UNIVERSIDAD DE SALAMANCA
MÁSTER UNIVERSITARIO EN MODELIZACIÓN MATEMÁTICA

Uso de aprendizaje profundo para la predicción del uso diario de sistemas de préstamo de bicicletas en relación con diferentes eventos sociales para la creación de una ciudad más sostenible

ALUMNO: Julián Sáez Carrión

TUTORES: Dr. Alfonso González Briones
Dr. Guillermo Hernández González

Curso 2022-2023

UNIVERSIDAD DE SALAMANCA
MÁSTER UNIVERSITARIO EN MODELIZACIÓN MATEMÁTICA

Uso de aprendizaje profundo para la predicción del uso diario de sistemas de préstamo de bicicletas en relación con diferentes eventos sociales para la creación de una ciudad más sostenible

ALUMNO: Julián Sáez Carrión

TUTORES: Dr. Alfonso González Briones
Dr. Guillermo Hernández González

Curso 2022-2023

MÁSTER UNIVERSITARIO EN MODELIZACIÓN MATEMÁTICA
Facultad de Ciencias
Universidad de Salamanca
e-mail: mumoma@usal.es
https://www.usal.es/modelizacion-matematica

CERTIFICADO DE LOS/AS TUTORES/AS DE TRABAJO DE FIN DE MÁSTER

MÁSTER UNIVERSITARIO EN MODELIZACIÓN MATEMÁTICA

D. Alfonso González Briones y D. Guillermo Hernández González, profesores del Departamento de Informática y Automática de la Universidad de Salamanca

HACEN CONSTAR

Que el trabajo titulado "Uso de aprendizaje profundo para la predicción del uso diario de sistemas de préstamo de bicicletas en relación con diferentes eventos sociales para la creación de una ciudad más sostenible", que se presenta, ha sido realizado por Julián Sáez Carrión, con DNI número ***12411C y constituye la memoria del trabajo realizado para la superación de la asignatura Trabajo Fin de Máster del Máster Universitario en Modelización Matemática de la Universidad de Salamanca.

Salamanca, a fecha de la firma electrónica

GONZALEZ BRIONES ALFONSO - 52415611Z

Firmado digitalmente por GONZALEZ BRIONES ALFONSO - 52415611Z
Fecha: 2023.07.09 17:57:58 +02'00'

Firmado. Alfonso González Briones

Firmado por HERNANDEZ GONZALEZ GUILLERMO - ***1124** el día 10/07/2023 con un certificado emitido por AC FNMT Usuarios

Firmado. Guillermo Hernández González

Índice general

Resumen

En este Trabajo de Fin de Máster, *Uso de aprendizaje profundo para la predicción del uso diario de sistemas de préstamo de bicicletas en relación con diferentes eventos sociales para la creación de una ciudad más sostenible*, se trata de predecir la demanda de bicicletas del servicio público de préstamo Salenbici, de la ciudad de Salamanca, considerando un horizonte de predicción de dieciséis semanas, a través del uso del aprendizaje profundo, una herramienta que ha demostrado proporcionar muy buenos resultados en tareas de regresión y predicción. En caso de disponer de buenas predicciones de la demanda, la empresa proveedora del servicio podría mejorar la calidad del servicio y la eficiencia de su gestión.

En este trabajo, se consideran primeramente, a través de una revisión bibliográfica, los fundamentos matemáticos del aprendizaje profundo y las herramientas, algoritmos y técnicas detrás del entrenamiento de redes neuronales, analizando las que más comúnmente se usan en problemas de regresión y predicción de series temporales. Se implementan y adaptan al problema seis redes neuronales distintas que se plantean en la bibliografía, haciendo uso de la librería Keras, de Python. Se concluye que es posible predecir, con gran precisión, el número de préstamos cada día a partir de información externa, como los calendarios vacacionales y de festivos, los fines de semana o información climática, permitiendo verificar la gran utilidad que tiene el aprendizaje profundo a la hora de tratar este tipo de problemas.

Abstract

In this Final Master's Thesis, titled *Use of deep learning to predict daily usage of bike sharing systems related to different social events to create a more sustainable city*, the goal is to predict the demand for bicycles of the public loan service Salenbici, of the city of Salamanca, considering a prediction horizon of sixteen weeks, making use of deep learning, a tool that has shown very good results in regression and prediction tasks. If good demand predictions are available, the service provider company could improve the service quality and management efficiency.

In this Master's Thesis, the mathematical foundations of deep learning and the tools, algorithms, and techniques behind neural network training are initially considered through a literature review, analyzing the most commonly used for regression and time series prediction problems. Six different neural networks mentioned in the literature are implemented and adapted to the problem using the Keras library in Python. It is concluded that it is possible to predict the number of daily bike loans with high accuracy using external information such as holiday and vacation calendars, weekends, or weather data. This highlights the great utility of deep learning in addressing these types of problems.

Capítulo 1

Introducción

En la actualidad, los servicios de préstamo de bicicletas se han expandido por todo el mundo. En 2019, el Observatorio Público por la Bicicleta Pública [1] en España concluyó que existían 43 sistemas públicos de préstamo de bicicletas en España, en ciudades y áreas urbanas cuya suma de poblaciones era de 14.911.563 habitantes.

El uso de estos servicios se presenta como una alternativa más limpia, más sana y, en algunos contextos, más conveniente y cómoda que los vehículos a motor, reduciendo también la congestión vehicular. Para mejorar la experiencia de los usuarios, es de vital importancia para la empresa proveedora del servicio prever la demanda para, entre otras tareas, garantizar una buena disponibilidad de bicicletas en las estaciones. En este contexto, la inteligencia artificial en general y el aprendizaje profundo (*deep learning*) en particular se presentan como unas herramientas potentes para mejorar la predicción y gestión de estos servicios.

La inteligencia artificial se ha desarrollado en las últimas décadas hasta llegar a formar parte de multitud de herramientas presentes en nuestra vida diaria: vehículos autónomos, aplicaciones de recomendación, escritura de texto o generación de imágenes... A pesar de la popularidad del término, no es posible realizar una definición más precisa que "una aplicación que hace algo inteligente", evolucionando además el término "inteligente" a lo largo de los años. Recientemente, este término se ha usado fundamentalmente para referirnos a algoritmos de aprendizaje automático (*machine learning*), cuya característica fundamental es que ajustan sus parámetros en respuesta a un conjunto de datos. De entre estos algoritmos, nosotros nos centraremos en aplicar redes neuronales artificiales, una técnica en aprendizaje automático que toma como inspiración el sistema nervioso humano y la estructura del cerebro.

Las redes neuronales artificiales tratan de resolver problemas que no se pueden modelar con otras herramientas matemáticas. Un ejemplo clásico es la clasificación de imágenes: realizar un programa estático capaz de clasificar imágenes de especies animales sería completamente inviable. Una imagen (por simplicidad, normalizada entre 0 y 1) de resolución 1280×960 contendría un total de 1.228.800 píxeles y 3.686.400 parámetros independientes, con lo que un programa que tratase de clasificar esta imagen tendría que definir una función $f : [0, 1]^{3.686.400} \to \{a | a \text{ es una especie animal}\}$. Por esta enorme dependencia dimensional, y las características arbitrarias que pueden definir cada animal posible, escribir dicho programa desde cero sería prácticamente imposible.

Es aquí donde las redes neuronales artificiales (a menudo referidas simplemente como redes neuronales) pueden ser de gran utilidad. En lugar de tratar de encontrar la función f antes descrita, trataríamos de aproximarla con otra función $F : [0,1]^{3,686,400} \to \{a|a$ es una especie de entre una colección predeterminada de animales$\}$, que ajustaríamos a partir de datos etiquetados (es decir, multitud de imágenes de animales, cada una con la especie a la que pertenece el animal). Comenzaríamos este proceso con una función inicial F_0, y se realizarían una serie de iteraciones de entrenamiento (frecuentemente denominadas *training epochs*). La red comenzaría calculando una función F_1, que en general será una muy mala aproximación de f, y compara las clasificaciones hechas por esta función con imágenes previamente clasificadas por seres humanos. Si programamos que la red se adapte para obtener progresivamente mejores tasas de clasificación, acabaremos con un programa que clasifique nuevas imágenes de manera consistente y satisfactoria.

Este método ha obtenido resultados muy exitosos en la última década, y hoy cuenta con un muy amplio abanico de aplicaciones, incluyendo la resolución de problemas relacionados con la predicción de series temporales, como la que atañe al escenario de predicción en el contexto de los servicios de préstamo de bicicletas. A pesar de este enorme desarrollo práctico, nuestro conocimiento teórico de por qué y cómo es que estas técnicas funcionan es todavía muy limitado, existiendo una gran brecha entre nuestro conocimiento práctico y teórico de la materia [2].

En este Trabajo, nos centraremos en el problema de modelizar los préstamos del servicio público de préstamo de bicicletas de Salamanca, Salenbici [3], por lo que analizaremos redes neuronales con aplicación en predicción de series temporales. El problema que trataremos podrá tener, por tanto, una gran utilidad para aumentar la eficiencia de la gestión del servicio y optimizar el uso de los recursos de la empresa. Trataremos de plantear, optimizar y comparar distintos modelos que aparecen en la bibliografía.

Capítulo 2

Estado de la cuestión

El uso de aprendizaje profundo ha demostrado ser de gran utilidad para el modelizado de sistemas de préstamo de bicicletas. Son muchas las ciudades que poseen servicios de este tipo, y las empresas encargadas de dicho servicio son capaces de recoger mucha información sobre los trayectos de los usuarios. Esta información puede ser usada para entrenar modelos que estimen el número diario de préstamos, el destino de cada viaje individual, la cantidad de préstamos que comenzarán y acabarán cada día en cada estación, cuándo las estaciones se quedarán sin bicicletas...

Para un mismo conjunto de datos de desplazamientos, podemos formular varios problemas distintos, todos ellos de un potencial gran interés para la empresa que gestiona el servicio. Por ejemplo, en [4] tratan el problema de, a partir de la información geográfica de todas las estaciones y todos los datos de recogida de bicicletas, predecir el destino más probable de un nuevo desplazamiento para una estación específica, a través del uso de una red neuronal convolucional.

Otra investigación [5] propone usar redes espaciotemporales de fusión (STFNet) y mecanismos de atención para predecir el flujo de pasajeros a corto plazo en los sistemas de préstamo de bicicletas de Shenzhen y Pekín. Estos sistemas se organizan sin estaciones, esto es, las bicicletas se pueden recoger y dejar en cualquier sitio válido.

En [6] tratan un problema parecido al que nosotros planteamos, realizando una discretización espacial de la ciudad de Nankín, y a partir de datos a gran escala de desplazamientos en bicicleta durante dos semanas, proponen hacer uso de redes neuronales del tipo LSTM para predecir las salidas y entradas de cada área utilizando datos de observaciones pasadas. En su caso, emplean la discretización dado que el servicio de préstamo que estudian no dispone de estaciones.

Nosotros nos centraremos en el problema de predecir la demanda de préstamo de bicicletas en la ciudad de Salamanca, a partir de datos de cada desplazamiento individual e información externa, como el clima, los días festivos, el día de la semana... Se han realizado multitud de estudios que analizan la influencia de parámetros externos sobre el uso de servicios de préstamo de bicicletas (por ejemplo, [7] analiza la influencia de las condiciones climáticas en los desplazamientos en el sistema de bicicletas compartidas de Washington D. C., y en [8] se analizan, en general, los factores que influyen en la toma de bicicletas compartidas en Beijing). Se ha

descrito el efecto de características de la ciudad, como la población, el gasto gubernamental en infraestructuras viales y la estructura de estas; del sistema de préstamo de bicicletas, como el número de bicicletas disponibles o el número de usuarios del servicio; del clima... En general, para el caso de una ciudad, podemos considerar que algunos de estos factores, como la calidad de las vías o la población de la ciudad, se mantienen constantes en intervalos de tiempo de pocos años, por lo que podemos centrarnos en la influencia que la temperatura, el viento, las precipitaciones, las festividades o el día de la semana pueden tener en la demanda de bicicletas y despreciar los factores de variación lenta.

Capítulo 3

Contexto del trabajo

El objetivo del trabajo es diseñar y optimizar modelos para predecir el préstamo diario de bicicletas del servicio de bicicletas Salenbici, el servicio de préstamo de bicicletas de la ciudad de Salamanca, que es la capital de la provincia homónima, en Castilla y León (España), en función de factores externos como la temperatura, las precipitaciones, el día de la semana, la existencia de festivos, las vacaciones universitarias...

Todos estos datos son conocidos de antemano (para cada día, por ejemplo, sabemos si existe o no un festivo) o pueden predecirse a corto plazo (temperaturas o precipitaciones, haciendo uso de servicios meteorológicos), por lo que un servicio podría usar un modelo de predicción de préstamos que le podría ser de gran utilidad.

En nuestro caso, nos centraremos en realizar predicciones de 16 semanas (112 días), un enfoque a un plazo más largo de lo que se suele plantear en la bibliografía. El mayor problema de un planteamiento a tan largo plazo es la imposibilidad de obtener datos meteorológicos fiables. Si lo seguimos es, fundamentalmente, para poder valorar la calidad de las predicciones en un intervalo de validación lo suficientemente extenso. Este enfoque puede tener poca aplicabilidad para ciertas cuestiones (por ejemplo, es imposible usar el modelo para prever qué hora una cierta estación se quedará sin bicicletas), aunque sí que existen algunas aplicaciones de una perspectiva a largo plazo. Algunas formas en las que disponer de un modelo de predicción del préstamo podría mejorar el servicio son:

- Planificación del mantenimiento: Cuando sea necesario revisar y arreglar las bicicletas, la empresa puede minimizar el impacto en el servicio realizando los mantenimientos en las fechas en las que el modelo prevea una menor demanda.
- Optimización del uso de recursos: En los días en los que se prevea una demanda baja, la empresa puede optar por no gastar recursos en distribuir bicicletas, y preservar esos esfuerzos para las ocasiones en las que sí se prevea una demanda mayor.
- Minimización de costes: Los gestores del servicio pueden decidir posponer un gasto en adquirir nuevo material o extenderse en caso de que prevean que la demanda no será suficiente como para que tal gasto merezca la pena.

Si la empresa que gestiona el servicio es capaz de mejorar su calidad y comodidad, el número de usuarios interesados en él aumentaría, reduciendo potencialmente el uso de vehículos

y mejorando la sostenibilidad de la ciudad.

Disponemos de datos de desplazamientos recogidos por Salenbici, de entre el año 2016 y 2020. En el año 2020, la ciudad de Salamanca tenía un total de 329.245 habitantes empadronados (159.929 hombres y 169.316 mujeres), con una edad media de 45,15 años [9]. Salamanca, además, por ser una ciudad universitaria, tiene un importante número de habitantes no censados, con una media de edad más baja, que cabe esperar que sean más propensos a usar servicios de préstamo de bicicletas.

Para cada uno de los desplazamientos, disponemos de información de su fecha de inicio y fecha de finalización (en formato día/mes/año hora:minuto), de qué bicicleta se usó (cada una tiene un código identificativo), de la estación de origen y de destino y de los candados de origen y destino. En total, disponemos de datos de 253.173 desplazamientos.

En lo que respecta a factores externos, consideramos primeramente datos climatológicos, obtenidos de la Agencia Estatal de Meteorología (AEMET). Desde su página web [10] se pueden obtener los datos climatológicos de la ciudad de Salamanca, en formato .txt. Para cada día, haremos uso de datos de temperaturas, precipitaciones y viento. No se dispone de cierta información climatológica en Salamanca para algunos días, por lo que tenemos que, bien borrar la información de días en los que no dispongamos de algún dato usamos la información recogida en el puesto de medida más cercano, bien sustituir los datos necesarios por los obtenidos en alguna estación cercana a Salamanca. Nosotros hemos optado por la segunda opción, usando cuando es necesario información tomada en el Aeropuerto de Salamanca, localizado a 17 km de la ciudad, o de la localidad de Vitigudino, localizada a 21,2 km de Salamanca, cuando tampoco disponemos de datos tomados en el Aeropuerto.

También consideramos factores como los días festivos de cada año, que se pueden consultar en el Boletín Oficial del Estado, o el día de la semana. Un factor difícil de tratar que podría haber afectado a la demanda de bicicletas es el confinamiento ocurrido en el año 2020 a raíz de la pandemia de COVID-19, dado que las limitaciones del movimiento variaron mucho entre las diferentes etapas de las restricciones. Precisamente por la enorme dificultad de modelizar el préstamo de bicicletas durante la pandemia, y la poca utilidad de tener en cuenta sus efectos en un contexto pospandemia, hemos desechado los datos del año 2020 (salvo los días 1 y 2 de enero) y nos centraremos en tratar con los años 2016, 2017, 2018 y 2019.

Es importante hacer un análisis del contexto geográfico de la ciudad. Por un lado, tenemos que en buena parte del centro histórico de la ciudad disponemos de un terreno plano, que facilita el uso de bicicletas. Sin embargo, alrededor del centro histórico y en zonas más alejadas existen áreas con colinas y cuestas muy pronunciadas, que pueden suponer un desafío para la movilidad. La ciudad dispone de abundantes carriles bici, que acomodan los desplazamientos en bicicleta.

El servicio de préstamo de bicicletas se estructura mediante estaciones, de tal manera que cada desplazamiento comienza en una estación y finaliza en otra. En el periodo de estudio, el servicio tenía un total de 33 estaciones repartidas por el área urbana de Salamanca. La localización de las estaciones queda representada en la Figura 3.1.

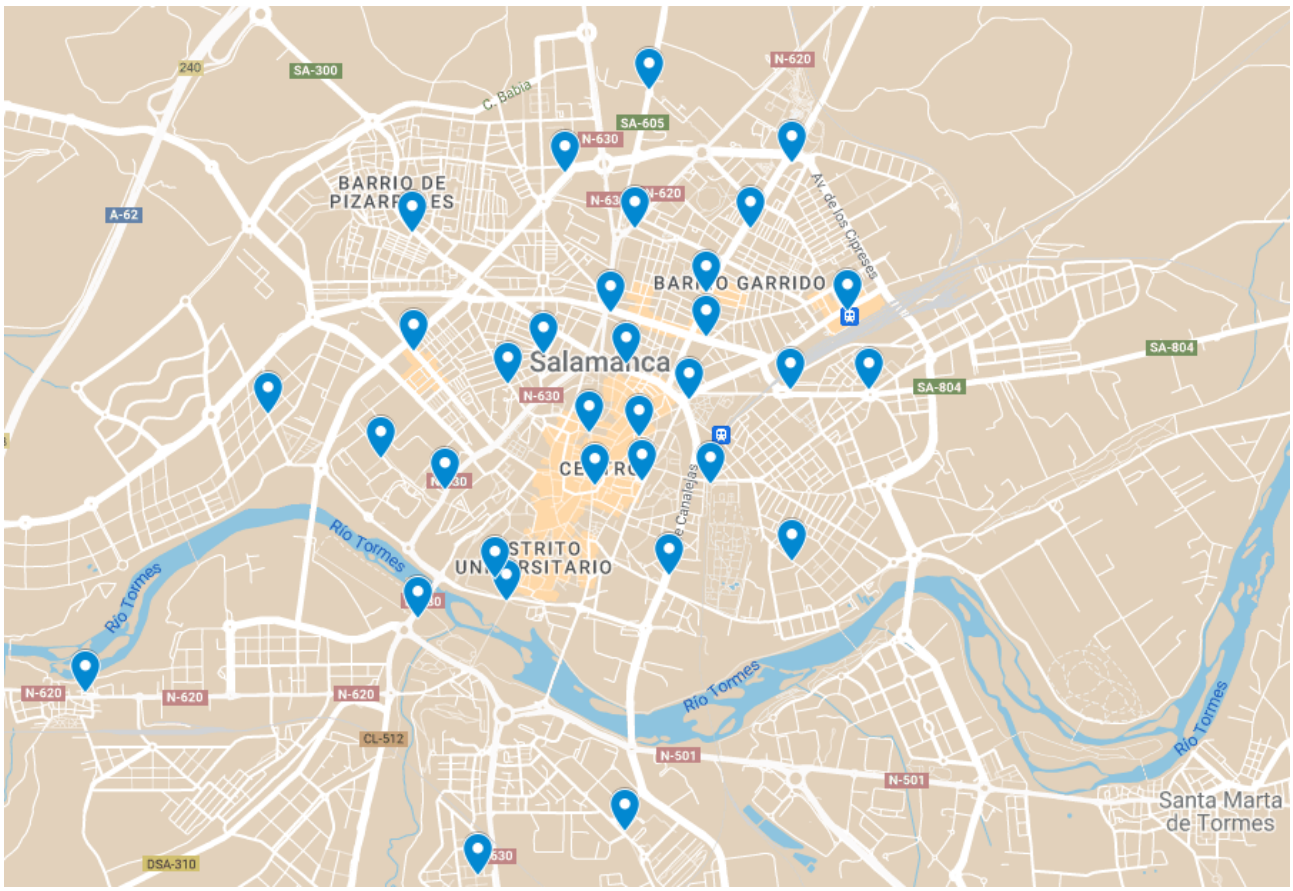


Figura 3.1: Localización de las estaciones del servicio de préstamo de bicicletas de Salenbici disponibles entre 2016 y el 2 de enero de 2020.

El servicio de bicicletas tiene una serie de normas que influyen en su uso. El horario de funcionamiento del servicio es de 7:00 a 23:00 de lunes a viernes y de 10:00 a 23:00 los sábados, domingos y festivos, existiendo penalizaciones a los usuarios que devuelvan la bicicleta después de la hora límite. El máximo tiempo permitido para cada préstamo es de una hora, y no existe límite de préstamos [3].

Capítulo 4

Objetivos del trabajo

El objetivo principal de este Trabajo es la aplicación de distintos algoritmos en aprendizaje profundo en un problema concreto de regresión y predicción de series temporales, usando datos reales: los préstamos del servicio público de bicicletas de Salamanca, Salenbici. Se busca comparar el rendimiento y la calidad de las predicciones de los distintos modelos, y mejorarlos a través de la optimización de hiperparámetros. Se realizará una predicción de 16 semanas de la demanda del servicio.

Un objetivo secundario del Trabajo es entender los fundamentos matemáticos del aprendizaje profundo, a través del análisis de las distintas técnicas y algoritmos en los que se basa el aprendizaje profundo: qué son los pesos y cuál es su función, cómo funciona un proceso de aprendizaje y qué herramientas usamos para llevarlo a cabo, qué son las funciones de activación y qué influencia tienen en el modelo, qué teoremas garantizan el funcionamiento de las redes neuronales... En lo que respecta a este objetivo, deberemos tener en cuenta que existe una gran brecha entre el desarrollo matemático y el desarrollo técnico, por lo que nos centraremos en un único tipo de redes neuronales, las prealimentadas.

Además del aspecto puramente teórico de las redes neuronales, otro objetivo secundario es el análisis de los distintos algoritmos, métodos, arquitecturas, estrategias y técnicas que se utilizan en la práctica en aprendizaje profundo, presentando de forma somera cómo funciona el proceso de aprendizaje, qué métodos se usan, qué problemas pueden presentar y qué soluciones pueden plantearse; qué arquitecturas se implementan en regresión y en predicción de series temporales; y qué estrategias podemos seguir para mejorar los resultados de nuestros modelos. Todos estos conocimientos se aplicarán usando la librería Keras [11], por lo que otro objetivo secundario es entender cómo usarla para construir modelos.

Por último, otro objetivo secundario es analizar, sin hacer uso del aprendizaje profundo, información que podemos obtener de forma sencilla del conjunto de datos del que disponemos: la importancia relativa de cada una de las estaciones, la relación entre factores climatológicos y los préstamos, cómo varían los desplazamientos semanalmente, cómo afectan los días festivos y los fines de semana en la distribución horaria de los desplazamientos... A través de este análisis, podremos llegar a algunas conclusiones sobre cómo los usuarios usan el servicio de bicicletas, y podrá ser también de utilidad a la hora de interpretar nuestras predicciones.

Capítulo 5

Conceptos teóricos

Para el estudio de los datos de los que disponemos, se han empleado diversos algoritmos de aprendizaje profundo, esto es, redes neuronales profundas (*Deep Neural Network, DNN*). Se han realizado modificaciones en propuestas encontradas en la bibliografía, con la finalidad de observar qué modelos predicen mejor los datos experimentales de los que disponemos.

La implementación de modelos de aprendizaje profundo se suele realizar tratándolas como si fuesen una caja negra, sin prestar atención al estado de cada neurona de la red. En la práctica, lo que se hace es plantear un modelo y tratar de mejorarlo siguiendo diversas técnicas de optimización. Este proceso no es sencillo, y se pueden presentar multitud de problemas durante el entrenamiento, que nos llevan a resultados subóptimos.

En la sección 5.1, analizamos la arquitectura más simple en aprendizaje profundo, las llamadas redes neuronales prealimentadas (*feedforward networks*). Estas redes permiten un entendimiento matemático de los fundamentos del aprendizaje profundo, y cuál es la función de cada una de las herramientas que usamos. Presentaremos además numerosos teoremas que demuestran su funcionalidad y trataremos conceptos como los pesos, tendencias o funciones de activación, que son de gran importancia en todas las redes neuronales. La perspectiva en este capítulo será fundamentalmente matemática, sin centrarnos en cómo aplicar una red neuronal prealimentada en la práctica. A lo largo de esta sección, nos apoyaremos en el trabajo de L. Ferreira Guilhoto [2].

En la sección 5.2, se analizan las arquitecturas, algoritmos y técnicas de aprendizaje profundo que podemos usar para el problema que analizamos. Aunque tratemos las matemáticas detrás de cada arquitectura y técnica, lo haremos de manera superficial, pues es poco el desarrollo teórico realizado por el momento. Nos basaremos en todo momento en el trabajo de A. Shrestha y A. Mahmood [12].

5.1. Fundamentos matemáticos del Aprendizaje Profundo: las redes neuronales prealimentadas

5.1.1. Estructura

Una red neuronal prealimentada es un algoritmo que trabaja en capas. En cada capa, existe un número de nodos dado, a veces conocidos como perceptrones o neuronas (por analogía con el cerebro biológico). Los nodos son simplemente representaciones de números almacenados en el programa. Empezando por la capa de entrada, hasta la capa de salida, los valores almacenados (llamados la *activación* del nodo) en la n-ésima capa servirán para calcular los valores almacenados en los nodos de la $(n+1)$-ésima capa. Matemáticamente, las activaciones almacenadas en una capa con k nodos puede ser representada por un vector de dimensión k.

Al hablar de la estructura de una red neuronal, normalmente usamos el término *profundidad* para referirnos al número de capas ocultas en la red. La característica que diferencia las redes profundas de las poco profundas es su número de capas: a mayor número de capas, más profunda es la red.

En la práctica, cada capa puede especializarse en detectar un determinado aspecto o característica de la información de entrada. Por ejemplo, en reconocimiento facial, la primera capa podría centrarse en detectar bordes, la segunda en diferenciar partes de la cara (orejas, ojos...) y la tercera podría aprender características aún más concretas.

En analogía con el término de profundidad de una red, también podemos hablar de la anchura, que hace referencia al número de nodos en cada capa oculta de la red, aunque solo tiene sentido si todas las capas ocultas tienen el mismo número de nodos. En la Figura 5.1 se muestra la representación de una red prealimentada.

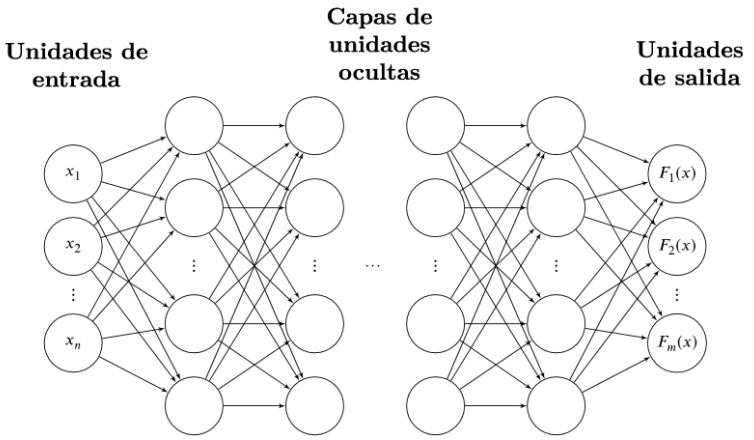


Figura 5.1: Representación visual de una red neuronal prealimentada que aproxima la función $f : \mathbb{R}^n \to \mathbb{R}^m$ a través de calcular la función $F(x) = (F_1(x), \ldots, F_m(x))$ [2]. Desde este enfoque, la red se muestra como un grafo direccionado con pesos. Nótese que se adopta la notación $x = (x_1, \ldots, x_n)$.

Este sistema se construye inspirándose en la red biológica de neuronas, en la que cada neurona, al actuar (transmitiendo una señal eléctrica), provoca una reacción en las neuronas adyacentes (propagando la señal eléctrica). Una diferencia significativa entre estos dos sistemas es que, en las redes biológicas, una neurona toma dos estados (activada o desactivada), mientras que en las redes neuronales artificiales pueden tener un continuo de activaciones: por ejemplo, un número real entre 0 o 1 en las redes sigmoides, un número no-negativo en las redes rectificadoras lineales (*rectified linear unit, ReLu*), o incluso números no reales [13].

Cada neurona en la n-ésima capa tiene asociados una serie de pesos, que miden la fuerza con la que cada neurona está conectada a cada nodo de la siguiente capa, y que son números reales. Llamemos $w^n_{a,b} \in \mathbb{R}$ al peso entre el nodo a-ésimo de la $(n-1)$-ésima capa y el b-ésimo nodo de la n-ésima capa. Podemos representar todos los pesos conectando estas dos capas usando una matriz:

$$W_n \equiv \begin{bmatrix} w^n_{1,1} & \cdots & w^n_{1,k} \\ \vdots & \ddots & \vdots \\ w^n_{j,1} & \cdots & w^n_{j,k} \end{bmatrix}. \tag{5.1}$$

También añadimos para cada nodo una tendencia (*bias*), o límite (*threshold*), que será una constante $b \in \mathbb{R}^j$, y una función de activación $\sigma : \mathbb{R} \to \mathbb{R}$ que convierte el valor calculado por los pesos y tendencias en un número real que puede ser almacenado en la red. Al final, para calcular el estado A_n de la n-ésima capa, usamos la expresión

$$A_n = \sigma\left(W_n A_{n-1} + b_n\right) = \sigma\left(\begin{bmatrix} w^n_{1,1} & \cdots & w^n_{1,k} \\ \vdots & \ddots & \vdots \\ w^n_{j,1} & \cdots & w^n_{j,k} \end{bmatrix} \begin{bmatrix} a^{n-1}_1 \\ \vdots \\ a^{n-1}_k \end{bmatrix} + \begin{bmatrix} b^n_1 \\ \vdots \\ b^n_j \end{bmatrix}\right). \tag{5.2}$$

En la expresión anterior, hemos usado la siguiente notación para la imagen de la función de activación sobre matrices:

$$\sigma\left(\begin{bmatrix} a_{1,1} & \dots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{j,1} & \dots & a_{j,k} \end{bmatrix}\right) = \begin{bmatrix} \sigma(a_{1,1}) & \dots & \sigma(a_{1,k}) \\ \vdots & \ddots & \vdots \\ \sigma(a_{j,1}) & \dots & \sigma(a_{j,k}) \end{bmatrix}. \tag{5.3}$$

Como los cálculos se realizan a lo largo de las capas de la red, la función final F calculada por una red de profundidad N es

$$F(x) = \sigma\left(W_N \sigma\left(\dots \sigma\left(W_2 \sigma\left(W_1 x + b_1\right) + b_2\right)\dots\right) + b_N\right). \tag{5.4}$$

5.1.2. Proceso de aprendizaje

Las redes neuronales se usan para estimar funciones de complejidad arbitraria. Tratemos qué métodos usamos para que las funciones se adapten a los datos de entrenamiento.

Función de coste

Supongamos que queremos aproximar una función $f : \mathbb{R}^n \to \mathbb{R}^m$, y nuestra red, en su estado actual, calcula una función $F : \mathbb{R}^n \to \mathbb{R}^m$. Para determinar la calidad de la predicción realizada

por el algoritmo, definimos una función de coste, que mide las discrepancias entre las funciones f y F. Para hacer esto, podemos alimentar con alguna entrada x, de imagen $f(x)$ conocida, a la red, y observar cuál es la salida. En general, introduciremos un conjunto $(x_1, \ldots, x_N)$, cuyas imágenes $(f(x_1), \ldots, f(x_n))$ conocemos. Esta definición acepta varias definiciones distintas, aunque lo más común es usar el llamado error medio cuadrático, o función de coste cuadrático, que se define como

$$C \equiv \frac{1}{2N} \sum_{i=1}^{N} \|f(x_i) - F(x_i)\|^2, \tag{5.5}$$

donde $\|\cdot\|$ representa la norma euclídea en $\mathbb{R}^m$:

$$\|y\| \equiv \sqrt{y_1^2 + \cdots + y_m^2} \quad \text{para } y = (y_1, \ldots, y_m) \in \mathbb{R}^m. \tag{5.6}$$

Esta función decrece conforme F va aproximando mejor f. Además, definiendo $C_x \equiv \frac{1}{2N}\|f(X) - F(x)\|^2$, tendremos

$$C = \sum_{i=1}^{N} C_{x_i}. \tag{5.7}$$

Descenso de gradiente

Para obtener el valor mínimo del funcional de error de f, el método de descenso del gradiente propone comenzar con un punto $x_0 \in \mathbb{R}^n$, calcular el valor de $\vec{\nabla} f(x_0)$, y con él calcular un nuevo punto $x_1 \equiv x_0 - \lambda \vec{\nabla} f(x_0)$, donde a $\lambda > 0$ la conocemos como tasa de aprendizaje. Cuando iteramos este proceso, creando una secuencia $\{x_i\}$ que dependerá de nuestra elección inicial de x_0. Esta estrategia enfrenta algunos problemas:

- El mínimo local obtenido no tiene por qué ser el mínimo global de la función, que es el que verdaderamente nos interesa.
- Tenemos que realizar pasos discretos, que variarán en longitud en función de la tasa de aprendizaje. Esto puede provocar que oscilemos en torno a un mínimo local o que nos lo saltemos.

A pesar de estos problemas, este método ha demostrado ser muy eficaz en multitud de aplicaciones, y se ha convertido en el método más utilizado para entrenar redes neuronales prealimentadas, y en otros algoritmos de aprendizaje automático.

Haciendo uso de que nuestra función de coste indica la calidad de la aproximación realizada por la red neuronal a la función dada, a través de calcular el gradiente de la función de coste con respecto a los pesos y tendencias de la red, y ajustando esos parámetros en la dirección opuesta al gradiente, lograremos reducir el error y, por tanto, acercarnos a una red que nos proporcione, en general, mejores resultados.

Propagación hacia atrás

El mayor problema a la hora de aplicar el algoritmo de descenso de gradiente a redes neuronales es el cálculo de las derivadas parciales a la función de coste con respecto a cada peso y

tendencia individuales (es decir, $\frac{\partial C}{\partial w^l_{i,j}}$ y $\frac{\partial C}{\partial b^l_{i,j}}$). Aquí es donde se hace uso de la propagación hacia atrás: este algoritmo nos indica cómo calcular esos valores para la última capa de conexiones, de tal manera que podamos calcular las derivadas parciales de cada capa, procediendo hacia atrás, hasta llegar a la primera capa de la red. Por ahora, consideremos que disponemos de una función de coste que compara con un conjunto de datos de entrenamiento compuesto de un único dato etiquetado x.

Haciendo uso de la propiedad de la Ecuación 5.7, dado que conocemos el valor de $\vec{\nabla} C_x$ para un punto etiquetado x, podremos escribir

$$\vec{\nabla} C = \vec{\nabla} \left(\sum_{i=1}^{N} C_{x_i} \right) = \sum_{i=1}^{N} \vec{\nabla} C_{x_i}, \tag{5.8}$$

lo que significa que podemos realizar este proceso para cada dato y, posteriormente, añadirlos al gradiente a través de una suma. Esto es relevante porque usar propagación hacia atrás en un conjunto grande de datos para cada etapa del aprendizaje es muy costoso computacionalmente. En su lugar, podemos seleccionar algunos elementos del conjunto de entrenamiento, calcular el gradiente, actualizar la red y repetir el proceso hasta que la red llegue a resultados satisfactorios.

Nos será útil considerar qué valores son enviados, en cada capa, por la capa anterior antes de que la función de activación fuese aplicada. Consideremos

$$z^l_j \equiv \sum_k w^l_{j,k} a^{l-1}_k + b^l_j, \quad \text{de tal manera que} \quad a^l_j = \sigma\left(z^l_j\right) \quad \text{y} \quad A^l = \sigma\left(Z^l\right). \tag{5.9}$$

En la ecuación anterior, $Z^l \equiv \sum_k z^l_k e_k$ es un vector con entradas correspondientes a los valores z^l_j, siendo e_i los vectores de la base. Además, consideremos las cantidades:

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j} \quad \text{y} \quad \Delta^l \equiv \sum_k \delta^l_k e_k. \tag{5.10}$$

Estos valores serán muy útiles para propagar el algoritmo hacia atrás a través de la red, y están directamente relacionados con las derivadas parciales $\frac{\partial C}{\partial w^l_{i,j}}$ y $\frac{\partial C}{\partial b^l_{i,j}}$, a través de la regla de la cadena, de la siguiente manera:

$$\frac{\partial C}{\partial w^l_{i,j}} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{i,j}} = \delta^l_j a^{l-1}_i \quad \text{y} \quad \frac{\partial C}{\partial b^l_j} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial b^l_j} = \delta^l_j. \tag{5.11}$$

Dado que conocemos a^{l-1}_j para cada nodo de la red, si somos capaces de calcular el valor de cada δ^l_j, seremos capaces de obtener el gradiente buscado. El primer paso consistirá en calcular este valor para la última capa de la red, es decir, δ^L_j para una red con L capas. No es difícil darnos cuenta de que, dado que $a^L_j = \sigma(z^L_j)$, por la regla de la cadena tendremos

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} \frac{\partial a^L_j}{\partial z^l_j} = \frac{\partial C}{\partial a^L_j} \sigma'\left(z^L_j\right). \tag{5.12}$$

Usando la función del error medio cuadrático, observando que $F(x) = A^L = (a^L_1, \ldots, a^L_k)$ y escribiendo $f(x) = (y_1, \ldots, y_k)$, obtenemos que

$$\delta^L_j = \left(a^L_j - y_j\right) \sigma'\left(z^L_j\right), \tag{5.13}$$

que puede ser fácilmente calculado por un ordenador si sabemos cómo calcular σ', tarea que debería ser sencilla para cualquier función de activación práctica. Es común escribir este resultado en forma vectorial, para lo que se suele usar la notación

$$\Delta^L = \nabla_{A^L} \odot \sigma'(Z^L), \tag{5.14}$$

donde $\nabla_{A^L} \equiv \left(\frac{\partial C}{\partial a_1^L}, \ldots, \frac{\partial C}{\partial a_k^L}\right)$ es el gradiente de C tomado respecto a los elementos de A^L, y $\odot$ es el producto de Hadamard, que multiplica dos matrices elemento a elemento, de la forma

$$\begin{bmatrix} a_{1,1} & \ldots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{j,1} & \ldots & a_{j,k} \end{bmatrix} \odot \begin{bmatrix} b_{1,1} & \ldots & b_{1,k} \\ \vdots & \ddots & \vdots \\ b_{j,1} & \ldots & b_{j,k} \end{bmatrix} \equiv \begin{bmatrix} a_{1,1}b_{1,1} & \ldots & a_{1,k}b_{1,k} \\ \vdots & \ddots & \vdots \\ a_{j,1}b_{j,1} & \ldots & a_{j,k}b_{j,k} \end{bmatrix}. \tag{5.15}$$

Ahora, propagamos hacia detrás en la red, con el objetivo de conseguir δ_j^{L-1}. Para lograrlo, aplicamos de nuevo la regla de la cadena:

$$\delta_j^{L-1} = \frac{\partial C}{\partial z_j^{L-1}} = \nabla_{Z^L} C \cdot \frac{\partial Z^L}{\partial z_j^{L-1}} = \sum_i^k \frac{\partial C}{\partial z_i^L} \frac{\partial z_i^L}{\partial z_j^{L-1}} = \sum_i^k \delta_i^L \frac{\partial z_i^L}{\partial z_j^{L-1}}. \tag{5.16}$$

Centrándonos en el término $\frac{\partial z_i^L}{\partial z_j^{L-1}}$, observamos que

$$\begin{aligned} \frac{\partial z_i^L}{\partial z_j^{L-1}} &= \frac{\partial \left(\sum_k w_{i,k}^L a_k^{L-1} + b_i^L\right)}{\partial z_j^{L-1}} = \frac{\partial \left(\sum_k w_{i,k}^L \sigma\left(z_k^{L-1}\right)^+ b_i^L\right)}{\partial z_j^{L-1}} = \\ &= \frac{\partial \left(w_{i,j}^L \sigma\left(z_j^{L-1}\right)\right)}{\partial z_j^{L-1}} = w_{i,j}^L \sigma'\left(z_j^{L-1}\right), \end{aligned} \tag{5.17}$$

que puede ser fácilmente calculado por un ordenador. Por tanto, obtenemos finalmente

$$\delta_j^{L-1} = \sum_i^k \delta_i^L w_{i,j}^L \sigma'\left(z_j^{L-1}\right). \tag{5.18}$$

Esta expresión nos dice cómo calcular cualquier δ_j^l en la red, conocida Δ^{l+1}. Dado que la Ecuación 5.14 indica cómo inicializar Δ^L en la última capa de la red, el algoritmo queda completo.

5.1.3. Aproximación universal

Teoremas del análisis funcional

Muchas de las demostraciones relacionadas con la aproximación universal se basan en resultados del análisis funcional. Sus demostraciones están estandarizadas y se pueden encontrar en multitud de manuales de análisis funcional (por ejemplo, [14]). Nosotros nos limitaremos a enunciar los teoremas.

Teorema 5.1.1 (Teorema de convergencia dominada de Lebesgue)**.** *Sea X un espacio de medida, μ una medida de Borel en X, $g : X \to \mathbb{R}$ de tipo L^1 y $\{f_n\}$ una secuencia de funciones*

medibles de $X \to \mathbb{R}$ *tales que* $\|f_n(x)\| \leq g(x)$ *para todo* $x \in X$ *y* $\{f_n\}$ *converja puntualmente a una función* f*. Entonces,* f *es integrable y*

$$\lim_{n\to\infty} \int f_n(x)\, d\mu(x) = \int f(x)\, d\mu(x). \tag{5.19}$$

Teorema 5.1.2 (Teorema de Hahn-Banach - Forma geométrica)**.** *Sea* V *un espacio vectorial normado y* A*,* $B \subset V$ *dos subconjuntos no vacíos, cerrados, disjuntos y convexos tales que uno de ellos es compacto. Entonces, existe un funcional lineal y continuo* $f \not\equiv 0$*, algún* $a \in \mathbb{R}$ *y un* $\epsilon > 0$ *tales que* $f(x) \leq \alpha - \epsilon$ *para algún* $x \in A$ *y* $f(y) \geq \alpha + \epsilon$ *para algún* $y \in B$*.*

De este teorema, se deriva el siguiente corolario.

Corolario 5.1.2.1. *Sea* V *un espacio vectorial normado sobre* $\mathbb{R}$ *y* $U \subset V$ *un subespacio lineal tal que* $\overline{U} \neq V$*. Entonces, existe una aplicación lineal* $f : V \to \mathbb{R}$ *con* $f(x) = 0$ *para algún* $x \in U$*, y* $f \not\equiv 0$*.*

Definición 5.1.1. Dado un espacio topológico Ω, definimos

$$C(\Omega) \equiv \{f : \Omega \to \mathbb{R} \text{ tales que } f \text{ es continua}\}. \tag{5.20}$$

Teorema 5.1.3 (Teorema de representación de Riesz)**.** *Sea* Ω *un subespacio de* $\mathbb{R}^n$ *y* $F : C(\Omega) \to \mathbb{R}$ *un funcional lineal en el espacio de las funciones continuas reales con dominio en* Ω*. Entonces, existe una medida signada de Borel* μ *en* Ω *tal que para cualquier* $f \in C(\omega)$ *tenemos que*

$$F(f) = \int_\Omega f(x)\, d\mu(x). \tag{5.21}$$

Una vez expuestos estos resultados preliminares, realicemos algunas definiciones necesarias para tratar el problema de la aproximación universal en redes neuronales.

Definición 5.1.2. Dada una función de activación $f : \mathbb{R} \to \mathbb{R}$, se define el conjunto:

$$\Sigma_n(f) = \text{span}\{f(y \cdot x + \theta) : y \in \mathbb{R}^n, \theta \in \mathbb{R}\}, \tag{5.22}$$

donde $y \cdot x$ representa el producto escalar en $\mathbb{R}^n$.

El conjunto $\Sigma_n(f)$ contiene todas las funciones que pueden ser calculadas por una red neuronal con una única capa oculta y una función de activación f.

Definición 5.1.3 (Aproximador universal)**.** Sea Ω un espacio topológico y $f : \mathbb{R} \to \mathbb{R}$. Decimos que una red neuronal con función de activación f es un aproximador universal en Ω si $\Sigma_n(f)$ es denso en $C(\Omega)$, el subconjunto de funciones continuas de Ω en $\mathbb{R}$.

Definición 5.1.4 (Función de activación n-discriminatoria)**.** Sea n un número natural. Decimos que una función de activación $f : \mathbb{R} \to \mathbb{R}$ es n-discriminatoria si la única medida signada de Borel μ tal que

$$\int f(y \cdot x + \theta)\, d\mu(x) = 0 \text{ para todo } y \in \mathbb{R}^n, \text{ y } \theta \in \mathbb{R} \tag{5.23}$$

es la medida cero.

Definición 5.1.5 (Función de activación discriminatoria)**.** Decimos que una función de activación $f : \mathbb{R} \to \mathbb{R}$ es discriminatoria si es n-discriminatoria para cualquier $n \in \mathbb{N}$.

Definición 5.1.6. A una función $f : \mathbb{R} \to \mathbb{R}$ se la denomina sigmoide si satisface las siguientes dos propiedades:

$$\lim_{x \to +\infty} f(x) = 1 \text{ y } \lim_{x \to -\infty} f(x) = 0. \tag{5.24}$$

Definición 5.1.7. Sea n un número natural. Definimos:

$$I_n \equiv [0,1]^n = \{x = (x_1, \ldots, x_n) \in \mathbb{R}^n : x_i \in [0,1] \text{ para cualquier } i = 1, \ldots, n\}. \tag{5.25}$$

Definición 5.1.8 (Función rectificadora lineal ($ReLU$))**.** La función rectificadora lineal es una función $\mathbb{R} \to \mathbb{R}$ definida por:

$$\text{ReLU}(x) \equiv \text{máx}(0, x). \tag{5.26}$$

Teorema de Aproximación Universal para funciones de activación Sigmoides o ReLU

Un artículo publicado por G. Cybenko en 1989, titulado "Aproximación por Superposición de Funciones Sigmoideas" [15] dio comienzo a una serie de trabajos a lo largo de los próximos años, que trataban de determinar qué funciones de activación conducían a la propiedad de aproximación universal. En dicho artículo, Cybenko demostró que las funciones sigmoides continuas cumplen tal propiedad, a través de lo que en este trabajo presentamos como el Teorema 5.1.4 y el Lema 5.1.5.

Teorema 5.1.4. *Sea f una función discriminatoria continua. Entonces, una red neuronal con f como función de activación es un aproximador universal.*

Demostración. Demostremos el teorema por reducción al absurdo. Supongamos que $\Sigma_n(f)$ no es densa en $C(I_n)$, de lo que se sigue que $\overline{\Sigma_n(f)} \neq C(I_n)$. Aplicando aquí el corolario del teorema de Hahn-Banach (Corolario 5.1.2.1), se concluye que existe algún funcional lineal continuo $F : C(I_n) \to \mathbb{R}$ tal que $F \neq 0$ siempre que no tengamos $g \in \overline{\Sigma_n(f)}$, caso en el que tenemos $F(g) = 0$. Por el Teorema de Representación de Riesz (Teorema 5.1.3), existirá alguna medida de Borel μ tal que

$$F(g) = \int_{I_n} g(x)\, d\mu(x)\, \forall g \in C(I_n). \tag{5.27}$$

Ahora bien, dado que para cualesquiera y y θ la función $f(y \cdot x + \theta)$ es un elemento de $\overline{\Sigma_n(f)}$, esto significa que para todo $y \in \mathbb{R}^n$, y para $\theta \in \mathbb{R}$, se tiene $\int f(y \cdot x + \theta)\, d\mu(x) = 0$, lo que, dado que f es discriminatoria, significa que $\mu = 0$ y, por tanto, que $F(g) = 0$ para cualquier $g \in C(I_n)$. Esto contradice el Corolario 5.1.2.1, con lo que queda demostrado el teorema. □

Lema 5.1.5. Todas las funciones de Borel sigmoides, acotadas y medibles son discriminatorias.

Demostración. Sea f una función de Borel sigmoide, acotada y medible, y asumamos que, para una medida $\mu \in M(I_n)$ dada, tenemos que

$$\int_{I_n} f(y \cdot x + \theta)\, d\mu(x) = 0\, \forall y \in \mathbb{R}^n, \quad \forall \theta \in \mathbb{R}. \tag{5.28}$$

Nuestro objetivo es mostrar que $\mu = 0$. Para tal fin, consideremos una función γ, obtenida a través de fijar $\gamma, \phi \in \mathbb{R}$ y tomando el límite

$$\gamma(x) = \lim_{\lambda \to \infty} f\left(\lambda(y \cdot x + \theta) + \phi\right) = \begin{cases} 1, & \text{si } y \cdot x + \theta > 0 \\ f(\phi), & \text{si } y \cdot x + \theta = 0 \\ 0, & \text{si } y \cdot x + \theta < 0 \end{cases}. \tag{5.29}$$

Por el Teorema de Convergencia Dominada de Lebesgue (Teorema 5.1.1), tenemos que

$$\int_{I_n} \gamma(x)\, d\mu(x) = \lim_{\lambda \to \infty} \int_{I_n} f\left(\lambda(y \cdot x + \theta) + \phi\right)\, d\mu(x) = 0. \tag{5.30}$$

De lo que derivamos que

$$\begin{aligned} \int_{I_n} \gamma(x)\, d\mu(x) &= \int_{H^+_{y,s}} 1\, d\mu(x) + \int_{\Pi,\beta} f(\phi)\, d\mu(x) + \int_{H^-_{\gamma,\theta}} 0\, d\mu(x) = \\ &= \mu\left(H^+_{y,\theta}\right) + f(\phi)\mu\left(\Pi_{y,\theta}\right) = 0, \end{aligned} \tag{5.31}$$

donde

$$\begin{aligned} H^+_{y,\theta} &\equiv \{x \in I_n \mid y \cdot x + \theta > 0\}, \\ \Pi_{y,\theta} &\equiv \{x \in I_n \mid y \cdot x + \theta = 0\} \text{ y} \\ H^-_{y,\theta} &\equiv \{x \in I_n \mid y \cdot x + \theta < 0\}. \end{aligned} \tag{5.32}$$

Esto es cierto para cualesquiera y, θ. Dado que, además, es cierto para cualquiera ϕ, y $f(\phi) \to 1$ cuando $\phi \to \infty$, se obtiene que

$$\mu\left(H^+_{y,\theta}\right) + \mu\left(\Pi_{y,\theta}\right) = 0. \tag{5.33}$$

De una manera similar, si hacemos $\phi \to -\infty$, vemos que $f(\phi) \to 0$ y

$$\mu\left(H^+_{y,\theta}\right) = 0. \tag{5.34}$$

Consideremos ahora el funcional $F : L^\infty(\mathrm{R}) \to \mathrm{R}$, definido como:

$$F(h) \equiv \int_{I_n} h(y \cdot x)\, d\mu(x). \tag{5.35}$$

Dado que $L^\infty(\mathbb{R})$ es el espacio de las funciones acotadas, esta integral (y, por tanto, el funcional que define) está bien definida para cualesquiera $h \in L^\infty(\mathbb{R})$. Si tomamos $\mathbf{1}_{[\theta,\infty)}$ como la función indicadora del intervalo $[\theta, \infty)$, se obtiene

$$F\left(\mathbf{1}_{[\theta,\infty)}\right) = \int_{I_n} \mathbf{1}_{[\theta,\infty)}(y \cdot x)\, d\mu(x) = \mu\left(H^+_{y,\theta}\right) + \mu\left(I_{y,\theta}\right) = 0. \tag{5.36}$$

De manera similar, para cualquier intervalo abierto (θ, ∞), tenemos

$$F\left(\mathbf{1}_{(\theta,\infty)}\right) = \int_{I_x} \mathbf{1}_{(\theta,\infty)}(y + x) d\mu(x) = \mu\left(H^+_{y,\theta}\right) = 0. \tag{5.37}$$

Por linealidad, se obtiene que $F(h) = 0$ para la función indicadora h de cualquier intervalo. Se sigue que $F(h) = 0$ para cualquier función simple. Usando que las funciones simples son densas en $L^\infty(\mathbb{R})$, obtenemos que $F = 0$. Además, dado que las funciones seno y coseno son elementos de $L^\infty(\mathbb{R})$, podemos escribir

$$F(\cos) + iF(\text{sen}) = \int_{I_n} (\cos(y \cdot x) + i \,\text{sen}(y \cdot x))\; d\mu = \int_{I_n} \exp(iy \cdot x)\; d\mu(x) = 0. \tag{5.38}$$

Esto es cierto para todo $y \in \mathbb{R}^n$, lo que significa que la transformada de Fourier de μ es 0. Esto solo es posible si $\mu = 0$, por lo que queda completada la demostración. □

Con esto, hemos demostrado que la propiedad de aproximación universal es cierta para redes con funciones sigmoides continuas. Sin embargo, existen multitud de algoritmos que utilizan funciones de activación ReLU. Demostraremos que la función ReLU es 1-discriminatoria, y posteriormente usaremos el Lema 5.1.7 para mostrar que esto conlleva la propiedad de aproximación universal para funciones con entradas de dimensión finita.

Lema 5.1.6. La función ReLU es 1-discriminatoria.

Demostración. Sea μ una medida de Borel signada, y asumamos que, para todo $y \in \mathbb{R}$ y $\theta \in \mathbb{R}$, se cumple

$$\text{ReLU}(y \cdot x + \theta)\, d\mu(x) = 0. \tag{5.39}$$

Buscamos mostrar que $\mu = 0$. Para eso, construiremos una función sigmoidea acotada, continua (y por tanto medible de Borel) a partir de substraer dos funciones ReLU con diferentes parámetros. En particular, consideramos la función

$$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \in [0,1]. \\ 1 & \text{si } x > 1 \end{cases} \tag{5.40}$$

Por tanto, cualquier función del tipo $g(x) = f(yx+\theta)$ con y no nula puede describirse como

$$g(x) = \text{ReLU}(yx + \theta_1) - \text{ReLU}(yx + \theta_2) \tag{5.41}$$

definiendo $\theta_1 = -\theta/y$ y $\theta_2 = (1-\theta)/y$. Si $y = 0$, en su lugar tendremos

$$g(x) = f(\theta) = \begin{cases} \text{ReLU}(f(\theta)) & \text{si } f(\theta) \geq 0 \\ -\text{ReLU}(-f(\theta)) & \text{si } f(\theta) \leq 0 \end{cases}, \tag{5.42}$$

lo que significa que, para cualesquiera $y \in \mathbb{R}, \theta \in \mathbb{R}$

$$\begin{aligned} \int f(yx+\theta) d\mu(x) &= \int (\text{ReLU}(yx + \theta_1) - \text{ReLU}(yx + \theta_2))\, d\mu(x) = \\ &= \int \text{ReLU}(yx + \theta_1)\, d\mu(x) - \int \text{ReLU}(yx + \theta_2)\, d\mu(x) = 0 - 0 = 0. \end{aligned} \tag{5.43}$$

Y, por el Lema 5.1.5, f es discriminatoria, por lo que $\mu = 0$. □

Lema 5.1.7. Si $\Sigma_1(f)$ es denso en $C([0,1])$, entonces $\Sigma_n(f)$ es denso en $C([0,1]^n)$.

Demostración. Usemos que el sistema generador del conjunto $g(a \cdot x) : a \in \mathbb{R}^n, g \in C([0,1])$ es denso en $C([0,1]^n)$. Esto significa que, dada cualquier función $h \in C([0,1]^n)$ y $\epsilon > 0$, existen funciones g_k en $C([0,1])$ tales que

$$\left| h(x) - \sum_{k=1}^{N} g_k \left(a_k \cdot x \right) \right| < \frac{\varepsilon}{2}. \tag{5.44}$$

Si ahora examinamos cada función $g_k \left(a_k \cdot x \right)$, y usamos la suposición de que $\Sigma_1(f)$ es densa en $C([0,1])$, llegamos a la conclusión de que para cada una de esas funciones, existe una suma de funciones que cumple

$$\left| g_k \left(a_k \cdot x \right) - \Sigma_{i=1}^{N_k} f \left(y_{k,i} \cdot x + \theta_{k,i} \right) \right| < \frac{\varepsilon}{2k}. \tag{5.45}$$

A través de la aplicación de la desigualdad triangular, llegamos a que

$$\begin{aligned} \left| h(x) - \sum_{k=1}^{N} \sum_{i=1}^{N_k} f \left(y_{k,i} \cdot x + \theta_{k,i} \right) \right| < \left| h(x) - \sum_{k=1}^{N} g_k \left(a_k \cdot x \right) \right| + k(\epsilon/2k) < \\ < \epsilon/2 + \epsilon/2 = \epsilon. \end{aligned} \tag{5.46}$$

Es decir, podemos estar arbitrariamente cerca de cualquier función en $C\left([0,1]^n\right)$ a través de usar funciones en $\Sigma_n(f)$. □

Como consecuencia directa del Teorema 5.1.4 y de los lemas anteriores, llegamos al resultado buscado:

Corolario 5.1.7.1 (Teorema de Aproximación Universal)**.** *Las redes neuronales con funciones de activación ReLU o sigmoides continuas son aproximadores universales.*

Con esto, se ha demostrado que una red neuronal con una única cada oculta y un número de nodos lo suficientemente alto es capaz de aproximar cualquier función continua. Sin embargo, este diseño difiere drásticamente de las arquitecturas empleadas para redes neuronales, en las cuales disponemos de multitud de capas. Se puede demostrar que una red de anchura acotada mantiene la propiedad de aproximación universal, a expensas de necesitar una capa más profunda:

Teorema 5.1.8. *Sea $f : [0,1]^d \to [0,1]$ la función calculada por una red neuronal con función de activación ReLU de una única capa con dimensión de entrada d, dimensión de salida 1 y n nodos ocultos. Entonces, existe otra red neuronal con función de activación ReLU, con n capas ocultas y anchura $d+2$, que calcula la misma función f.*

Dado que la demostración de este Teorema tiene escasa relevancia práctica para la construcción de redes neuronales, presentamos únicamente el resultado.

5.2. Algoritmos en Aprendizaje Profundo

Las redes neuronales son una técnica en aprendizaje automático, que toma como inspiración el sistema nervioso humano y la estructura del cerebro. Están formadas por unidades de

procesamiento organizadas en capas de entrada, capas ocultas y capas de salida. Estos nodos o unidades en cada capa están conectadas a los nodos de las capas adyacentes, y cada conexión tiene asignado un número al que llamaremos peso. Las entradas se multiplican por sus pesos respectivos y se suman en cada unidad. La suma, posteriormente, es transformada por una función de transformación, que usualmente es una función sigmoide, una tangente hiperbólica o una unidad ReLU. Estas funciones se usan por tener derivadas sencillas, lo que facilita el cálculo de las derivadas parciales del error respecto a cada peso. Las funciones sigmoide y tangente hiperbólica también aplastan la entrada, haciendo que la salida pueda tomar como valores $\{0, 1\}$ ó ± 1. Además, implementan una no linealidad saturada a medida que las salidas se estabilizan o se saturan antes o después de los umbrales respectivos. Por otro lado, ReLU muestra comportamientos tanto saturados como no saturado. La salida de la función es, posteriormente, devuelta como entrada en la unidad consecuente de la siguiente capa. El resultado de la salida final se toma como solución del problema.

La implementación de una red neuronal consta de los siguientes pasos:

1. Adquirir datos para el entrenamiento y la comprobación del modelo.
2. Entrenar la red con dichos datos.
3. Realizar predicciones con los datos de comprobación.

En esta sección nos centraremos en las redes neuronales que pueden ser de utilidad para problemas de regresión y de predicción de series temporales, así como los algoritmos que permiten su implementación.

5.2.1. Clasificación, implementación y entrenamiento de redes neuronales

Las redes neuronales pueden clasificarse en distintas tipologías, entre las que se incluyen:

- Redes neuronales prealimentadas (*Feedforward Neural Network, FNN*)
- Redes neuronales recurrentes (*Recurrent Neural Network, RNN*)
- Redes neuronales de base radial (*Radial Basis Function Neural Network*)
- Redes neuronales autoorganizadas de Kohonen (*Kohonen Self Organizing Neural Network*)
- Redes neuronales modulares (*Modular Neural Network*)

Ya hemos tratado las características de las redes neuronales prealimentadas en la Sección 5.1. De los restantes tipos de redes neuronales, solo utilizamos las recurrentes. En dichas redes, las unidades de procesamiento pueden formar un ciclo, es decir, la salida de una capa puede introducirse como entrada de la siguiente, mientras que la salida de la capa se convierte en una entrada de si misma, formando un ciclo de retroalimentación. Esto permite a la red tener memoria sobre estados pasados, y usar esa memoria para influir sobre las siguientes salidas. La consecuencia de esto es que, a diferencia de lo que sucede en una red neuronal prealimentada,

una RNN puede tomar una secuencia de entradas y generar una secuencia de valores de salida. En la Figura 5.2 se muestra las iteraciones de una RNN a lo largo del tiempo. En esa Figura, x_t representa la entrada en tiempo t; U, V y W son los parámetros aprendidos que comparten todos los pasos; O_t es la salida a tiempo t; S_t representa el estado a tiempo t y su expresión es:

$$S_t = f(Ux_t + Ws_{t-1}), \tag{5.47}$$

donde f es la función de activación.

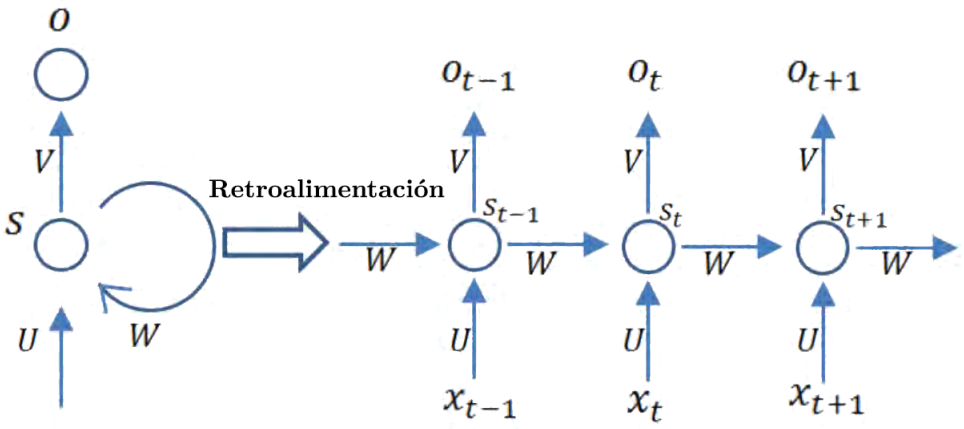


Figura 5.2: Ejemplo de red neuronal recurrente [12].

Las DNN se enfrentan a dos grandes problemas a la hora de entrenarse: la convergencia prematura y el sobreajuste (*overfitting*).

Convergencia prematura: Ocurre cuando los pesos y la tendencia de la DNN se mantiene en un estado que es solo óptimo local, y deja de acercarse al óptimo global del espacio multidimensional.

Sobreajuste: Es el estado en el que la DNN se vuelve demasiado rígido, dando muy buenos para los datos de entrenamiento, pero siendo incapaz de describir correctamente los datos de prueba.

Se han desarrollado distintas estructuras y librerías para facilitar la creación de modelos en *machine learning*. Estas herramientas contienen las funciones matemáticas, los algoritmos de entrenamiento y la modelización estadística necesarias para entrenar modelos, sin la necesidad de que el usuario las programe. Algunas de estas herramientas proporcionan capacidades de procesamiento distribuido y en paralelo, y diversas herramientas de desarrollo. En nuestro caso, hacemos uso de Keras [11], por su simplicidad de uso, por su modularidad y flexibilidad (es compatible con backends como TensorFlow, Theano y CNTK), por estar muy bien documentado y por tener una buena portabilidad.

5.2.2. Arquitecturas para el aprendizaje profundo

Las redes neuronales profundas constan de varias capas de nodos. Se han desarrollado arquitecturas especializadas en la resolución de determinados tipos de problemas: por ejemplo,

las CNN se usan especialmente en la visión computarizada y en el reconocimiento de imágenes, y las RNN se usan para problemas de series temporales y predicción. Por otro lado, algunos problemas pueden requerir distintos enfoques dependiendo de sus características particulares, como es el caso de los problemas de clasificación.

En nuestro trabajo, aplicamos los siguientes modelos:

- Una red RNN simple, cuyas características ya hemos tratado.
- Una red de memoria de largo a corto plazo (*Long-Short Term Memory*, LSTM), que es una implementación de las redes RNN que presenta algunas ventajas frente a éstas.
- Una red perceptrón multicapa (*Multilayer Perceptron*, MLP), una implementación de las redes prealimentadas con sus capas completamente conectadas.
- Una red CNN multicanal que, a diferencia de las redes CNN monocanal, además de trabajar con el dato a predecir (los desplazamientos), también hace uso de las demás columnas de la tabla de datos.
- Una red híbrida LSTM codificador-decodificador, que se diferencia de una red codificadora automática en que se implementa una red LSTM en el decodificador, permitiéndole así conocer predicciones pasadas y acumular estados.
- Una red híbrida CNN-LSTM codificador-decodificador, en la que la red CNN funciona como codificador.

Como ya hemos tratado las redes RNN y las MLP son redes FNN con sus capas completamente conectadas, tratemos las redes LSTM, las redes CNN y los codificadores automáticos.

Memoria de largo a corto plazo (*Long Short-Term Memory, LSTM*)

Las LSTM son implementaciones de las redes neuronales recurrentes (RNN), y su principal característica es que son capaces de retener información de anteriores estados, lo que las hace aptas en problemas que requieren memoria, como las plataformas de reconocimiento de voz. Por otro lado, las LSTM resuelven una de las problemáticas de las RNN: el problema de que los gradientes se desvanezcan, a través de permitir que los gradientes pasen de estado en estado sin alterarse.

Como se puede ver en la Figura 5.3, las LSTM consisten en bloques de células de memoria a través de las cuales fluye una señal regulada por las puertas de entrada, olvido y salida. Estas puertas controlan lo que se almacena, lee y escribe en la celda.

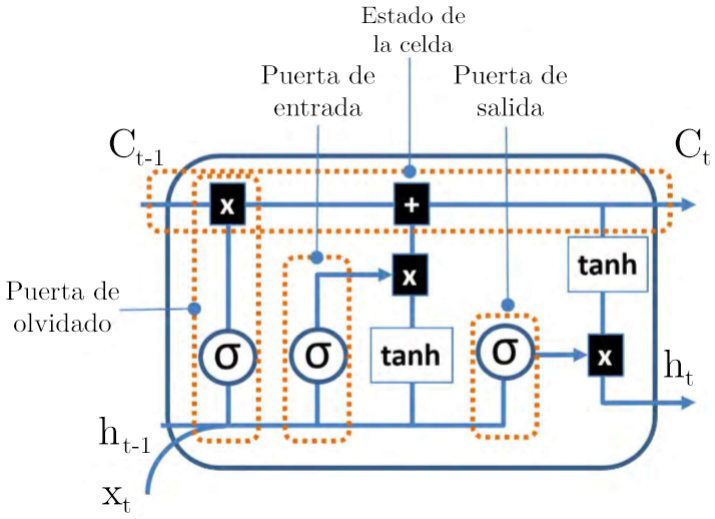


Figura 5.3: Puertas y células de memoria de un bloque LSTM [12].

En la Figura 5.3, C hace referencia a las celdas, mientras que x y h son los valores de entrada y de salida, respectivamente. El subíndice t denota el paso temporal en el que nos encontramos, y $t-1$ representa el bloque LSTM a tiempo $t-1$. Los operadores $+$ y $\times$ hacen referencia a la suma y al producto elemento a elemento, respectivamente. El símbolo σ representa la función sigmoidea; y tanh, la función tangente hiperbólica:

$$\sigma(t) = \frac{1}{1+\exp(-t)} \tag{5.48}$$

$$\tanh(t) = \frac{\exp(t)-\exp(-t)}{\exp(t)+\exp(-t)} \tag{5.49}$$

Para cada tiempo t, los valores de los vectores en las puertas de olvidado (f), de entrada (i) y de salida (o) son:

$$f_t = \sigma\left(W_f x_t + w_f h_{t-1} + b_f\right) \tag{5.50}$$

$$i_t = \sigma\left(W_i x_t + w_i h_{t-1} + b_i\right) \tag{5.51}$$

$$o_t = \sigma\left(W_o x_t + w_o h_{t-1} + b_o\right) \tag{5.52}$$

donde W, w y b representan los pesos de la entrada, los pesos de las salidas recurrentes y las tendencias, respectivamente. La función h_t es, en cada paso de tiempo:

$$h_t = o_t \cdot \sigma_h\left(c_t\right) \tag{5.53}$$

donde $\cdot$ denota el producto elemento a elemento (el producto escalar) y c_t es:

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \sigma_c\left(W_c x_t + w_c h_{t-1} + b_c\right) \tag{5.54}$$

Los bloques de una LSTM mantienen información de estados anteriores: por su capacidad para trabajar con dependencias a largo plazo, son muy útiles para construir modelos que dependen del contexto y de estados anteriores. Las puertas de entrada, salida y olvido indican qué datos de la celda salen, cuáles se olvidan y qué datos nuevos entran en ella.

Redes neuronales convolucionales (*Convolutional Neural Network, CNN*)

Estas redes se inspiran en la corteza visual humana y se usan muy a menudo en problemas de reconocimiento de imágenes y de vídeo, en procesamiento de lenguaje natural, en investigación de medicamentos... Una red CNN consta de varias capas de convolución y submuestreo, seguidas de una capa totalmente conectada y otra de normalización (por ejemplo, mediante una función softmax). En la Figura 5.4 se muestra la arquitectura CNN de siete capas *LeNet-5*, desarrollada por LeCun et al. [16] para el reconocimiento de dígitos.

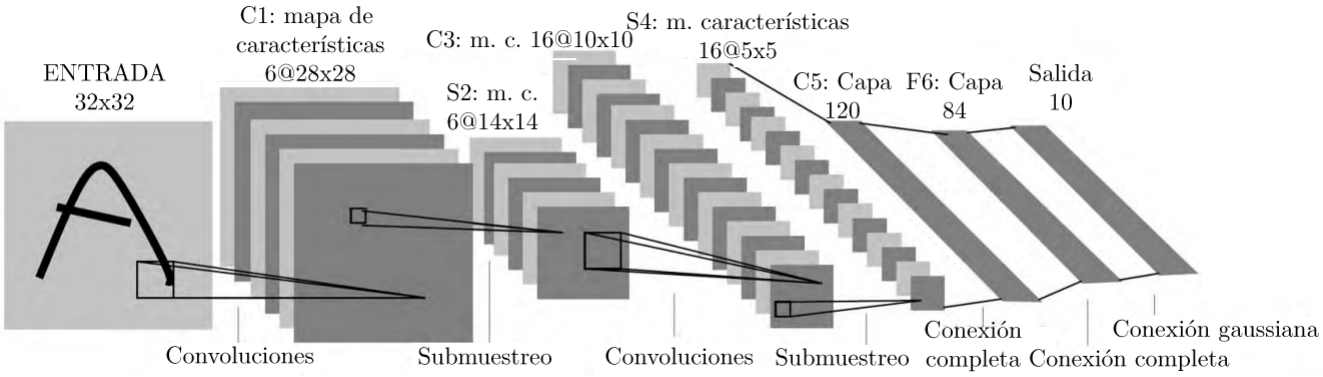


Figura 5.4: Arquitectura CNN de 7 capas para el reconocimiento de caracteres [12].

Las múltiples capas de convolución extraen progresivamente características de la entrada capa a capa, desde la capa de entrada a la capa de salida. Las capas totalmente conexas, que realizan la clasificación, preceden a las capas de convolución. Las capas de submuestreo suelen insertarse entre los pares de capas de convolución. Esta CNN toma como entrada una imagen de dos dimensiones, cuadrada y de n píxeles de lado. Cada capa consta de grupos de neuronas bidimensionales llamadas filtros o núcleos. A diferencia de lo que sucede en otras redes neuronales, las neuronas de cada capa de extracción de características no están conectadas a las neuronas de las capas adyacentes, si no que solamente están conectadas a las neuronas de la capa anterior, que están asociadas a su imagen de entrada o a un mapa de características. Esta región en la entrada se conoce como mapa receptivo local.

Cuanto menor sea el número de conexiones, menor será el tiempo de entrenamiento y la probabilidad de que aparezca sobreajuste. Todas las neuronas en un filtro están conectadas al mismo número de neuronas de la capa de entrada o el mapa de características anterior, y están constreñidas a tener la misma secuencia de pesos y tendencias. Estas características de la red aumentan la velocidad de aprendizaje y reducen las necesidades de memoria. Por tanto, cada neurona en un filtro específico busca el mismo patrón de características en distintas partes de la imagen de entrada. Las capas de submuestreo reducen el tamaño de la red. La capa de agrupación max/mean y los filtros de promediado local se usan a menudo para lograr submuestreos. Las capas finales de la CNN se ocupan de la clasificación, y las neuronas entre sus capas están completamente conectadas.

Una CNN profunda puede implementarse con varias capas de convolución con pesos compartidos y con capas de submuestreo. Dar a una CNN la característica de profundidad redunda

en que se mantenga la localidad y en que dispongamos de representaciones de alta calidad.

En la mayoría de las situaciones, la propagación hacia atrás se usa únicamente para entrenar todos los parámetros (pesos y tendencias) en la CNN. Expongamos el algoritmo de propagación hacia atrás:

La función coste, que mide la precisión con la que un modelo se ajusta a los datos de prueba, con respecto a un ejemplo de entrenamiento individual (x, y) en capas ocultas se define como:

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{w,b}(x) - y \right\|^2 . \tag{5.55}$$

Para el término de error δ para la capa l, tenemos:

$$\delta^{(l)} = \left(\left(W^{(l)} \right)^T \delta^{(l+1)} \right) \cdot f' \left(z^{(l)} \right), \tag{5.56}$$

donde $\delta^{(l+1)}$ es el error de la capa $(l+1)$-ésima de una red cuya función de coste es $J(W, b; x, y)$. $f'(z^{(l)})$, y $f'(z^l))$ representa la derivada de la función de activación. Por otro lado, tenemos que:

$$\nabla_{w^{(l)}} \cdot J(W, b; x, y) = \delta^{(l+1)} \left(a^{(l+1)} \right)^T, \text{ y que} \tag{5.57}$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)} \tag{5.58}$$

donde a es la entrada, tal que $a^{(1)}$ es la entrada para la primera capa (es decir, la imagen real) y $a^{(l)}$ es la entrada de la l-ésima capa. El error de la capa de submuestreo es:

$$\delta_k^{(l)} = \text{upsample} \left(\left(W_k^{(l)} \right)^T \delta_k^{(l+1)} \right) \cdot f' \left(z_k^{(l)} \right), \tag{5.59}$$

donde k representa el índice del filtro en la capa. En la capa de submuestreo, el error debe propagarse en la dirección contraria: por ejemplo, cuando se usa mean pooling, upsample distribuirá el error a las unidades de entrada anteriores. Finalmente, los gradientes de los mapas de características son:

$$\nabla_{w_k^{(l)}} J(W, b; x, y) = \sum_{i-1}^{m} \left(a_i^{(l)} \right) * \text{rot}\, 90 \left(\delta_k^{(l+1)}, 2 \right), \text{ y} \tag{5.60}$$

$$\nabla_{b_k^{(l)}} J(W, b; x, y) = \sum_{a,b} \left(\delta_k^{(l+1)} \right)_{a,b}, \tag{5.61}$$

donde $\left(a_i^{(l)} \right) * \delta_k^{(l+1)}$ representa la convolución entre el error y la i-ésima entrada en la l-ésima capa respecto al k-ésimo filtro.

Codificador automático

Los codificadores automáticos (*autoencoders*) son redes neuronales que usan algoritmos no supervisados para aprender la representación del conjunto de datos de entrada para realizar reducción de la dimensionalidad, con la finalidad de recrear el conjunto de datos de entrada. El

algoritmo de aprendizaje se basa en la implementación de la propagación hacia atrás.

Los codificadores automáticos extienden la idea del análisis de la componente principal (*Principal Component Analysis, PCA*). Como se observa en la Figura 5.5, una PCA toma datos multidimensionales y los transforma en una representación lineal. En particular, en la figura partimos de un dato de entrada bidimensional y se convierte en un vector lineal usando PCA.

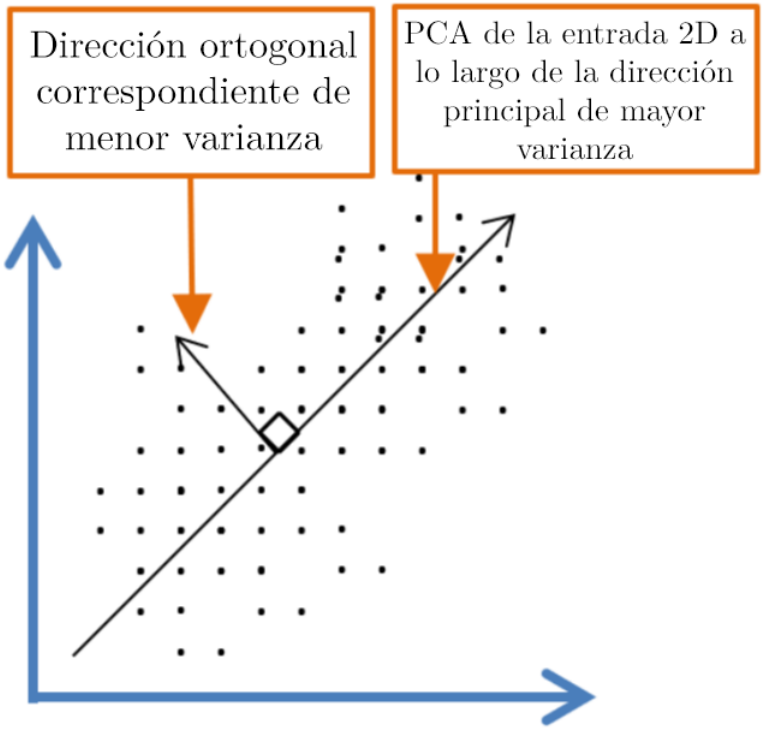


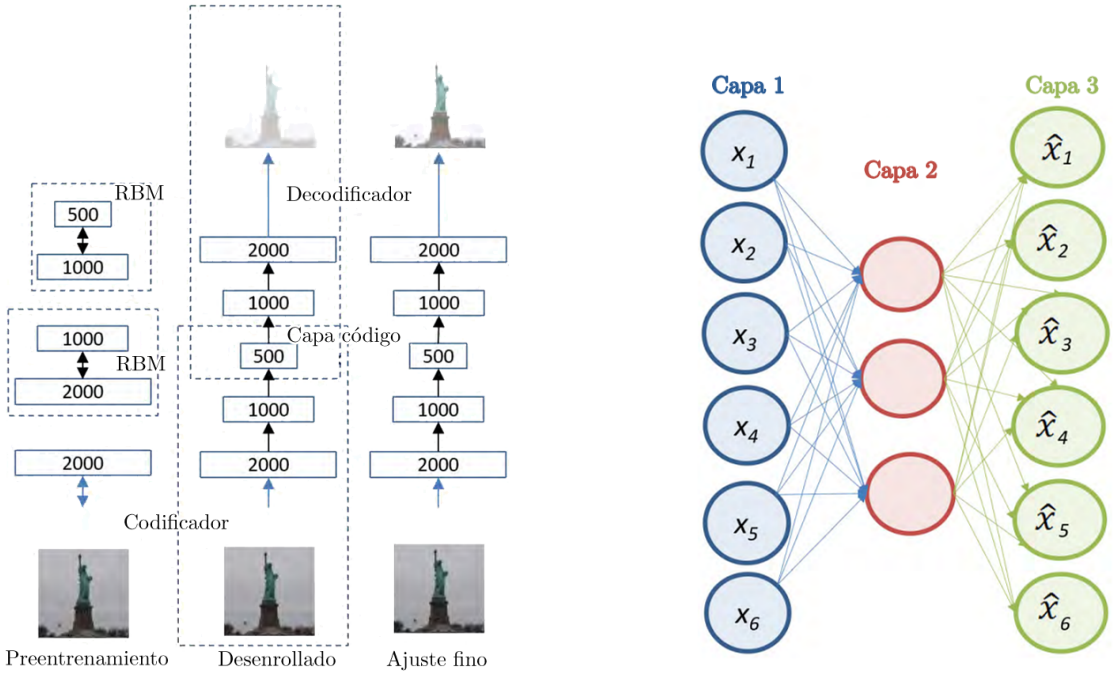
Figura 5.5: Representación lineal de un conjunto de datos bidimensional haciendo uso de una red PCA [12].

Un codificador automático puede ir más allá, produciendo representaciones no lineales. La PCA determina un conjunto de variables lineales en la dirección de mayor varianza. Los puntos p-dimensionales de los datos de entrada se representan como m direcciones ortogonales, con $m \leq p$, y se construye un espacio dimensional de dimensión menor que m. Los puntos de los datos originales se proyectan en la dirección principal, omitiendo por tanto información que corresponde a las direcciones ortogonales. Las PCAs se centran más en las varianzas que en las covarianzas y correlaciones: busca la función lineal de mayor varianza. El objetivo es determinar la dirección con menor error cuadrático medio, que tendrá, por tanto, el menor error de reconstrucción.

Los autocodificadores usan bloques de codificadores y decodificadores de capas ocultas no lineales para realizar una reducción de la dimensionalidad y posteriormente reconstruir los datos originales. Se usa un preentrenamiento no supervisado capa por capa conocido como greedy, y se realiza un ajuste usando propagación hacia atrás. A pesar de usar la propagación hacia atrás, que se usa generalmente en entrenamiento no supervisado, los autocodificadores se consideran

entrenamiento no supervisado, dado que regeneran la propia entrada $x^{(i)}$, es decir, tenemos $y^{(i)} = x^{(i)}$.

En la Figura 5.6a, se muestra un ejemplo de bloques detectores de características de RBMs compuestos de una sola capa que pueden ser usados en el preentrenamiento. Tras el entrenamiento, se realiza un desenrollado. Este desenrollado combina los grupos de RBMs para crear un bloque codificador, y entonces revierte el bloque codificador para crear una sección decodificadora. Finalmente, la red está ajustada con propagación hacia atrás. La Figura 5.6b ilustra una representación sencilla de cómo los autocodificadores pueden reducir la dimensión de los datos de entrada y aprender a recrearlos en la capa de salida.



(a) Etapas de entrenamiento en un autocodificador. (b) Nodos de un autocodificador.

Figura 5.6: Etapas de entrenamiento y nodos en un autocodificador [12].

La expresión que describe el promedio de la función de activación $a_j^{(2)}$ de la j-ésima unidad de la segunda capa cuando la x-ésima entrada activa la neurona es:

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)} x^{(i)} \right] . \tag{5.62}$$

Se introduce un parámetro de dispersión ρ pequeño (por ejemplo, $\rho = 0{,}03$), con $\hat{\rho} = \rho$. Para asegurarnos de que $\hat{\rho} = \rho$, se añade también un término de penalización de Kullback-Leibler $KL(\rho||\hat{\rho}_j)$, que es una función en la que $KL(\rho||\hat{\rho}_j) = 0$ cuando $\rho = \hat{\rho}_j$ y aumenta monótonamente conforme la diferencia entre los dos valores diverge. La función de coste considerando este parámetro es:

$$J_{\text{dispersa}}\,(W, b) = J(W, b) + \beta \sum_{j=1}^{s2} KL\,(\rho \| \hat{\rho}_j)\,, \tag{5.63}$$

donde $s2$ es el número de unidades en la segunda capa y β es el parámetro que controla el peso del término de penalización de dispersión.

5.2.3. Algoritmos para el entrenamiento

Los algoritmos de aprendizaje constituyen la parte central del aprendizaje profundo. El objetivo de estos algoritmos es encontrar los valores óptimos de los vectores de pesos para resolver una clase concreta de problema en el dominio que nos interese. Algunos algoritmos frecuentemente usados son:

- Descenso de gradiente (*Gradient Descent, GD*), fundamento detrás de la mayor parte de los algoritmos de aprendizaje automático y aprendizaje profundo.
- Descenso estocástico de gradiente (*Stochastic Gradient Descent, SGD*)
- Momento (*Momentum*)
- Algoritmo de Levenberg-Marquardt (*Levenberg-Marquardt Algorithm, LMA*), que se usa fundamentalmente para resolver problemas de mínimos cuadrados no lineales, como es el caso del ajuste a curvas.
- Propagación hacia atrás en el tiempo (*Backpropagation through time, BPTT*)

Analicemos en qué se fundamentan aquellos que pueden resultar de interés en problemas de regresión y predicción de series temporales.

Descenso de gradiente

El descenso de gradiente (*Gradient Descent, GD*) se basa en el método o algoritmo de Newton-Raphson, un método numérico para encontrar las raíces de funciones reales de una variable real.

De manera similar a como se hace en el algoritmo de Newton, desplazamos la solución a lo largo de un determinado camino en un espacio de pesos multidimensional (una dimensión por cada peso), buscando reducir la función de coste hasta que la función de error deje de decrecer. Existe el inconveniente de que, si usamos este método en funciones no convexas, podemos atascarnos en un mínimo local. Este problema puede provocar que el conjunto de pesos obtenido sea subóptimo.

La función de coste de este algoritmo es:

$$C = \frac{1}{2}\left(y_{\text{esperado}} - y_{real}\right)^2, \tag{5.64}$$

donde y_{esperado} es la salida deseada y y_{real} es la salida obtenida.

El método de propagación hacia atrás usa el método GD. En cada iteración, se calcula el incremento del error para los cambios en el valor de cada peso. Dichos pesos son, tras cada

iteración, ajustados para reducir la función de coste. El resultado final es un espacio multidimensional de pesos. El proceso se repite hasta que la función de coste deja de reducirse.

En la Figura 5.7 se muestra el proceso de cálculo de las derivadas de los errores en relación con las salidas en cada capa oculta.

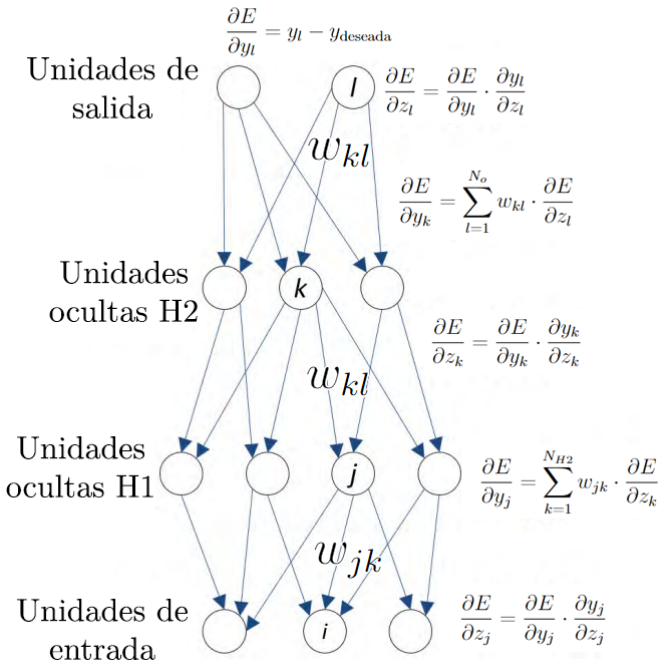


Figura 5.7: Cálculo de los errores en una red neuronal con dos capas ocultas [12].

En la figura, N_o y N_{H2} hacen referencia al número de unidades de salida y en H2, respectivamente.

Descenso estocástico de gradiente

El descenso estocástico de gradiente (*Stochastic Gradient Descent, SGD*) es la variación e implementación más común del descenso de gradiente. Mientras que en el descenso de gradiente se realiza el algoritmo a todos los datos del conjunto de datos antes de actualizar los pesos, al realizar SGD las actualizaciones se realizan tras haber trabajado con un lote de n muestras. Dado que actualizamos los pesos más frecuentemente que al realizar GD, la convergencia al mínimo global es más rápida.

Momento

En un SGD estándar, la tasa de aprendizaje se usa como un multiplicador del gradiente para calcular la actualización a los pesos. Esto puede provocar que el algoritmo pase por alto un posible mínimo si el gradiente es demasiado empinado, o que la convergencia se retrase si el gradiente varía poco.

Una potencial solución surge al usar la idea física de momento: el algoritmo de momento usa una variable de velocidad v, que es una función exponencial decreciente del promedio del gradiente. Para cada paso de tiempo, actualizamos la velocidad y el gradiente conforme a las siguientes expresiones:

$$\text{Actualización de la velocidad: } v \leftarrow \alpha v - \epsilon g \tag{5.65}$$

$$\text{Actualización del gradiente: } \theta \leftarrow \theta + v \tag{5.66}$$

donde el parámetro de momento es α, con $\alpha \in [0, 1)$, y ϵ es la tasa de aprendizaje.

Propagación hacia atrás en el tiempo

La propagación hacia atrás en el tiempo (*Backpropagation through time, BPTT*) es el método estandarizado en el entrenamiento de redes neuronales recurrentes. Como se mostró en la Figura 5.2, el desarrollo de una RNN en el tiempo es similar al de una red retroalimentada. Sin embargo, una RNN desenrollada tiene los mismos pesos para cada capa. El proceso de retroceso temporal calcula en cada paso el gradiente respecto a pesos específicos en cada capa, y luego promedia las actualizaciones para el mismo peso para incrementos de tiempo distintos y los cambia para que el valor de los pesos sea uniforme en cada capa.

5.2.4. Defectos de los algoritmos de aprendizaje

Pueden aparecer diversos problemas al emplear los algoritmos de aprendizaje antes descritos en redes neuronales profundas. Comentemos los más comunes [17]:

Gradientes desvanecientes o explosivos

Las redes neuronales profundas a menudo gradientes que se desvanecen (se hacen muy pequeños en poco tiempo) o explosivos (se hacen muy grandes en poco tiempo) debido a la forma en la que calculamos derivadas: al calcularlas capa por capa en cascada, las derivadas crecen o decrecen de manera exponencial. Los pesos se incrementan o disminuyen basándose en los gradientes con el fin de reducir la función de coste. Gradientes muy pequeños pueden provocar que la red tarde demasiado tiempo en entrenarse, mientras que gradientes demasiado grandes pueden provocar que el entrenamiento diverja. Este efecto empeora con el uso de funciones de activación no lineales, como es el caso de la sigmoide o la tangente hiperbólica, cuyas imágenes están en un intervalo pequeño.

Ese problema puede mitigarse eligiendo otros valores iniciales para los pesos, o mediante el empleo de funciones lineales de activación, como puede ser la ReLU.

Mínimo local

En funciones convexas, el mínimo local es siempre el mínimo global, lo que hace que el algoritmo de descenso de gradiente funcione. Sin embargo, en funciones no convexas, existe el problema de que la propagación hacia atrás puede provocar convergencias a mínimos no globales, que el algoritmo confunde con mínimos globales.

Regiones planas y puntos de silla

De manera similar a lo que sucede en locales mínimos, las regiones planas y los puntos de silla pueden resultar problemáticos en la optimización basada en gradientes descendientes, si trabajamos con funciones no convexas de dimensiones altas.

Bordes empinados

En caso de que la cuenca de un mínimo global sea demasiado estrecha, el algoritmo puede pasarse tal mínimo por alto.

Tiempo de entrenamiento

El tiempo de entrenamiento es un factor importante a la hora de valorar la eficiencia de un algoritmo. La mayoría de los modelos requieren cantidades desorbitadas de tiempo y conjuntos de datos enormes para tener buenas predicciones.

Sobreajuste (*Overfitting*)

Aumentar el número de neuronas de una DNN aumenta la capacidad de resolución de problemas de la red, permitiéndole trabajar con problemas más complejos. Sin embargo, esto puede provocar que el modelo se vuelva demasiado rígido y ajustado a los datos de entrenamiento, dando malos resultados en los datos de entrenamiento. Por ejemplo, en los problemas de clasificación o clusterizado, el sobreajuste puede crear una salida polinómica de alto orden que separe el límite de decisión para el conjunto de entrenamiento, que tomará más tiempo y provocará resultados de peor calidad para los conjuntos de prueba.

Una forma de contrarrestar el sobreajuste consiste en tratar de optimizar el número de neuronas en la capa oculta, ajustándolo a las características del problema. Aunque existen algoritmos que pueden usarse para aproximar el número de neuronas, lo más usual es experimentar con diferentes números hasta alcanzar un valor óptimo.

5.2.5. Optimización de algoritmos de aprendizaje

El objetivo de las redes neuronales profundas es optimizar la precisión del modelo en los datos de prueba. Los algoritmos de entrenamiento se diseñan con el fin de alcanzar ese objetivo tomando como medida de calidad la función de coste.

De las cinco problemáticas comentadas en la sección anterior, tres son provocadas debido a que los algoritmos, para funcionar correctamente, requieren que la función solución del problema sea convexa. Por otro lado, los otros dos problemas surgen por tener un número demasiado alto de nodos, o que los pesos de dichos nodos puedan adoptar un abanico de pesos demasiado amplio.

Mientras que los pesos se aprenden durante el proceso de entrenamiento sobre el conjunto de datos de entrenamiento, hay otros parámetros adicionales, llamados hiperparámetros, que no se aprenden directamente de dicho conjunto. Estos hiperparámetros pueden tener valores en

un rango amplio, añadiendo complejidad a la tarea de encontrar una arquitectura y un modelo óptimos.

A continuación, trataremos algunas formas populares de mejorar la precisión de las DNNs, y que nosotros aplicamos a nuestros modelos.

Optimización de hiperparámetros

Los hiperparámetros más frecuentes en DNN son la tasa de aprendizaje y los parámetros de regularización. La tasa de aprendizaje determina el ritmo al cual los pesos se actualizan, y el propósito de la regularización es evitar que el sobreajuste y los parámetros de regularización afecten al grado de influencia de la función de pérdida. Las redes complejas tienen más hiperparámetros: el número de filtros, las formas de los filtros, el número de capas de *dropout* y las formas de agrupación máximas en cada capa de convolución, el número de nodos en capas totalmente conexa...

Estos parámetros son muy importantes para entrenar y modelar una DNN. Encontrar un conjunto óptimo de valores para estos parámetros es, en general, un proceso complicado, dado que iterar a través de cada combinación de valores de hiperparámetros es computacionalmente muy costoso.

Por ejemplo, supongamos que el entrenamiento y evaluación de una DNN, con todos los datos del conjunto, tarda diez minutos. Si disponemos de siete hiperparámetros, cada uno con ocho valores potenciales, tardaremos $8^7 \cdot 10\,\text{min}$, casi 40 años, en entrenar exhaustivamente y evaluar la red en todas las combinaciones de los valores de hiperparámetro.

En nuestro caso, definiremos un espacio de hiperparámetros para cada red neuronal propuesta, y el algoritmo tomará puntos al azar, tratando de encontrar aquél cuyo modelo asociado ofrezca un error cuadrático medio menor. Existen otros algoritmos que usan criterios para tomar puntos, agilizando la búsqueda de los hiperparámetros óptimos, pero nosotros elegimos el más sencillo por no disponer de suficiente potencia computacional.

Tasas de aprendizaje adaptativas

Las tasas de aprendizaje son de gran importancia en el entrenamiento de redes neuronales profundas. El uso de tasas de aprendizaje que varíen a lo largo del entrenamiento puede acelerar el proceso de entrenamiento, o solventar los problemas asociados a funciones con superficies planas o no convexas. La técnica de tasas de aprendizaje adaptativas consiste en variar las tasas de aprendizaje de los parámetros en función del gradiente y el impulso. Describamos las técnicas propuestas más destacables:

1. **Algoritmo Delta-Bar:** La tasa de aprendizaje de un parámetro aumenta si la derivada parcial de la función problema con respecto a él no varía en signo, y disminuye si cambia de signo.

2. **AdaGrad:** Es un algoritmo más sofisticado que el Delta-Bar, y no funciona en el entrenamiento de todas las redes neuronales profundas. Se usan distintas tasas de aprendizaje

para los parámetros, teniendo en cuenta la raíz cuadrada de los gradientes acumulados de cada parámetro. Al tener una magnitud acumulativa, es probable que los parámetros converjan.

3. **RMSProp:** Se trata de una modificación del algoritmo de AdaGrad que lo hace efectivo en espacios de problema no convexos. Se reemplaza la suma de gradientes al cuadrado de AdaGrad por un promedio móvil exponencialmente decreciente de los gradientes, eliminando la influencia de gradientes históricos.

4. **Adam (*Adaptative Moment Stimation*):** Integra las ideas de AdaGrad, RMSProp y momento. Al igual que AdaGrad y RMSProd, Adam otorga una tasa de aprendizaje individual para cada parámetro. Incluye las ventajas de los dos métodos anteriores y, además, maneja mejor los objetivos no estacionarios y problemas de gradientes ruidosos y dispersos. Adam usa el primer momento (es decir, la media aritmética) y segundo momento (varianza no centrada) de los gradientes, utilizando el promedio de la evolución de una función exponencial de los gradientes al cuadrado.

En la Figura 5.8, se muestra el desempeño relativo de los diversos métodos de tasa de aprendizaje adaptativo, donde se observa que Adam supera al resto.

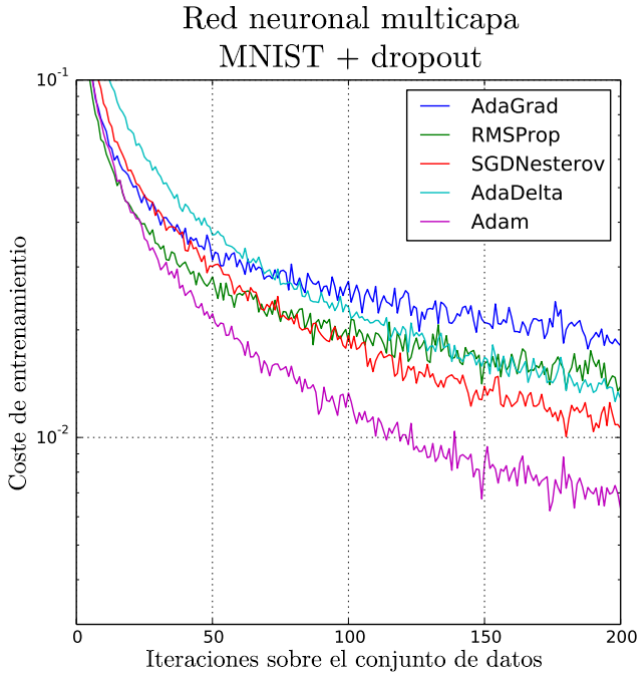


Figura 5.8: Costo de entrenamiento de red multicapa en el conjunto de datos MNIST (base de datos de imágenes de dígitos) usando diferentes algoritmos de aprendizaje adaptativo [12].

Dado que Adam ha demostrado ser la técnica para modular las tasas de aprendizaje con mejores resultados en la mayoría de las ocasiones, y por sus ventajas frente al resto de métodos, la usaremos en nuestros modelos.

Dropout

El dropout es uno de los pocos métodos capaces de reducir el riesgo de sobreajuste. En esta técnica, se eligen al azar unidades y se anulan sus pesos y salidas, de tal manera que no influyan en el paso hacia adelante ni en la propagación hacia atrás.

Uso de la nube y de GPUs para acelerar los entrenamientos

El tiempo de entrenamiento es uno de los principales indicadores del rendimiento de un proceso de aprendizaje automático. En este sentido, resultan muy eficientes la computación en la nube y las GPUs. Trabajar con servidores desde la nube nos proporciona un poder de cómputo enorme, y los principales proveedores de servidores para aprendizaje automático en la nube incluyen en su servicio servidores dotados de GPUs que pueden ser utilizados para entrenar DNN bajo demanda a precios competitivos. En nuestro caso, usamos Google Colaboratory, una plataforma de Google que permite trabajar con Python y dispone de un uso gratuito limitado de GPUs.

Capítulo 6

Metodología

La metodología del trabajo es la que se sigue comúnmente en cualquier proceso de entrenamiento de una red neuronal [18]. Primeramente, debemos construir una tabla de datos con información diaria sobre desplazamientos y factores externos. Mientras que los factores externos ya vienen recogidos como datos diarios, cada desplazamiento es un dato individual, por lo que es necesario realizar sumas diarias. La información de los desplazamientos viene recogida en siete archivos .csv distintos, con un formato como el que se observa en el Cuadro 6.1.

Fecha inicio	Fecha fin	Bicicleta	Origen	Destino	Candado origen	Candado destino
30/06/2016 21:44	30/06/2016 21:49	b19	Campus Unamuno	Estación de autobuses	c1305	c1004
30/06/2016 21:31	30/06/2016 21:54	b97	María Auxiliadora	María Auxiliadora	c808	c810
30/06/2016 21:19	30/06/2016 21:24	b86	Parque Zona Carrefour	Estación de autobuses	c702	c1007
30/06/2016 21:13	30/06/2016 21:25	b42	Huerta Otea	Puente Romano	c1805	c306
30/06/2016 21:12	30/06/2016 21:42	b74	Plaza Gabriel y Galán	Puente Romano	c2206	c307
30/06/2016 21:12	30/06/2016 21:21	b11	Glorieta Alto del Rollo	Plaza Poeta Iglesias	c2006	c116

Cuadro 6.1: Formato de los datos de desplazamientos.

Para nuestro análisis, las columnas que hacen referencia a la bicicleta, los candados y las estaciones de origen y destino no son relevantes, y solamente hacemos uso de las fechas. Asumiremos que se cumple el reglamento de entrega de las bicicletas, por lo que podemos tomar como fecha de cada desplazamiento la fecha de inicio de éste.

En lo que respecta a los datos climáticos, la AEMET dispone los datos colocados entre corchetes para cada día. Para cada día, los datos que nos interesan y que incluimos en nuestros datos temporales son la temperatura media, las precipitaciones y la velocidad media del viento.

Del resto de los datos externos, el día de la semana se calcula en función de la fecha, y los días festivos, con eventos y las vacaciones se incluyen manualmente, comprobando la prensa, el BOE y los calendarios académicos de la Universidad, respectivamente. Como días con eventos, hemos considerado los grandes eventos que suceden anualmente en la ciudad de Salamanca: la Fiesta de las Águedas, el Festival Internacional de las Artes de Castilla y León, la Semana Santa de Salamanca, San Juan de Sahagún, el Festival Luz y Vanguardias, las Ferias y Fiestas

de septiembre, y la Nochevieja universitaria.

Una vez recogidos y tratados los datos, tenemos que el conjunto de datos que utilizamos tiene el formato que se muestra en el Cuadro 6.2.

Fecha	Temp. (ºC)	Precip. (mm)	Viento (km/h)	Festivo	Evento	Fin de semana	Vacaciones universitarias	Desplazamientos
01/01/2016	9.0	11.2	3.6	1	0	0	0	0
02/01/2016	7.8	0.0	2.5	0	0	1	0	57
03/01/2016	9.8	6.3	4.4	0	0	1	0	19
04/01/2016	10.4	22.2	3.9	0	0	0	0	27
05/01/2016	4.8	0.0	3.3	0	0	0	0	60

Cuadro 6.2: Formato del conjunto de datos final. Como temperatura tomamos la temperatura media diaria; como precipitaciones, las totales a lo largo del día; y, como velocidad del viento, la velocidad media diaria.

Este conjunto de datos se separa en conjuntos de entrenamiento, testeo y validación. Para entrenar los modelos, se normalizan los datos: se calcula, para el conjunto de datos de entrenamiento, el promedio y la desviación típica de cada columna, y a cada elemento de la tabla se le resta el promedio y se lo divide entre la desviación típica. Para entrenar usamos 209 semanas (1.351 días, desde el 1 de enero de 2016 al 11 de septiembre de 2019), de los cuales el 20 % de los días se usan como testeo (los últimos del conjunto), mientras que usamos 16 semanas (112 días) para validar los resultados, del 12 de septiembre de 2019 al 2 de enero de 2020. El motivo por el que tomamos dos días de 2020 es para poder separar los datos semanalmente (que el número total de días sea múltiplo de 7).

En la sección 6.1 nos enfocaremos en analizar los modelos concretos de los que hemos hecho uso, tratando cada una de sus capas. En la sección 6.2, abordaremos el proceso de optimización de hiperparámetros de los modelos.

6.1. Redes neuronales implementadas

Hemos implementado, usando Keras, un total de seis redes neuronales distintas: una RNN, una LSTM, una MLP, una CNN multicanal, y dos modelos híbridos: un codificador-decodificador LSTM y un codificador-decodificador CNN-LSTM. Los modelos basados en RNN, LSTM, MLP y CNN tienen relativamente pocas capas y son sencillos, mientras que los híbridos tienen más capas y son más complejos. En general, las redes pequeñas tienen un menor riesgo de sufrir sobreajuste que las grandes, por lo que cabe esperar que proporcionen mejores resultados.

La red MLP es planteada por F. Chollet [19] y es aplicada a un problema de regresión de precios de casas. Los demás modelos son implementados por J. Brownlee [20] para tratar el problema de predecir el consumo eléctrico de una casa. Todos los modelos han sufrido algunas modificaciones para adecuarlos a nuestro problema. En particular, nosotros añadimos

capas Dropout a todos los modelos, con tasas elegidas ad hoc para intentar evitar el sobreajuste.

En todos los modelos, usamos la función de pérdida del error medio cuadrático, y como algoritmo de ajuste del descenso de gradiente usamos Adam, con una tasa de aprendizaje 0,001, que es la que Keras toma por defecto, salvo cuando optimicemos hiperparámetros, cuando variaremos dicha tasa. Como tamaño de los lotes de entrenamiento tomamos 30, aproximadamente un mes, y realizamos 500 epochs de entrenamiento, dado que empíricamente se observa que las pérdidas de validación ya no disminuyen (o incluso crecen) tras realizar ese número de epochs. Nosotros tomaremos aquél modelo que minimice la pérdida de validación.

Siguiendo la misma perspectiva que F. Chollet, las redes se entrenarán con datos agrupados semanalmente. No existe una manera *correcta* para agrupar los datos, pues la agrupación óptima dependerá del modelo y de los datos. En nuestro caso, tomamos grupos de 7 días porque las semanas tienen una periodicidad bien marcada, como se puede comprobar en la Figura 7.4b. Es posible que otra estructuración diferente de los datos ofrezca mejores resultados.

A continuación, trataremos cada uno de los modelos. Los hiperparámetros empleados corresponden a los usados en los modelos en los que no se ha efectuado optimización de hiperparámetros.

Redes RNN y LSTM

Como red RNN se ha tomado una implementación de una LSTM y simplemente se han cambiado la capa LSTM por RNN, por lo que las trataremos de forma conjunta. Estos modelos constan de una capa oculta SimpleRNN o LSTM de 200 unidades y activación ReLU seguida de una capa Dropout con tasa 0,8, de una capa completamente conectada con 100 nodos que interpretará las características aprendidas por la capa SimpleRNN o LSTM, otra capa Dropout, también con tasa 0,8, y una capa de salida densa con siete unidades (porque precedimos vectores de siete componentes, uno por día) y activación lineal ($f(x) = x$).

De esta forma, la red implementada con una capa SimpleRNN dispone de 62.407 parámetros entrenables, mientras que la que implementamos con una capa LSTM tiene 187.207.

Red MLP

La red consta de tres capas ocultas: dos densas completamente conectadas con activación ReLU y 64 unidades, seguidas de una Dropout con tasa 0,8, y la capa de salida es una densa con una neurona, dado que la salida será un único número, y de activación lineal. Con esta configuración tan simple, la red tiene 4.737 parámetros entrenables.

Red CNN univariada

Tras la capa de entrada, tenemos una capa Conv1D con 16 filtros, tamaño del núcleo 3, y activación ReLU. Esto se sigue de una capa MaxPooling1D con un tamaño de agrupación de 2, una capa Dropout con tasa 0.5, una capa que aplana el resultado y una capa Dense de 10 unidades y activación ReLU, que interpretarán las características de la entrada. La capa

de salida es densa y tiene 7 salidas, con activación lineal. Esta red dispone de 759 parámetros entrenables.

Modelo codificador-decodificador LSTM

La primera capa oculta es una LSTM con 200 unidades de activación ReLU. Esta capa será el decodificador, que lea la secuencia de entrada y tenga como salida un vector de 200 elementos (una salida por unidad), capturando características de la secuencia de entrada.

Se usa una arquitectura codificador-decodificador. Primeramente, se repite varias veces la representación interna de la secuencia de entrada (una por cada paso de tiempo en la secuencia de salida, es decir, 7 veces). Esto se hace añadiendo una capa RepeatVector. También incluimos una capa Dropout de tasa 0,8, para reducir el sobreajuste. La secuencia de vectores se introducirá al decodificador LSTM.

Definimos el decodificador como una capa LSTM oculta con 200 unidades y activación ReLU. El decodificador tendrá como salida la secuencia entera, de tal manera que cada una de las 200 unidades tendrá como salida un valor para cada uno de los siete días.

Para interpretar cada paso de tiempo de la secuencia de salida, se añade una capa completamente conectada, con 100 unidades y activación ReLU, y añadimos después una capa Dropout con tasa 0,8 para reducir el sobreajuste. La capa de salida predecirá un único paso de tiempo en la secuencia de salida, y no secuencias de siete días, esto es, la capa totalmente conectada y la capa de salida se usarán para procesar cada paso de tiempo proporcionado por el decodificador. Para conseguir esto, envolveremos las capas de interpretación y de salida en envolturas TimeDistributed, que permiten que las capas envueltas se usen para cada paso de tiempo desde el decodificador. La red tiene, en total, 507.401 parámetros entrenables.

Modelo codificador-decodificador CNN-LSTM

Una red convolucional puede usarse como codificador en una arquitectura codificador-decodificador. Una CNN no admite directamente secuencias como entrada; en su lugar, una CNN 1D es capaz de leer a través de la secuencia de entrada y aprender automáticamente las características salientes. Esto puede ser entonces interpretado por un decodificador LSTM. Conocemos estos modelos híbridos como modelos CNN-LSTM.

Nuestra primera capa oculta es una capa convolucional que lee a través de la secuencia de entrada y proyecta los resultados en mapas de características. La segunda es otra capa convolucional de iguales características, que realiza la misma operación sobre el mapa de características creado por la primera capa, tratando de amplificar las características salientes. Se añade una capa Max Pooling, que simplifica los mapas de características manteniendo 1/4 de los valores con la máxima señal. El mapa de características resultante se aplana en un vector largo que se usará como entrada en el proceso de decodificación. El decodificador es el mismo que en el modelo LSTM codificador-decodificador. En total, la red tendrá 245.961 parámetros evaluables.

6.2. Optimización de hiperparámetros de los modelos

La optimización de hiperparámetros es un proceso computacionalmente muy costoso, y, como ya vimos, puede tomarse tiempos muy largos. Es, además, un proceso estocástico, que puede proporcionarnos hiperparámetros distintos cada vez que lo ejecutemos (no hemos fijado semillas, y, por tanto, se toman al azar en cada ejecución). Hemos realizado un proceso de optimización de hiperparámetros a cada uno de los modelos planteados. Para no entrar en procesos demasiado largos, hemos tratado de tomar pocas opciones para cada uno de los hiperparámetros.

En todos los casos, como ajustador (*tuner*) se ha elegido el algoritmo RandomSearch. En este algoritmo, nosotros definimos un espacio acotado de valores posibles de los hiperparámetros, y el algoritmo selecciona aleatoriamente puntos en ese dominio.

Como hiperparámetros a ajustar, se han considerado las unidades de las capas SimpleRNN, LSTM y Dense (exceptuando las capas densas de salida, cuyas unidades deben venir determinadas por cómo agrupamos los datos), las tasas de las capas Dropout, el número de filtros de las capas Conv1D y la tasa de aprendizaje que se fija al método Adam. En el Cuadro 6.3, se muestran los espacios de hiperparámetros planteados para cada red. En todos los modelos, la tasa de aprendizaje del método Adam tiene valor mínimo 10^{-4}, valor máximo 10^{-2} y muestreo logarítmico.

Redes RNN y LSTM: espacio de hiperparámetros

Capas	Magnitud	Valor mínimo	Valor máximo	Valor inicial	Paso
SimpleRNN/LSTM	Unidades	150	250	200	10
Dense	Unidades	50	150	100	25
Dropout	Tasa	0,3	0,9	0,8	0,1

Red MLP: espacio de hiperparámetros

Capas	Magnitud	Valor mínimo	Valor máximo	Valor inicial	Paso
Dense	Unidades	16	128	64	16
Dropout	Tasa	0,3	0,9	0,8	0,1

Red CNN univariada: espacio de hiperparámetros

Capas	Magnitud	Valor mínimo	Valor máximo	Valor inicial	Paso
Conv1D	Filtros	8	32	16	4
Dense	Unidades	5	20	5	10
Dropout	Tasa	0,3	0,9	0,8	0,1

Codificador-decodificador LSTM: espacio de hiperparámetros

Capas	Magnitud	Valor mínimo	Valor máximo	Valor inicial	Paso
LSTM	Unidades	100	300	200	25
Dense	Unidades	50	150	100	25
Dropout	Tasa	0,3	0,9	0,8	0,1

Codificador-decodificador CNN-LSTM: espacio de hiperparámetros

Capas	Magnitud	Valor mínimo	Valor máximo	Valor inicial	Paso
Conv1D	Filtros	16	128	64	16
LSTM	Unidades	100	300	200	25
Dense	Unidades	50	150	100	25
Dropout	Tasa	0,3	0,9	0,8	0,1

Cuadro 6.3: Espacios de hiperparámetros planteados para cada red.

Durante el proceso de búsqueda de los mejores hiperparámetros, para cada combinación de hiperparámetros se realizan un máximo de 100 epochs, se prueban 100 combinaciones distintas y cada combinación se prueba una única vez. Para cada combinación, se configuran dos llamadas de retorno de detención temprana: se detendrá el entrenamiento y se pasará al siguiente si tras 25 iteraciones el modelo no mejora o si en la iteración número 50 el modelo proporciona una pérdida de validación mayor a la que obtiene el modelo sin optimización de hiperparámetros. El objetivo de implementar esto es la de agilizar el proceso de optimización, evitando evaluar modelos que presentan sobreajuste o que no son mejores que el modelo inicial.

Por supuesto, todo esto podría hacerse de manera más exhaustiva, si dispusiésemos de más potencia computacional. Un aumento del número de epochs para cada combinación de hiperparámetros podría reducir el sobreajuste del modelo final, al igual que la eliminación de las llamadas de detención. Aumentar el número de combinaciones significaría explorar más posibles modelos, pudiendo encontrar algunos que proporcionasen mejores resultados. Evaluar varias veces cada modelo disminuiría notablemente la probabilidad de descartar aquellos modelos que proporcionan buenos resultados de manera consistente frente a otros modelos que solo dan buenos resultados pocas veces (recordemos que el proceso de entrenamiento es estocástico). Por último, también convendría aumentar el espacio de hiperparámetros, aumentando así el número de modelos a evaluar.

Los hiperparámetros que hemos obtenido, y cuyos resultados se muestran en esta memoria, quedan recogidos en el Cuadro 6.4.

Hiperparámetros tras el proceso de optimización

Modelo RNN		**Modelo LSMT**		**Modelo MLP**	
Unidades RNN	220	Unidades LSTM	100	Unidades dense 1	64
Tasa dropout 1	0,4	Tasa 1 dropout	0,3	Unidades dense 2	64
Unidades dense	120	Unidades dense	150	Tasa dropout	0,7
Tasa dropout 2	0,4	Tasa 2 dropout	0,6	Learning rate	$6{,}59{\cdot}10^{-3}$
Tasa aprendizaje	$4{,}71{\cdot}10^{-4}$	Tasa aprendizaje	$4{,}17{\cdot}10^{-3}$		

Modelo CNN univariado		**Modelo C-D LSTM**		**Modelo C-D CNN-LSTM**	
Filtros Conv1D	32	Unidades LSTM 1	125	Filtros Conv1D 1	64
Tasa dropout	0,4	Tasa dropout 1	0,7	Filtros Conv1D 2	48
Unidades dense	20	Unidades LSTM 2	100	Tasa dropout 1	0,3
Tasa aprendizaje	$8{,}04{\cdot}10^{-3}$	Unidades dense	75	Unidades LSTM 1	150
		Tasa dropout 2	0,5	Unidades dense	50
		Tasa aprendizaje	$1{,}99\cdot10^{-3}$	Tasa dropout 2	0,6
				Tasa aprendizaje	$3{,}68\cdot10^{-3}$

Cuadro 6.4: Hiperparámetros resultantes del proceso de optimización. Este proceso es estocástico, así que cabe esperar obtener resultados diferentes cada vez que se ejecute el código.

Capítulo 7

Resultados

En este capítulo, mostraremos los resultados que hemos obtenido a partir de nuestros datos. Mostraremos la eficiencia del entrenamiento de cada uno de los modelos, así como los resultados predichos, en comparación con los observados.

7.1. Resultados preliminares

Mostremos primeramente resultados preliminares espaciales del movimiento entre estaciones, analizando las salidas y entradas a cada estación; y temporales, analizando cómo varían los desplazamientos con el tiempo, comparando esta evolución temporal con factores externos.

Los promedios semanales de la cantidad de veces que se ha salido o entrado con una bicicleta de cada estación quedan recogidos en el Cuadro 7.1, y se muestra la gráfica KDE del promedio semanal de las salidas y entradas de cada estación en la Figura 7.1.

Estación	Código	Salidas	Entradas
Plaza Poeta Iglesias (Plaza Mayor)	C1	80,23	94,82
Plaza de los Bandos	C2	47,39	56,63
Puente Romano	C3	33,90	55,93
Hospital Virgen de la Vega - Parking	C4	23,96	18,13
Barrio de San José (Junto Caja España-Duero)	C5	10,89	12,07
Tejares (Junto a UNICAJA)	C6	6,64	12,89
Parque Calle La Bañeza	C7	55,08	31,78
Avenida de Federico Anaya - El Corte Inglés	C8	56,16	42,71
Estación de trenes (Vialia)	C9	58,56	62,11
Estación de Autobuses	C10	55,11	58,32
Plaza Iglesia Pizarrales	C11	25,98	14,50
Calle La Marina (Parque Jesuitas)	C12	55,20	35,98
Campus Unamuno - Facultad de Medicina	C13	30,46	45,51
Campus Ciencias - Plaza de la Merced	C14	34,33	53,81
Campus Claretianos (Ciudad Jardín)	C15	31,41	28,60
Calle Vergara - Edificio de la JCyL	C16	30,14	38,75
Barrio del Zurguén - Calle Siega Verde	C17	10,61	7,75
Huerta Otea	C18	19,38	20,37
Glorieta CYL - Barrio Garrido	C19	54,62	51,00
Glorieta Alto del Rollo	C20	30,26	19,64
Avenida de Federico Anaya - Los Tilos	C21	48,53	34,66
Plaza Gabriel y Galán	C22	58,27	45,78
Plaza de Toros	C23	29,46	23,63
Plaza del Oeste	C24	47,40	37,65
Paseo Canalejas (Colegio Calasanz)	C25	15,45	24,97
Plaza de San Julián (Gran Vía)	C26	14,38	29,69
Parque Elio Antonio de Nebrija - Salas Bajas	C27	26,97	39,50
Parque de la Alamedilla	C28	47,88	49,44
Tunel de la radio - Avenida de Comuneros	C29	30,70	22,76
Plaza de Santa Eulalia	C30	0,17	0,22
Glorieta de la Unión Deportiva Salamanca	C31	0,28	0,24
Avenida de Villamayor - Mercadona	C32	0,13	0,10
Avenida de Portugal - Los Ovalle	C33	0,12	0,14

Cuadro 7.1: Promedios semanales de entradas y salidas de cada estación del servicio Salenbici.

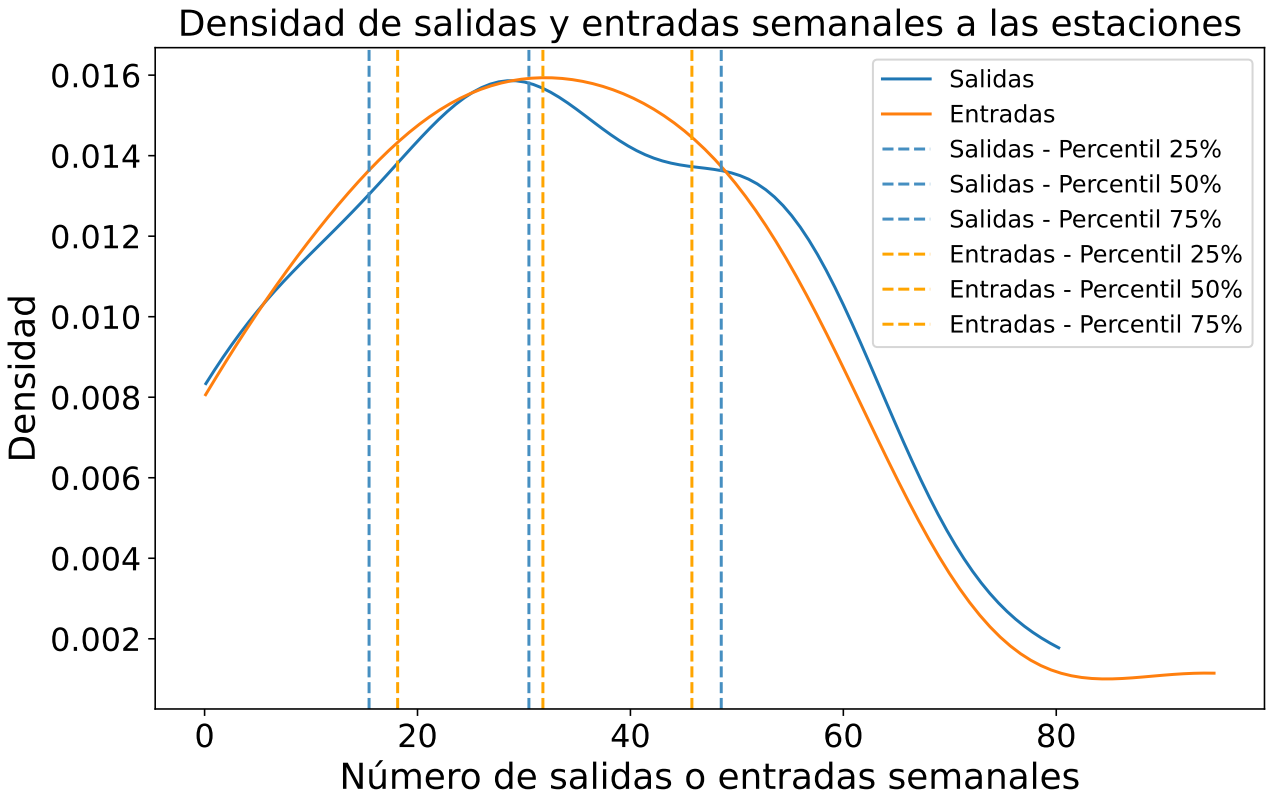


Figura 7.1: Estimación de la densidad del núcleo del promedio semanal de las entradas y salidas de las estaciones.

Por otro lado, resulta de interés comparar la cantidad de préstamos con factores climáticos como la temperatura o las precipitaciones. En las Figuras 7.2 y 7.3 representamos la evolución temporal de la cantidad de préstamos mensuales y la temperatura y las precipitaciones, respectivamente.

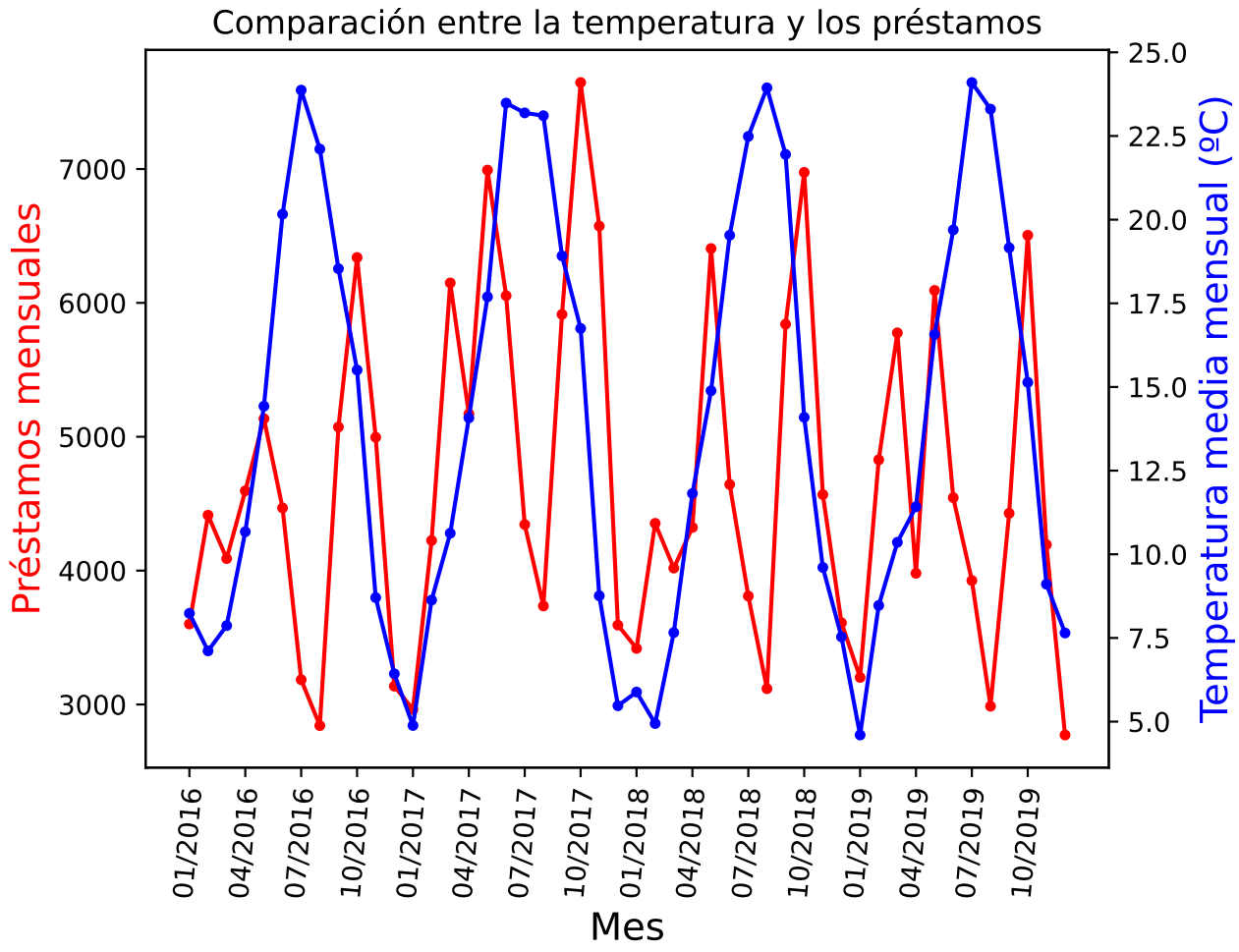


Figura 7.2: Representación de la cantidad de préstamos mensuales (en rojo) y la temperatura media mensual (en azul) en función del tiempo.

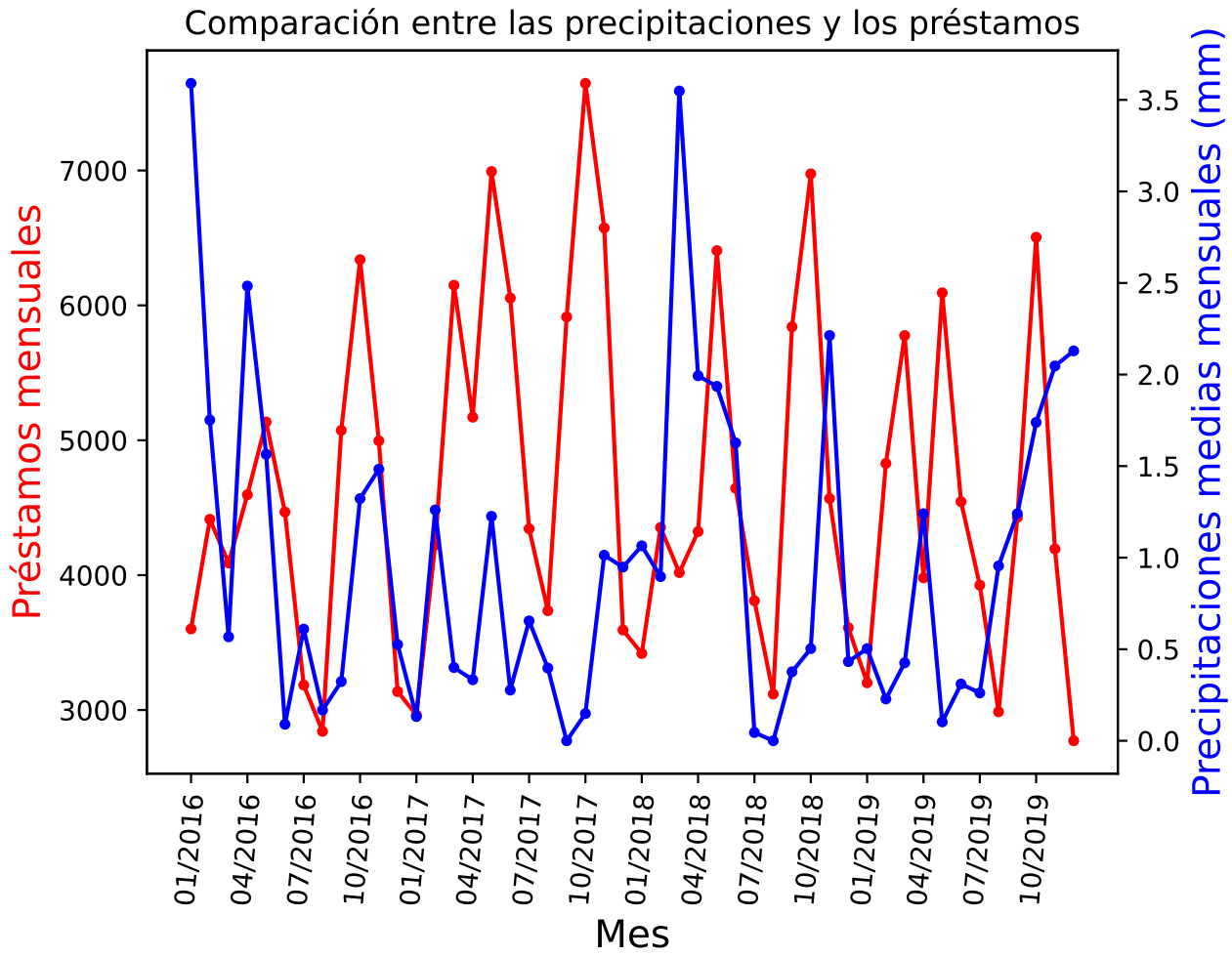
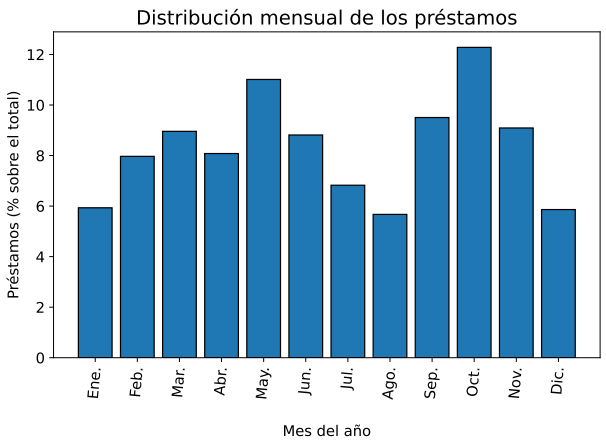
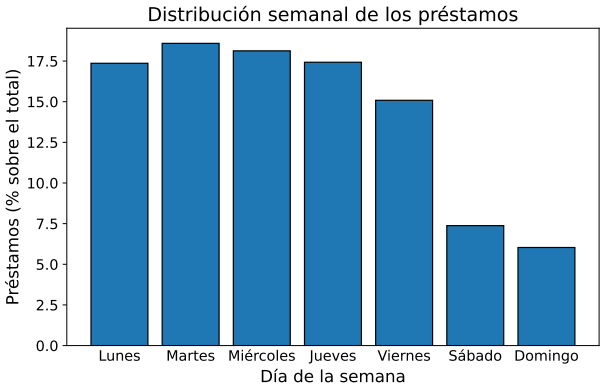


Figura 7.3: Representación de la cantidad de préstamos mensuales (en rojo) y las precipitaciones medias mensuales (en azul) en función del tiempo.

En la Figura 7.4a, se muestra qué porcentaje de préstamos ocurren cada mes del año; la distribución semanal de los préstamos, en la Figura 7.4b; y, en las Figuras 7.5a, 7.5b y 7.5c, la relación que guardan los préstamos con la temperatura, las precipitaciones y el viento, respectivamente. En la Figura 7.6a, se muestra la distribución horaria de los préstamos los días festivos y los no festivos; mientras que en la Figura 7.6b tenemos la distribución horaria de los préstamos los fines de semana y los días de entre semana. En lo que respecta a los parámetros de correlación entre los distintos factores considerados, se han representado en la Figura 7.7. Finalmente, se puede observar la distribución de la duración de los préstamos (es decir, el tiempo que transcurre desde que se toma la bicicleta hasta que se deja) en la Figura 7.8.



(a) Porcentaje de los préstamos ocurrido cada mes del año.



(b) Porcentaje de los préstamos ocurridos cada día de la semana.

Figura 7.4: Distribución anual y semanal de los desplazamientos.

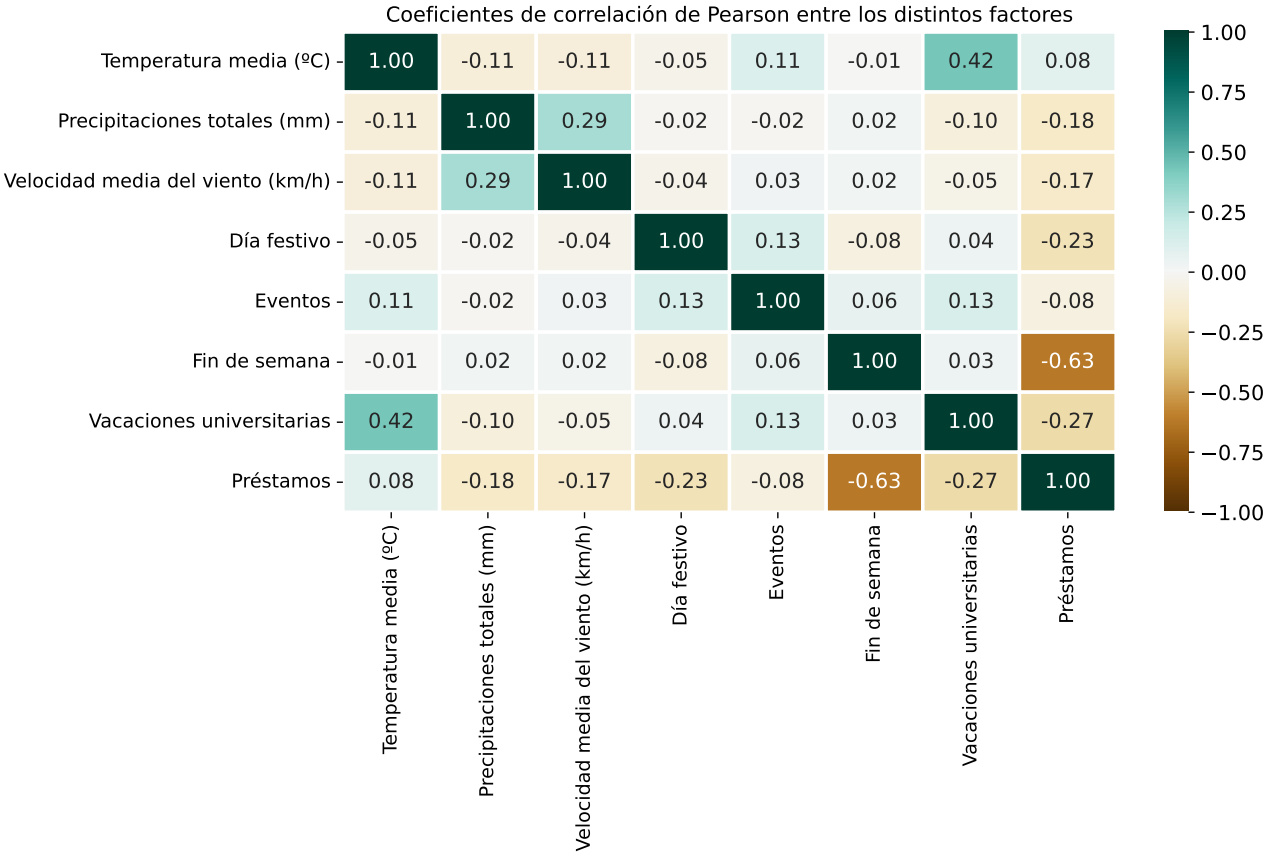
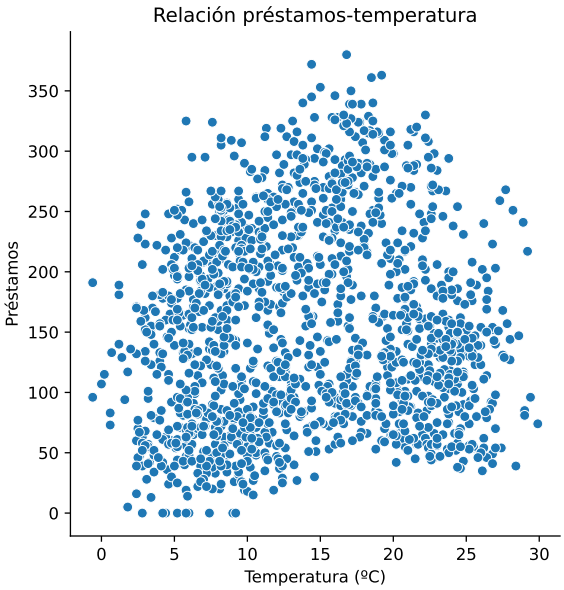
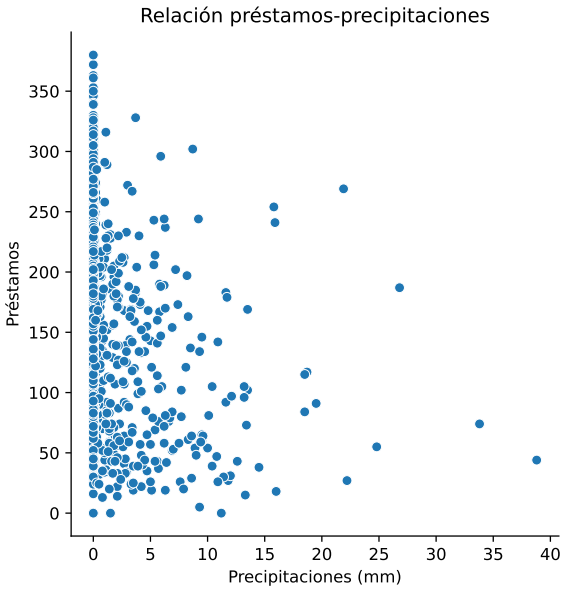


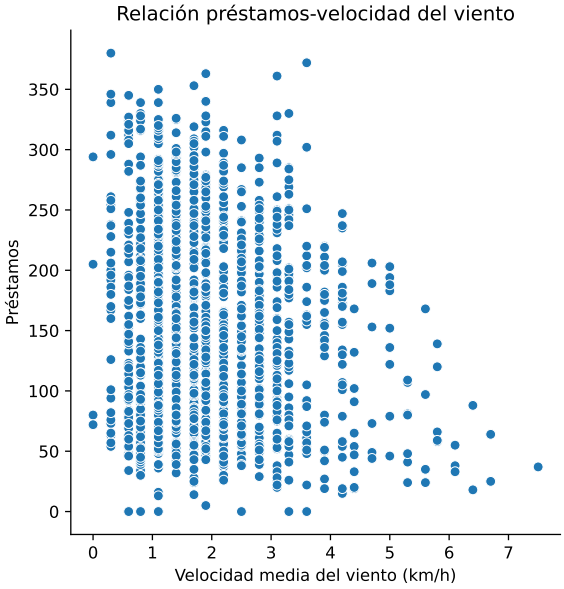
Figura 7.7: Coeficientes de correlación entre los distintos factores.



(a) Relación entre préstamos y temperatura.

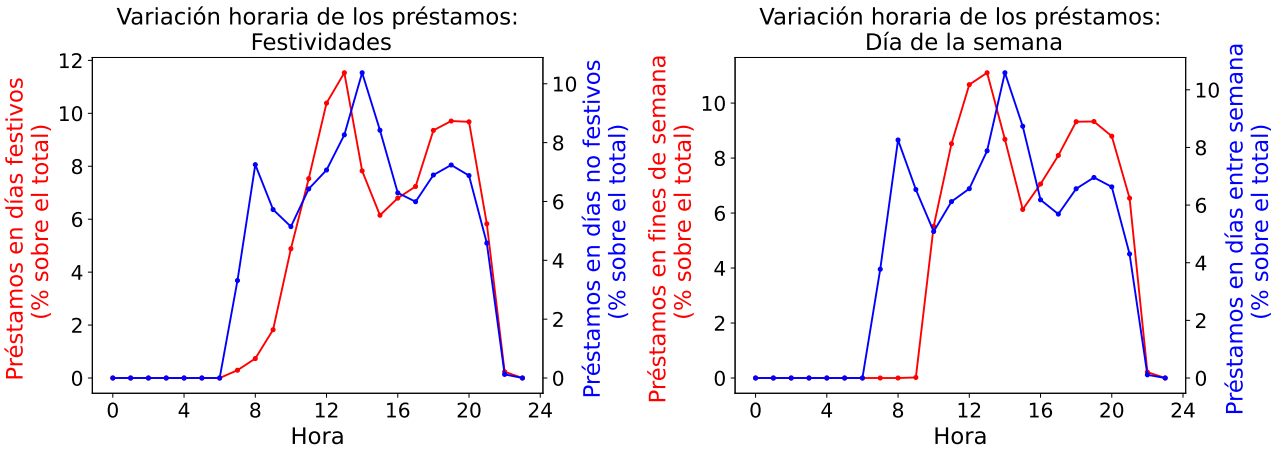


(b) Relación entre préstamos y precipitaciones.



(c) Relación entre préstamos y velocidad del viento.

Figura 7.5: Relación entre los préstamos y factores meteorológicos.



(a) Porcentaje de los préstamos ocurridos cada hora, los días festivos y los días no festivos.

(b) Porcentaje de los préstamos ocurridos cada hora, los fines de semana y los días entre semana.

Figura 7.6: Variación de los préstamos con las festividades y los fines de semana.

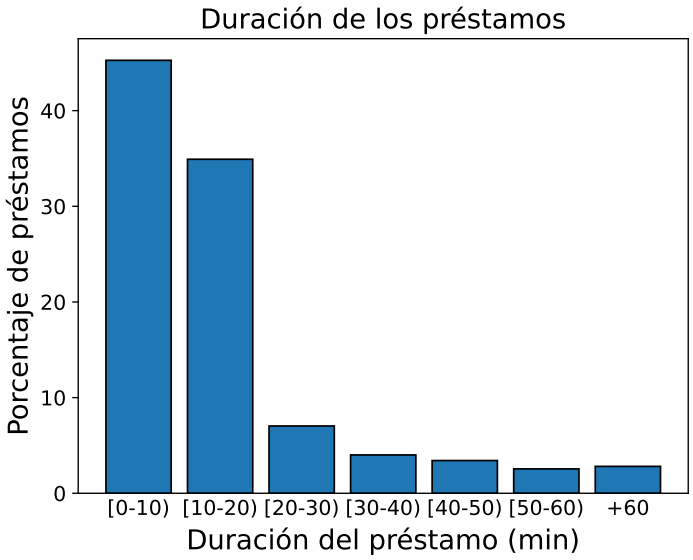


Figura 7.8: Distribución de la duración de los préstamos. Recordemos que la bicicleta debe entregarse, como máximo, 60 minutos después de que se tome.

7.2. Procesos de entrenamiento

Comparemos el proceso de entrenamiento de los modelos. Para cada modelo, se han realizado 500 epochs, y aunque no se ha implementado detención temprana para poder realizar una comparación entre los entrenamientos, se toma como modelo final para las simulaciones aquél que ofrece una menor pérdida de validación media. En los entrenamientos se busca minimizar la pérdida de validación, aunque también se muestran los errores absolutos medios de entrenamiento y de validación. Las Figuras 7.9, 7.10, 7.11, 7.12 muestran, para los distintos modelos, la evolución de la pérdida de entrenamiento, la pérdida de validación, el error absoluto medio

de entrenamiento y el error absoluto medio de validación, respectivamente, para los modelos en los que no se ha efectuado optimización de hiperparámetros. En las Figuras 7.13, 7.14, 7.15, 7.16 se muestra lo mismo, esta vez para los modelos con optimización de hiperparámetros. Recordemos que los entrenamientos se realizan con datos normalizados, por los que los valores de las pérdidas y los errores absolutos medios no están en las mismas escalas que los datos originales.

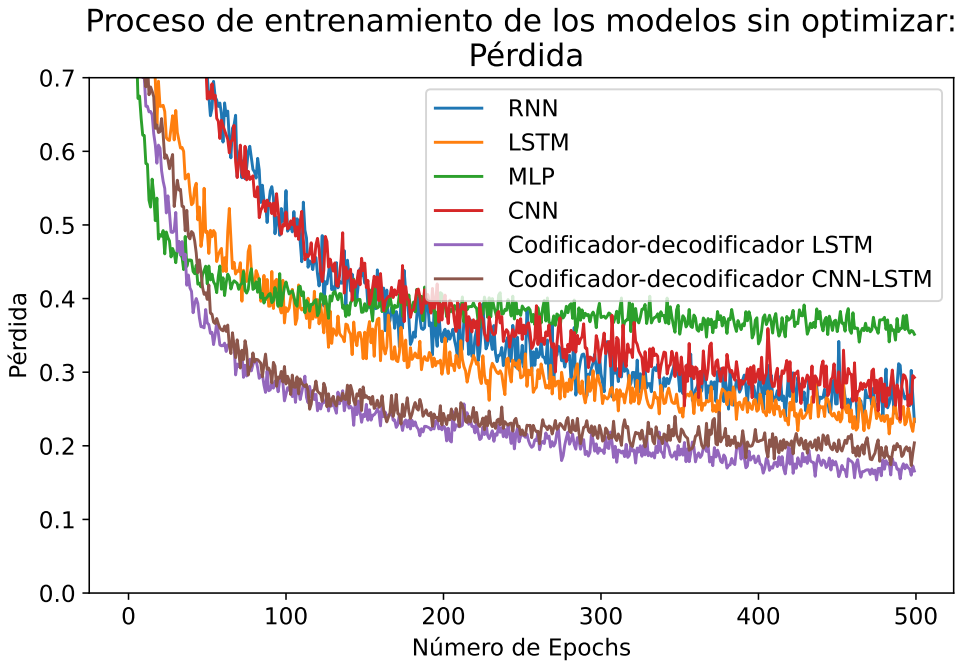


Figura 7.9: Evolución durante el entrenamiento de la pérdida de entrenamiento para los distintos modelos, sin realizarse una optimización de hiperparámetros.

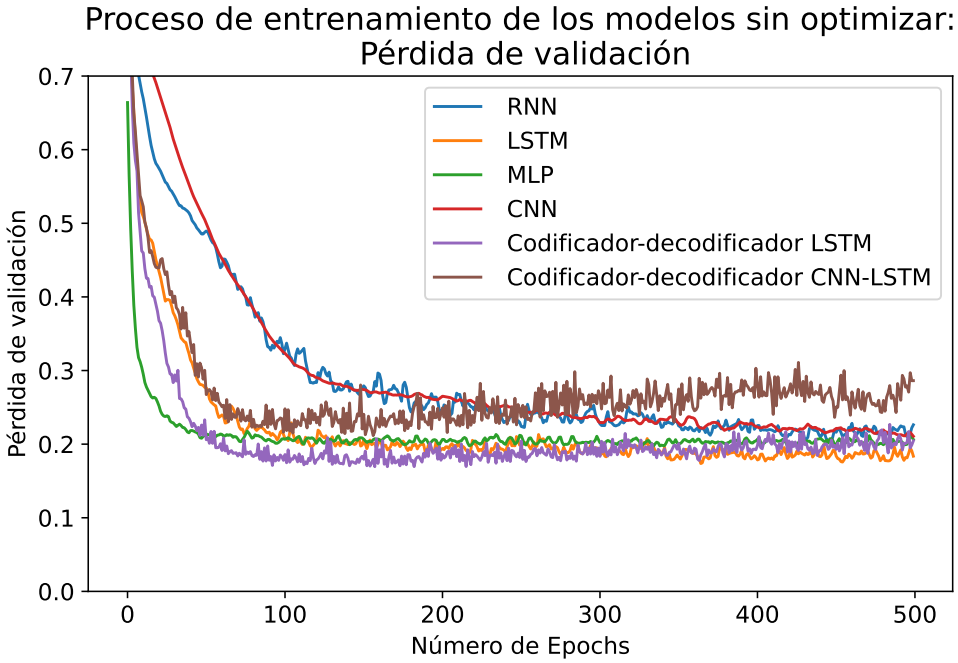


Figura 7.10: Evolución durante el entrenamiento de la pérdida de validación para los distintos modelos, sin realizarse una optimización de hiperparámetros.

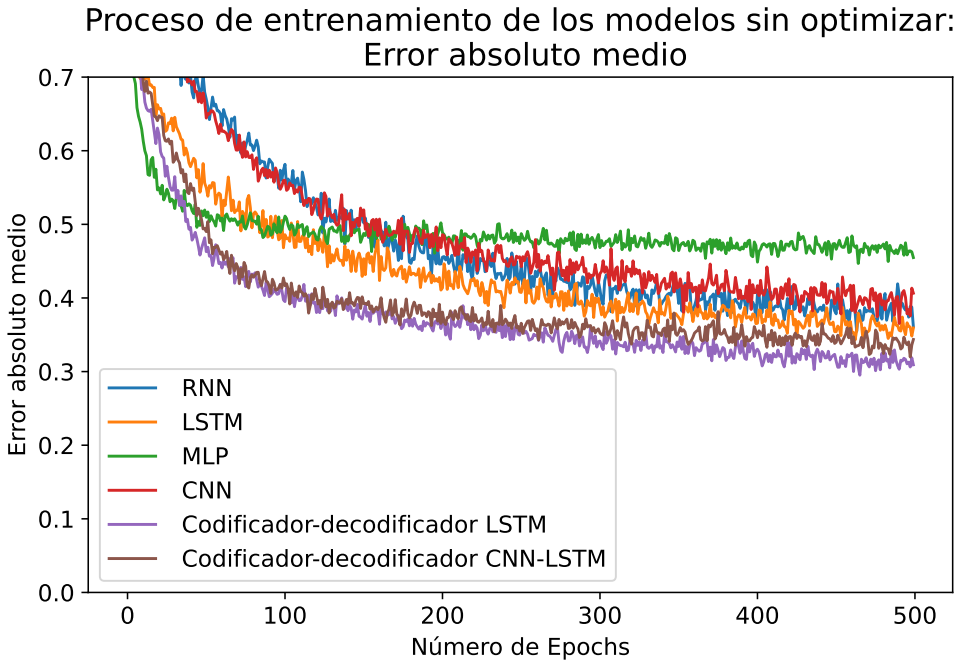


Figura 7.11: Evolución durante el entrenamiento del error medio absoluto de entrenamiento para los distintos modelos, sin realizarse una optimización de hiperparámetros.

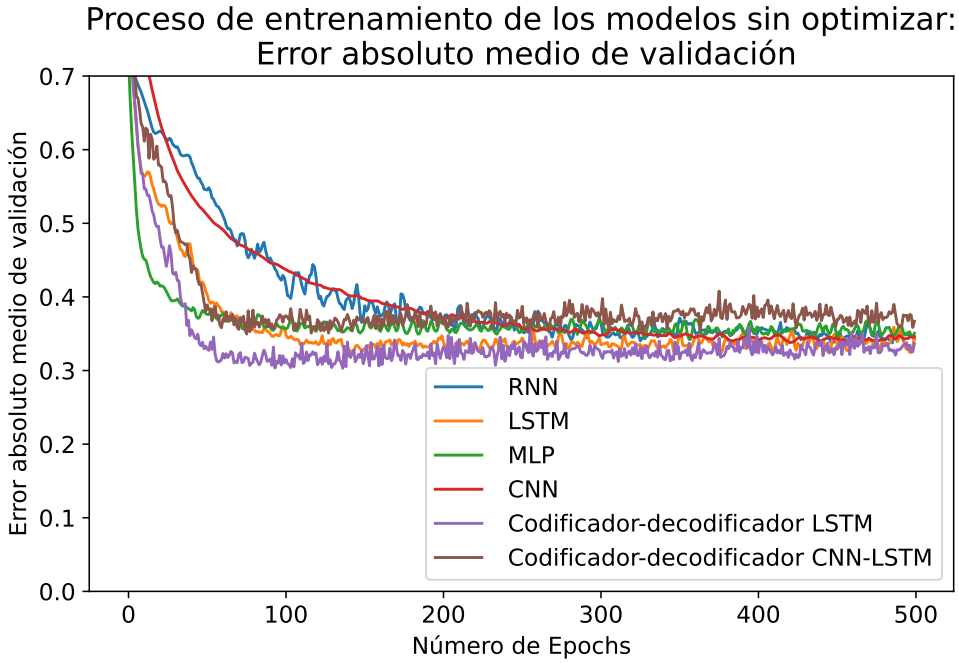


Figura 7.12: Evolución durante el entrenamiento del error medio absoluto de validación para los distintos modelos, sin realizarse una optimización de hiperparámetros.

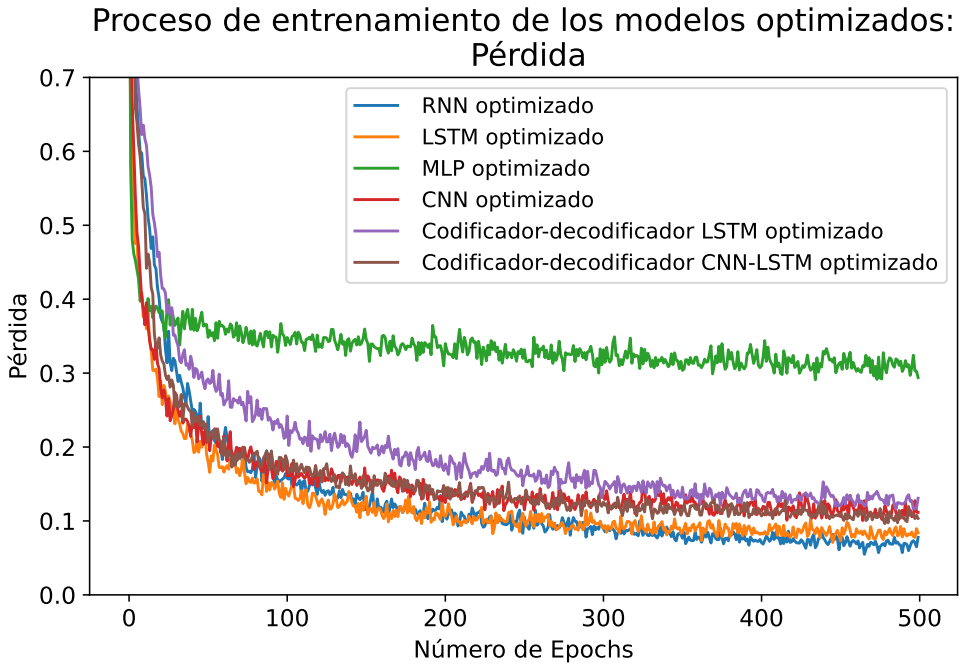


Figura 7.13: Evolución durante el entrenamiento de la pérdida de entrenamiento para los distintos modelos, tras un proceso de optimización de hiperparámetros.

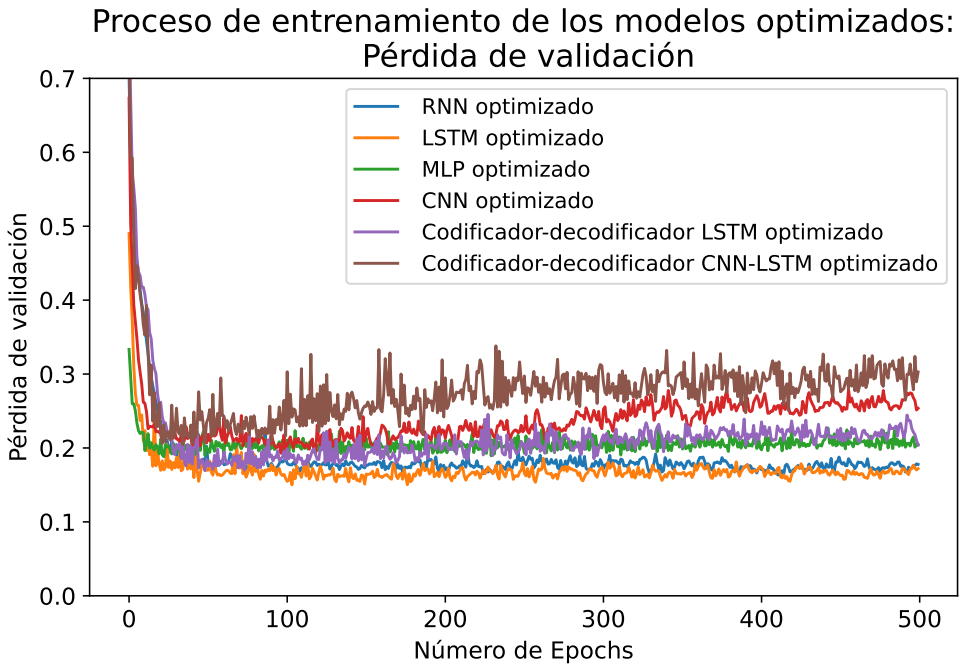


Figura 7.14: Evolución durante el entrenamiento de la pérdida de validación para los distintos modelos, tras un proceso de optimización de hiperparámetros.

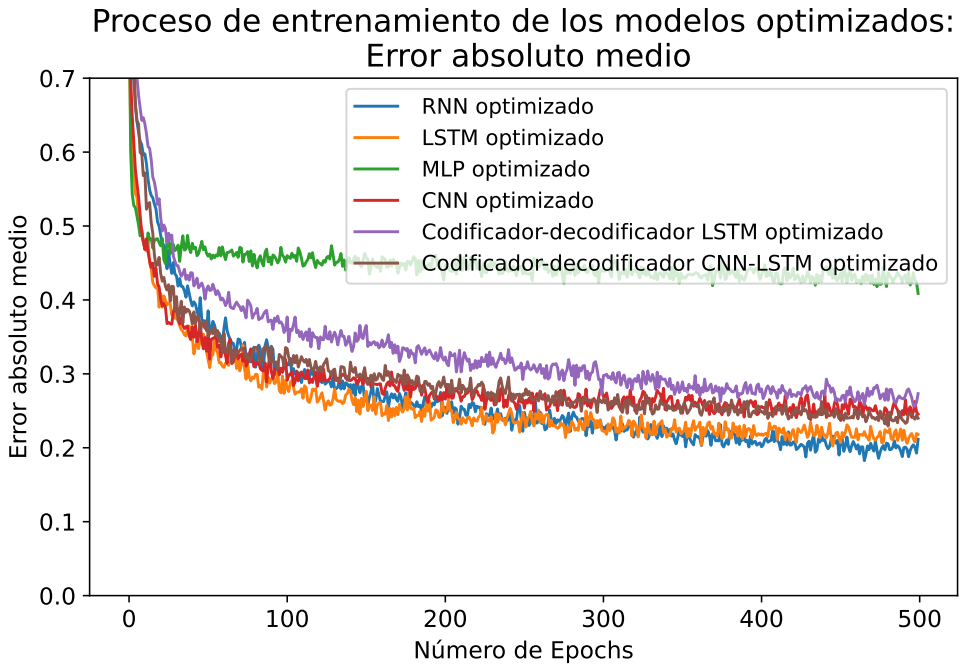


Figura 7.15: Evolución durante el entrenamiento del error medio absoluto de entrenamiento para los distintos modelos, tras un proceso de optimización de hiperparámetros.

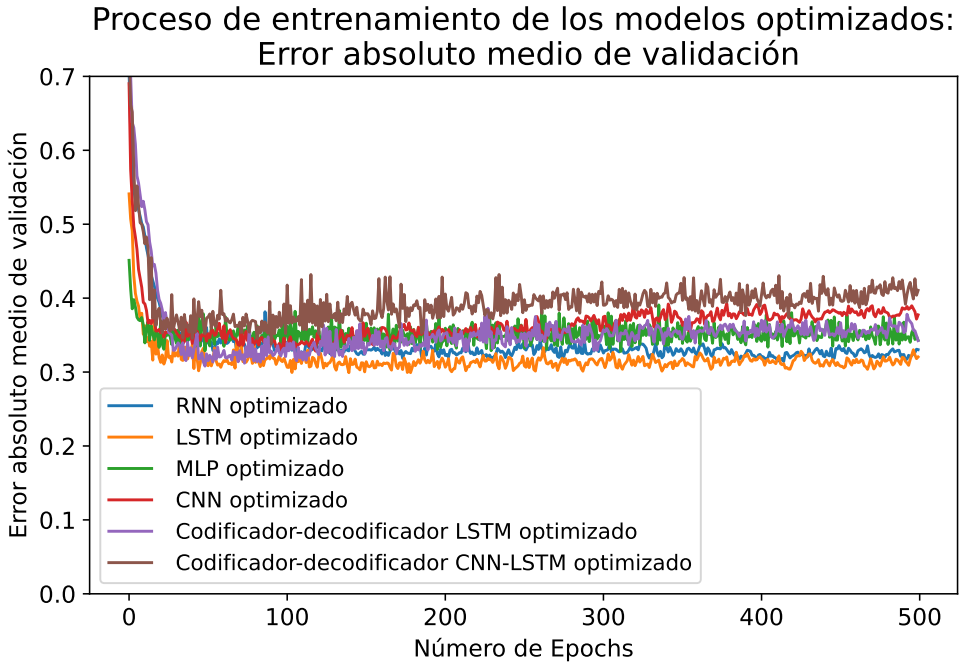


Figura 7.16: Evolución durante el entrenamiento del error medio absoluto de validación para los distintos modelos, tras un proceso de optimización de hiperparámetros.

7.3. Predicciones de los modelos

Las Figuras 7.17 y 7.18 muestran la comparación entre los distintos modelos y los datos de validación, para los modelos no optimizados y optimizados, respectivamente.

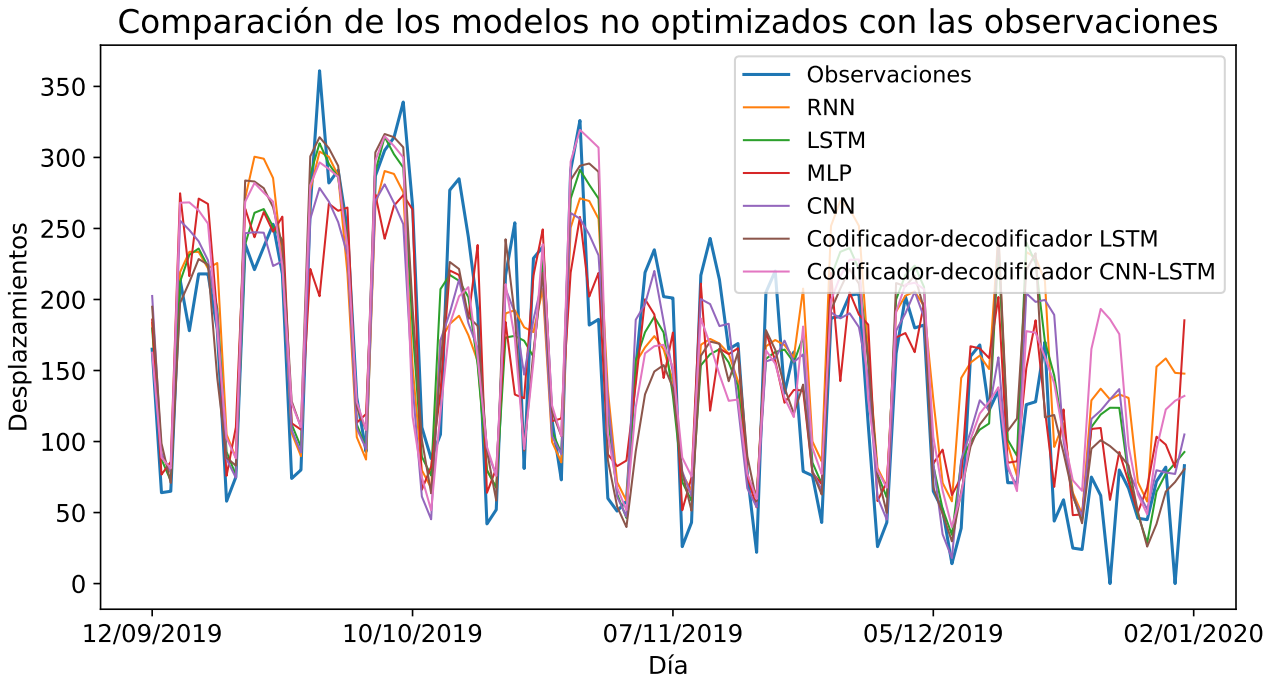


Figura 7.17: Comparación de las predicciones de los distintos modelos (sin optimizar) y las observaciones.

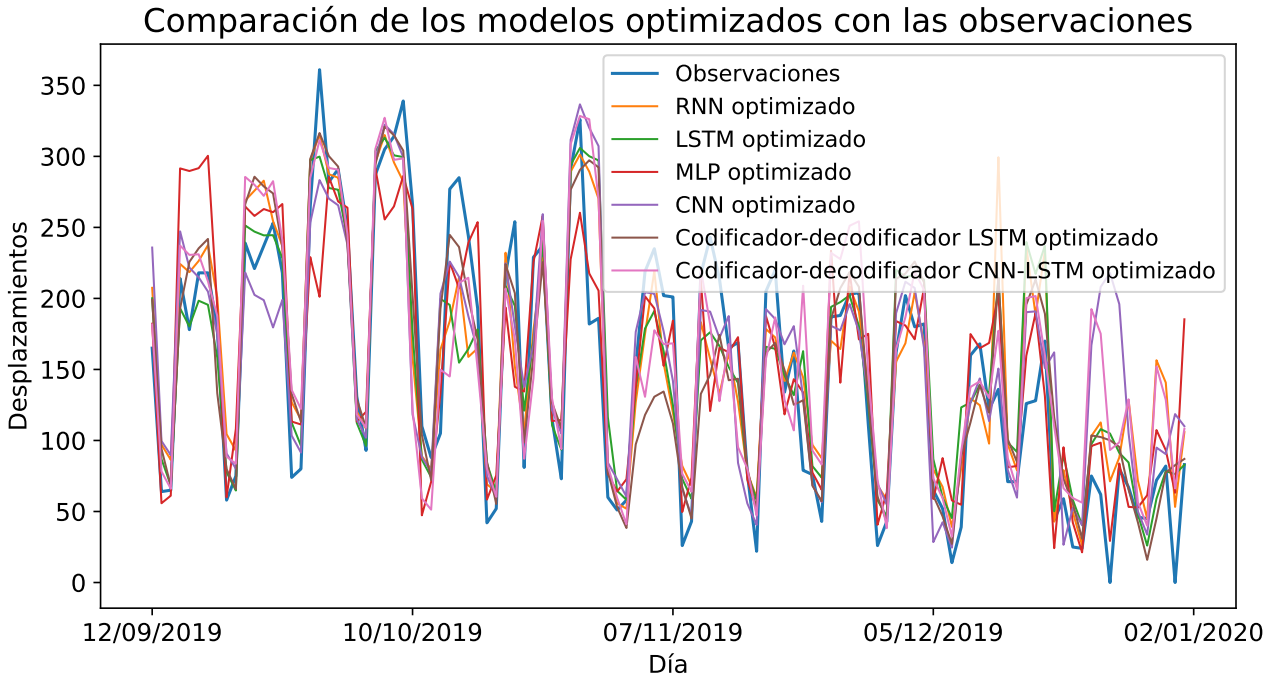


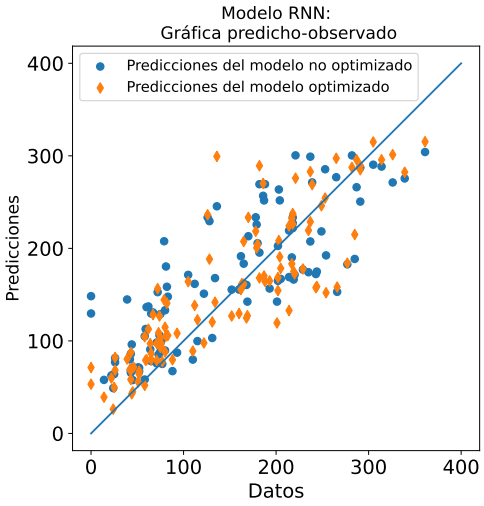
Figura 7.18: Comparación de las predicciones de los distintos modelos (optimizados) y las observaciones.

Los valores de la media, la raíz cuadrada del error cuadrático medio y la desviación media de los errores para cada modelo quedan recogidos en el Cuadro 7.2.

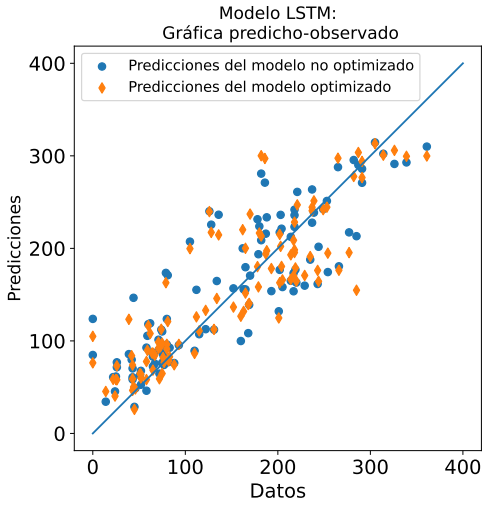
Modelo	Optimización de hiperparámetros	Error absoluto medio	Error absoluto mediano	Raíz del error cuadrático medio	Desviación media del error
RNN	No optimizado	44,01	38,58	7,36	25,31
	Optimizado	34,56	25,50	6,73	22,62
LSTM	No optimizado	35,47	28,08	6,78	23,55
	Optimizado	32,47	22,14	6,63	22,85
MLP	No optimizado	34,08	28,59	6,56	19,35
	Optimizado	32,40	23,39	6,55	21,14
CNN	No optimizado	34,01	25,91	6,64	21,71
	Optimizado	35,96	24,44	7,16	25,50
C-D LSTM	No optimizado	34,21	29,94	6,61	22,03
	Optimizado	33,12	27,11	6,58	22,03
C-D CNN-LSTM	No optimizado	39,63	32,14	7,25	25,43
	Optimizado	37,38	29,49	7,03	25,05

Cuadro 7.2: Coeficientes estadísticos de los errores de cada uno de los modelos planteados.

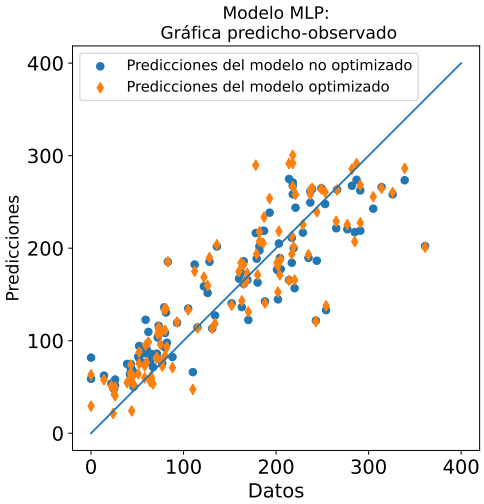
En las Figuras 7.19a, 7.19b, 7.19c, 7.19d, 7.19e y 7.19f se muestran gráficas predicho observado para los distintos modelos planteados, y en las Figuras 7.20a, 7.20b, 7.20c, 7.20d, 7.20e, 7.20f se muestra la distribución de errores entre las observaciones y las predicciones para cada modelo.



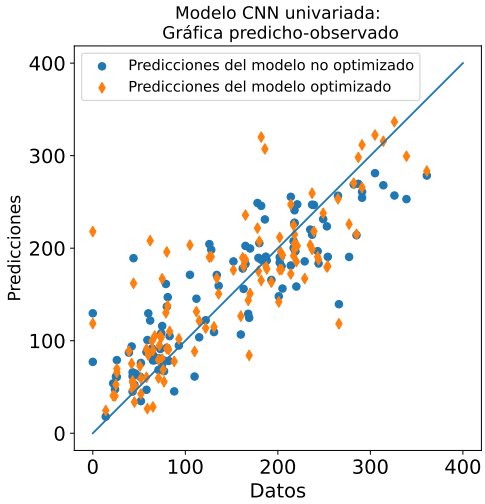
(a) Modelos RNN.



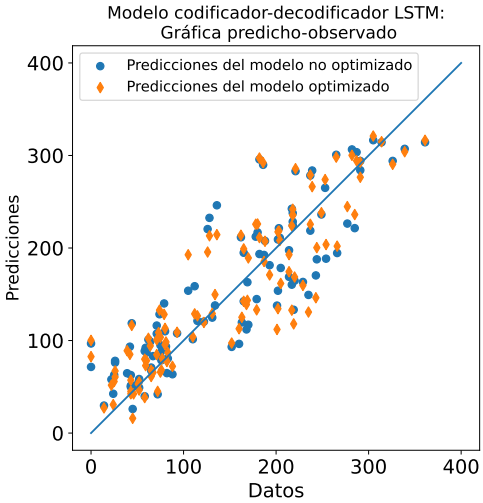
(b) Modelos LSTM.



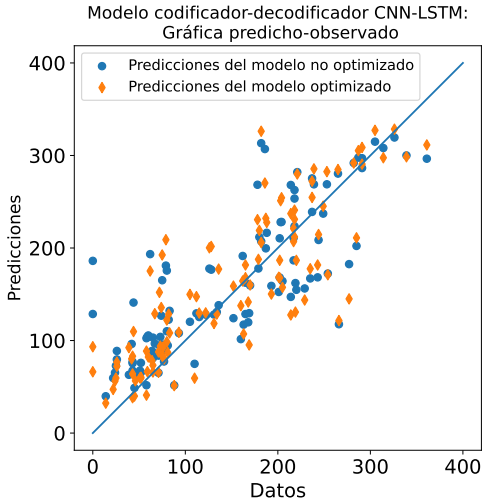
(c) Modelos MLP.



(d) Modelos CNN.

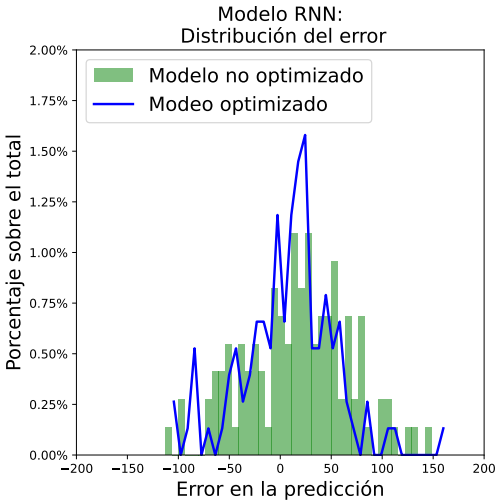


(e) Modelos codificador-decodificador LSTM.

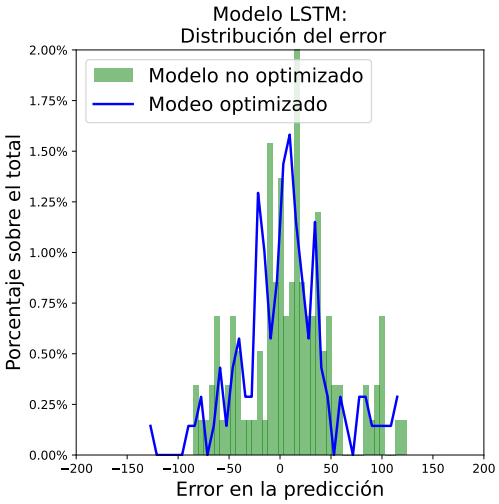


(f) Modelos CNN-LSTM codificador-decodificador.

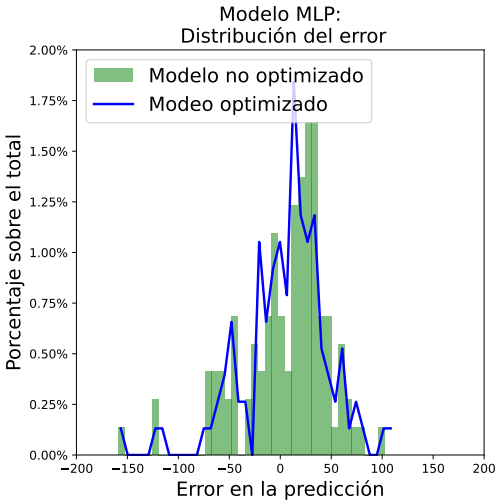
Figura 7.19: Gráficas predicho-observado.



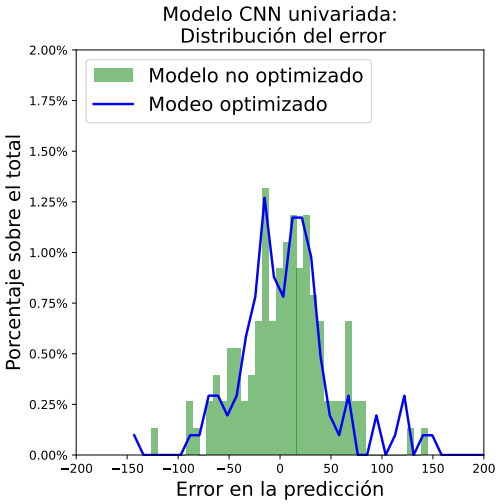
(a) Modelos RNN.



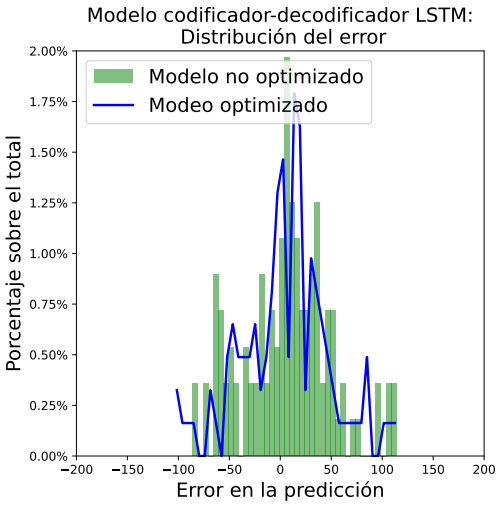
(b) Modelos LSTM.



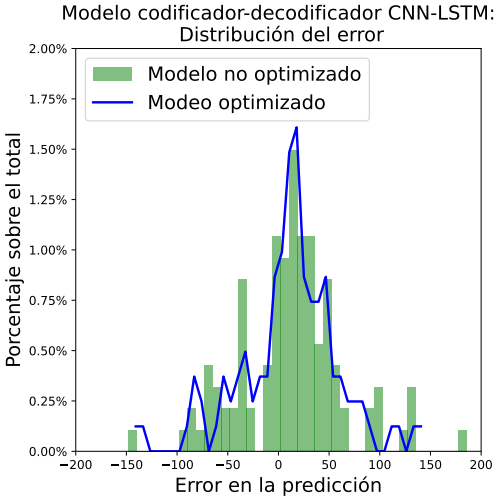
(c) Modelos MLP.



(d) Modelos CNN.



(e) Modelos codificador-decodificador LSTM.



(f) Modelos CNN-LSTM codificador-decodificador.

Figura 7.20: Distribución de errores entre las predicciones y las observaciones.

Capítulo 8

Discusión y análisis de los resultados

En este capítulo, discutiremos los resultados que hemos obtenido a través de los distintos análisis efectuados y herramientas empleadas.

8.1. Datos preliminares

Valoremos primeramente los datos preliminares obtenidos, que nos proporcionan una visión general de cuáles son los factores que intervienen en la demanda de bicicletas, cuál es su importancia y cómo están relacionados.

En primer lugar, podemos observar cómo varía el número de desplazamientos a lo largo de los meses en la Figura 7.4a. Se observa un descenso en el uso del servicio durante los meses de invierno y verano. Podemos suponer que esto se debe principalmente a las condiciones de temperatura: como se observa en la Figura 7.5a, existen máximos del número de desplazamientos para temperaturas entre 15 y 20 °C, reduciéndose los desplazamientos conforme la temperatura aumenta o disminuye de esas cifras.

Sin embargo, es necesario considerar otras causas que puedan contribuir a esta observación. Por ejemplo, en los meses que contienen Navidad y Semana Santa, así como en los meses de verano, las vacaciones universitarias pueden reducir notablemente el número de usuarios del servicio, por lo que tiene sentido haber considerado también las vacaciones universitarias como variable.

En la Figura 7.5b observamos que las precipitaciones reducen los desplazamientos, aunque no demasiado (al menos, las precipitaciones ligeras). Existen pocos datos de días con muchas precipitaciones (solo seis días con más de 20 mm), por lo que no podemos extraer buenas conclusiones de qué sucede los días de mucha lluvia, aunque parece que afectan mucho a los desplazamientos, como cabría esperar. Hay que considerar que, probablemente, los días de poca lluvia solo afecten a los desplazamientos durante el tiempo que esté lloviendo, lo que explica el bajo impacto en la demanda. Por otro lado, según se puede observar en la Figura 7.5c, un exceso de viento reduce los desplazamientos, aunque, de nuevo, esto solo afectará a los desplazamientos cuando el viento sostenido tenga velocidades altas o haya muchas rachas, y, además, el viento puede estar correlacionado con otros factores que también afecten a los desplazamientos.

Es posible, a partir de los datos preliminares, obtener conclusiones sobre el uso que se le da al servicio de préstamo de bicicletas. Por ejemplo, en la Figura 7.4b observamos que la mayor parte de los desplazamientos ocurren de lunes a viernes, y que, por tanto, muy buena parte de los desplazamientos son rutinarios, y no tanto de ocio. Esta interpretación se ve reforzada por las Figuras 7.6b y 7.6a, donde se observa que la distribución horaria de los préstamos los fines de semana y los días festivos es similar. En estas dos Figuras se observa, además, que los picos de préstamos tanto los días no festivos como los días entre semana ocurren a las 08:00, a las 14:00 y a las 19:00, correspondiendo con las horas de entrada y salida del trabajo. Los picos de demanda los días festivos y los fines de semana ocurren de 11:00 a 13:00 y de 18:00 a 20:00.

En lo que respecta a los coeficientes de Pearson, en la Figura 7.7 vemos que el factor más correlacionado con el número de préstamos es si el día es sábado o domingo o no, seguido de los días festivos, siendo ambos coeficientes negativos, es decir, los fines de semana y los festivos reducen la demanda. Las precipitaciones y la velocidad del viento tienen coeficientes muy similares, también negativos. Los coeficientes de correlación menores en módulo se corresponden a los relacionados con los eventos y a la temperatura media, aunque no son en absoluto despreciables.

Al interpretar estos coeficientes, debemos tener en cuenta que presentan algunos problemas. Por un lado, como con la temperatura media el efecto es no lineal, el coeficiente de Pearson no refleja bien la correlación entre la temperatura y la demanda. Además, un mismo fenómeno puede incluirse en dos variables distintas: por ejemplo, la Semana Santa se modeliza a la vez como vacaciones universitarias y como evento. Todo esto hace que debamos ser cautelosos al extraer conclusiones a partir de los coeficientes de Pearson.

Finalmente, en la Figura 7.8 se comprueba que la inmensa mayoría de los préstamos se dan para desplazamientos cortos, de menos de 20 minutos. En préstamos de mayor duración, no podemos concluir si se trataron de un solo trayecto largo, o de una sucesión de varios cortos, dado que no conocemos la trayectoria seguida por cada bicicleta, solo su origen y destino.

8.2. Procesos de entrenamiento

En cuanto a lo que pérdida sobre los datos de entrenamiento se refiere (recordemos que hemos tomado como función de pérdida el error cuadrático medio), observamos en la Figura 7.9 que la red codificador-decodificador LSTM ofrece los mejores resultados, seguido del codificador-decodificador CNN-LSTM: a priori, parece que las redes más complejas van a ofrecer mejores resultados. Las redes LSTM, RNN y CNN tienen resultados similares en lo que a pérdida se refiere, siendo el mejor de los tres modelos el LSTM. La red con peores resultados de pérdida es la MLP, con alrededor del doble de la que obtenemos con la red codificador-decodificador LSTM.

Sin embargo, en la Figura 7.10, podemos observar que, en la práctica, las distintas redes presentan precisiones más parejas. La red que muestra el mejor resultado es el codificador-decodificador LSTM, aunque también se observa cierto grado de sobreajuste en este caso. Los modelos MLP y LSTM ofrecen precisiones similares a las del modelo codificador-decodificador LSTM. Los modelos con peores resultados son el codificador-decodificador CNN-LSTM, que es el más complejo que hemos planteado y en el que también aparece el sobreajuste, el CNN y el

RNN.

De las Figuras 7.11 y 7.12 se obtienen las mismas conclusiones que con las Figuras 7.9 y 7.10. La red que genera el modelo de menor error absoluto de validación es la codificador-decodificador LSTM; mientras que los demás modelos proporcionan precisiones muy parejas.

En lo que respecta al proceso de optimización de hiperparámetros, podemos concluir que mejora notablemente los resultados que proporcionan todas las redes, y que también varía la precisión relativa de cada una de ellas. De las Figuras 7.14 y 7.16 concluimos que el modelo con mejores resultados es el LSTM, tanto en pérdida como en error absoluto medio. La peor red vuelve a ser la CNN-LSTM codificador-decodificador, que, junto con la LSTM codificador-decodificador y la CNN, presenta sobreajuste.

Podemos achacar los problemas de la red CNN-LSTM codificador-decodificador a su complejidad: la hace más propensa al sobreajuste, y su alto número de hiperparámetros provoca que su optimización requiera más tiempo que las demás redes, que no le hemos dedicado. Resulta extraño, sin embargo, que el modelo codificador-decodificador LSTM tenga los mejores resultados, teniendo en cuenta que solo tiene un hiperparámetro menos.

8.3. Predicciones de los modelos

Obtener conclusiones generales sobre el rendimiento de los distintos modelos a partir de las gráficas de desplazamientos (Figuras 7.17 y 7.18) no tiene sentido, ya que algunos modelos pueden tener buenos resultados en ciertos días, pero malos en otros. Sí que podemos darnos cuenta, de que, salvo las últimas semanas, todos los modelos son capaces de ajustarse muy bien a los descensos y ascensos de cada fin de semana, dado que probablemente sea la correlación más simple entre la demanda y un factor externo.

De las gráficas de errores (Figuras 7.20a a 7.20f) cabe notar, primeramente, que los errores tienen la apariencia de una campana de Gauss, con un centro bien definido y una cierta simetría. En cuanto a los coeficientes estadísticos de los errores (Cuadro 7.2), vemos que los errores absolutos medios y medianos son de varias decenas de desplazamientos diarios, en comparación con la media de préstamos en las 16 semanas bajo estudio, de 148 préstamos diarios. Tenemos, por tanto, que los errores medios y medianos relativos son grandes: de entre el 21,89 % y el 29,74 % en el caso de los errores absolutos medios, y de entre el 14,96 % y el 26,07 % en el caso de los errores absolutos medianos.

Todas las redes mejoran varios parámetros estadísticos tras el proceso de optimización de hiperparámetros, aunque, en el caso de la CNN, el error absoluto medio y la raíz del error cuadrático medio aumentan; y, para la red MLP, aumenta la desviación media del error. En lo que respecta a error absoluto medio y mediano, el modelo con mejores resultados es el MLP optimizado. La red MLP optimizada también ofrece los mejores resultados en lo que respecta a la raíz del error cuadrático medio, y presenta la segunda menor desviación media del error, por lo que podemos considerarlo como el modelo de mayor precisión. Por otro lado, el modelo

de peor precisión es el RNN no optimizado, por tener errores absolutos medio y mediano muy superiores a los de los demás modelos.

Recordemos que el modelo MLP optimizado, a pesar de su buena precisión, es el que presenta una mayor pérdida de entrenamiento en la Figura 7.13, y no presenta errores absolutos medios de validación destacables frente a los demás modelos en la Figura 7.16. Para valorar la calidad de un modelo, no basta con observar únicamente su validez en los datos de entrenamiento y testeo, si no también realizar predicciones y compararlos con unos datos de validación.

Algunos de las redes tienen mejoras pequeñas cuando se optimizan sus hiperparámetros, lo que indica que se debería haber dedicado más tiempo al proceso de optimización de hiperparámetros.

Capítulo 9

Conclusiones y desarrollos futuros

9.1. Conclusiones

En este Trabajo Fin de Máster, hemos empleado distintas técnicas de aprendizaje profundo, como las RNN, las LSTM, las MLP, las CNN y modelos híbridos, para tareas de regresión y predicción de series temporales, específicamente en lo que respecta a demanda de un servicio de préstamo de bicicletas.

Hemos hecho uso de datos recogidos por la empresa proveedora del servicio, de información de eventos y días festivos y de datos climatológicos, que hemos tenido que tratar por existir datos faltantes. El hecho de usar datos de otras estaciones cercanas a la de Salamanca difícilmente puede haber tenido un efecto demasiado grave a la calidad de los modelos, dado que son pocos los datos que hemos tenido que sustituir.

La selección de variables (temperatura media diaria, precipitaciones totales diarias, velocidad media del viento, si el día es festivo o si hubo un evento, si el día es fin de semana y las vacaciones universitarias) ha resultado ser buena, y ninguno de los factores es despreciable, como podemos comprobar en el análisis previo de datos que hemos realizado.

También hemos podido comparar la conveniencia de cada uno de los modelos planteados a la hora de realizar predicciones, hemos verificado la gran importancia de realizar un buen ajuste de hiperparámetros, pues pueden mejorar en mucho la precisión de los modelos, y hemos valorado qué modelos pueden resultar más útiles en la práctica.

Consideramos que los modelos planteados, tras ser ligeramente modificados para adaptarse a periodos temporales que permitan la previsión meteorológica, podrían resultar de gran utilidad a empresas proveedoras de servicios de préstamo de bicicletas, pues disponer de capacidad de previsión puede permitirles optimizar factores como la selección de fechas para reparar el material o el disponer de un número suficiente de bicicletas en las estaciones.

En resumen, podemos concluir que el aprendizaje profundo es una herramienta poderosa y útil a la hora de predecir la demanda de servicios de préstamo de bicicletas, que la demanda se puede predecir a partir de datos climáticos y sociales y que es importante plantear cuidadosamente los modelos que vayamos a usar.

9.2. Desarrollos futuros

La predicción de la demanda de cualquier servicio es un tema amplio y profundo que brinda un campo de investigación y desarrollo muy vasto. Son muchos los enfoques y técnicas que haber empleado para mejorar la calidad de nuestras predicciones y hacerlas más aplicables a la realidad, de haber contado con más tiempo y capacidad computacional.

Primeramente, podríamos haber implementado arquitecturas de aprendizaje profundo más avanzadas. El desarrollo de la investigación en aprendizaje profundo ha avanzado rápidamente, y se han puesto arquitecturas más complejas que pueden ser usadas en problemas de regresión y predicción de series temporales. Por ejemplo, en [21], los autores combinan modelos GAN, que entran dentro del aprendizaje semiautomático, con redes CNN y puertas difusas en tareas de regresión; en [22] se utilizan modelos GAN para predecir el deterioro de freidoras de aceite; [23] muestra el uso de transformadores de fusión temporal para predicción de series temporales a largo plazo...

Hasta donde sabemos, ninguno de estos tres métodos se ha llegado a aplicar problemas relacionados con la predicción de la demanda de bicicletas, por lo que sería un enfoque innovador al problema. La implementación de modelos más avanzados podría mejorar la precisión y capacidad de adaptación de los modelos, proporcionando así resultados más confiables y precisos.

En lo que respecta a los datos usados, combinar información de servicios de préstamo de bicicletas de distintas ciudades, si tuviesen características geográficas, climáticas y sociales similares, podría proporcionarnos una visión más completa de las variables que afectan la demanda de bicicletas y mejorar la precisión de las predicciones. Esto puede ser problemático, pues pueden aparecer sesgos debido a factores de las otras ciudades que no estemos considerando.

Otra posibilidad es hacer uso de datos de otros servicios de la ciudad de Salamanca, por ejemplo, el transporte público en autobús. Esto puede resultar también problemático, pues algunos factores que reducen los desplazamientos en bicicleta pueden aumentar los que suceden en transporte público (por ejemplo, las precipitaciones), aunque sí podría ser posible detectar algunos patrones compartidos en eventos, festivos...

En lo que respecta al enfoque temporal, en nuestro caso, hemos tomado como tiempo de predicción un total de 16 semanas, ya que es una cantidad de tiempo lo suficientemente larga como para permitir comprobar la precisión de los modelos, aunque no es lo suficientemente corta como para poder disponer de información meteorológica. Hemos considerado la ciudad como un todo, sin realizar predicciones de lo que sucede en cada estación.

Otros enfoques temporales más largos o cortos también podrían servir de utilidad para la empresa proveedora del servicio. Un enfoque temporal de mayor alcance posibilitaría la elaboración de planes a largo plazo, mientras que un enfoque a corto plazo (de una semana, por ejemplo) podría ser útil para realizar las tareas diarias de dotar a las distintas estaciones de bicicletas. A priori, todo esto podría hacerse de manera sencilla modificando las entradas y salidas de los modelos. El mayor problema de una perspectiva a largo plazo sería la obtención de datos meteorológicos.

Otra posibilidad sería enfocar el problema realizando predicciones hora a hora y estación a estación. Esto resultaría tremendamente útil para el servicio, pues les permitiría retirar bicicletas de las estaciones donde no sean necesarias y colocarlas donde sí. Es posible que este enfoque no se pueda realizar con los pocos datos de los que disponemos.

Finalmente, podríamos haber realizado una interpretación de los modelos. En aprendizaje profundo es habitual tratar los modelos como cajas negras debido a su complejidad: nos enfocamos sus predicciones, sin pararnos a analizar cómo funciona exactamente el modelo. La interpretación de modelos busca describir las razones detrás de las predicciones realizadas por estos modelos. Existen multitud de métodos que nos permiten obtener un mayor entendimiento de las decisiones tomadas por un modelo de aprendizaje profundo. Estas técnicas buscan responder preguntas como "¿por qué el modelo hizo esta predicción?" o "¿qué características son más importantes para el modelo?". En nuestro caso, no hemos realizado ningún proceso de interpretación de los modelos, aunque podría resultar de gran interés tanto teórico como práctico.

Apéndice A

Códigos empleados

No podemos incluir los códigos que hemos programado en esta memoria por ser demasiado extensos, pero se pueden consultar desde Google Colaboratory a través de este enlace. En este cuaderno aparecen multitud de figuras que no se han incluido en esta memoria por no aportar demasiado al tratamiento y análisis del problema. El entrenamiento de todos los modelos puede tomar varias horas.

Hagamos una breve descripción de todas las librerías que usamos.

pandas: Se trata de una librería especializada en la manipulación y el análisis de datos, y permite estructurar y operar con datos organizados en tablas numéricas y series temporales [24].

NumPy: Permite la creación de vectores y matrices grandes multidimensionales, y ofrece multitud de funciones matemáticas para operar con ellas [25].

Matplotlib: Es una biblioteca para la creación de gráficos en dos dimensiones [26].

seaborn: Basada en Matplotlib, permite la creación de gráficos estadísticos [27].

Statistics: Cuenta con multitud de funciones para cálculos estadísticos de datos numéricos [28].

Math: Ofrece algunas funciones matemáticas [29]. Nosotros usamos las funciones math.ceil(x), que devuelve el entero más pequeño mayor o igual que x, y math.isnan, para buscar celdas sin datos.

Requests: Permite enviar peticiones HTTP/1.1 [30]. La usamos para descargar de Github las tablas de datos.

Datetime: Contiene funciones y clases para manipular fechas y tiempos [31].

JSON: Es una librería codificadora y decodificadora [32]. Nosotros únicamente la usamos para convertir listas en diccionarios.

Keras, TensorFlow y Sklearn: Tres módulos para el aprendizaje automático. La mayor parte de los modelos están programados con Keras [11], pero también usamos TensorFlow [33] para implementar el método Adam y Sklearn [34] para calcular errores cuadráticos medios.

Bibliografía

[1] Jenny Douch, "Análisis de los sistemas de bicicletas compartidas en España: sistemas públicos", Observatorio de la Bicicleta Pública en España, 2019. Disponible en: https://bicicletapublica.es/2019/04/27/estado-de-los-sistemas-de-bicicleta-compartida-en-espana/ (acceso: 9 de julio de 2023).

[2] G. Leonardo Ferreira, "An overview of artificial neural networks for mathematicians", Univ. Chicago, 2018. Disponible en: https://math.uchicago.edu/~may/REU2018/REUPapers/Guilhoto.pdf (acceso: 9 de julio de 2023).

[3] Área de Medio Ambiente del Ayuntamiento de Salamanca. "Sistema de préstamo de bicicletas *SALenBICI*". Sistema de préstamo de bicicletas "SALenBICI". [En línea]. Disponible en: medioambiente.aytosalamanca.es/es/movilidadsostenible/sistemadeprestamodebiciletassalenbici/ (acceso: 9 de julio de 2023).

[4] W. Wang, X. Zhao, Z. Gong, Z. Chen, N. Zhang y W. Wei, "An Attention-Based Deep Learning Framework for Trip Destination Prediction of Sharing Bike", *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4601-4610, jul. 2021. [En línea]. Disponible en: 10.1109/TITS.2020.3008935. Acceso: julio 2023.

[5] X. Chang, Z. Feng, J. Wu, H. Sun, G. Wang y X. Bao, "Understanding and Predicting the Short-Term Passenger Flow of Station-Free Shared Bikes: A Spatiotemporal Deep Learning Approach", *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 4, pp. 73-85, jul.-ago. 2022. [En línea]. Disponible en: 10.1109/MITS.2021.3049362. Acceso: julio 2023.

[6] C. Xu, J. Ji y P. Liu, "The station-free sharing bike demand forecasting with a deep learning approach and large-scale datasets", *Transportation Research Part C: Emerging Technologies*, vol. 905, pp. 47-60, oct. 2018. [En línea]. Disponible en: 10.1016/j.trc.2018.07.013. Acceso: julio 2023.

[7] K. Gebhart y R. B. Noland, "The impact of weather conditions on bikeshare trips in Washington, DC", *Transportation*, vol. 41, pp. 1205–1225, nov. 2014. [En línea]. Disponible en: 10.1007/s11116-014-9540-7. Acceso: julio 2023.

[8] A. A. Campbell, C. R. Cherry, M. S. Ryerson y X. Yang, "Factors influencing the choice of shared bicycles and shared electric bikes in Beijing", *Transportation Research Part C: Emerging Technologies*, vol. 67, pp. 399-414, jun. 2016. [En línea]. Disponible en: 10.1016/j.trc.2016.03.004. Acceso: julio 2023.

[9] Instituto Nacional de Estadística, "Salamanca: Población por municipios y sexo" INEbase. [En línea]. Disponible en: https://www.ine.es/jaxiT3/Tabla.htm?t=2891 (acceso: 9 de julio de 2023).

[10] Agencia Estatal de Meteorología, "Datos climatológicos" Servicios climáticos. [En línea]. Disponible en: https://www.aemet.es/es/serviciosclimaticos/datosclimatologicos (acceso: 9 de julio de 2023).

[11] F. Chollet y otros, "Keras", 2015. Disponible en: https://keras.io (acceso: 9 de julio de 2023).

[12] A. Shrestha y A. Mahmood, "Review of Deep Learning Algorithms and Architectures", *IEEE Access*, vol. 7, pp. 53040-53065, 2019. [En línea]. Disponible en: 10.1109/ACCESS.2019.2912200. Acceso: julio 2023.

[13] A. Hirose, *Complex-Valued Neural Networks*. 2ª ed. Tokyo: Springer, 2012.

[14] H. Brezis, *Functional Analysis, Sobolev Spaces and Partial Differential Equations*. Universitext. Nueva York: Springer, 2010.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Math. Control Signal Systems*, vol. 2, pp. 303–314, dic. 1989. [En línea]. Disponible en: 10.1007/BF02551274. Acceso: julio 2023.

[16] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, nov. 1998. [En línea]. Disponible en: 10.1109/5.726791. Acceso: julio 2023.

[17] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. 1ª ed. Cambridge: The MIT Press, 2016.

[18] T. Hastie, R. Tibshirani y J. Friedman, *The Elements of Statistical Learning*. 2ª ed. Stanford: Springer, 2009.

[19] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[20] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery, 2018.

[21] R. Nguyen, S. Kumar Singh y R. Rai, "FuzzyGAN: Fuzzy generative adversarial networks for regression tasks", *Neurocomputing*, vol. 525, pp. 88-110, mar. 2023. [En línea]. Disponible en: 10.1016/j.neucom.2023.01.015. Acceso: julio 2023.

[22] K. Ye, Z. Wang, P. Chen, Y. Piao, K. Zhang, S. Wang, X. Jiang y X. Cui, "A novel GAN-based regression model for predicting frying oil deterioration", *Sci Rep*, vol. 12, no.1, pp. 2045-2322, jun. 2022. [En línea]. Disponible en: 10.1038/s41598-022-13762-5. Acceso: julio 2023.

[23] B. Lim, S. O. Arik, N. Loeff y T. Pfister, "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting", *International Journal of Forecasting*, vol. 37, iss. 4, pp. 1748-1764, oct.-dic. 2021. [En línea]. Disponible en: 10.1016/j.ijforecast.2021.03.012. Acceso: julio 2023.

[24] Equipo de desarrollo de pandas, "pandas-dev/pandas: Pandas", *Zenodo*, feb. 2020. [En línea]. Disponible en: 10.5281/zenodo.3509134. Acceso: julio 2023.

[25] C. R. Harris. et al., "Array programming with NumPy", *Nature*, vol. 585, n.º 7825, pp. 357-362, sep. 2020. [En línea]. Disponible en: 10.1038/s41586-020-2649-2. Acceso: julio 2023.

[26] J. D. Hunter, "Matplotlib: A 2D graphics environment", *Computing in Science & Engineering*, vol. 9, n.º 3, pp. 90-95, 2007. [En línea]. Disponible en: https://doi.org/10.1109/MCSE.2007.55. Acceso: julio 2023.

[27] M. L. Waskom, "seaborn: statistical data visualization", *Journal of Open Source Software*, vol. 6, n.º 60, p. 3021, 2021. [En línea]. Disponible en: https://doi.org/10.21105/joss.03021. Acceso: julio 2023.

[28] Equipo de desarrollo de Statistics. "Statistics". Statistics. Disponible en: https://github.com/python/cpython/blob/3.11/Lib/statistics.py (acceso: 9 de julio de 2023).

[29] Python Software Foundation. "Math". Math. Disponible en: https://docs.python.org/3/library/math.html (acceso: 9 de julio de 2023).

[30] K. Reitz, Python Software Foundation. "Requests". Requests. Disponible en: https://pypi.org/project/requests/ (acceso: 9 de julio de 2023).

[31] Equipo de desarrollo de datetime. "datetime". datetime. Disponible en: https://docs.python.org/3/library/datetime.html (acceso: 9 de julio de 2023).

[32] Equipo de desarrollo de json. "json". json. Disponible en: https://github.com/python/cpython/blob/3.11/Lib/json/_init_.py (acceso: 9 de julio de 2023).

[33] M. Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". TensorFlow. Disponible en: https://www.tensorflow.org/ (acceso: 9 de julio de 2023).

[34] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011. [En línea]. Disponible en: https://doi.org/10.48550/arXiv.1201.0490. Acceso: julio 2023.