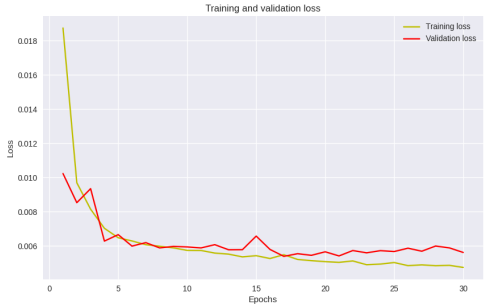
(b) Historial entrenamiento modelo st-RNN.



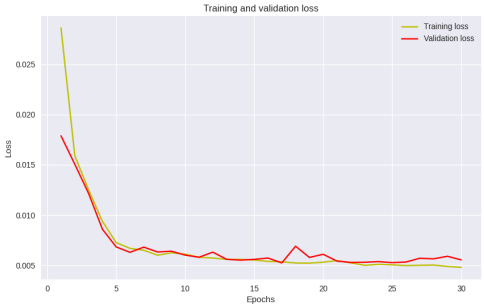
(c) Historial entrenamiento modelo LSTM.



(d) Historial entrenamiento modelo st-LSTM.



(e) Historial entrenamiento modelo Bi-LSTM.



(f) Historial entrenamiento modelo CNN+LSTM.

Figura 7.7 Historiales de los entrenamientos con los distintos modelos.

En los entrenamientos de los modelos RNN y st-RNN se aprecia un mayor coste de validación que de entrenamiento en las épocas finales. Esto significa que está habiendo *overfitting* a los datos de entrenamiento. No obstante, en el resto de modelos, los costes de validación y de entrenamiento son más parecidos.

7.6.2 Validación

Como se comenta anteriormente, se realiza una comparación cuantitativa del funcionamiento de los distintos modelos con dos ventanas temporales distintas en la Tabla 7.3.

El modelo que ha obtenido los peores resultados ha sido el modelo RNN con una ventana temporal de 96.

De nuevo, el modelo que ha obtenido los modelos resultados ha sido el formado por las redes LSTM, con una ventana temporal de 24 valores.

Tabla 7.3 Comparación de los modelos.

w_size	Métricas	RNN	st- RNN	LSTM	st- LSTM	Bi-LSTM	CNN + LSTM
24	RMSE train (kW)	94.21	92.71	99.78	100.67	94.69	96.28
	RMSE test (kW)	114.98	117.32	108.25	109.89	105.44	104.61
	MAE train (kW)	47.31	52.60	51.44	56.38	49.50	49.00
	MAE test (kW)	60.37	64.45	58.55	62.82	56.06	55.60
96	RMSE train (kW)	94.49	-	103.08	101.40	97.26	96.30
	RMSE test (kW)	119.57	-	107.35	115.08	108.44	110.75
	MAE train (kW)	50.88	-	53.04	57.09	55.40	52.00
	MAE test (kW)	67.46	-	58.01	69.35	61.05	62.35

Respecto al tamaño la ventana temporal se puede afirmar que, en general, los modelos han obtenido mejores resultados utilizando ventanas temporales de menor tamaño.

7.6.3 Test

En primer lugar se muestran las predicciones sobre todo el conjunto de datos de entrenamiento y los de validación con un horizonte temporal de 1 valor en el futuro. Los resultados se muestran en la Figura 6.11. Se ha mostrado sólo una franja de los datos para mejorar la visibilidad y mostrar más en detalle las predicciones. Las curvas verdes representan las predicciones de los modelos mientras que las azules representan los valores reales de la demanda.

Los resultados obtenidos han sido los esperados y muestran un buen funcionamiento cuando el horizonte temporal es de 1 valor. Sin embargo, el problema de predicción se complica cuando no es sólo 1 valor el que hay que predecir sino varios.

A continuación se realizan predicciones individuales a futuro con una secuencia de 24 valores de salida (1 día) cuyos resultados se muestran en la Figura 7.8. Para obtener estas gráficas se ha utilizado el modelo LSTM con una ventan temporal de 96 valores.

En este caso se observa que los valores predichos difieren más de los valores reales que cuando el horizonte temporal era de 1 valor. Sin embargo, teniendo en cuenta lo ambicioso que es el problema que se quiere resolver, los resultados son satisfactorios ya que, aunque no sean capaces de predecir las pequeñas fluctuaciones, son capaces de anticipar la tendencia de la serie temporal, que es lo que se necesita para el problema de optimización de la estación de vehículos eléctricos.

Siguiendo por la misma línea, se representan en la Figura 7.9 las predicciones conjuntas con un horizonte temporal de 24 horas, para ver cómo sería la predicción recursiva de más de un día en adelante.

Finalmente, se representan los resultados de la comparativa de predicciones para varios modelos en la Figura 7.10.

7.7 Conclusión

Se ha podido comprobar que es posible utilizar modelos basados en deep learning para solucionar el problema de predicción de producción fotovoltaica. El uso de datos multivariables enriquece más el problema de predicción y posibilita a los modelos aprender las relaciones no lineales entre ellos para generar predicciones.

En este capítulo, se ha hecho uso de datos reales de una planta fotovoltaica para entrenar a los modelos tras un previo procesado de los mismos. Se ha realizado un estudio comparativo de distintos

modelos, incluyendo los modelos ANN, RNN, LSTM, Bi-LSTM y CNN+LSTM implementándolos en dos escenarios distintos con distintos tamaños de ventanas temporales después de haber sido entrenados mediante un algoritmo de aprendizaje supervisado. Se han buscado los hiperparámetros óptimos de los modelos para conseguir el mejor funcionamiento. Métricas como MAE y RMSE han sido utilizadas para evaluar el funcionamiento de los modelos.

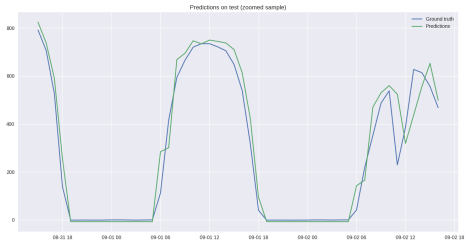
Los resultados muestran que los modelos de deep learning se han ajustado adecuadamente y son efectivos a la hora de predecir la producción en el futuro de una manera precisa para la optimización de un sistema de gestión de potencia dinámica. Dentro de los modelos de deep learning utilizados, el modelo LSTM se ha mostrado superior al resto de modelos y se demuestra que cumple correctamente con la tarea de predecir la producción en un plazo de un día.

De nuevo se justifica la importancia de la cantidad y de la calidad de los datos. La falta de datos y los datos atípicos provocados por fallos en los sensores ha hecho que fuese necesario un preprocesamiento de los datos, que posiblemente haya dificultado el entrenamiento a los modelos.

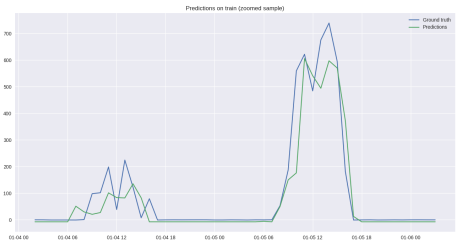
La versatilidad de los modelos de redes neuronales se ha visto reflejada, ya que se ha comprobado la fácil adaptación de los modelos a los distintos problemas. Los modelos construidos para el capítulo anterior se hicieron de manera que admitiesen secuencias de entrada de datos multivariables. Se ha mostrado también la utilidad de haber desarrollado todo un marco de trabajo en el capítulo anterior, con la automatización de los entrenamientos y la validación.

La resolución del problema de predecir en un horizonte temporal de 24h es compleja y ambiciosa. Las técnicas y soluciones adoptadas pueden ser mejoradas severamente. No obstante, lo más importante es que se ha realizado una implementación extremo a extremo de modelos de redes neuronales y se ha hecho un estudio de cómo les afecta el tamaño de las ventanas temporales. Además, se ha comprobado la capacidad de los modelos de captar las tendencias y la temporalidad de los datos y la versatilidad de los mismos al poder ser aplicados en problemas muy variados.

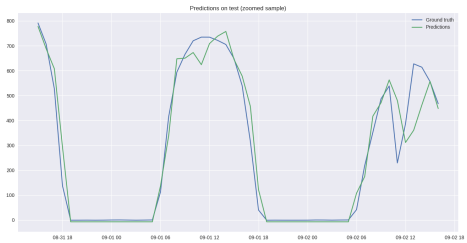
(a) Predicciones entrenamiento modelo RNN .



(b) Predicciones test modelo RNN.

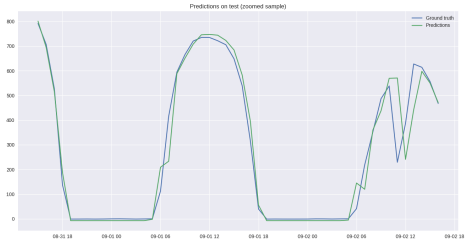


(c) Predicciones entrenamiento modelo RNN-st .



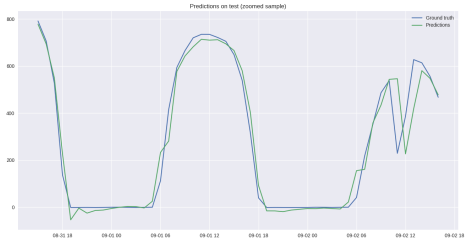
(d) Predicciones test modelo RNN.

(e) Predicciones entrenamiento modelo LSTM .

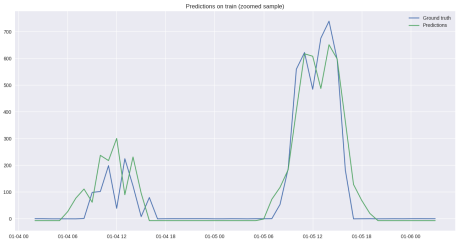


(f) Predicciones test modelo RNN.

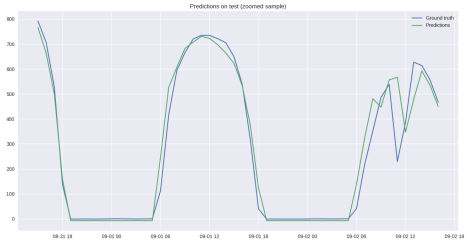
(g) Predicciones entrenamiento modelo LSTM-st .



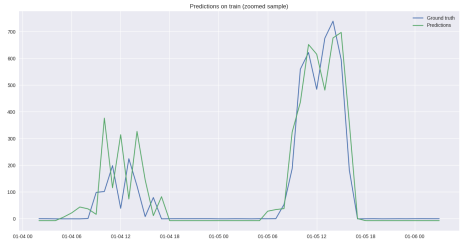
(h) Predicciones test modelo LSTM-st.



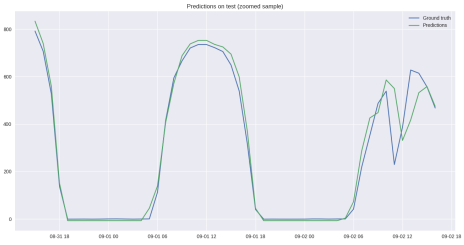
(i) Predicciones entrenamiento modelo Bi-LSTM .



(j) Predicciones test modelo CNN.



(k) Predicciones entrenamiento modelo CNN .



(l) Predicciones test modelo CNN.

Figura 7.7 Resultados predicciones (zoom). Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

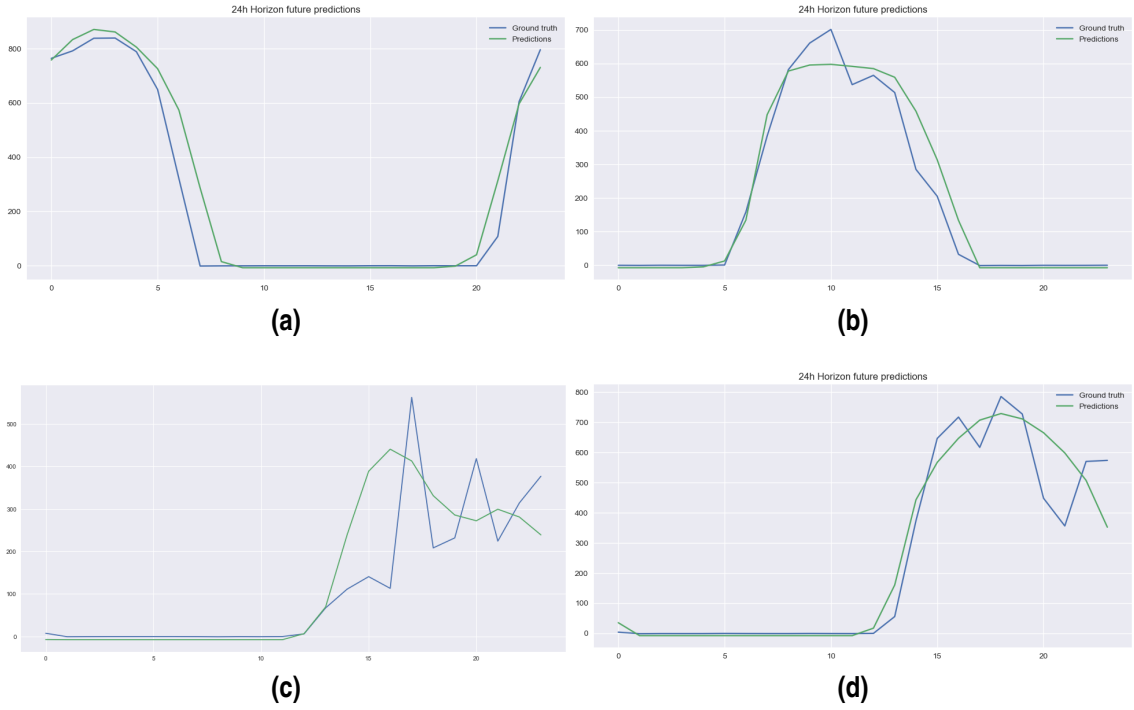
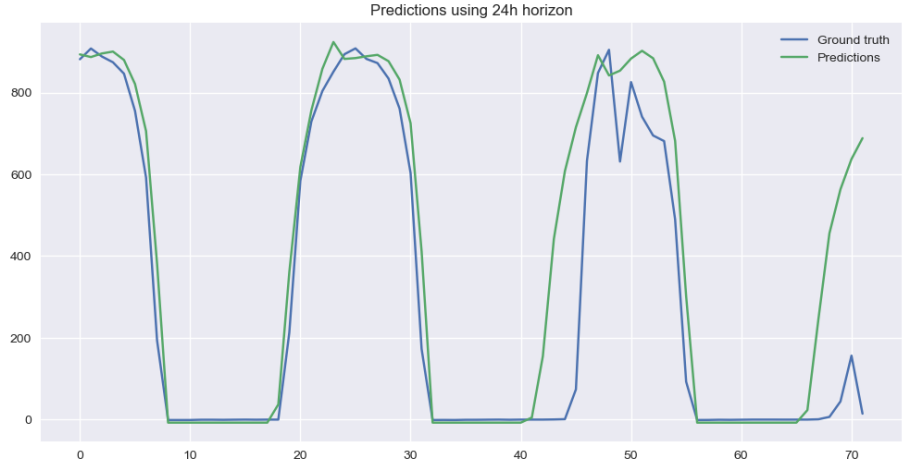
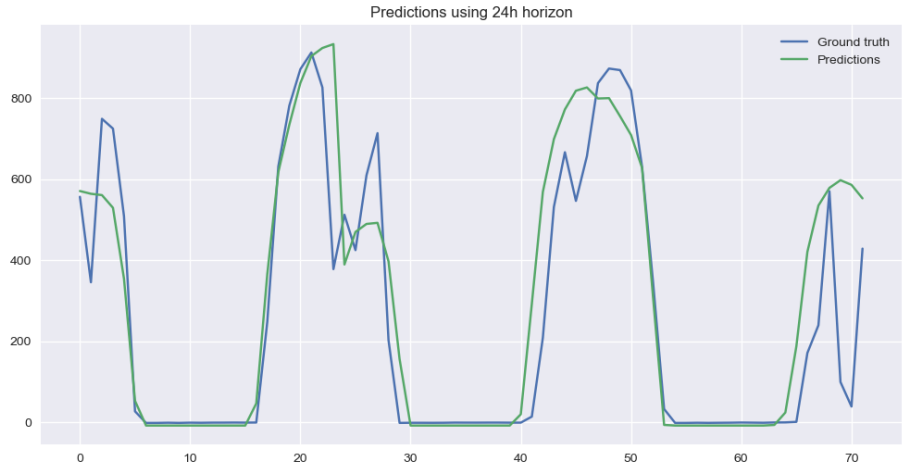


Figura 7.8 Resultados predicciones con horizonte de 24h. Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

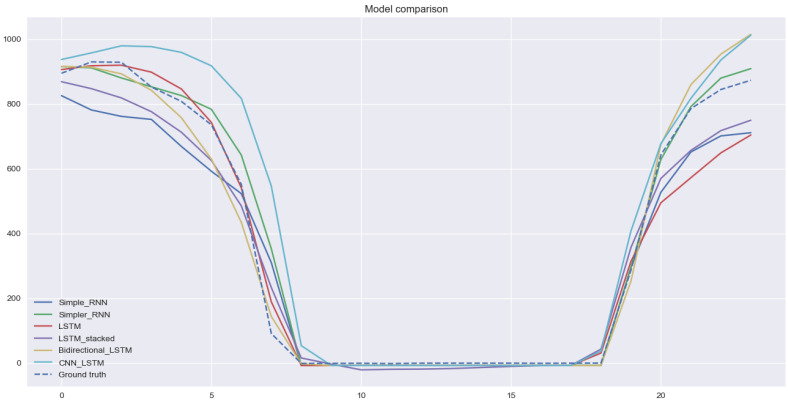


(a)

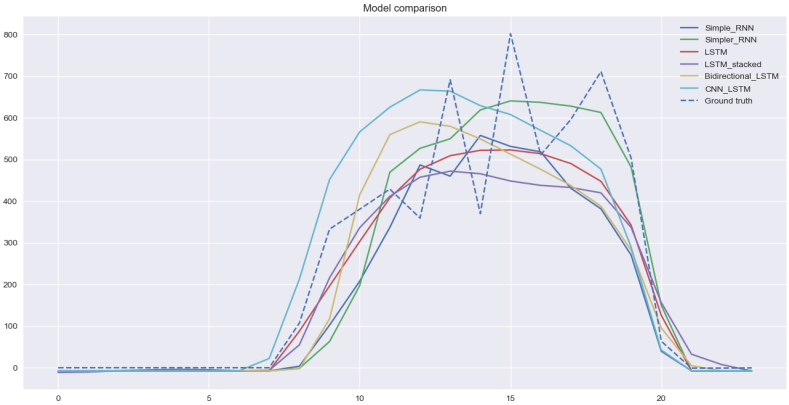


(b)

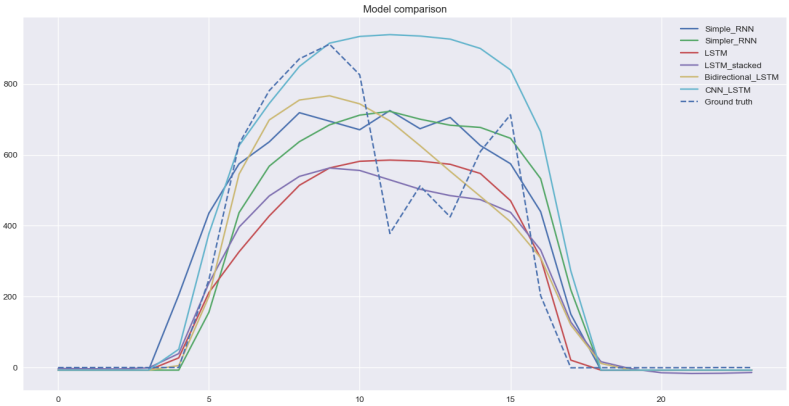
Figura 7.9 Resultados de varias predicciones consecutivas con horizonte de 24h.



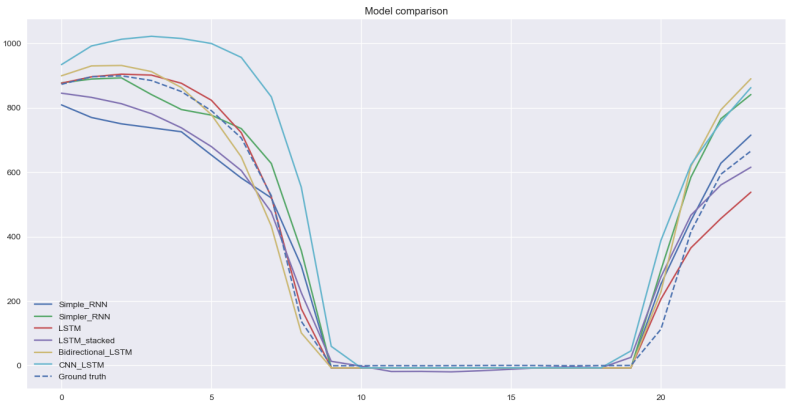
(a)



(b)



(c)



(d)

Figura 7.10 Resultados comparación de predicciones con horizonte de 24h. Eje X: tiempo (h) - Eje Y: potencia demandada (kW).

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] (2020) What's the difference between ev charging levels? https://freewiretech.com/difference-between-ev-charging-levels/.

[2] (2020) Different types of ev charging connectors. https://bestchargers.eu/blog/different-types-of-ev-charging-connectors.

[3] Connector types for ev charging around the world. https://www.evexpert.eu/eshop1/knowledge-center/connector-types-for-ev-charging-around-the-world.

[4] (2021) Electric vehicle (ev) charging standards and how they differ. https://electrek.co/2021/10/22/electric-vehicle-ev-charging-standards-and-how-they-differ/.

[5] (2022) ¿qué es la iso 15118? https://www.lugenergy.com/que-es-la-iso-15118/.

[6] (2022) El sistema eléctrico español. https://www.ree.es/sites/default/files/publication/2022/03/downloadable/Avance_ISE_2021.pdf.

[7] ¿cómo funcionan las plantas fotovoltaicas? https://www.iberdrola.com/sostenibilidad/que-es-energia-fotovoltaica.

[8] ¿qué es la energía eólica, cómo se transforma en electricidad y cuáles son sus ventajas? https://www.iberdrola.com/sostenibilidad/energia-eolica.

[9] F. M. Shakeel and O. P. Malik, "Vehicle-to-grid technology in a micro-grid using dc fast charging architecture," *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*, pp. 1–4, 2019.

[10] K. Tan, V. K. Ramachandaramurthy, and J. Y. Yong, "Bidirectional battery charger for electric vehicle," 05 2014, pp. 406–411.

[11] A. Arancibia and K. Strunz, "Modeling of an electric vehicle charging station for fast dc charging," in *2012 IEEE International Electric Vehicle Conference*, 2012, pp. 1–6.

[12] Seldon. (2021) Supervised vs unsupervised learning explained. https://www.seldon.io/supervised-vs-unsupervised-learning-explained.

[13] Y. Zhang, J. Qin, D. S. Park, W. Han, C.-C. Chiu, R. Pang, Q. V. Le, and Y. Wu, "Pushing the limits of semi-supervised learning for automatic speech recognition," 2020. [Online]. Available: https://arxiv.org/abs/2010.10504

[14] G. T. Hudson and N. A. Moubayed, "Muld: The multitask long document benchmark," 2022. [Online]. Available: https://arxiv.org/abs/2202.07362

[15] B. Tan, Z. Yang, M. AI-Shedivat, E. P. Xing, and Z. Hu, "Progressive generation of long text with pretrained language models," 2020. [Online]. Available: https://arxiv.org/abs/2006.15720

[16] H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and T. Zhao, "SMART: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization," 2020. [Online]. Available: https://doi.org/10.18653%2Fv1%2F2020.acl-main.197

[17] D. Jones, "Macsen: A voice assistant for speakers of a lesser resourced language," in *Proceedings of the 1st Joint Workshop on Spoken Language Technologies for Under-resourced languages (SLTU) and Collaboration and Computing for Under-Resourced Languages (CCURL)*. Marseille, France: European Language Resources association, May 2020, pp. 194–201. [Online]. Available: https://aclanthology.org/2020.sltu-1.27

[18] G. Belani, "The use of artificial intelligence in cybersecurity: A review," https://www.computer.org/publications/tech-news/trends/the-use-of-artificial-intelligence-in-cybersecurity.

[19] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen, "Glide: Towards photorealistic image generation and editing with text-guided diffusion models," 2021. [Online]. Available: https://arxiv.org/abs/2112.10741

[20] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. d. M. dÁutume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals, "Competition-level code generation with alphacode," 2022. [Online]. Available: https://arxiv.org/abs/2203.07814

[21] H. Marius, "State-of-the-art image classification algorithm: Fixefficientnet-l2," https://towardsdatascience.com/state-of-the-art-image-classification-algorithm-fixefficientnet-l2-98b93deeb04c.

[22] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, "Object detection with deep learning: A review," 2018. [Online]. Available: https://arxiv.org/abs/1807.05511

[23] E. Vahdani and Y. Tian, "Deep learning-based action detection in untrimmed videos: A survey," 2021. [Online]. Available: https://arxiv.org/abs/2110.00111

[24] V. Kuznetsov and Z. Mariet, "Foundations of sequence-to-sequence modeling for time series," *CoRR*, vol. abs/1805.03714, 2018. [Online]. Available: http://arxiv.org/abs/1805.03714

[25] H. B. Barlow, "Unsupervised learning," in *Neural computation, Vol.1, 2014 6th International Conference of*, 1989, pp. 295–311.

[26] S. Dridi, "Unsupervised learning - a systematic literature review," 12 2021.

[27] W. S. Sarle, "Neural networks and statistical models," 1994.

[28] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[29] Deep learning specialization. https://www.coursera.org/specializations/deep-learning.

[30] J. Du, "The frontier of sgd and its variants in machine learning," *Journal of Physics: Conference Series*, vol. 1229, p. 012046, 05 2019.

[31] G. Ian, B. Yoshua, and C. Aaron, *Deep Learning.*, ser. Adaptive Computation and Machine Learning. The MIT Press, 2016. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=2565107&lang=es&site=ehost-live&scope=site

[32] Cnn convolution on rgb images. https://datahacker.rs/convolution-rgb-image/.

[33] C. Kiourt, G. Pavlidis, and S. Markantonatou, "Deep learning approaches in food recognition," 2020.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[35] M. Elgendy, *Deep Learning for Vision Systems*. Manning, Oct. 2020.

[36] F. Chollet, *Deep Learning with Python*. Manning, Nov. 2017.

[37] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[39] (2018) Introduction to sequence models - rnn and lstm. https://franklinwu19.github.io/2018/08/27/rnn-lstm/.

[40] Y. Li, L. N. Harfiya, K. Purwandari, and Y.-D. Lin, "Real-time cuffless continuous blood pressure estimation using deep learning model," *Sensors*, vol. 20, 09 2020.

[41] A. L. Llamas, in *Optimización de carga de vehículos eléctricos con asistencia solar fotovoltaica y almacenamiento de energía*, 2022.

[42] M. Bemporad, A. y Morari, in *Advances in Neural Information Processing Systems*, vol. 35. Automatica, 1999, pp. 407–427.

[43] Z. J. Lee, T. Li, and S. H. Low, "ACN-Data: Analysis and Applications of an Open EV Charging Dataset," in *Proceedings of the Tenth International Conference on Future Energy Systems*, ser. e-Energy '19, Jun. 2019.

Índice alfabético