Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA DE SEGOVIA

Grado en Ingeniería Informática de Servicios y Aplicaciones

Análisis de diferentes modelos de aprendizaje automático para estimar los tiempos de llegada de las aeronaves

---

Alumno: Miguel de Santiago Gilsanz

Tutores: Miguel Ángel Martínez Prieto
Jorge Silvestre Vilches

Análisis de diferentes modelos de aprendizaje automático para estimar los tiempos de llegada de las aeronaves

Miguel de Santiago Gilsanz

Agradecimientos

En primer lugar, me gustaría dar las gracias a mis tutores, Jorge Silvestre Vilches y Miguel Ángel Martínez Prieto, y al profesor Aníbal Bregón Bregón por permitirme trabajar con ellos en este proyecto y por toda su ayuda.

También me gustaría dárselas a mis padres, Luis y Esther, a mi hermana Paula, y a Eva por su apoyo constante y su cariño durante estos años, ya que sin ellos no habría sido posible llegar hasta aquí.

Y finalmente, pero no por ello menos importante, agradecer al resto de mi familia y a todos mis amigos de la parcela, del instituto, de la universidad, del tenis y del Erasmus el haberme acompañado todo este tiempo y todas las risas que hemos compartido, porque sin ellos no habría sido lo mismo.

Resumen

El sistema ADS-B (*Automatic Dependent Surveillance-Broadcast*) es una tecnología de vigilancia que las aeronaves llevan incorporada, permitiendo su seguimiento en tiempo real gracias al envío periódico de información. Una de las aplicaciones de esta información es la reconstrucción de las trayectorias 4D (latitud, longitud, altitud y tiempo) seguidas por las aeronaves en vuelo. Aplicado a un vuelo en curso, este completo perfil de vuelo nos permite evaluar la marcha de este con respecto a la información histórica disponible sobre el mismo trayecto, o sobre vuelos realizados en rutas similares. De esta manera, es posible detectar retrasos con respecto a la planificación inicial del vuelo, o estimar el tiempo de llegada de acuerdo con las condiciones actuales del vuelo en tiempo real. En este Trabajo Fin de Grado se abordará el estudio de las redes neuronales y su funcionamiento con el fin de poder entender qué son las redes neuronales LSTM (*Long Short-Term Memory*); y se implementarán en el lenguaje de programación Python varios modelos basados en redes LSTM que permitirán estimar los tiempos de llegada para vuelos con el aeropuerto Adolfo Suárez Madrid-Barajas como destino. Además, se evaluarán diferentes representaciones de la información de vigilancia recibida de las aeronaves para optimizar el desempeño de los modelos desarrollados.

Tras los ensayos realizados, queda patente que la incorporación de información adicional a las trayectorias 4D, tales como la distancia de las aeronaves al aeropuerto y su velocidad, resultan claves para obtener mejores predicciones, ya que las estimaciones de los tiempos de llegada se han mejorado hasta en 20 segundos utilizando esta información.

Este proyecto se organizará y desarrollará utilizando la metodología de enseñanza ágil UVagile, que está basada en el marco de trabajo ágil *Scrum*. Esta metodología plantea una organización iterativa e incremental del proyecto en la que el trabajo se divide en espacios temporales denominados 'sprints'. En concreto, para este proyecto se realizarán 5 *sprints*, cada uno de tres semanas, donde al final de cada *sprint* se entregará un incremento del producto en desarrollo, es decir, un prototipo que incluirá un subconjunto completo de requisitos y la memoria técnica correspondiente. Esta forma de trabajo permitirá que la carga de trabajo de cada *sprint* se adecúe al ritmo real del proyecto, y derivará en un resultado final que se ajustará más y mejor a los objetivos propuestos al inicio del proyecto.

Palabras claves: aprendizaje automático, redes neuronales LSTM, redes convolucionales, UVagile, *sprint*.

Abstract

The ADS-B system (*Automatic Dependent Surveillance-Broadcast*) is a surveillance technology that aircraft have incorporated, allowing its monitoring in real time thanks to the periodic sending of information. One of the applications of this information is the reconstruction of the 4D trajectories (latitude, longitude, altitude and time) followed by the aircraft in flight. Applied to a flight in progress, this complete flight profile allows us to evaluate its progress with respect to the historical information available on the same route, or on flights carried out on similar routes. In this way, it is possible to detect delays with respect to the initial flight planning, or to estimate the arrival time according to the current flight conditions in real time. In this Final Degree Project, the study of neural networks and their operation will be addressed in order to understand what LSTM (*Long Short-Term Memory*) neural networks are; and several models based on LSTM networks will be implemented in the Python programming language that will allow estimating arrival times for flights with Adolfo Suárez Madrid-Barajas airport as destination. In addition, different representations of the surveillance information received from the aircraft will be evaluated to optimize the performance of the models developed.

After the tests carried out, it is clear that the incorporation of additional information to the 4D trajectories, such as the distance of the aircraft to the airport and their speed, are key to obtaining better predictions, since the estimates of arrival times have been improved. up to 20 seconds using this information.

This project will be organized and developed using the UVagile agile teaching methodology, which is based on the agile Scrum framework. This methodology proposes an iterative and incremental organization of the project in which the work is divided into temporary spaces called 'sprints'. Specifically, for this project, 5 sprints will be carried out, each lasting three weeks, where at the end of each sprint an increment of the product under development will be delivered, that is, a prototype that will include a complete subset of requirements and the corresponding technical memory. This way of working will allow the workload of each sprint to adapt to the real rhythm of the project, and will lead to a final result that will adjust more and better to the objectives proposed at the beginning of the project.

Keywords: machine learning, LSTM neural networks, convolutional networks, UVagile, sprint.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Desde los primeros diseños de un avión realizados por Leonardo da Vinci en el siglo XV, pasando por el primer avión autopropulsado creado en 1890 por el francés Clément Ader [1]; este medio de transporte ha ido evolucionando progresivamente hasta ser hoy en día uno de los más seguros [2]. Además, en las últimas décadas, el transporte aéreo anual de pasajeros ha experimentado un crecimiento exponencial (Figura 1.1), y con ello, el número de vuelos diarios que se realizan, por lo que también es uno de los medios más utilizados en todo el mundo.

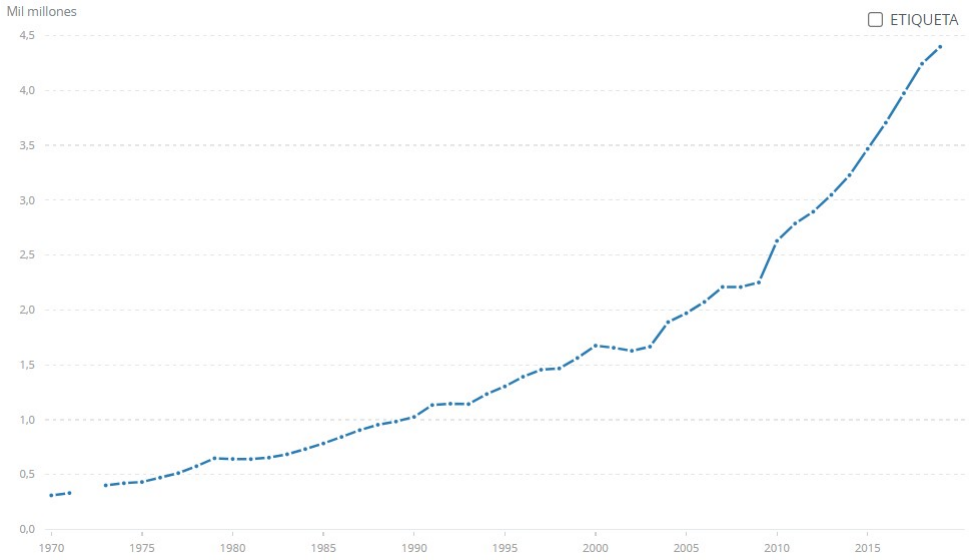


Figura 1.1: Número de pasajeros anuales trasportados al año desde 1970 hasta 2019. Fuente: [3].

Sin embargo, en el año 2020, como consecuencia del virus SARS-CoV-2, se produjo una disminución del tráfico aéreo de pasajeros (y del número de vuelos) en todo el mundo debido a las restricciones de movilidad existentes. Concretamente, por los aeropuertos españoles solo pasaron 7,6 millones de pasajeros y solo se registraron 1,1 millón de vuelos, lo que supone un 72,4 % menos y un 53,4 % menos, respectivamente, que en el año

2019 [4]. A pesar de esto, se espera un aumento significativo de las cifras de pasajeros y vuelos diarios en los próximos años, por lo que los aeropuertos deben contar con una infraestructura y una organización apropiadas para su correcto funcionamiento, y se hace necesario una correcta Gestión del Tráfico Aéreo o ATM (por sus siglas en inglés *Air Traffic Management*).

Además de esto, una pieza clave para mejorar la seguridad, la capacidad y la fluidez del espacio aéreo, así como para facilitar la gestión del tráfico aéreo, incrementar las ganancias de las aerolíneas y reducir el impacto medioambiental, es la predicción precisa de la hora de llegada de los vuelos comerciales [5]. Esta predicción permitiría lograr una mayor previsibilidad de la situación futura del espacio aéreo, lo que derivaría en una mejor adaptación a los cambios de planificación y una mejor gestión y organización del espacio aéreo.

Sin embargo, esta tarea resulta muy compleja debido a la naturaleza variable de los factores ambientales o del resto del tráfico aéreo, puesto que el tiempo meteorológico o la congestión del espacio aéreo pueden afectar a los aterrizajes, a las rutas o a los despegues de las aeronaves (Figura 1.2). A pesar de esta complejidad, las predicciones realizadas deben ser lo más precisas posibles, ya que si son demasiado inexactas, pueden provocar retrasos en vuelos posteriores que derivarán en conflictos entre pasajeros y compañías aéreas generando pérdidas económicas y limitando el desarrollo de la industria aérea, o en situaciones de riesgo para la seguridad [6, 7].

Figura 1.2: Tráfico aéreo sobre la Península Ibérica el 14 de abril de 2021 a las 14:00h. Fuente: [8].

1.1. Motivación

En la actualidad, el sistema de transporte aéreo tiene un papel cada vez más importante en las economías locales, nacionales e internacionales [9]. Esto se aprecia notablemente en el caso de España, no solo porque más del 80 % del turismo extranjero que llega al país lo hace por vía aeroportuaria, sino porque el turismo representa un sector clave en la economía española, ya que genera dos millones de empleos anuales y representa el 11 % del PIB (Producto Interior Bruto) del país. Además, cabe mencionar que España es el segundo país del mundo (solo por detrás de Estados Unidos) en volumen de ingresos procedentes del turismo internacional [10].

A pesar de que los retrasos en los vuelos pueden ser provocados por múltiples factores, se puede intentar predecir el tiempo de llegada de las aeronaves. Esta predicción permitiría reducir los retrasos en vuelos posteriores y las pérdidas millonarias para las aerolíneas que dichas demoras conllevan. Además, esto también permitiría impulsar (aún más) el turismo internacional, ya que los pasajeros podrían organizar sus viajes de manera más razonable de acuerdo con la hora estimada de llegada de los vuelos, lo que ahorraría en gran medida su tiempo de espera, y les generaría mayor satisfacción para seguir utilizando el medio aéreo en futuros viajes.

Hoy en día existen diversas aproximaciones para hacer predicciones en los tiempos de llegada de los vuelos comerciales. Entre las principales propuestas, destacan los métodos basados en redes neuronales LSTM (*Long Short-Term Memory*) para predecir los retrasos de estos vuelos empleando para ello datos históricos de tráfico aéreo de salidas y llegadas de los aeropuertos (debido a que este tipo de redes permiten explotar las características temporales de estos datos). Sin embargo, en muchos de estos ensayos no se utilizan los datos espaciales y temporales en ruta de las aeronaves, lo cual genera predicciones inexactas [5].

Por este motivo, aprovechando los mensajes ADS-B (por sus siglas en inglés Automatic Dependent Surveillance - Broadcast) que las aeronaves en ruta envían de forma automática y periódica con información de su posición, altitud, velocidad, etc., se quiere construir un modelo de red neuronal capaz de extraer de estos mensajes las dependencias espaciales y temporales, logrando así una predicción más precisa a la hora de estimar el tiempo de llegada de las aeronaves. Esto derivaría principalmente en una mejora en la seguridad aérea y en la eficiencia en la gestión del tráfico aéreo; asociado a un importante impulso del sector aéreo, y una reducción en el impacto medioambiental y en las pérdidas económicas de las aerolíneas.

1.2. Objetivos y restricciones

El presente proyecto se enmarca en la problemática asociada a la predicción de los tiempos de llegada de las aeronaves con destino aeropuerto Adolfo Suárez Madrid-Barajas empleando para ello los mensajes ADS-B procedentes de dichas aeronaves. En esta línea de trabajo se identifican varios objetivos:

- **OBJ-1:** Estudiar cómo se lleva a cabo la gestión del tráfico aéreo en España, así como los aspectos más importes del aeropuerto Adolfo Suárez Madrid-Barajas.
- **OBJ-2:** Realizar un análisis sobre las redes neuronales, en particular sobre las redes neuronales LSTM, y los principales parámetros e hiperparámetros que afectan a su configuración y su entrenamiento.
- **OBJ-3:** Desarrollar modelos predictivos basados en redes neuronales LSTM adaptados a la operativa del aeropuerto Adolfo Suárez Madrid-Barajas. Este objetivo se subdivide en los siguientes subobjetivos para facilitar su consecución:
  - **OBJ-3.1:** Implementar un modelo utilizando solo neuronas LSTM similar al propuesto en [11] para poder ser comparado, tanto con este estudio, como con los nuevos modelos a diseñar.
  - **OBJ-3.2:** Construir un nuevo modelo basado en redes neuronales LSTM que permita explotar las dependencias, tanto temporales como espaciales, de los datos.
  - **OBJ-3.3:** Elaborar un estudio sobre el impacto que tiene utilizar diferentes características para predecir el tiempo de llegada de las aeronaves.
- **OBJ-4:** Efectuar predicciones con los modelos construidos sobre diferentes conjuntos de datos y evaluar los resultados obtenidos.

Las principales limitaciones que tenemos para alcanzar estos objetivos y que pueden afectar al correcto desarrollo del proyecto son los siguientes:

- Alcance y duración condicionados: este proyecto se desarrolla como parte de la asignatura 'Trabajo Fin de Grado' del Grado de Ingeniería Informática de Servicios y Aplicaciones de Segovia (Universidad de Valladolid). Por esta razón, tanto el alcance como la duración de este están limitados por la carga de trabajo que establece dicha asignatura.
- Conjunto de datos reducido: aunque el volumen de datos generados durante la gestión del tráfico aéreo es enorme, el entrenamiento de redes neuronales con grandes volúmenes de datos supone un reto de escalabilidad que no podemos asumir con los recursos disponibles para el proyecto. Por ello, se va a trabajar con una muestra reducida de los datos (2-8 de febrero de 2018), ya que contar con varios días nos permitirá aprovechar en parte el potencial de las redes LSTM para encontrar patrones o periodicidad en los vuelos (aunque es probable que se puedan obtener mejores predicciones considerando un período de tiempo mayor). Además, nos vamos a centrar en un único aeropuerto para facilitar el modelado de las trayectorias y la efectividad del modelo de predicción. Esto se debe a que si utilizan más aeropuertos, se tendría una mayor variabilidad de los datos, por lo que la red neuronal requeriría de una cantidad de datos mucho mayor para su correcto entrenamiento.

- Falta de receptores ADS-B sobre el océano Atlántico: su inexistencia implica la pérdida de todos los mensajes ADS-B de vuelos transatlánticos mientras estos atraviesan el océano.

1.3. Organización del documento

Los capítulos que conforman este documento y su contenido son:

- **Capítulo 1. Introducción:** se establecerá el contexto en el que se desarrolla este proyecto, así como la motivación del mismo y los objetivos que se pretenden conseguir tras su finalización.
- **Capítulo 2. Dominio del problema:** se explicará en qué consiste la gestión del tráfico aéreo y la tecnología ADS-B. También se analizará el tráfico aéreo del aeropuerto Adolfo Suárez Madrid-Barajas y se expondrán los estudios previos sobre los que se basa este proyecto.
- **Capítulo 3. Background:** se definirá qué son el aprendizaje automático y las redes neuronales, y se profundizará en este último concepto. Además, se presentarán algunos ejemplos de redes neuronales que se usarán en los siguientes capítulos.
- **Capítulo 4. Metodología de trabajo:** se describirá la metodología, las herramientas y las tecnologías empleadas durante el desarrollo del proyecto.
- **Capítulo 5. Planificación:** se presentarán las tareas a realizar durante el transcurso del proyecto y su planificación temporal. Además, también se mostrará el desglose de los costes asociados al proyecto, y se expondrá el balance final (tanto temporal como de costes) tras la finalización del mismo.
- **Capítulo 6. Implementación de los modelos:** se detallarán los conjuntos de datos de los que se dispone, el proceso llevado a cabo para construir los modelos de aprendizaje, y los análisis realizados para comparar su precisión.
- **Capítulo 7. Evaluación de los modelos:** se probarán los modelos entrenados en el capítulo anterior sobre nuevos conjuntos de datos para evaluar realmente su precisión.
- **Capítulo 8. Conclusiones y trabajo futuro:** se enunciarán las conclusiones obtenidas, las posibles líneas de trabajo futuras que podrían abordarse más adelante, y el aprendizaje personal que ha supuesto el proyecto.
- **Apéndices:** se hallará información adicional relativa al desarrollo del proyecto y a la entrega del mismo. Estará conformado por:
  - **Apéndice A. Resultados de las retrospectivas:** se mostrarán los resultados de las retrospectivas llevadas a cabo durante el desarrollo del proyecto.

- **Apéndice B. Experimentos para la optimización de los modelos:** se explicarán los experimentos llevados a cabo para optimizar los modelos construidos.
- **Apéndice C. Contenido adjunto:** se detallará el material entregado junto a esta memoria y su explicación.
- **Apéndice D. Acrónimos y siglas:** recogerá los acrónimos y siglas que se utilizan en este documento.

Capítulo 2

Dominio del problema

En este capítulo se explicará en qué consiste la gestión del tráfico aéreo (Sección 2.1) y la tecnología ADS-B (Sección 2.2). Asimismo, se evaluará el tráfico aéreo del aeropuerto Adolfo Suárez Madrid-Barajas (Sección 2.3), y se analizarán los principales trabajos previos que suponen el punto de partida de este proyecto (Sección 2.4).

2.1. Gestión del tráfico aéreo

La gestión del tráfico aéreo o ATM conforma la base técnica de la navegación aérea, ya que hace referencia a todos los sistemas y procedimientos que facilitan el despegue y aterrizaje de las aeronaves en un aeródromo, y su tránsito por el espacio aéreo. Entre estos sistemas se incluyen el *Control de Tráfico Aéreo* o ATC (por sus siglas en inglés *Air Traffic Control*), el *Servicio de Tráfico Aéreo* o ATS (del inglés *Air Traffic Services*), la *Gestión del Espacio Aéreo* o ASM (acrónimo de *Airspace Management*), y la *Gestión del Flujo y la Capacidad del Tráfico Aéreo* o ATFCM (de su nombre en inglés *Air Traffic Flow and Capacity Management*) [12].

El crecimiento exponencial que ha experimentado el tráfico aéreo en las últimas décadas y su esperado incremento en los próximos veinte años, hacen que esta gestión sea crucial e indispensable para mejorar las rutas de vuelo existentes, ya que las rutas ineficientes provocan una mayor contaminación acústica y aérea, un mayor uso de combustibles, pérdidas de productividad y retrasos que implican perdidas millonarias en el sector aéreo [13]. Sin embargo, la existencia de agentes circunstanciales externos, tales como fenómenos meteorológicos, la capacidad y estructura de los aeródromos, el volumen del tráfico aéreo o los diversos protocolos y procedimientos de control que se emplean, pueden afectar a dichos sistemas de gestión, haciendo que la labor de la gestión del tráfico aéreo adquiera un papel aún más importante y complicado.

En España, la empresa pública *Enaire*, adscrita al Ministerio de Transporte, Movilidad y Agenda Urbana, es la encargada de la gestión de la navegación aérea. Controla más de dos millones de kilómetros cuadrados de espacio aéreo, y es el proveedor oficial de servicios

de comunicaciones, navegación y vigilancia en el espacio aéreo español y en los aeropuertos de la red Aena [14].

Para ello, existen cinco centros de control situados en Madrid, Barcelona, Gran Canaria, Sevilla y Palma que prestan a todos los vuelos servicio de control en ruta y de aproximación a todos los aeropuertos del país. Asimismo, Enaire ofrece servicios de control de aeródromo a los principales aeropuertos del país; y servicios de comunicación, navegación y vigilancia a las torres de control situadas en estos aeropuertos [15].

Para prestar los servicios ATS, Enaire cuenta con el sistema *SACTA* (Sistema Automatizado de Control de Tránsito Aéreo) que se encarga de la gestión del control del tráfico aéreo en tiempo real, garantizando su fluidez y separación. Este sistema destaca por su gestión coherente y coordinada de todos los datos, ya que integra todos los centros de control de ruta, aproximación y aeródromo españoles. Además, SACTA permite la comunicación automática con centros de control extranjeros, minimiza las actuaciones manuales, descubre de forma automática posibles conflictos y aporta flexibilidad para reconfigurar el espacio aéreo operacional [16].

SACTA está en continua evolución, y su última versión (SACTA-iTEC 4.0) incorpora nuevas funcionalidades que permiten a Enaire gestionar una mayor cantidad de vuelos, mejorar la seguridad, y reducir las emisiones innecesarias. Además, esta nueva versión se encuadra dentro de la Alianza iTec, de la que forman parte los proveedores de servicios de navegación aérea de España, Reino Unido (NATS), Alemania (DFS), Lituania (Oro Navigacija), Noruega (AVINOR), Holanda (LVNL) y Polonia (PANSA). Con esta alianza se quiere lograr un tráfico aéreo más fluido, eficiente y limpio en toda Europa, con una perfecta interoperabilidad entre todos los centros de control [17, 18].

2.2. Tecnología ADS-B

La tecnología de transmisión de vigilancia dependiente automática o ADS-B (por sus siglas en inglés *Automatic Dependent Surveillance - Broadcast*) es un sistema de vigilancia empleado para gestionar el tráfico aéreo. Permite a las aeronaves enviar automáticamente y de forma periódica información de su posición (obtenida gracias a los satélites) y otros datos a cualquier otra aeronave o estación equipada con un sistema receptor ADS-B que se encuentre dentro de su radio de influencia (Figura 2.1). OpenSky Network es una red de receptores ADS-B que cuenta con el mayor conjunto de datos de vigilancia del tráfico aéreo en la actualidad ya que posee más de 25 millones de mensajes recopilados a lo largo de todo el mundo desde el año 2013 [19]. También existen otras fuentes de obtención de datos importantes como ADSBHub (que cubre principalmente la zona de Europa y EEUU) [20] o Frambuesa, receptor ADS-B propiedad de Boeing Research & Technology Europe situado en el aeropuerto Adolfo Suárez Madrid-Barajas [21].

El empleo de esta tecnología facilita la gestión del tráfico aéreo ya que permite mejorar la eficiencia y seguridad en la toma de decisiones, y obtener información más exacta y en tiempo real. Además, permite a los controladores aéreos y a los pilotos tener una visión del espacio aéreo más precisa [22].

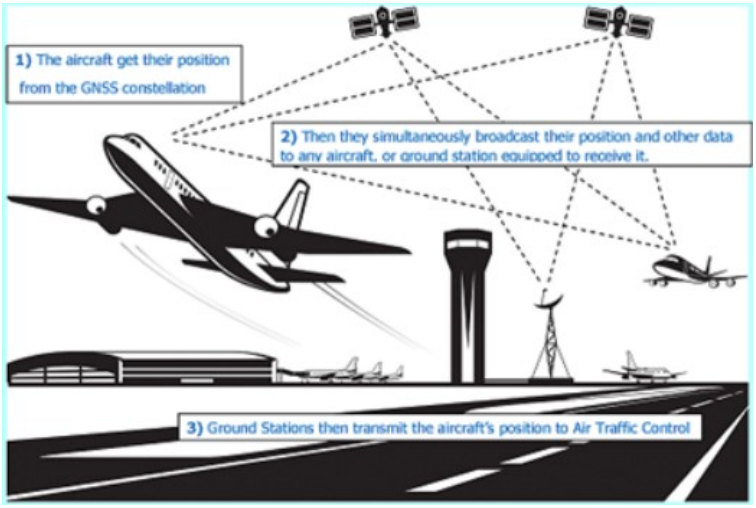


Figura 2.1: Esquema comunicación ADS-B. Fuente: [23].

El sistema ADS-B es mucho más fiable que el radar tradicional debido a que los datos son enviados por la propia aeronave. Además, la velocidad de transmisión es mayor y la precisión de la información no se degrada con la distancia o las condiciones meteorológicas [22,24]. Sin embargo, dado que está en fase de implantación, existen puntos de la superficie terrestre, como por ejemplo el océano Atlántico, donde no existen receptores de mensajes ADS-B, lo que dificulta o impide en ocasiones realizar un correcto seguimiento de los vuelos. En estas zonas, el control de las aeronaves se lleva a cabo gracias a los informes de posición emitidos por los pilotos cada 10-15 minutos [25].

Para enviar y recibir mensajes ADS-B existen dos tipos de sistemas [24, 26]:

- ADS-B OUT: encargado de emitir la información ADS-B (latitud, longitud, altitud, velocidad vertical y horizontal, número de vuelo, presión barométrica, etc.).
- ADS-B IN: encargado de recibir información procedente de otras estaciones OUT.

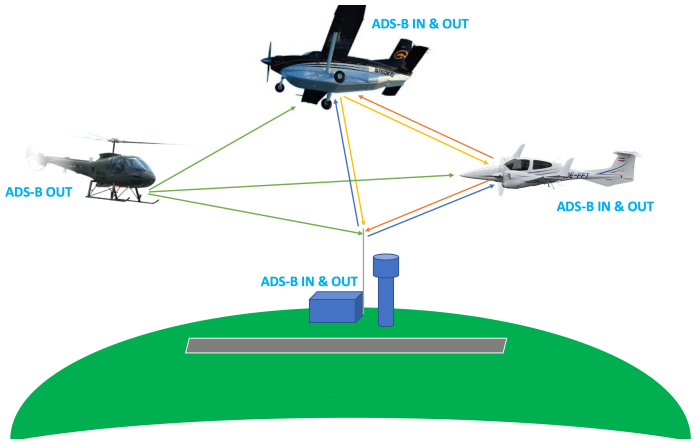


Figura 2.2: Ejemplo con la diferencia entre ADS-B OUT y ADS-B IN. Fuente: [26].

2.3. Aeropuerto Adolfo Suárez Madrid-Barajas

El aeropuerto Madrid-Barajas, o como se le conoce desde 2014, Adolfo Suárez Madrid-Barajas (código IATA: MAD, código OACI: LEMD), está situado estratégicamente en el centro de la Península Ibérica, en la Comunidad Autónoma de Madrid, y es actualmente el que gestiona más tráfico de pasajeros, carga aérea y número de operaciones en España y el 5º en Europa [27].

Localizado a una distancia de 12 km del centro de la capital madrileña y ocupando una superficie aproximada de 1.925 hectáreas, este aeropuerto cuenta con cuatro terminales de pasajeros, una terminal ejecutiva, un centro de carga aérea y dos zonas principales de hangares. Además dispone de una pista de estacionamiento y 2 parejas de pistas paralelas donde se realizan los despegues y aterrizajes. La primera pareja de pistas se denomina 14L-32R y 14R-32L, mientras que la segunda 18L-36R y 18R-36L [27, 28]. El número de estos nombres indica la dirección que apunta la pista, es decir, la orientación de la pista respecto al polo norte magnético de la Tierra (en decenas de grados magnéticos y redondeado), mientras que la letra indica la localización de la pista (izquierda, L, o derecha, R) y permite diferenciar extremos de pistas con el mismo número [29].

El empleo de las pistas en una dirección u otra viene determinado por las condiciones meteorológicas (principalmente la dirección y velocidad del viento), motivos de seguridad, o la inoperatividad de alguna pista; pero con el fin de minimizar la afección acústica sobre el entorno, el aeropuerto utiliza una configuración norte de forma preferente tanto de día como de noche como se puede ver en las Figuras 2.3 y 2.4 [28].

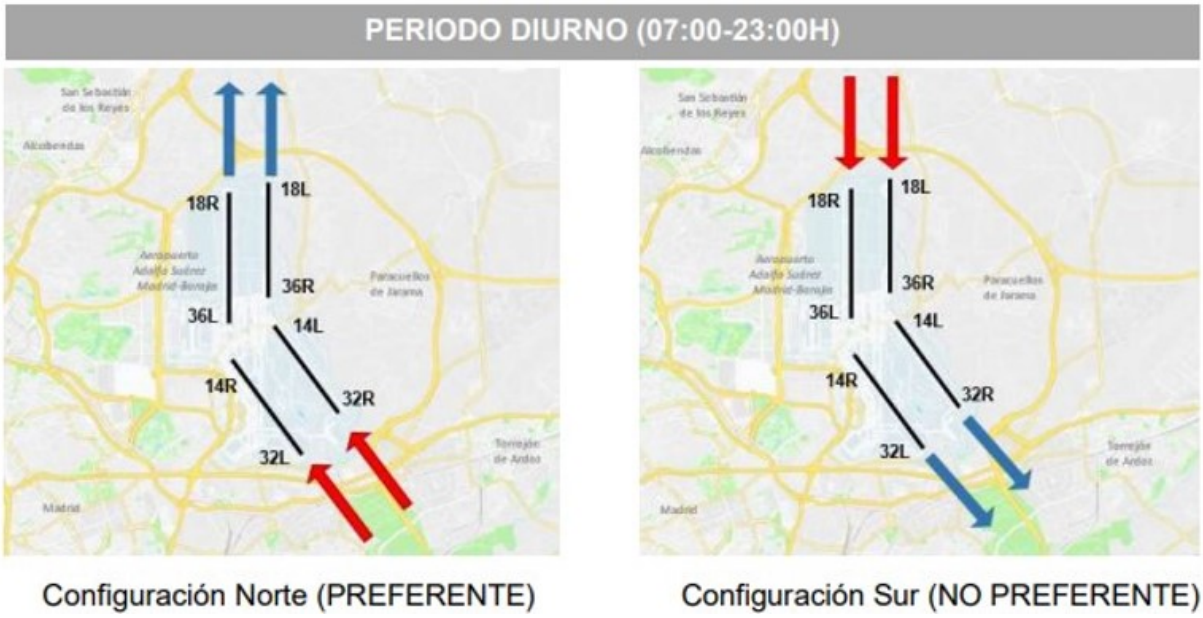


Figura 2.3: Configuración periodo diurno (07:00h-23:00h). Fuente: [28].

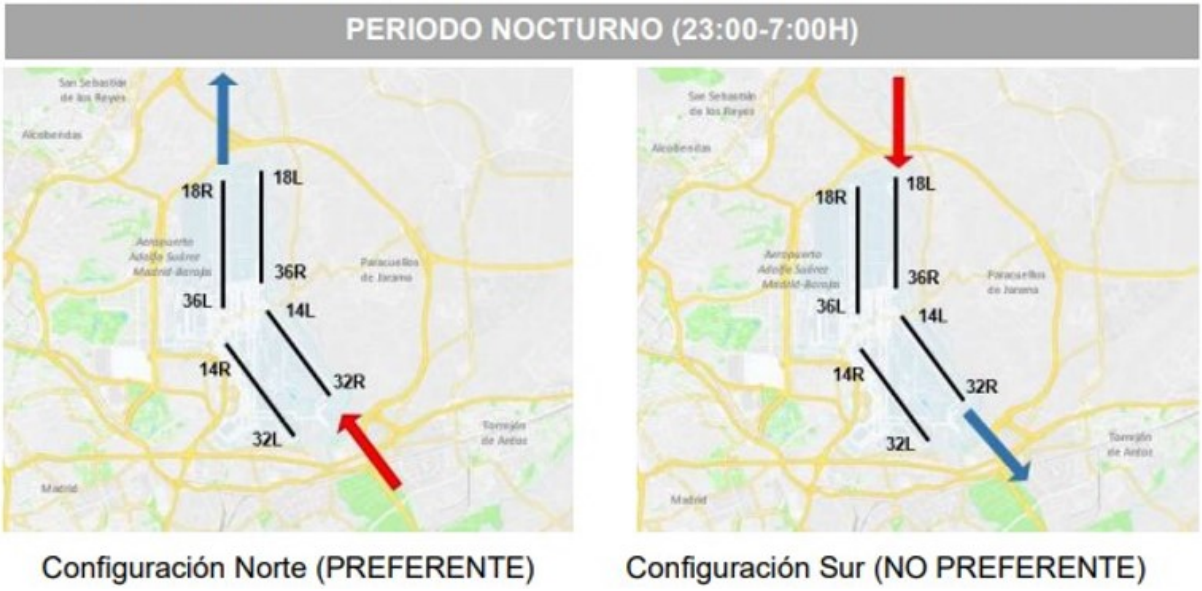


Figura 2.4: Configuración periodo nocturno (23:00h-07:00h). Fuente: [28].

Como consecuencia del virus SARS-CoV-2, el tráfico aéreo en el año 2020 se desplomó hasta niveles mínimos. Por esta razón, no se tiene en cuenta para comparar datos y se utilizan los datos del año 2019. A lo largo de ese año, este aeropuerto registró un total de 426.376 operaciones, donde casi el 50 % de ellas fueron realizadas por las compañías Iberia, Air Europa y Air Nostrum. También cabe destacar que la mayoría de ellas se realizaron en el periodo diurno (07:00h.-23:00h.) como se puede ver en la Figura 2.5 [28].

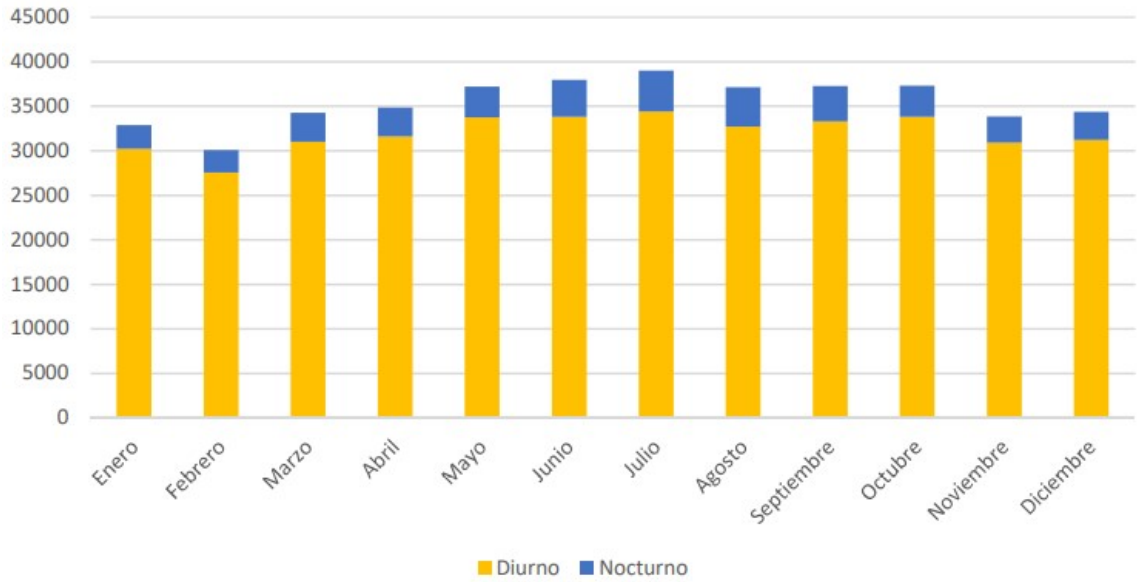


Figura 2.5: Número de operaciones mensuales dividido en periodo diurno (07:00h.-23:00h.) y nocturno (23:00h.-07:00h.). Fuente: [28].

Además, en la Figura 2.6 se puede observar el número de operaciones mensuales por configuración que se registró en el aeropuerto en el año 2019, y en la Figura 2.7 el porcentaje de uso por cada configuración dicho año.

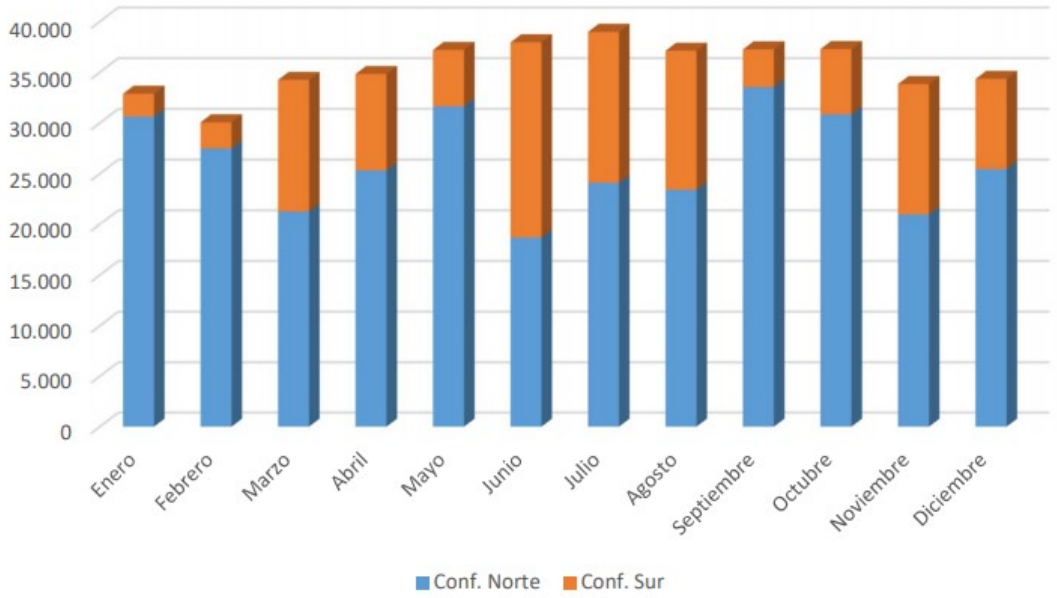


Figura 2.6: Número operaciones mensuales por configuración. Fuente: [28].

2019	DESPEGUES		ATERRIZAJES		DESPEGUES		ATERRIZAJES		% PERIODO
	36 L	36 R	32 L	32 R	14 L	14 R	18 L	18 R	
Día (07:00h-23:00h)	14,5	18,6	17,0	15,3	6,3	6,1	6,9	5,5	**90,2**
Noche (23:00h-07:00h)	3,4	0,4	0,6	3,8	0,6	0,1	0,9	0,0	**9,8**
% Conf.	**Conf. Norte: 73,6**				**Conf. Sur: 26,4**				**100,0**

Figura 2.7: Porcentaje de operaciones según configuración y periodo. Fuente: [28].

Aunque en el primer semestre del año 2021 Enaire ha gestionado un 66,5 % menos de vuelos que en el mismo periodo del año 2019 [30], el informe sobre el impacto del virus SARS-CoV-2 en el aeropuerto Adolfo Suárez Madrid-Barajas publicado el 12 de julio de 2021 por Eurocontrol (Organización Europea para la Seguridad de la Navegación Aérea que da apoyo a la Aviación Europea) [31] revela un crecimiento en el tráfico aéreo en el segundo trimestre del año 2021 que invita al optimismo, como se puede observar en la Figura 2.8.

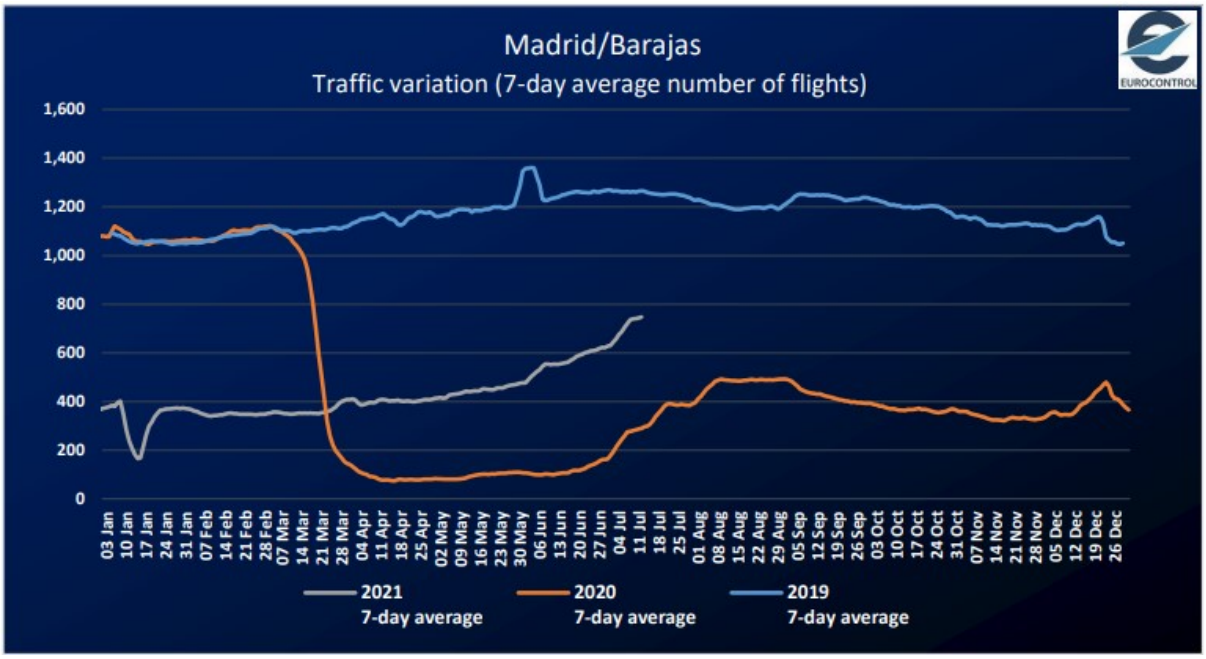


Figura 2.8: Variación del trafico aéreo en el aeropuerto Madrid-Barajas. Fuente: [32].

2.4. Estudios previos

En esta sección se va a abordar el estudio de algunas soluciones sobre las que se basará nuestro proyecto, ya que existen numerosos estudios que abordan la problemática de la predicción de tiempos de llegada de las aeronaves, y otros estudios relacionados con problemas similares que pueden ayudarnos a alcanzar los objetivos propuestos.

- ***Predicting Estimated Time of Arrival for Commercial Flights*** **de Samet Ayhan et al. (2018)** [**7**]: este artículo propone un nuevo modelo de aprendizaje automático para realizar una predicción precisa de la hora estimada de llegada en vuelos comerciales. Este modelo es capaz de aprender de las trayectorias históricas y utiliza los puntos de cuadrícula 3D pertinentes para recopilar características clave como los parámetros meteorológicos, el tráfico aéreo y los datos del aeropuerto a lo largo de la ruta de los vuelos.

  El resultado de este estudio es muy prometedor ya que el modelo propuesto es capaz de predecir con un margen de error de unos cuatro minutos el tiempo de llegada de cualquier vuelo comercial en España antes del comienzo del vuelo.

- ***Spatio-Temporal Data Mining for Aviation Delay Prediction*** **de Kai Zhang et al. (2020)** [**5**]: en este artículo se presenta un innovador sistema de predicción de retrasos en la aviación que se basa en redes LSTM apiladas. Este modelo aprende de las trayectorias históricas reconstruidas a partir de la tecnología

ADS-B y utiliza las geolocalizaciones de las aeronaves para recopilar características a tener en cuenta como el tráfico del espacio aéreo o los elementos climáticos.

Los experimentos verifican que este modelo es capaz de predecir un retraso en la llegada de un vuelo comercial a los Estados Unidos de forma bastante precisa.

- ***Una propuesta basada en aprendizaje automático para la mejora de la predicción de tiempos de llegada*** **de Petar Georgiev (2020)** [**11**]: en este Trabajo Fin de Grado de la Universidad de Valladolid se busca predecir los tiempos de llegada de las aeronaves utilizando trayectorias reconstruidas a partir de la tecnología ADS-B que influyen en el trayecto de los vuelos. Para realizar las predicciones utiliza varios modelos entre los que se encuentran las redes LSTM y determina que este tipo de redes son las que obtienen los mejores resultados.

- ***A multi-step airport delay prediction model based on spatial-temporal correlation and auxiliary features*** **de Hao Zhang et al. (2021)** [**6**]: este estudio propone un nuevo modelo para predecir los retrasos en los aeropuertos. En él se pone de manifiesto que estos retrasos tiene fuertes dependencias espacio-temporales dinámicas, de modo que las dependencias espaciales pueden ayudar a mejorar la precisión de los modelos de predicción, especialmente para la predicción del estado del tráfico a largo plazo.

  Además, en este artículo se refleja que el uso combinado de una red convolucional y una red LSTM mejora los resultados obtenidos por redes LSTM para secuencias espacio-temporales, ya que las capas convolucionales pueden extraer la correlación espacial y las LSTM pueden capturar las correlaciones temporales.

- ***Spatiotemporal Recurrent Convolutional Networks for Traffic Prediction in Transportation Networks*** **de Haiyang Yu et al. (2017)** [**33**]: en este artículo se propone un modelo de red convolucional recurrente espacio-temporal para la previsión del tráfico en la red de transporte de Beijing (China). El modelo que proponen combina las ventajas de las redes convolucionales, ya que estas permiten capturar las dependencias espaciales del tráfico en toda la red, con las ventajas de las redes LSTM que son capaces de aprender la dinámica temporal.

  Este estudio concluye que este modelo mejora las predicciones a corto y largo plazo realizadas con otros modelos (entre los que incluye las redes LSTM y las redes convolucionales por separado) en términos de error porcentual absoluto medio y error cuadrático medio.

Capítulo 3

Background

A lo largo de este capítulo se van a presentar algunos conceptos básicos de las redes neuronales que son necesarios para comprender los modelos que se van a construir y evaluar en los siguientes capítulos.

En primer lugar se define qué es el *Machine Learning* (Sección 3.1) y el *Deep Learning* (Sección 3.3), y después se explica qué tiene que ver las redes neuronales con el *Machine Learning* y se profundiza en estas redes (Sección 3.2).

3.1. *Machine Learning*

El *Machine Learning* o Aprendizaje Automático es una de las ramas de la Inteligencia Artificial basada en la idea de que una máquina puede aprender a identificar patrones a partir de una serie de datos y tomar decisiones con una mínima intervención humana [34]. Para ello, se centra en el diseño y desarrollo de algoritmos que permiten a los computadores mejorar su desempeño en la realización de una tarea a partir de la experiencia.

Los orígenes del *Machine Learning* datan de los años 50, cuando Arthur Samuel escribió el primer programa de aprendizaje automático que simulaba a una persona jugando a las "damas", y que conforme jugaba más partidas, mejoraba su estrategia. Sin embargo, el boom del *Machine Learning* no fue hasta la década de los 90, con la disponibilidad de hardware más rápido, conjuntos de datos más grandes y el auge de la informática. A partir de 2010, las grandes empresas como Google, IBM, Facebook, Amazon y Microsoft comenzaron a realizar sus propios desarrollos, y gracias a ello, esta tecnología está en la actualidad presente en multitud de aplicaciones como las sugerencias de Netflix o Spotify, o el habla de los asistentes personales Siri y Alexa [35].

A diferencia de la programación habitual, en este paradigma de programación los algoritmos no se programan, sino que se entrenan: al ordenador se le proporcionan datos de entrada (y en ocasiones, la salida esperada para esos datos de entrada) y él es el encargado de "crear" un algoritmo que permita resolver el problema. Además, una vez que se ha creado y entrenado el algoritmo, este será capaz de extraer similitudes en nuevos conjuntos de datos para los que no ha sido entrenado previamente y obtener de ellos una respuesta.

En la actualidad el *Machine Learning* tiene numerosos usos: por ejemplo, la seguridad informática, el reconocimiento de imágenes y/o patrones, o los análisis en el mercado de valores [36].

Desde el punto de vista del aprendizaje, la mayoría de autores establecen tres tipos de *Machine Learning* [36–39] :

1. Aprendizaje supervisado o *Supervised Learning*: este tipo de aprendizaje consiste en entrenar la máquina a partir de datos etiquetados con el objetivo de realizar predicciones o clasificar, donde las etiquetas son clases o categorías conocidas para los datos de entrada. Los principales usos del aprendizaje supervisado son la clasificación (clasificar en grupos a partir de los datos históricos etiquetados: identificar dígitos, diagnósticos, detectar fraudes de identidad, etc.) y la regresión (predecir valores continuos sabiendo los datos históricos etiquetados: predicciones del tiempo, crecimiento, etc.). Cabe destacar que este tipo de aprendizaje es el más utilizado por las redes neuronales.

2. Aprendizaje no supervisado o *Unsupervised Learning*: en este tipo de aprendizaje, a diferencia del aprendizaje supervisado, las máquinas se entrenan con datos sin etiquetar, es decir, sin conocer la salida esperada. El objetivo de este tipo de aprendizaje es que las máquinas aprendan a buscar patrones y similitudes por lo que se suele usar en problemas de *clustering*, reconocimiento facial, etc.

3. Aprendizaje por refuerzo o *Reinforcement Learning*: este tipo de aprendizaje consiste en el uso del método prueba-error para que la máquina mejore su rendimiento y alcance la mejor forma de completar una tarea dada. En él, la máquina aprende gracias al uso de "recompensas" para reforzar el comportamiento deseado, sin necesidad de programarla para que lo realice de alguna manera en particular. Actualmente este tipo de aprendizaje se usa para competir en todo tipo de juegos, o para descubrir tratamientos óptimos para enfermedades y tratamientos farmacológicos [40, 41].

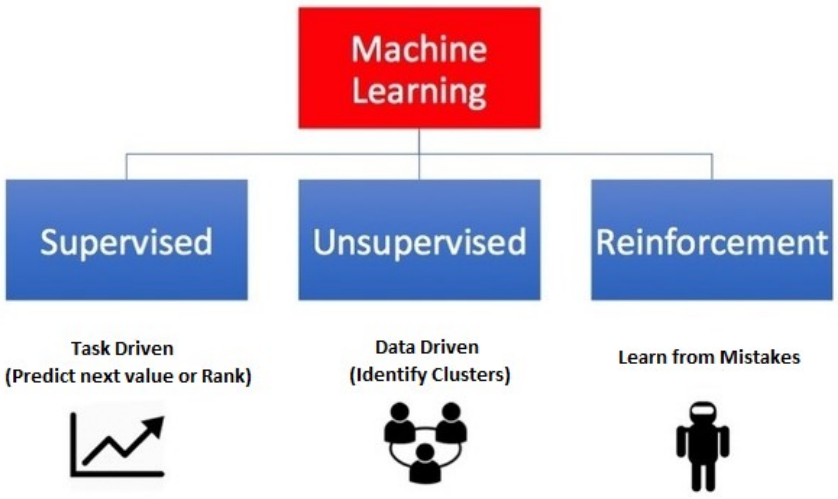


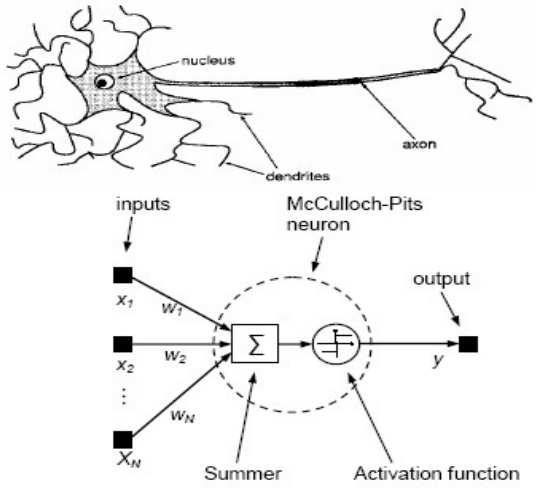
Figura 3.1: Resumen tipos de *Machine Learning*. Fuente: [39].

3.2. Redes neuronales

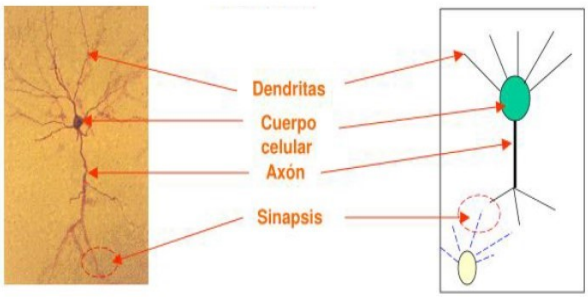
Uno de los métodos más exitosos dentro del *Machine Learning* son las redes neuronales artificiales que fueron planteadas por primera vez como método de aprendizaje a mediados del siglo XX. A pesar de ello, no se empezaron a usar fuera del ámbito académico hasta la década de los 60 [42]. Actualmente, y gracias a la potencia de los ordenadores modernos y la aparición de diferentes arquitecturas de redes neuronales, este tipo de redes tienen numerosos usos en el campo de la medicina [43], en el reconocimiento de patrones a partir de imágenes aéreas, en el marketing [44], etc.

El principal objetivo de una red neuronal es buscar modelos que solucionen problemas complejos que no pueden ser resueltos mediante las técnicas algorítmicas convencionales, y de esta manera, automatizar procesos que en un principio solo podrían ser resueltos por las personas.

Las redes neuronales tratan de imitar la forma de procesar la información del cerebro humano utilizando para ello un gran número de neuronas artificiales interconectadas y que transmiten señales entre sí. Es decir, son un conjunto de nodos de procesamiento relacionados que se asemejan de forma abstracta a las neuronas biológicas, y de ahí su nombre. En la Figura 3.2 podemos ver una comparación entre una neurona biológica y una neurona artificial (de forma rigurosa a la izquierda, y de forma simplificada a la derecha).



(a) Fuente: [45].



(b) Fuente: [46].

Figura 3.2: Analogía entre una neurona biológica y una artificial.

En general, el modelo estándar de una neurona artificial establecido en 1949 por McCulloch-Pitts [46] (y que podemos ver representado en la Figura 3.3) consiste según el propio McCulloch-Pitts y [46–48] en:

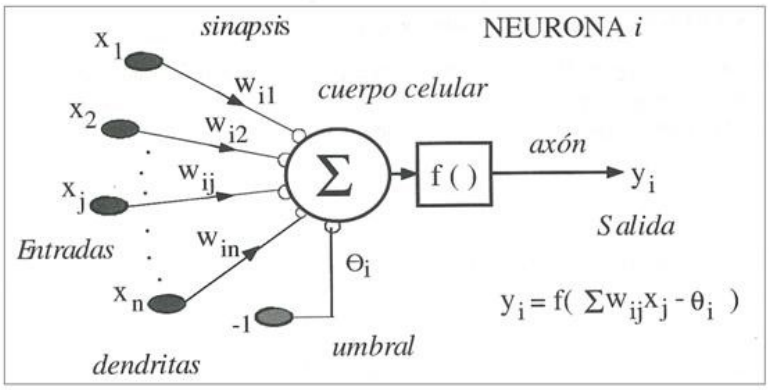


Figura 3.3: Modelo de neurona artificial estándar. Fuente: [47].

- Un conjunto de entradas x_j y unos pesos sinápticos w_j, con j = 1,...,n. Las entradas son el análogo de las dendritas en una neurona biológica y representan los valores de entrada que recibe la neurona. Por su parte, cada peso está asociado a una entrada y representa el impacto que tiene dicha entrada en la neurona. Durante la fase de entrenamiento de una neurona o red neuronal será necesario modificar los pesos para lograr un correcto aprendizaje.

- Un sesgo, umbral o *bias* denotado como Θ que establece la predisposición de una neurona para devolver un valor u otro independientemente del valor de los pesos. Esto provoca que, si este valor es muy grande, los pesos deban ser grandes para compensarlo. Al igual que los pesos, durante la fase de entrenamiento de la red neuronal también será necesario modificar este valor para lograr un correcto aprendizaje.

- Una regla de propagación h_i definida a partir de las entradas, los pesos sinápticos y el sesgo. Su notación habitual es $h_i(x_1, ..., x_n, w_{i1}, ..., w_{in})$. La regla de propagación más habitual cuando se trabaja con una red neuronal es la función sumatoria. Esta función sumatoria se basa en una suma ponderada de las entradas con los pesos sinápticos y el sesgo, y se puede calcular con la fórmula 3.1.

$$h_i(x_1, ..., x_n, w_{i1}, ..., w_{in}) = \sum_{j=1}^{n} x_j w_{ij} - \Theta \tag{3.1}$$

- Una función de activación f que limita la amplitud de la salida de la neurona. Esta función de activación es la análoga de la tasa de potencial de acción de una neurona biológica. Generalmente, la salida de la neurona se denota por y y se expresa como:

$$y = f(h_i(x_1, ..., x_n, w_{i1}, ..., w_{in}))$$

  Entre las funciones de activación clásicas destacan la función umbral, la función signo, la función sigmoide y la función lineal que se pueden ver en la Figura 3.4. En la Subsección 3.2.1 se explicará en detalle las funciones de activación que se utilizan normalmente en la actualidad.

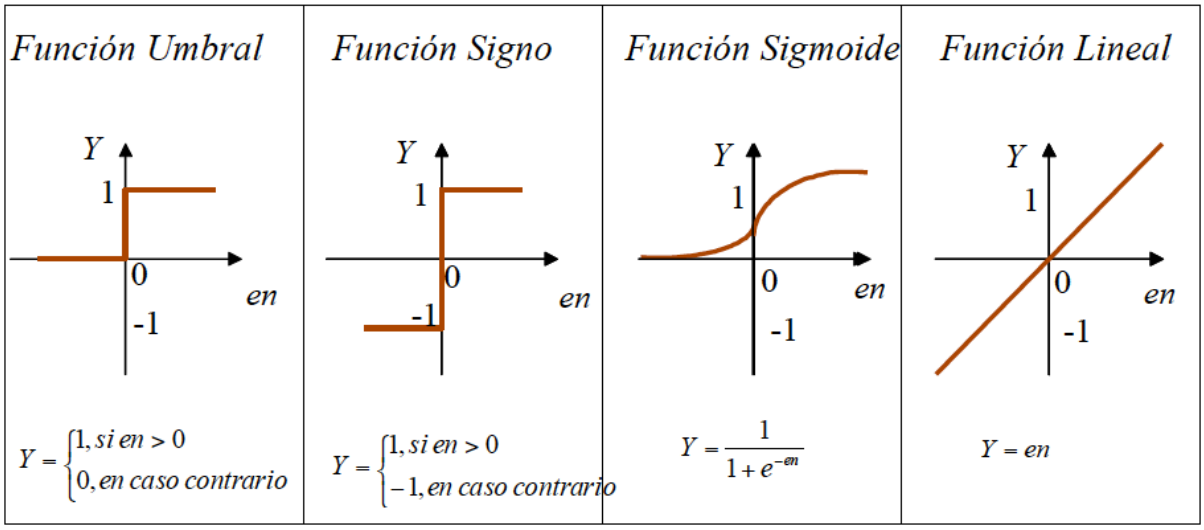


Figura 3.4: Funciones de activación clásicas. Fuente: [46].

Aunque una red neuronal puede estar formada por una sola neurona, es más común combinar múltiples neuronas en varias capas con el fin de aumentar la capacidad expresiva de la red. En la Figura 3.5 podemos ver la estructura jerárquica de una red neuronal.

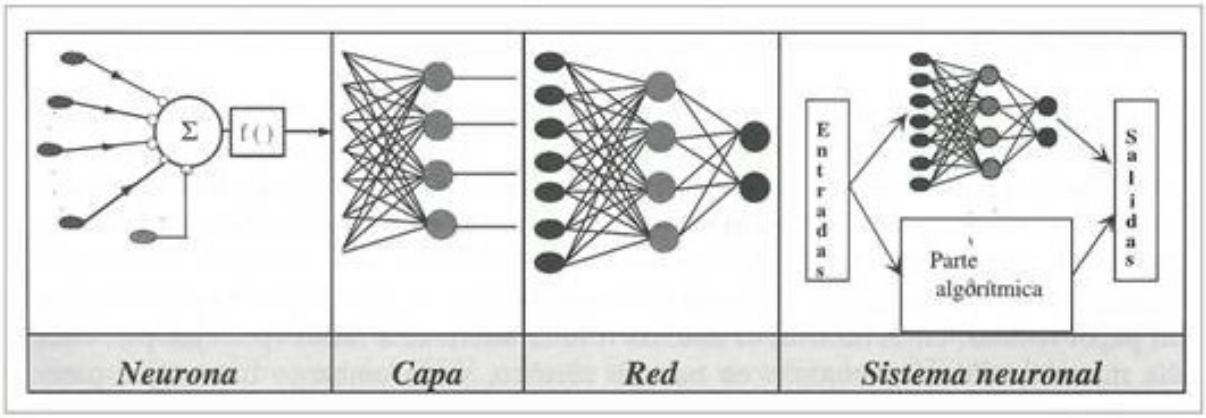


Figura 3.5: Estructura jerárquica de un sistema basado en una red neuronal artificial. Fuente: [47].

Cuando se utiliza esta estructura en capas, las neuronas de cada capa reciben su entrada de la salida de las neuronas de la capa anterior. Cada uno de los nodos de una capa está conectado a todos los nodos de la capa anterior (como se puede ver en la Figura 3.6), y cuando esto ocurre, se dice que las neuronas están densamente conectadas o que forman capas densas. Con las capas densas se pueden realizar transformaciones y cambios de dimensión entre los vectores de entrada y salida [49]. Dentro de una red neuronal estructurada en capas, se pueden distinguir tres tipos de capas [47, 50]:

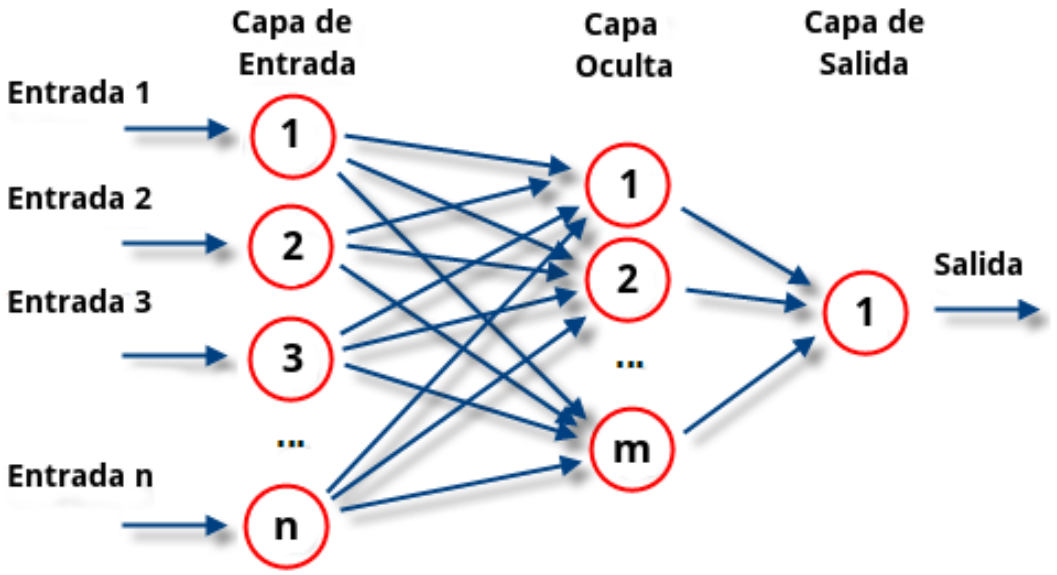


Figura 3.6: Capas dentro de una red neuronal. Fuente: [51].

1. Capa de entrada: es la primera capa de la red y está formada por todas aquellas neuronas cuya función consiste en recibir datos o señales procedentes del entorno. Generalmente, las neuronas que componen esta capa no realizan ningún tipo de procesamiento.

2. Capa(s) oculta(s): está formada por aquellas neuronas que no tienen una conexión directa con el entorno. En una red neuronal, puede haber una o varias capas de este tipo y su función principal es recibir como datos de entrada los datos de la salida de la capa anterior, procesarlos, y enviarlos a la siguiente capa.

3. Capa de salida: es la última de las capas que conforman la red. Está formada por aquellas neuronas que recogen la salida de la última capa oculta de la red y proporcionan la respuesta de la red neuronal.

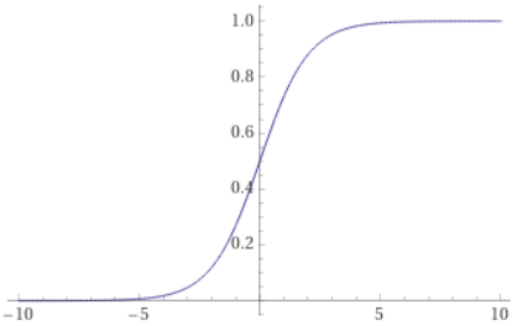
Para entrenar este tipo de redes, se utilizan estimadores de errores que tratan de minimizar una función de coste como veremos en la Subsección 3.2.4.

3.2.1. Funciones de activación

Como se ha comentado anteriormente, el principal objetivo de las funciones de activación es limitar la amplitud de la salida de las neuronas. Para ello devuelven una salida (cuyo rango de valores suele estar en (0,1) o (-1,1)) a partir de un valor de entrada. En general, se usan funciones cuya derivadas son sencillas de calcular con el fin de minimizar costes computacionales [52].

Las funciones de activación clásicas son la función umbral, la función signo y la función lineal, pero las más utilizadas hoy en día son las que se detallan a continuación [46]:

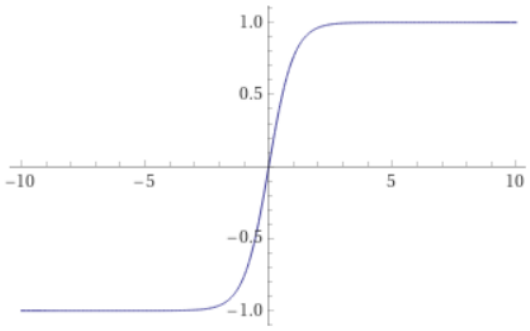
- Función sigmoide: esta función definida por la ecuación 3.2 y representada en la Figura 3.7, devuelve una salida en el rango (0,1). Sus principales características son su lenta convergencia y su buen rendimiento en la última capa, motivo por el cual ha sido una de las más usadas a lo largo de la historia. Sin embargo, el uso de valores de entrada muy grandes o muy pequeños puede derivar en el problema del "desvanecimiento del gradiente" o *vanishing gradient* (explicado en la Sección 3.4).



$$f(x) = \frac{1}{1 + e^{-x}} \qquad (3.2)$$

Figura 3.7: Función sigmoide.

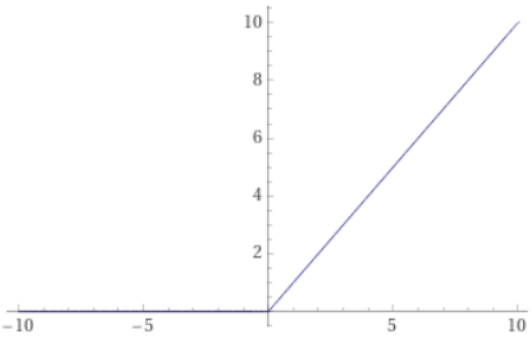
- Función tangente hiperbólica: esta función también se utiliza frecuentemente (sobre todo en redes recurrentes) ya que se trata de una versión escalada de la función sigmoide, y generalmente se escoge cuando se quiere discernir entre dos opciones. Está definida por la ecuación 3.3, y como podemos ver en la Figura 3.8, devuelve una salida en el rango (-1,1). Sin embargo, puede seguir causando el problema del *vanishing gradient* al igual que la función sigmoide.



$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (3.3)$$

Figura 3.8: Función tangente hiperbólica.

- Función unidad lineal rectificada o ReLU (del inglés *Rectified Linear Unit*): esta función es la más utilizada en *Deep Learning* ya que deja intacta la entrada si es un valor positivo, y devuelve 0 si es un valor negativo. Está definida por la ecuación 3.4 y se puede ver su representación en la Figura 3.9. Esta función tiene una implementación muy simple, y al no tener límite superior, es mucho más rápida de entrenar. Además, cabe destacar que para valores de entrada positivos, no provoca el problema del *vanishing gradient*.



$$f(x) = max(0, x) \qquad (3.4)$$

Figura 3.9: Función ReLU.

3.2.2. Aprendizaje de una neurona

El objetivo principal cuando se entrena una neurona es encontrar una combinación de sus pesos sinápticos y de su sesgo de forma que su comportamiento generalice lo mejor posible los datos de entrenamiento y prueba. Es decir, que sea capaz de devolver salidas correctas en la mayoría de ocasiones.

Antes de comenzar a entrenar una neurona, se debe definir unas condiciones de parada o criterios de convergencia para poder terminar el entrenamiento si se cumple alguna de ellas [47]. Algunos de estas causas pueden ser:

- Se ha alcanzado una cota de error que está por debajo de un determinado umbral.
- Se ha alcanzado el número máximo de iteraciones (épocas) que se había definido como tope para el conjunto de datos de entrenamiento.
- Se ha obtenido un error igual a cero, por lo que no es necesario seguir entrenando.
- La neurona se ha saturado y por más que se entrene no se puede reducir el error.

Una vez definidos estos criterios de convergencia, se utiliza el siguiente algoritmo (que se puede ver representado en la Figura 3.10) para entrenar a una neurona [46]:

1. En primer lugar, se inicializan los pesos iniciales w_1,...,w_n y el sesgo Θ a valores aleatorios en el rango [-0.5,0.5].
2. Después, suponiendo que se dispone de un conjunto de ejemplos de entrenamiento (x_1,y_1),(x_2,y_2),... con x_t=(x_{t1},x_{t2},...,x_{tn}) e $y_t \in \{0,1\}$; para cada ejemplo de entrenamiento se realizan los siguientes pasos:
   a) Se clasifican las entradas (x_{t1},x_{t2},...,x_{tn}) con los pesos y el sesgo que se tengan en ese momento.
   b) Se compara el valor obtenido (y_{obt}) con el valor esperado (y_t), y se calcula el error existente como e_t=y_t-y_{obt}

c) En caso de que el error e_t sea distinto de 0, se debe realizar una corrección de los pesos y del sesgo. Es decir, modificar sus valores para lograr un correcto aprendizaje de la neurona. Para lograrlo, se utiliza un factor de aprendizaje o *learning rate* (α) que corresponde con una constante positiva pequeña (generalmente entre 0 y 1) que permite regular la velocidad de aprendizaje de esta. La elección de este valor es muy importante porque un valor muy pequeño puede provocar que la neurona aprenda muy despacio, mientras que un valor muy grande puede provocar un comportamiento errático (en la Figura 3.11 podemos ver varios ejemplos de la velocidad de aprendizaje de la neurona en función del valor que toma esta constante). Una vez fijado el *learning rate*, se calcula un ajuste de todos los pesos (Δw_i) y del sesgo ($\Delta\Theta$) de la neurona con el fin de reducir el error existente. Este ajuste se calcula de la siguiente manera:

- Para los pesos: $\Delta w_i = \alpha \cdot x_{ti} \cdot e_t$ $\forall$i $\in$\{1,...,n\}
- Para el sesgo: $\Delta\Theta = \alpha \cdot e_t$

d) Finalmente, se actualizan los pesos y el sesgo de la siguiente forma:

- Los pesos: $w_i = w_i + \Delta w_i$ $\forall$i $\in$\{1,...,n\}
- El sesgo: $\Theta = \Theta + \Delta\Theta$

3. Finalmente se repite el paso 2 con todos los ejemplos de entrenamiento tantas veces como sea necesario hasta cumplir alguno de los criterios de convergencia previamente establecidos.

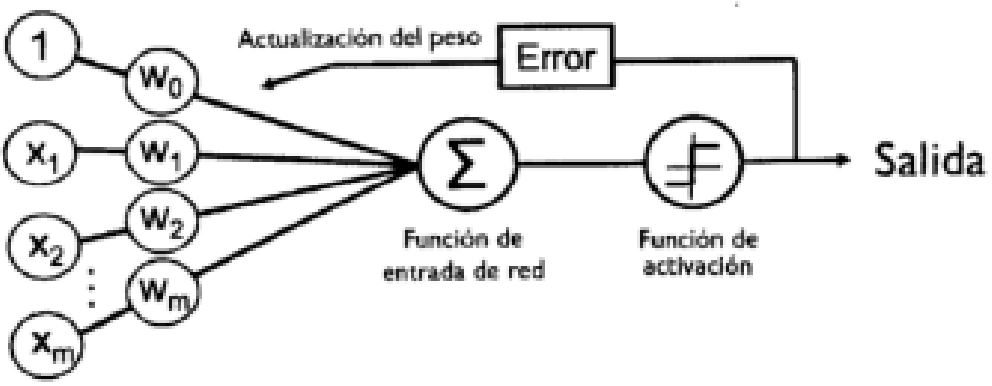
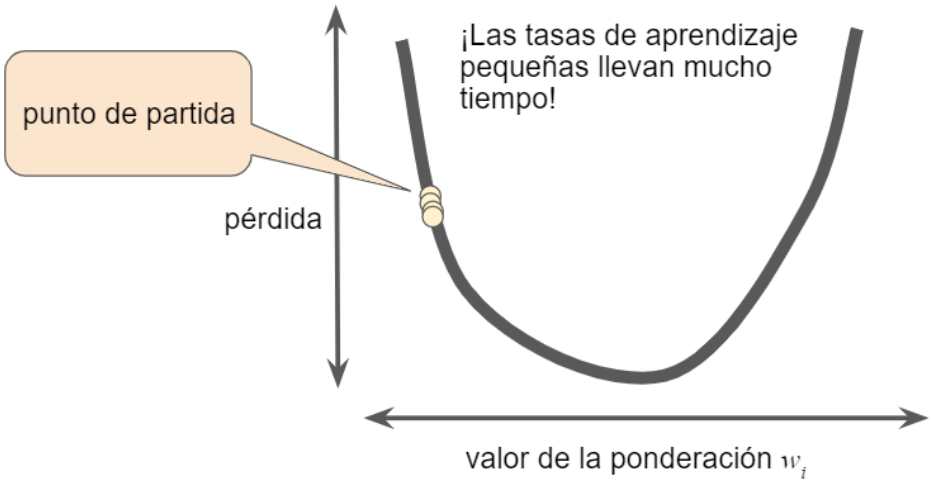
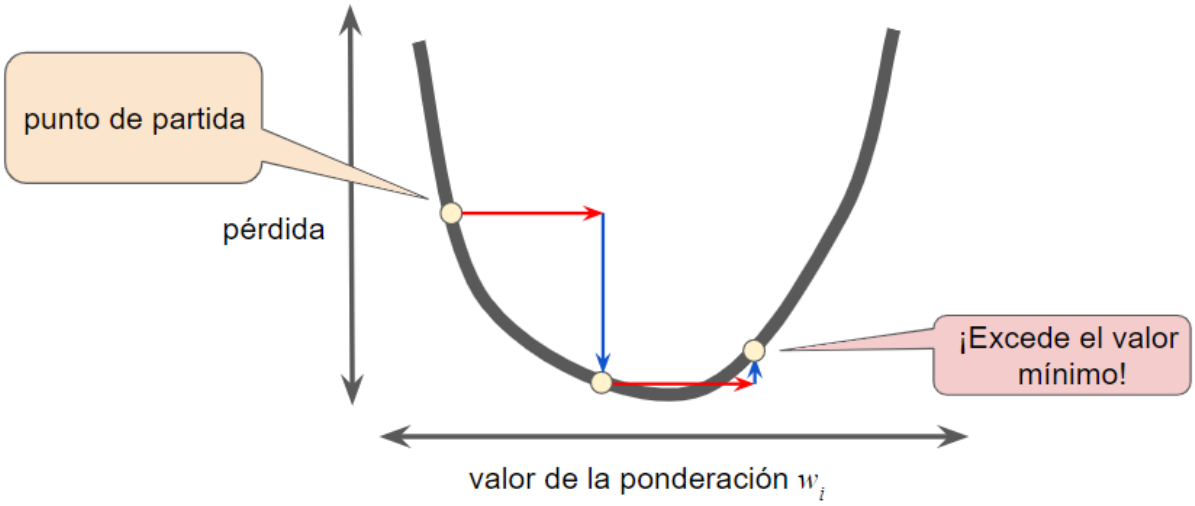


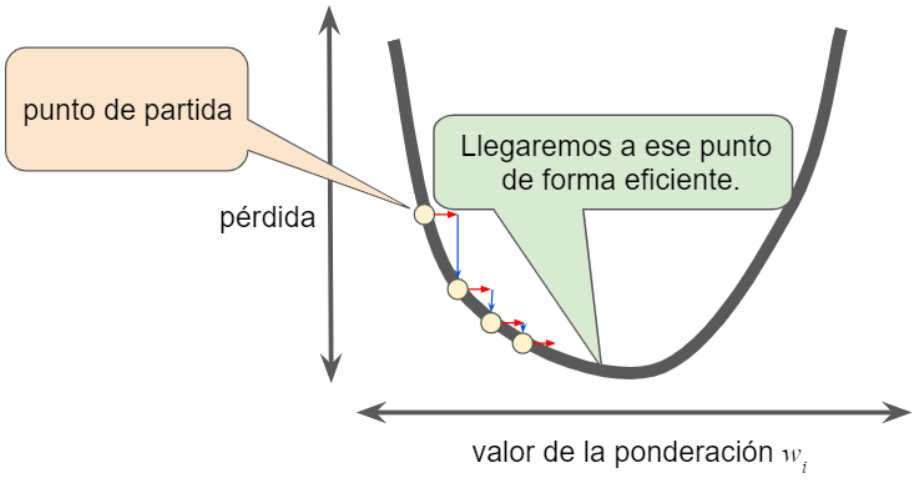
Figura 3.10: Esquema del entrenamiento de una neurona artificial. Fuente: [53].



(a) *learning rate* muy pequeño.



(b) *learning rate* muy grande.



(c) *learning rate* adecuado.

Figura 3.11: Velocidad de aprendizaje de la neurona en función del valor del *learning rate*. Fuente: [54].

3.2.3. Aprendizaje de una red neuronal

Aunque entrenar una neurona es algo bastante simple, cuando queremos entrenar una red neuronal para usarla en problemas más complejos debemos tener en cuenta que las entradas de las neuronas no van a depender solamente de las entradas del problema, sino que también van a depender de las salidas de las neuronas que las preceden en la red neuronal. Esto provoca que el error cometido por una neurona no solo dependa de si sus pesos sinápticos y su sesgo son correctos o no, ya que también depende del error que trae acumulado del resto de neuronas que le preceden [47].

Por ello, para entrenar una red neuronal se llevan a cabo el algoritmo de retropropagación en el tiempo (también conocido como BPTT por sus siglas en inglés *BackPropagation Through Time*) que está compuesto por tres fases: propagación hacia delante (en inglés *forward propagation*), la de propagación hacia atrás (en inglés *backpropagation*), y la del descenso por gradiente (en inglés *gradient descent*).

Este algoritmo de retropropagación se aplica al aprendizaje supervisado que se ha explicado en la Sección 3.1 ya que se necesita conocer de antemano la salida que se quiere obtener, y se realiza de la siguiente manera [55]:

- La primera fase de propagación hacia delante (o *forward propagation*) consiste en que cada una de las neuronas de la red recibe los datos de entrada procedentes de la capa anterior, los procesa, y los propaga hacia las siguientes capas hasta llegar a la última. Una vez que la última capa de la red devuelve la salida, se calcula el error cometido entre la salida esperada y la proporcionada por la red, y comienza la segunda fase. La "función de coste" trata de determinar y optimizar este error de diversas maneras (esto se explica en detalle en la Subsección 3.2.4).

- La segunda fase de propagación hacia atrás (o *backpropagation*) consiste en propagar este error desde la última capa hasta la primera. En cada capa se calcula un valor de error para cada neurona a partir de los errores de la siguiente capa. De esta forma, se calcula el error para todas las neuronas de la red y puede dar comienzo la tercera fase.

- La tercera fase del descenso por gradiente (o *gradient descent*) consiste en calcular las actualizaciones necesarias para los pesos de las neuronas que forman la red neuronal y aplicar dichas actualizaciones. Para ello, se calcula el gradiente (derivada parcial) de la función de coste con respecto a todos los parámetros de la red (los pesos de todas las neuronas). El gradiente que se obtiene es un vector que indica la dirección y sentido en el que la función de coste crece más rápido, por lo tanto, para disminuir el error se debe avanzar en el sentido opuesto a este gradiente. En la fórmula 3.5 podemos ver la expresión matemática del descenso por gradiente y cómo se actualizan los parámetros de la red:

$$\Phi = \Phi - \alpha \cdot \nabla f \;\; ; \;\; \nabla f = \begin{bmatrix} \frac{\partial C}{\partial \Phi_1} \\ \vdots \\ \frac{\partial C}{\partial \Phi_N} \end{bmatrix} \tag{3.5}$$

Donde:
∇f: vector gradiente.
C: función de coste.
Φ: matriz de parámetros de la red neuronal.
Φ_i: vector de parámetros de la red neuronal en la capa i con i $\in$ {1,...,N}, y siendo N la última capa de la red neuronal.
α: *learning rate* o factor de aprendizaje.

Cuando se realizan las tres fases del algoritmo de retropropagación, se vuelve a realizar el algoritmo completo con los pesos actualizados, y al igual que sucedía en el caso del entrenamiento de una neurona, se repetirá tantas veces como sea necesario para llegar a algún criterio de convergencia que se haya establecido al inicio del entrenamiento.

Los criterios de convergencia más habituales son los mismos que los explicados en la Sección 3.2.2, pero cambiando el último de los criterios por uno que sea que la red neuronal se ha saturado y ya no es capaz de reducir el error por mucho que se entrene.

3.2.4. Función de coste

Como se ha explicado anteriormente, la función de coste trata de calcular el error cometido entre la salida esperada y la proporcionada por la red neuronal con el objetivo de optimizar los pesos de las neuronas que conforman la red [56].

Entre los principales estimadores de error de la función de coste destacan los siguientes:

- Raíz del error cuadrático medio (también denominada RMSE por sus siglas en inglés: *Root Mean Squared Error*): esta medida se calcula como la raíz cuadrada de la media de los residuos (fórmula 3.6), entendiendo como residuos el cuadrado de la diferencia entre la salida esperada (y_{esp}) y la proporcionada por la red (y_{pro}). Esta medida en general funciona bien para optimizar regresiones, aunque los valores muy grandes penalizan mucho el resultado. Sin embargo, es difícil de interpretar.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_{esp} - y_{pro})^2}{N}} \tag{3.6}$$

Nota: N es el tamaño de la salida esperada.

- Error cuadrático medio (también denominada MSE por sus siglas en inglés: *Mean Squared Error*): esta medida se utiliza frecuentemente para calcular los gradientes ya que es idéntica al RMSE pero sin calcular la raíz cuadrada (fórmula 3.7).

$$MSE = \frac{\sum_{i=1}^{N}(y_{esp} - y_{pro})^2}{N} \tag{3.7}$$

  Nota: N es el tamaño de la salida esperada.

- Error absoluto medio (también denominado MAE por sus siglas en inglés: *Mean Absolute Error*): esta medida se calcula como la suma media de los valores absolutos de los residuos (fórmula 3.8). Esta medida es más sencilla de interpretar en comparación con la RMSE, pero en este caso, los valores grandes no penalizan tanto el resultado, lo que provoca una convergencia más lenta que con la RMSE.

$$MAE = \frac{\sum_{i=1}^{N}|y_{esp} - y_{pro}|}{N} \tag{3.8}$$

  Nota: N es el tamaño de la salida esperada.

Una vez que se ha escogido el estimador de error que utilizará la red neuronal, también es necesario escoger un optimizador que se encargue de reducirlo, ya que existen multitud de ellos. Cada uno tiene sus ventajas e inconvenientes, pudiendo obtener resultados totalmente distintos en función del optimizador escogido. Para los fórmulas 3.9, 3.10, 3.11 y 3.12 que definen los distintos optimizadores explicados, α denota el *learning rate*, W la matriz de pesos o parámetros, y ∇W la derivada de la matriz de pesos. Con esta notación, los optimizadores más utilizados en la actualidad son los siguientes [57, 58]:

- Descenso del Gradiente Estocástico (también conocido como SGD o *Stochastic Gradient Descent*): es el método iterativo más básico de optimización, y por ello se considera el algoritmo por defecto para entrenar redes neuronales. Los primeros prototipos de este algoritmo fueron publicados en 1951 por Herbert Robbins y Sutton Monro en el artículo "*A Stochastic Approximation Method*" [59]. SGD utiliza la técnica de Descenso por Gradiente (explicada previamente en la Subsección 3.2.3), pero en vez de realizarlo sobre toda la matriz de pesos de las neuronas que conforman la red, lo realiza sobre una pequeña muestra aleatoria (o estocástica, de ahí su nombre) para reducir el número de cálculos y reducir los tiempos de computación. En este método, el *learning rate* se mantiene constante durante todo el entrenamiento y el paso de actualización de los pesos viene dado por la fórmula:

$$W = W - \alpha \cdot \nabla W \tag{3.9}$$

- Adagrad (acrónimo de *Adaptative Gradient Algorithm*): este es un método adaptativo que, a diferencia del método SGD, utiliza un *learning rate* que se ajusta a los parámetros de la red (de forma que las actualizaciones más grandes se realizan sobre parámetros que casi no cambian, y las más pequeñas sobre los que varían constantemente). Para ello, este método parte de un *learning rate* inicial, y lo escala y adapta para cada peso en función del gradiente acumulado en cada iteración. Para ello, utiliza una especie de *caché* (denotada como C) que mantiene la suma de los cuadrados de los gradientes para cada peso. Además, hace uso de un valor positivo muy pequeño (denotado como ϵ) para evitar posibles divisiones entre 0 ya que este método viene dado por las fórmulas:

$$\begin{aligned} C &= C - \alpha \cdot \nabla W^2 \\ W &= W - \frac{\alpha \cdot \nabla W}{\sqrt{C + \epsilon}} \end{aligned} \tag{3.10}$$

- Adadelta: este método es una mejora del método Adagrad. En él, la caché se calcula solo a partir de los últimos n gradientes. Esto reduce los efectos negativos de la acumulación global de los cachés pero no tiene en cuenta todos los gradientes desde el inicio del aprendizaje. En este caso, las fórmulas que utiliza este método son las mismas que las que utiliza el método Adagrad.

- RMSprop (acrónimo de *Root Mean Square Propagation*): este método es similar al método Adagrad pero utiliza un parámetro denominado tasa de descomposición, tasa de decadencia o *decaying rate* (denotada como d). Este parámetro, que suele valer 0.9 aunque se puede modificar, trata de corregir el efecto negativo de la acumulación global de los cachés haciendo que las nuevas entradas de la caché tenga una mayor influencia que las antiguas. Este método viene dado por las fórmulas:

$$\begin{aligned} C &= C + d \cdot C \cdot (1 - d) \cdot \nabla W^2 \\ W &= W - \frac{\alpha \cdot \nabla W}{\sqrt{C + \epsilon}} \end{aligned} \tag{3.11}$$

- Adam (acrónimo de *Adaptative Moment Estimation*): este método combina las ventajas de los métodos Adagrad y RMSprop. Es decir, emplea un *learning rate* que se ajusta a los parámetros de la red y que se ve afectado por la media (denotado por m) y la varianza (denotado por v) de los gradientes a lo largo del tiempo. En este método, se utilizan dos parámetros adicionales: β_1 y β_2 para calcular la media y la varianza respectivamente a lo largo del tiempo. Normalmente, β_1 vale 0.9 y β_2 0.99, aunque estos valores se pueden modificar. Este método viene dado por las fórmulas:

$$\begin{aligned} m &= \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla W \\ v &= \beta_2 \cdot v + (1 - \beta_2) \cdot \nabla W^2 \\ W &= W - \frac{\alpha \cdot m}{\sqrt{v + \epsilon}} \end{aligned} \tag{3.12}$$

3.3. *Deep Learning*

El *Deep Learning* o Aprendizaje Profundo es un campo que forma parte del *Machine Learning* y es muy efectivo para problemas que requieren el aprendizaje de patrones. En la Figura 3.12 podemos ver dónde se encuadran el *Machine Learning* y el *Deep Learning* dentro de la Inteligencia Artificial.

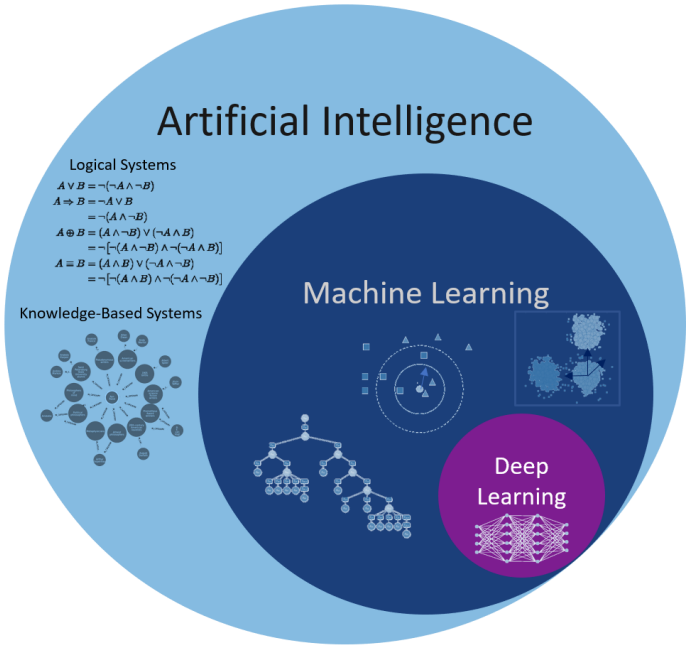


Figura 3.12: Visión del *Machine Learning* y del *Deep Learning* dentro de la Inteligencia Artificial. Fuente: [60].

En este campo, se necesitan conjuntos de datos más grandes y una mayor potencia de cálculo que en el *Machine Learning*, pero con ello se puede lograr una mejora automática y un alto nivel de precisión [61]. En la Tabla 3.1 se pueden ver las principales ventajas y desventajas entre el *Machine Learning* y el *Deep Learning*:

Machine Learning	***Deep Learning***
Buenos resultados con pequeños conjuntos de datos.	Requiere conjuntos de datos muy grandes.
Rápido para entrenar un modelo.	Computacionalmente intensivo.
Necesita probar diferentes características y clasificadores para lograr los mejores resultados.	Aprende características y clasificadores automáticamente.
Precisión limitada.	Precisión ilimitada.

Tabla 3.1: Ventajas y desventajas entre *Machine Learning* y *Deep Learning*.

En general, las técnicas de *Deep Learning* explotan la información almacenada en grandes conjuntos de datos de una forma más eficiente que las técnicas clásicas de *Machine Learning*, ya que en el *Deep Learning* no es necesario extraer manualmente las características del conjunto de los datos como ocurre en el *Machine Learning*. Es decir, la tarea de encontrar las características relevantes del conjunto de datos es parte del algoritmo y está automatizada [62]. Esta diferencia se puede ver en la Figura 3.13.

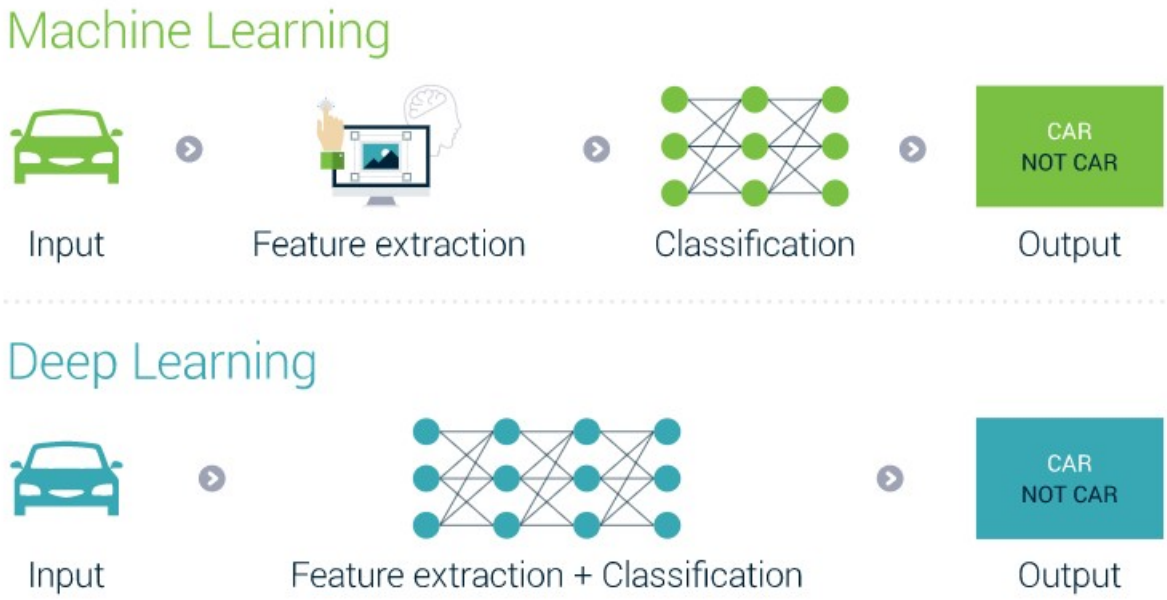


Figura 3.13: Principal diferencia entre *Machine Learning* y *Deep Learning*. Fuente: [62].

En la actualidad, el *Deep Learning* tiene multitud de aplicaciones, entre las que destacan: la detección de fraudes, el análisis de imágenes médicas, la localización de caras e identificación de emociones faciales, la clasificación de vídeos, el reconocimiento del habla (o *speech recognition* en inglés) o la visión computacional [63].

Aunque existen diferentes arquitecturas de redes neuronales en *Deep Learning*, las más utilizadas hoy en día son las redes neuronales recurrentes (explicadas en la Sección 3.4) y las redes neuronales convolucionales (explicadas en la Sección 3.5) [64, 65].

3.4. Redes neuronales recurrentes

Uno de los principales tipos de redes neuronales profundas son las redes neuronales recurrentes o RNN (por sus siglas en inglés: *Recurrent Neural Networks*) que surgieron en los años 80 de la mano de Rumelhart et al. [66], Werbos [67] y Elman [68].

Este tipo de redes no presentan una estructura de capas bien definidas, ya que se permite que las salidas de la red alimenten de nuevo a las entradas como se puede ver en

el ejemplo de la Figura 3.14. Esto provoca que el estado de la red presente un comportamiento dinámico, pudiendo alcanzar un estado estable, presentar oscilaciones, o incluso comportarse de forma caótica. Además, estas redes son más difíciles de entrenar debido a que se establecen conexiones arbitrarias entre las neuronas, donde la salida de algunas neuronas puede alimentar a sus propias entradas. Sin embargo, esto permite que la red neuronal presente memoria a corto plazo, y que se utilicen a menudo en aplicaciones que involucran análisis de secuencias (como puede ser el análisis de texto, sonido, vídeo, etc.), series temporales o de reconocimiento de voz humana y escritura a mano [69].

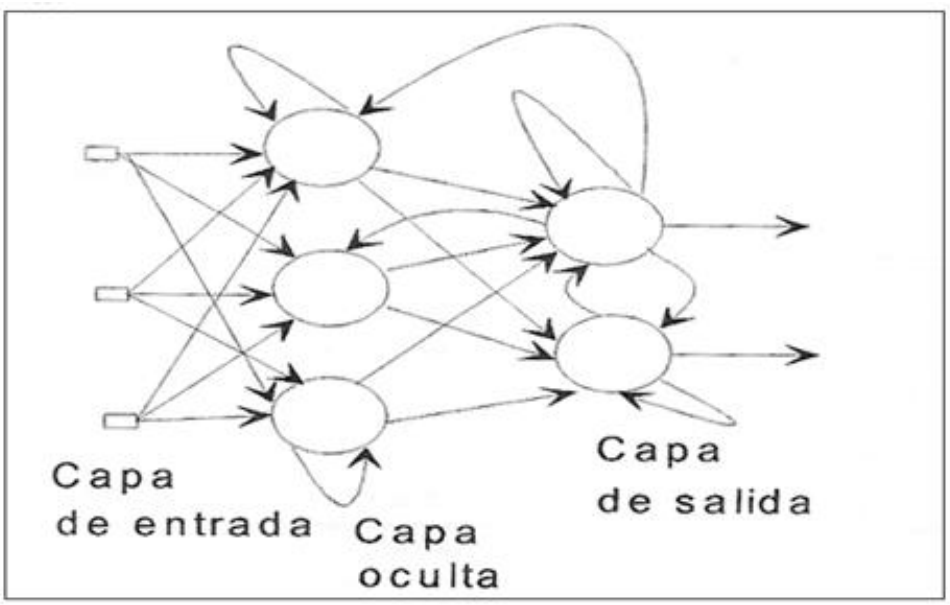


Figura 3.14: Ejemplo de funcionamiento de una RNN. Fuente: [47].

Debido a que en las RNN se pueden combinar las entradas y salidas según la necesidad del problema, se clasifican por el tamaño de los datos de entrada y de salida, pudiendo distinguir varios modelos de arquitectura (Figura 3.15).

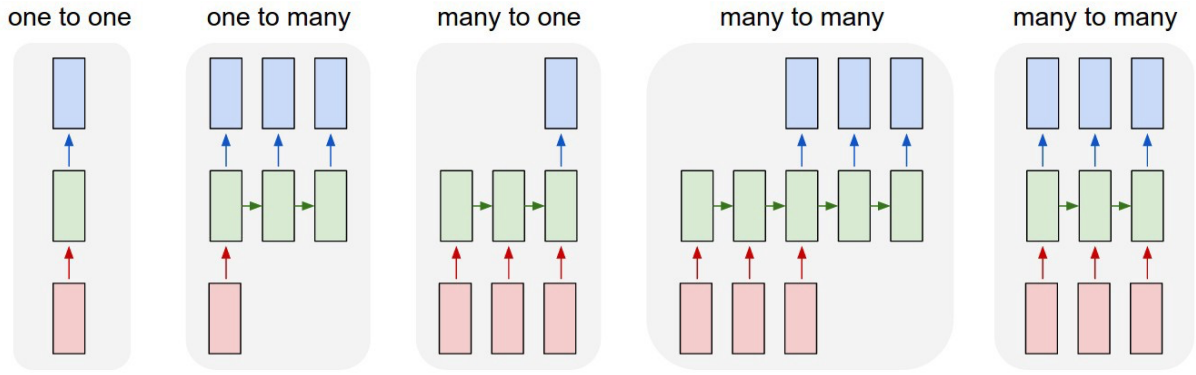


Figura 3.15: Principales modelos de arquitectura de una RNN. Fuente: [70].

Escoger entre un modelo de arquitectura u otro dependerá del uso que se le quiera dar a la red neuronal: "*one to one*" se usa para clasificar imágenes, "*one to many*" para subtitular imágenes, "*many to one*" para analizar sentimientos, y "*many to many*" para realizar traducciones, para clasificar vídeos en función de cómo estén dispuestas las entradas y salidas, y para problemas *seq2seq* (secuencia a secuencia) [46].

Una red neuronal recurrente simple (o también conocida como RNN *Vanilla*) equivale a un conjunto de redes neuronales simples que se encuentran concatenadas (Figura 3.16). Esta estructura recurrente nos permite usar la red en distintos instantes de tiempo, pasando la salida del instante anterior al instante actual, pero usando siempre la misma red, y de ahí su nombre de recurrente. Para este tipo de redes, los pesos de las capas ocultas se mantienen constantes para poder memorizar correctamente.

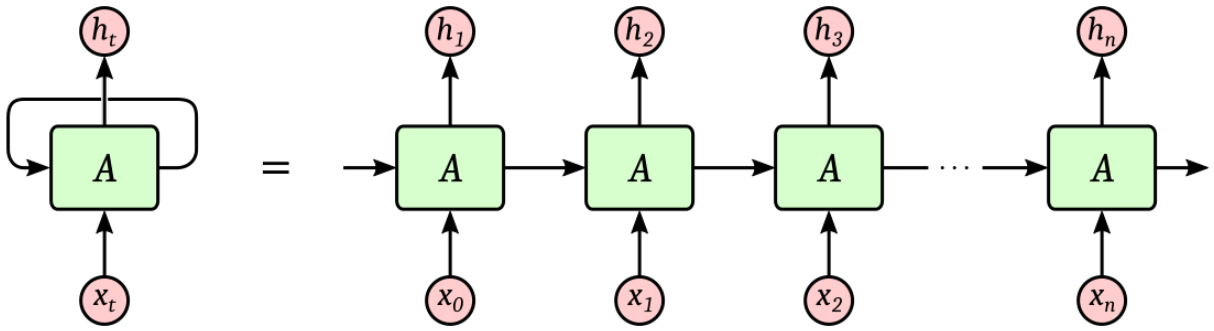


Figura 3.16: Ejemplo de RNN y su representación "desplegada" en el tiempo. Fuente: [71].

A pesar de que esta retroalimentación puede ser muy útil en muchos casos, cuando se entrena este tipo de redes neuronales con el algoritmo de retropropagación (explicado en la Subsección 3.2.3), durante la tercera fase del Descenso por Gradiente aparece un problema conocido como "desvanecimiento del gradiente" o *vanishing gradient*. Este problema consiste en que al calcular la derivada parcial de la función de coste (con respecto a todos los pesos de la red) para actualizar cada uno de los pesos, este valor puede ser cada vez más pequeño con el paso del tiempo, lo que provoca que los pesos de la red apenas se modifiquen, y que a veces no se complete el entrenamiento de la red [72].

Otra de las consecuencias de este problema es la que se puede observar en la Figura 3.17: en ella se puede ver como la sensibilidad a los valores de entrada en la red neuronal recurrente disminuye con el tiempo, llegando incluso a desaparecer. Este problema implica que la red neuronal recurrente no puede recordar largas dependencias temporales, y de ahí que solo presente memoria a corto plazo.

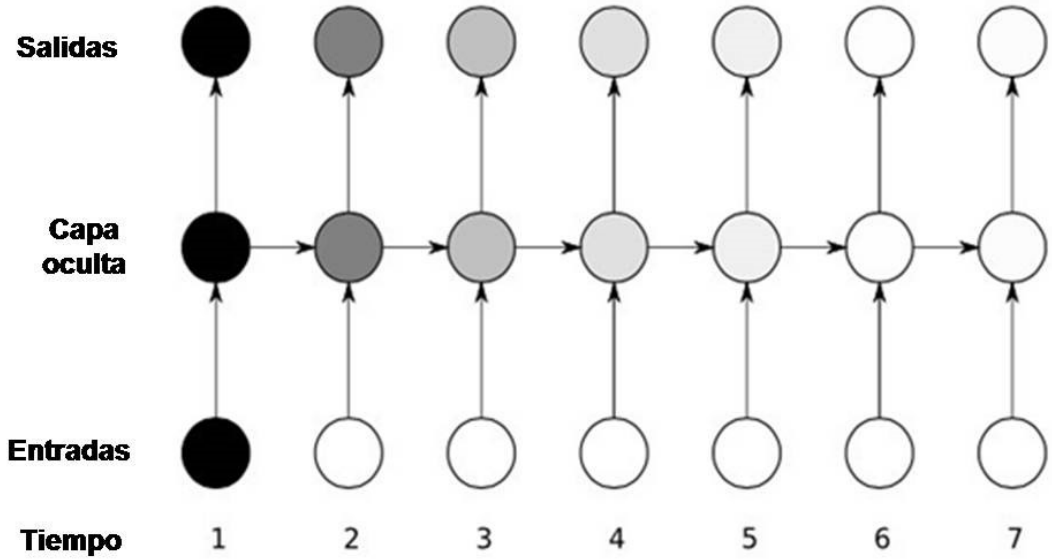


Figura 3.17: Sensibilidad a los valores de entrada en una RNN a lo largo del tiempo. El negro indica sensibilidad máxima y el blanco nula. Fuente: [73].

Para evitar este problema del desvanecimiento del gradiente, la solución pasa por usar redes neuronales recurrentes LSTM (por sus siglas en inglés *Long Short-Term Memory Networks*) [74]. Como se explicará a continuación, este tipo de redes son capaces de retener la información de entradas anteriores en el tiempo, consiguiendo así una memoria a largo plazo.

3.4.1. Redes LSTM

Las redes LSTM o redes de memoria a largo plazo son un tipo particular de redes neuronales recurrentes. Este tipo de redes, introducidas por Sepp Hochreiter y Jürgen Schmidhuber en 1997, son capaces de aprender dependencias a largo plazo, solucionando el problema del *vanishing gradient* de las RNN [69].

Su principal ventaja frente a las RNN tradicionales es que aparte de tener memoria a corto plazo, tienen memoria a largo plazo, y por esta razón son las más utilizadas en la actualidad para la clasificación de series temporales, el reconocimiento del habla y de la escritura, o la generación de texto. Sin embargo, son más difíciles de entender y de trabajar con ellas.

Las redes LSTM se pueden disponer en cadena al igual que las RNN estándar, pero la estructura de cada celda (neurona) es diferente en este tipo de redes (Figura 3.18) ya que cada celda LSTM posee cuatro capas de red neuronal que interactúan entre ellas.

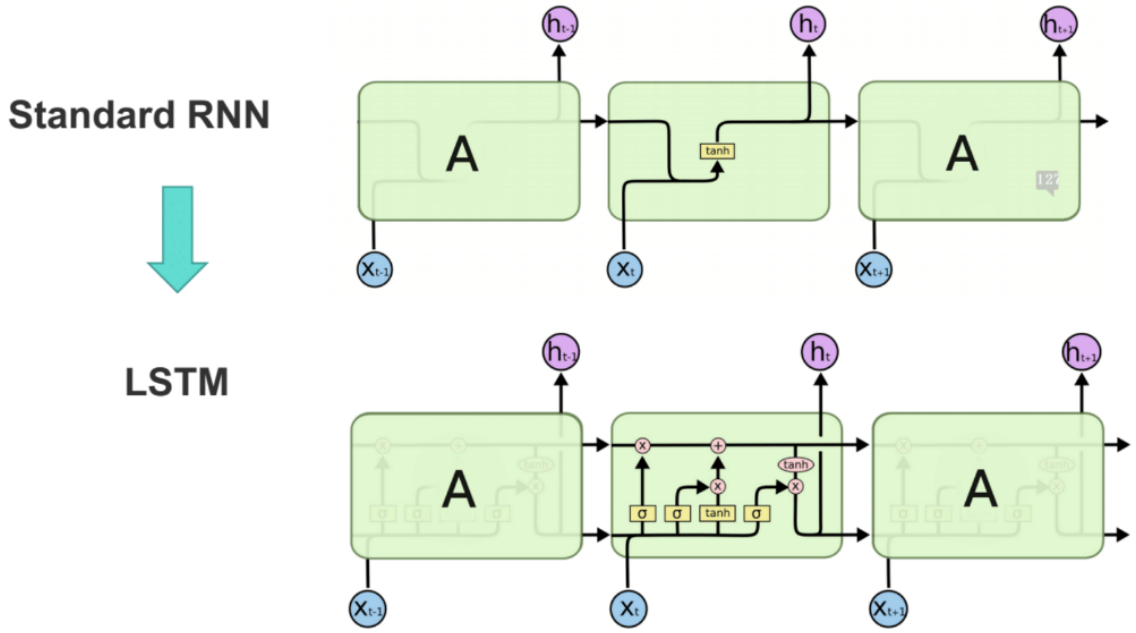


Figura 3.18: Estructura celda RNN estándar vs celda LSTM. Fuente: [75].

La clave principal de las redes LSTM es el uso de un nuevo estado interno, que se denomina "celda de estado" y que equivale a una cinta transportadora de información. Este estado se corresponde con la línea horizontal que se ejecuta en la parte superior de una celda LSTM y que se puede apreciar en la Figura 3.19.

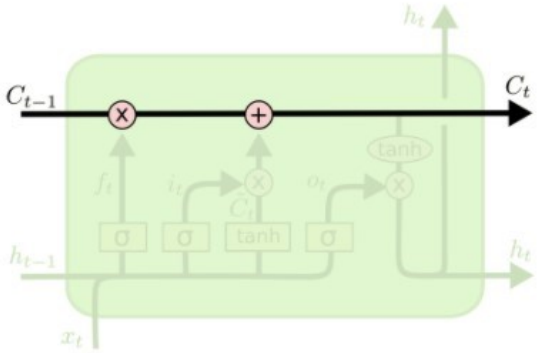


Figura 3.19: Celda de estado. Fuente: [76].

Gracias a la celda de estado es muy sencillo que la información pase de una celda a otra sin cambios bruscos, lo que permite que las redes LSTM puedan tener la memoria a largo plazo de la que carecían las RNN estándar. Además, cada celda LSTM estándar dispone de tres tipos de estructuras denominadas "puertas" que le otorgan la capacidad de añadir o eliminar información de la celda de estado [75, 76]:

- Puerta de olvido o *forget gate*: decide lo que debe ser eliminado de la celda de estado. La capa asociada a esta puerta tiene dos entradas: por una parte, el valor del estado oculto de la celda LSTM anterior (denotado por h_{t-1}), y por otro, el nuevo vector de observaciones que se han realizado (denotado por x_t). Utilizando la función de activación sigmoide, devuelve un número comprendido entre 0 y 1 para cada valor del estado oculto de la celda LSTM anterior. El 1 significa que esa información debe conservarse completamente, y el 0 que debe eliminarse por completo.

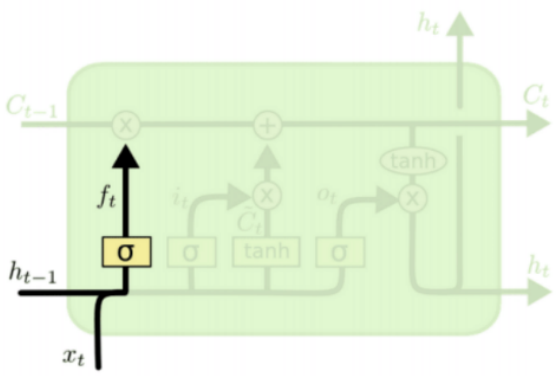


Figura 3.20: Puerta de olvido o *forget gate*. Fuente: [75].

- Puerta de entrada o *input gate*: decide qué valores de la entrada se deben utilizar para actualizar la memoria. Al igual que en la puerta de olvido, esta puerta toma de nuevo los valores h_{t-1} (valor del estado oculto de la celda LSTM anterior) y x_t (nuevo vector de observaciones realizadas). En este caso, encontramos dos capas: una primera capa con la función de activación sigmoide que decide qué valores actualizar en la memoria (de nuevo empleando un rango de valores comprendido entre 0 y 1); y una segunda capa con la función de activación tangente hiperbólica que crea un vector de nuevos valores candidatos. Finalmente, los valores devueltos por ambas capas se combinan y se agregan a la celda de estado.

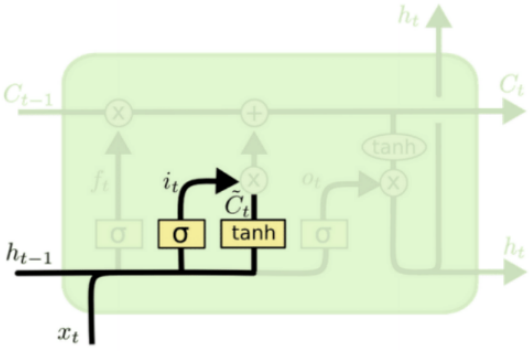


Figura 3.21: Puerta de entrada o *input gate*. Fuente: [75].

- Puerta de salida u *output gate*: decide la salida de la celda LSTM y su nuevo estado oculto interno. Para ello, combina la salida de una capa que usa la función de activación sigmoide aplicada de nuevo sobre los valores h_{t-1} (valor del estado oculto de la celda LSTM anterior) y x_t (nuevo vector de observaciones realizadas), y una versión filtrada de la celda de estado (dado que se aplica la función tangente hiperbólica a cada uno de sus valores antes de proceder a la combinación).

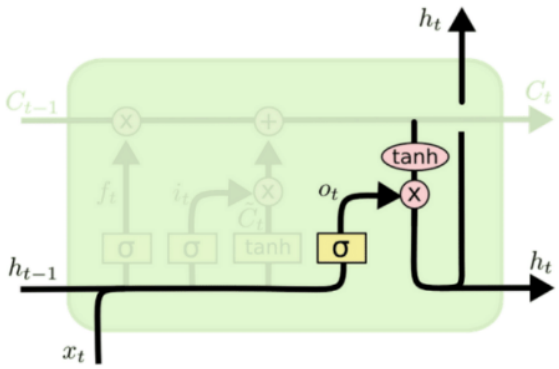


Figura 3.22: Puerta de salida u *output gate*. Fuente: [75].

Las dos arquitecturas más importantes para resolver este tipo de problemas son la arquitectura *Vanilla* y la *Encoder-Decoder* que se explican a continuación.

Red LSTM *Vanilla*

Esta arquitectura es la más básica que puede adoptar una red LSTM y fue definida por primera vez por Hochreiter y Schmidhuber en 1997 [74].

Se utiliza principalmente para problemas de predicción con secuencias pequeñas donde la salida se limita a un valor unidimensional, es decir, a partir de una secuencia de entradas trata de predecir únicamente el próximo valor de la secuencia. Por esta razón, suele estar formada por 3 capas: una capa de entrada, una capa LSTM oculta, y una capa densa de salida para poder generar la salida unitaria (Figura 3.23) [77].

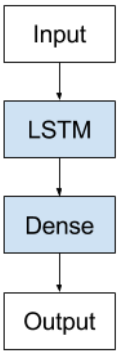


Figura 3.23: Arquitectura LSTM *Vanilla*. Fuente: [77].

Red LSTM *Encoder-Decoder*

Este tipo de arquitectura fue desarrollado en su origen para problemas de procesamiento de lenguaje natural y supuso un gran avance principalmente en el área de traducción de textos (concretamente en la traducción automática estadística) [78].

En la actualidad se utiliza principalmente para trabajar con problemas de predicción (conocidos comúnmente como *seq2seq* o secuencia a secuencia), es decir, para problemas en los que a partir de muchas entradas de observaciones se trata de predecir una secuencia de salida cuya longitud puede variar (ya que con la arquitectura *Vanilla* se restringe la salida a valores unidimensionales).

Las redes LSTM *Encoder-Decoder* están formada por dos modelos cuyo uso en conjunto dan a la arquitectura su nombre:

- Un modelo codificador o *encoder* que se encarga de leer la secuencia de entrada, codificarla, y devolver un vector de tamaño fijo.

- Un modelo decodificador o *decoder* que se encarga de decodificar el vector de longitud fija devuelto por el modelo *encoder*, y generar la secuencia predicha.

Como podemos ver en el ejemplo de la Figura 3.24, una red de este tipo puede estar formada por varias capas *encoder* y por varias capas *decoder*. Añadiendo capas *decoder*, podemos obtener secuencias de salidas de la red de tamaño arbitrario de ahí su gran utilidad para los problemas *seq2seq*. Cabe destacar el hecho de que se suele añadir una capa densa en la salida por cada capa *decoder* utilizada para realizar las transformaciones necesarias y poder generar salidas unitarias por cada capa.

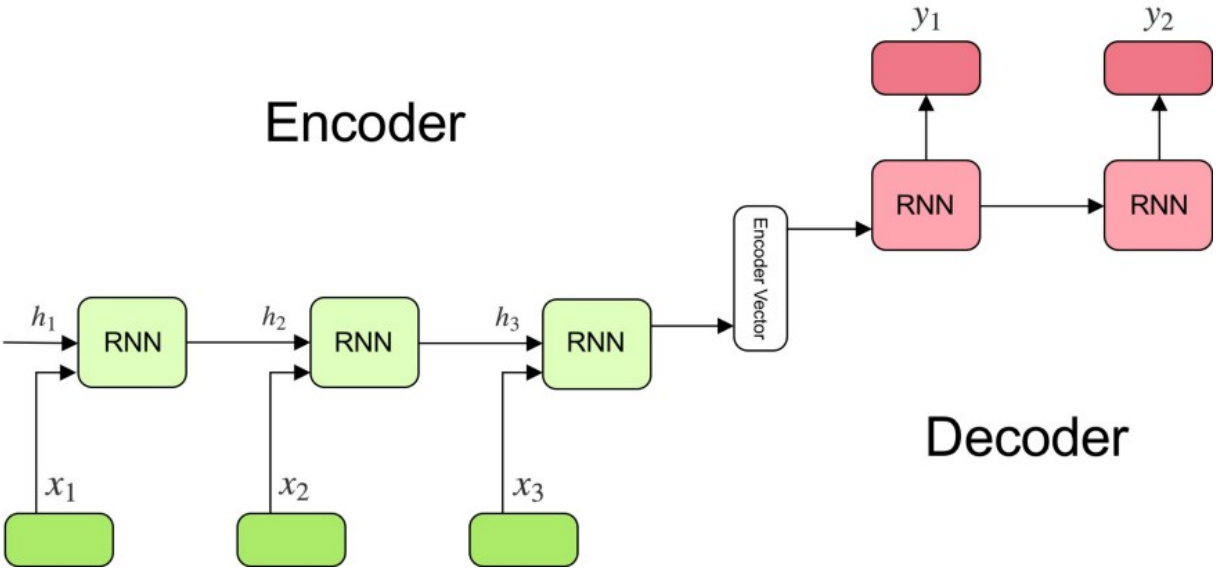


Figura 3.24: Arquitectura LSTM *Encoder-Decoder*. Fuente: [79].

3.5. Redes neuronales convolucionales

Las redes neuronales convolucionales, redes convolucionales o CNN (por sus siglas en inglés: *Convolutional Neural Networks*) son otro subtipo muy conocido de redes neuronales profundas que tratan de imitar al córtex visual del ojo humano con el fin de poder identificar las diferentes características de las entradas, y poder reconocer y distinguir objetos. Este tipo de redes se basan en el modelo de red neuronal desarrollado por Kunihiko Fukushima en 1980 en su artículo *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position* [80].

Históricamente, las redes CNN se han utilizado para la clasificación y el autoetiquetado de imágenes gracias a la naturaleza de las convoluciones, que hacen que estas redes sean aptas para la clasificación de datos en dos dimensiones que estén distribuidos de forma continua y homogénea a lo largo del mapa de entrada. No obstante, en la actualidad también se están utilizando redes convolucionales de una dimensión para la clasificación de series temporales o señales de audio; y de tres dimensiones para la clasificación de datos volumétricos [81, 82].

Los principales componentes de una red CNN son los siguientes [46, 83]:

- Capa de Convolución: es la encargada de extraer las características de la entrada. En esta capa se realiza la operación de *convolución* que consiste en la multiplicación de una parte de los datos de entrada por una matriz de pesos o ponderaciones (denominada filtro o *kernel*), y la suma de todos los valores obtenidos para conseguir un valor que represente al conjunto de datos de entrada. Esta operación se realiza por todo el conjunto de entrada, obteniendo una capa convolucionada o mapa de características (Figura 3.25).

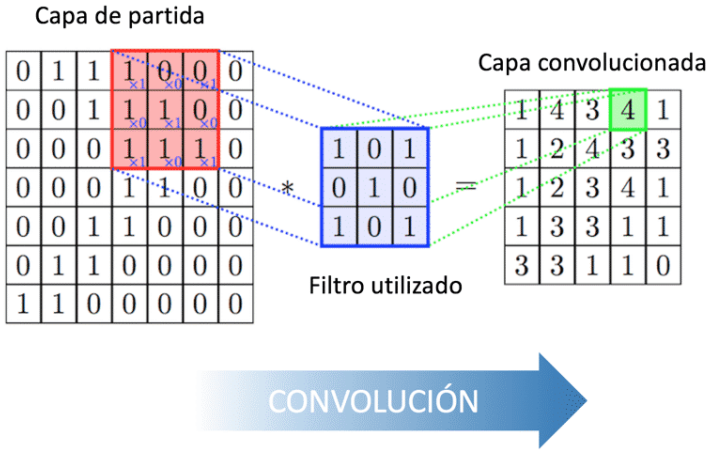


Figura 3.25: Proceso de obtención de una capa convolucionada. Fuente: [83].

Es habitual que sobre la capa de entrada se utilicen gran cantidad de filtros (normalmente 32, 64 o 128), ya que cada filtro permitirá obtener una capa convolucionada diferente.

- Función de activación (ReLU): la aplicación de la función de activación ReLU a la salida de la capa anterior permite transformar los valores negativos en ceros, y dejar los positivos sin modificar como se puede ver con el ejemplo de la Figura 3.26.

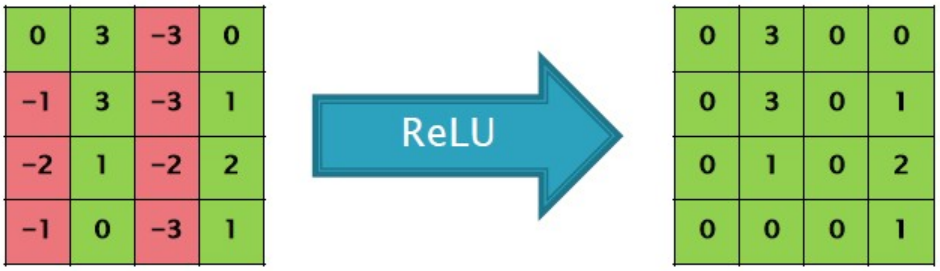


Figura 3.26: Aplicación ReLU sobre una capa convolucionada. Fuente: [46].

- Capa de agrupación o *Pooling*: permite reducir el tamaño de la salida de la capa anterior logrando así la disminución del número de neuronas a utilizar en la siguiente capa sin perder gran cantidad de información, y un entrenamiento más rápido de la red neuronal.

  Para ello, se aplica un filtro de *pooling* sobre dicha capa que puede ser de varios tipos: *Max Pooling* (selecciona el valor máximo), *Average Pooling* (selecciona el valor medio), y *Sum Pooling* (selecciona la suma de los valores). Generalmente el más utilizado es el *Max Pooling* (ejemplo en la Figura 3.27).

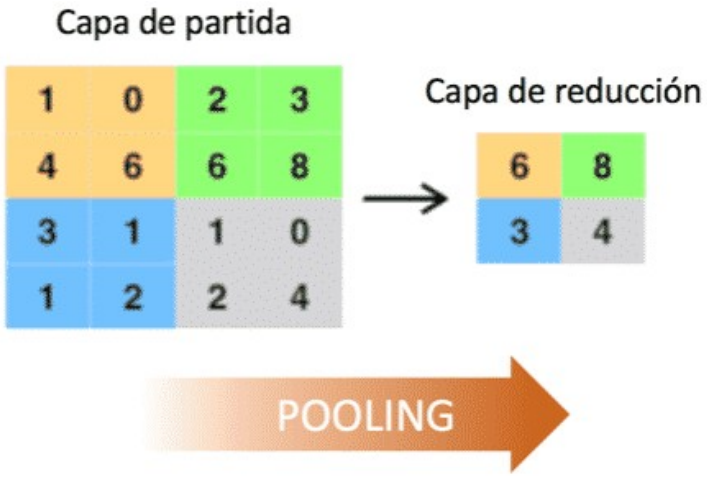


Figura 3.27: Aplicación del filtro *Max Pooling* sobre una capa. Fuente: [83].

- Capa *Fully Connected*: se trata de una capa densa que se utiliza para transformar la matriz tridimensional de características generada en la capa anterior en un vector unidimensional. Esta capa permite combinar de manera eficiente todas las características extraídas en las capas anteriores.

En general, en una red convolucional se suelen colocar varias capas convolucionales y de *Pooling* antes de la capa *Fully Connected* para lograr mejores resultados (ejemplo en la Figura 3.28). Por este motivo, se dice que estas capas trabajan de forma jerárquica, ya que las primeras capas tratan de identificar los elementos básicos, que se van combinando en las capas intermedias, con el fin de que las capas más profundas sean capaces de reconocer formas y objetos más complejos [46].

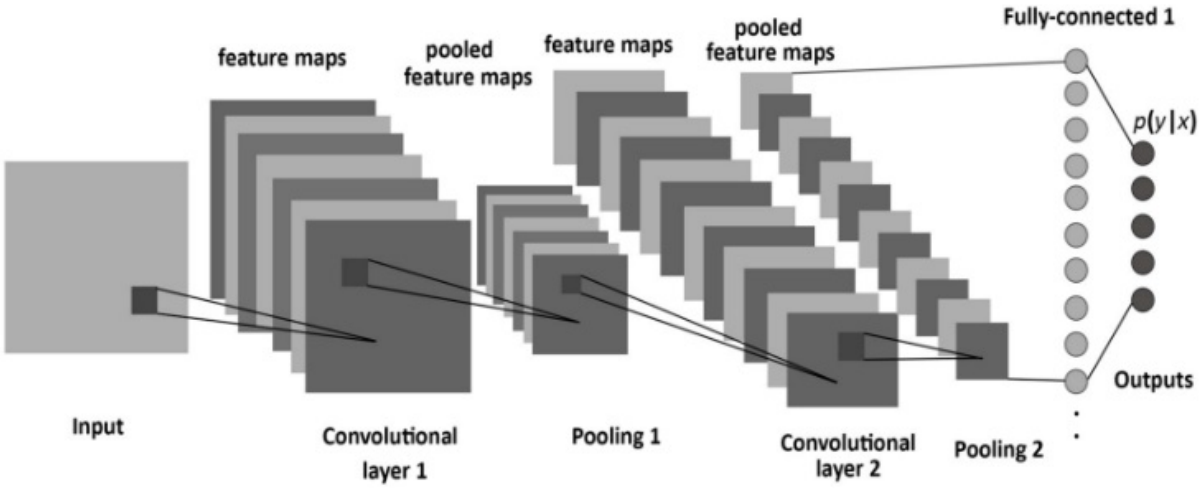


Figura 3.28: Esquema general de una red convolucional. Fuente: [84].

Para el lector interesado en ampliar la información de este tipo de redes neuronales se recomienda la lectura del artículo *A survey of the recent architectures of deep convolutional neural networks* de Asifullah Khan et al. [81].

Capítulo 4

Metodología de trabajo

Este Trabajo Fin de Grado se organizará y desarrollará utilizando la metodología de enseñanza ágil *UVagile*, que trata de adaptar el marco de trabajo ágil *Scrum* al ámbito académico, y plantea una organización iterativa e incremental del proyecto para lograr mejores resultados.

En este capítulo se presenta qué es *Scrum* (Sección 4.1) y cómo se ha adaptado a este trabajo siguiendo los principios de *UVagile* (Sección 4.2). Asimismo, se detallan las herramientas (Sección 4.3) y tecnologías (Sección 4.4) empleadas en el desarrollo del proyecto.

4.1. Marco de trabajo *Scrum*

Scrum [85] es un marco de trabajo ágil ligero que facilita a las personas, equipos y organizaciones la obtención de valor a través de soluciones adaptables para problemas complejos. Entre sus principales beneficios destaca la retroalimentación o *feedback* que se logra con la inclusión de las partes interesadas (*stakeholders*) en el desarrollo del proyecto, y que permite alcanzar mejores resultados en menos tiempo.

Scrum se centra en tres pilares fundamentales: la transparencia (el proceso y el trabajo deben ser visibles para todas las partes), la inspección (el progreso hacia los objetivos debe ser examinado frecuentemente para detectar problemas indeseados) y la adaptación (si algún proceso se desvía de los límites aceptables, se debe poder ajustar lo antes posible para minimizar el desvío adicional).

Esta forma de trabajo utiliza un enfoque iterativo e incremental para optimizar la previsibilidad y controlar los riesgos. Para lograrlo, *Scrum* define un *equipo Scrum* (Subsección 4.1.1), cinco eventos (Subsección 4.1.2) y tres artefactos (Subsección 4.1.3).

4.1.1. Equipo *Scrum*

La unidad básica de trabajo de *Scrum* la compone un pequeño grupo de personas (normalmente entre 5 y 11 personas) que se denomina *Equipo Scrum* o *Scrum Team*. Este equipo está conformado por un *Scrum Master*, un propietario de producto (*Product Owner*) y un grupo de desarrolladores, en el que no se definen subequipos ni jerarquías.

Las tareas que tienen asignadas son las siguientes:

- **Desarrolladores:** son las personas que se ocupan del desarrollo del producto. Se encargan de crear un plan de trabajo para poder cumplir con los objetivos de cada *sprint*, de asegurar la calidad del producto desarrollado, etc.

- ***Product Owner*:** es el responsable de maximizar el valor del producto que ha sido realizado por los desarrolladores. También se ocupa de que el trabajo pendiente del producto sea transparente, visible y comprendido; y de organizar y gestionar el *Product Backlog*.

- ***Scrum Master*:** es el líder del equipo de trabajo. Se responsabiliza de eliminar los problemas que puedan surgir durante el desarrollo del proyecto, de asegurarse de que se llevan a cabo todos los eventos de *Scrum*, de capacitar al resto de miembros en autogestión y multifuncionalidad, etc.

4.1.2. Eventos *Scrum*

Los eventos en *Scrum* permiten establecer regularidad y minimizar las reuniones no establecidas. Además, cada evento en *Scrum* permite analizar y ajustar los artefactos de *Scrum*, lo que deriva en una mejor adaptación del equipo al trabajo.

Los proyectos se dividen en eventos temporales denominados *sprints*. Un *sprint* es un marco temporal de longitud fija y de duración máxima un mes, cuya finalidad es entregar un incremento de producto funcional tras su finalización. Todos los *sprints* deben tener la misma duración, un objetivo a alcanzar, y las herramientas necesarias para lograrlo.

La entrega de un producto funcional al finalizar cada *sprint* permite identificar todo aquello que se ha hecho bien, y analizar las partes a mejorar. Para lograr esto, en cada *sprint* se realizan los siguientes eventos:

- ***Sprint Planning*:** es una reunión que se realiza al comienzo del *sprint* en la que participa todo el *Scrum Team* y que tiene como objetivo planificar el desarrollo del *sprint*. Tiene una duración máxima de 8 horas y en ella se decide en qué se va a trabajar en dicho *sprint* y cómo se va a llevar a cabo.

- ***Daily Scrum*:** es una reunión breve de duración máxima 15 minutos que se realiza todos los días laborables del *sprint*, en el mismo sitio y a la misma hora. En ella participan los desarrolladores del *Scrum Team* para planificar el trabajo de las próximas 24 horas y ver los impedimentos que surgieron en el día anterior.

Esta reunión no es el único momento del día en el que se reúnen los desarrolladores, ya que es frecuente que se junten en otros momentos del día para debatir de forma detallada parte del trabajo a desarrollar.

- ***Sprint Review*:** es una reunión de duración máxima 4 horas celebrada a la finalización del *sprint*. En ella el *Scrum Team* revisa con los *stakeholders* el resultado del *sprint* y se analiza el progreso hacia el objetivo del producto.
- ***Sprint Retrospective*:** es una reunión de duración máxima 8 horas realizada después del *Sprint Review*. En ella el *Scrum Team* analiza cómo se desarrolló el último *sprint* y cómo se podría aumentar la calidad y la eficacia del trabajo realizado.

4.1.3. Artefactos *Scrum*

Los artefactos de *Scrum* están diseñados para maximizar la transparencia de la información esencial, y representan trabajo o valor. Su uso permite organizar y gestionar el trabajo realizado en cada *sprint*, y las tareas necesarias para alcanzar el producto final.

Scrum define los siguientes artefactos:

- ***Product Backlog*:** lista priorizada que constituye la única fuente de funcionalidades, requisitos y mejoras que se deben realizar en el producto. El *Product Owner* es el encargado de crearla y mantenerla priorizada constantemente de acuerdo a la visión de negocio y las necesidades del cliente.
- ***Sprint Backlog*:** lista de elementos seleccionados del *Product Backlog* para un *sprint*. El equipo de desarrolladores es el encargado de gestionarla, ya que constituye la imagen visible de su avance y su consecución implica que se han cumplido todos los objetivos fijados en dicho *sprint*.
- **Incremento:** suma de todas las tareas pertenecientes al *Product Backlog* que han sido completados (durante un *sprint* y todos los anteriores).

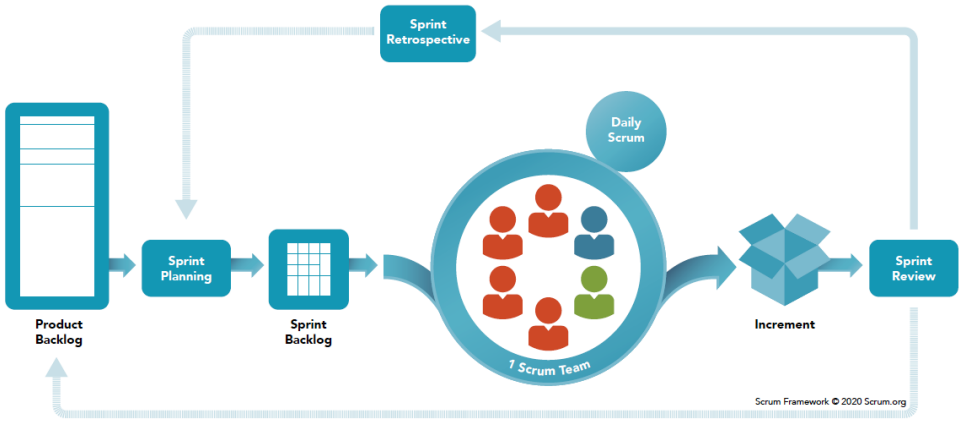


Figura 4.1: Framework de Scrum. Fuente: [86].

4.2. Metodología UVagile

UVagile [87] es un proyecto de innovación docente de la Universidad de Valladolid cuyo objetivo principal es implantar los valores y principios de *Agile* en el ámbito universitario. Esta metodología está basada en *Scrum* y con su uso se quiere fomentar el contacto entre alumnos y profesores, aportar un *feedback* temprano al alumnado, fomentar el aprendizaje activo, etc. [88].

En este Trabajo Fin de Grado se trabajará siguiendo los principios de UVagile para lograr una mejor adaptación de la carga de las tareas a realizar al ritmo real del proyecto. Con ello, se espera lograr un resultado final que se ajuste más y mejor a los objetivos propuestos al inicio del proyecto. Sin embargo, se deben realizar algunas modificaciones de esta metodología para adaptarlo al desarrollo de este proyecto, ya que UVagile se está utilizando hoy en día en la docencia del Grado de Ingeniería Informática de Servicios y Aplicaciones de Segovia (Universidad de Valladolid) [89], pero su uso no está consolidado para Trabajos Fin de Grado.

En este proyecto se utilizarán los mismos artefactos que define *Scrum*, y el ciclo de vida del proyecto será similar al de cualquier otro realizado con *Scrum*: en primer lugar, se creará un *Product Backlog* con todas las tareas que se deberán realizar. Después, al inicio de cada *sprint*, se escogerán los elementos que deberán ser abordados en dicho *sprint* conformando el *Sprint Backlog*, y durante cada *sprint* se realizarán algunas reuniones intermedias. Al finalizar cada *sprint*, se conseguirá un producto funcional que permitirá obtener *feedback* de todas las partes interesadas, y comenzará el siguiente *sprint*.

A continuación, se explica quiénes forman el equipo UVagile en la adaptación de esta metodología al desarrollo de un Trabajo Fin de Grado (Sección 4.2.1) y los eventos de UVagile llevados a cabo durante el transcurso del proyecto (Sección 4.2.2).

4.2.1. Equipo UVagile

Los roles definidos en un equipo UVagile son los mismos que los de un equipo *Scrum* (Sección 4.1.1). Sin embargo, para el desarrollo de este proyecto, el equipo UVagile está compuesto únicamente por 3 personas (el alumno y los tutores) que se distribuyen de la siguiente manera:

- **Desarrolladores:** el único miembro es el alumno. Será el encargado de llevar a cabo toda la realización del proyecto.

- ***Product Owner*:** aunque en un *Scrum Team* está figura solo la adopta una persona, en este caso es adoptada por los dos tutores. Se encargan de seleccionar y priorizar las tareas que deben realizar los desarrolladores (en este caso, el alumno). También son el nexo entre los desarrolladores y los *stakeholders* (el tribunal de la defensa, el comité de Grado, etc).

- ***Scrum Master*:** en este proyecto esta figura no está completamente definida. La mayor parte del tiempo el alumno adopta este rol ya que es el encargado de asegurarse de que se llevan a cabo todos los eventos UVagile, de solucionar los problemas que surjan, y de capacitar a los desarrolladores para que se autogestionen (en este caso, a él mismo). No obstante, a veces los tutores también pueden adoptar este rol para eliminar posibles contratiempos que el alumno no puede resolver por sí solo.

4.2.2. Eventos UVagile

Este proyecto se va a dividir en cinco *sprints*, cada uno de tres semanas de duración, comenzando el primero el día 03/03/2021, y el último el día 23/06/2021. En cada uno se van a realizar varios eventos similares a los definidos en *Scrum* con el fin de establecer una regularidad y reducir las reuniones no establecidas.

El primer evento de UVagile se denomina *Inicio del Proyecto*, y se trata de una reunión donde se junta todo el equipo UVagile para consolidar los objetivos del proyecto y definir la dinámica de trabajo. Después también se planifica el primer *sprint* del proyecto.

Por otro lado, dado que el equipo de desarrolladores solo está formado por una persona, el avance del proyecto de un día para otro no es significativo. Por esta razón, la *Daily Scrum* se sustituye por el siguiente evento:

- **Reunión de Sincronización (*Weekly*):** se trata de una reunión breve con una duración máxima 15 minutos que se realiza una vez por semana en la que participa todo el equipo UVagile. En ella se explican los avances desde la reunión anterior y los impedimentos que han surgido, y se planifica el trabajo de la siguiente semana.

Al terminar cada *sprint*, los eventos *Sprint Review*, *Sprint Retrospective* y *Sprint Planning* se concentran en una único evento denominado *Retroplanning*. Esta reunión tiene una duración aproximada de una hora, participa todo el equipo UVagile, y en ella se realizan los siguientes eventos:

- ***Review*:** constituye la primera parte de la *Retroplanning*. En ella se expone el trabajo realizado durante el *sprint* (se describe el incremento desarrollado al término del mismo), y en qué medida satisface los objetivos del *sprint*.

- **Retrospectiva:** se realiza a continuación y en ella se analizan las cosas que han ido bien durante el *sprint*, las que se deben mejorar, y los posibles planes de mejora para evitar que sucedan de nuevo.

- ***Planning*:** constituye la ultima parte de la *Retroplanning*. En ella se establecen los objetivos que se deben conseguir en el siguiente *sprint* y las tareas a desarrollar para poder alcanzarlo.

Finalmente, también se incluye un nuevo evento conocido con el nombre de *Reuniones de Socorro* en la que participan las partes necesarias del equipo UVagile. Estas reuniones

se llevan a cabo de forma excepcional durante el transcurso de cualquier *sprint* en caso de que exista algún problema que impidan el correcto avance de estos. Solo se deben solicitar cuando el contratiempo surgido no puede ser abordado en la *Weekly* por su complejidad, ni se puede esperar a la *Retroplanning* para afrontarlo.

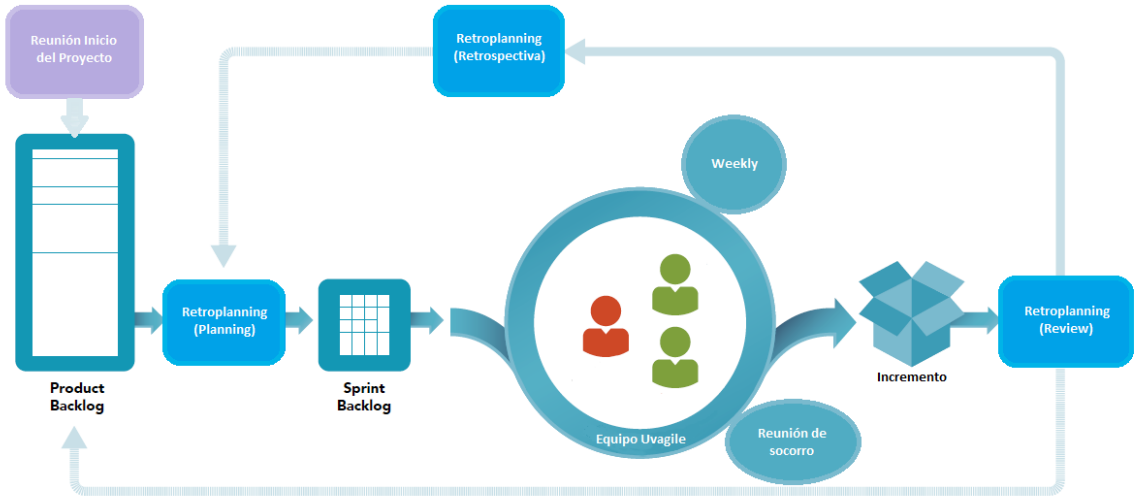


Figura 4.2: Metodología UVagile.

4.3. Herramientas utilizadas

Durante el desarrollo de este proyecto se han utilizado las siguientes herramientas:

- Jupyter Notebook: aplicación web que permite crear y compartir documentos que contienen una lista ordenada de celdas con entradas y salidas en formato JSON. Estas celdas pueden contener código (escrito en Python, Julia, R, etc.), gráficos, textos, etc. Fue diseñado en 2014 para desarrollar software libre y servicios de computación interactiva compatibles con diferentes lenguajes de programación, y entre sus principales usos destaca la creación y entrenamiento de modelos de aprendizaje automático, la visualización de datos y la modelización estadística [90].

- Overleaf: herramienta de publicación y redacción en línea que permite la colaboración en tiempo real entre personas para una sencilla creación, redacción, edición y publicación de documentos (generalmente de carácter científicos). Utiliza el sistema de composición de textos LaTeX [91].

- Google Colab o Colaboratory: servicio *cloud* gratuito creado por Google que permite ejecutar y programar código escrito en Python en un navegador. Se basa en el uso de los documentos de Jupyter Notebook, no requiere ninguna configuración, permite el uso gratuito de GPUs y TPUs de Google para acelerar los procesos, y posibilita la opción de compartir contenido fácilmente [92].

Además de estas, para una correcta planificación, gestión y coordinación durante el transcurso del proyecto siguiendo la metodología UVagile, se han utilizado también las siguientes herramientas:

- Microsoft Teams: espacio de trabajo *cloud* para desarrollar un proyecto en equipo que permite colaborar a los miembros de este mediante diferentes canales de comunicación. Además, permite la edición de archivos de forma colaborativa y la integración con otras aplicaciones [93].

  Esta herramienta permitirá al equipo UVagile comunicarse de forma no presencial cuando sea necesario, así como almacenar las versiones de los documentos del proyecto (memoria, presentaciones, etc.) de forma sencilla y efectiva.

- Trello: software en línea para la gestión y organización de proyectos cuya estructura está basada en un tablero Kanban (herramienta empleada para mapear y visualizar los flujos de trabajo en la metodología ágil Kanban [94]). Permite organizar ideas y tareas en diferentes columnas para lograr una rápida visión del avance de un proyecto [95].

  Por esta razón se usará para organizar las tareas a realizar en cada *sprint* de este proyecto. Para ello, cada tarea a realizar se representará en una tarjeta independiente, y se le agregará un título (y una descripción si fuera necesario) para diferenciarla del resto de tareas. Además, las tareas se podrán encontrar en diferentes columnas, donde cada columna representará un estado diferente por el que puede pasar una tarea para su realización en un determinado *sprint*.

  Para este proyecto se crearán cinco columnas en Trello: *Objetivos* (incluirá cada uno de los objetivos que se deben alcanzar al finalizar cada *sprint*), *Tareas planificadas* (incluirá las tareas que se deben realizar en el *sprint* pero que aún no se han comenzado), *Tareas comenzadas* (incluirá las tareas programadas durante un *sprint* que se han empezado a realizar), *Tareas bloqueadas* (incluirá las tareas que no se pueden realizar por algún motivo y que requieren la ayuda del *Product Owner*) y *Tareas finalizadas* (incluirá las tareas que ya han sido terminadas).

  En la Figura 4.3 se puede observar una captura de pantalla de este tablero durante el transcurso de este proyecto.

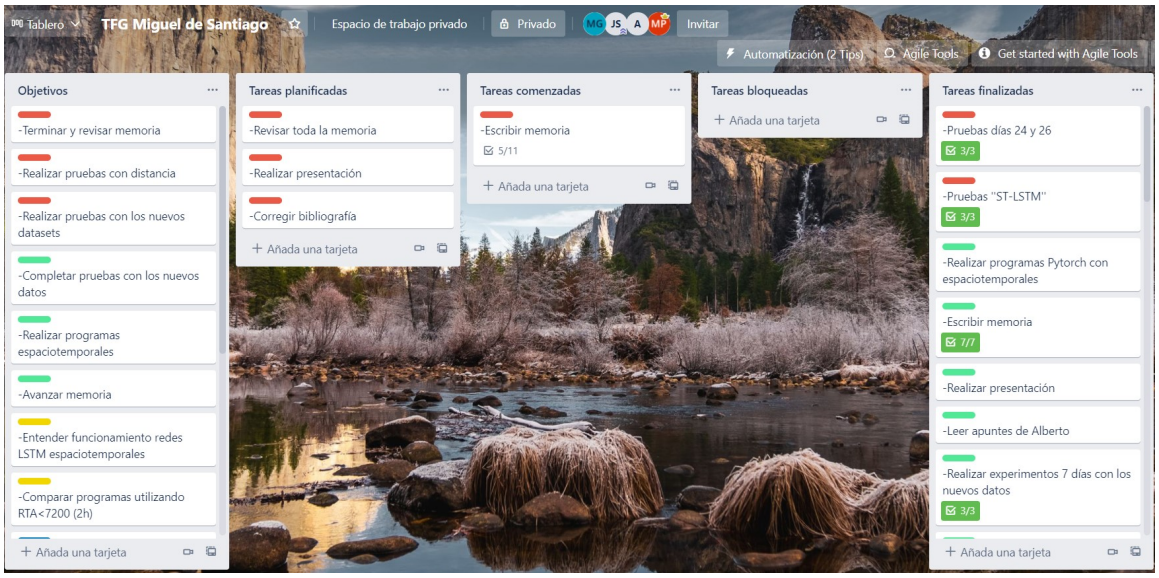


Figura 4.3: Tablero del proyecto.

- EasyRetro: plataforma en línea que permite realizar las retrospectivas de forma fácil y sencilla al finalizar los *sprints* con el fin de debatir lo que salió bien, lo que se podría mejorar, y los posibles planes de mejora. Además, permite exportar los resultados para usarlos en otras herramientas [96].

  La reunión de retrospectiva al finalizar cada *sprint* es fundamental tanto en *Scrum* como en UVagile, ya que permite identificar los posibles obstáculos que pueden haber surgido y las acciones de mejora para lograr mejores resultados a corto plazo. Los resultados de estas reuniones a lo largo de este proyecto se encuentran detallados en el Apéndice A.

4.4. Tecnologías utilizadas

Durante este proyecto se han utilizado las siguientes tecnologías:

- Python: lenguaje de programación de código abierto, multiparadigma e interpretado, que soporta la orientación a objetos, la programación imperativa y funcional, y cuya filosofía se enfoca en la legibilidad de su código. Fue creado por Guido van Rossum a finales de los años ochenta en los Países Bajos, y en la actualidad se considera uno de los principales lenguajes de programación por su gran versatilidad. Entre sus principales usos destaca el análisis y visualización de datos, y la creación de modelos de aprendizaje automático gracias a la gran cantidad de bibliotecas, librerías y módulos que posee [97].

  Para el desarrollo de este proyecto se han utilizado las siguientes librerías:

- NumPy: librería que incorpora funciones matemáticas de alto nivel para poder trabajar en computación científica con arrays multidimensionales (tensores) y matrices [98].
- Pandas: extensión de Numpy que proporciona estructuras de datos (denominadas *dataframes*) y operaciones para manipular y analizar tablas numéricas y series temporales de forma rápida y sencilla [99].
- Matplotlib: librería clásica de Python para la generación de gráficos sencillos de forma rápida a partir de listas o arrays. Permite la creación de gran cantidad de gráficos como histogramas, series temporales o diagramas de barras [100].
- Seaborn: librería de visualización de datos basada en Matplotlib que permite la creación de gráficos estadísticos. Se considera la principal referencia para la visualización exploratoria debido a que proporciona gráficos más atractivos visualmente y más fáciles de crear que los de Matplotlib [101].
- TensorFlow: librería de código abierto destinada al desarrollo y la implementación de modelos de aprendizaje automático [102].
- Keras: librería destinada al desarrollo de modelos de aprendizaje automático y que actúa como interfaz de Tensorflow. Simplifica el desarrollo de las redes neuronales profundas y soporta las redes concurrentes y convolucionales; de ahí su uso dentro de este proyecto [103].
- Math: permite realizar operaciones matemáticas de forma sencilla. Incluye funciones de potencias, de logaritmos, trigonométricas o de conversión angular entre otras muchas [104].
- Sklearn (o Scikit-learn): proporciona herramientas sencillas y eficientes para su uso en el aprendizaje automático y en el modelado estadístico. Se basa en Numpy y Matplotlib, e incluye funciones de clasificación, regresión, preprocesamiento, etc. [105].
- Haversine: mide la distancia (en kilómetros) entre dos puntos de la superficie terrestre utilizando su latitud y longitud [106].

Capítulo 5

Planificación

En este capítulo se incluye la estimación de esfuerzos efectuada al inicio del proyecto para poder dividir las tareas de forma equilibrada entre los diferentes *sprints* (Sección 5.1) y la planificación temporal inicial de estas tareas (Sección 5.2). Asimismo, se presenta un presupuesto inicial con los posibles gastos derivados del proyecto (Sección 5.3).

El capítulo concluye con el balance final (temporal y de costes) realizado tras la finalización del proyecto (Sección 5.4).

5.1. Estimación del esfuerzo

Para planificar todo el trabajo que se debe realizar es esencial analizar las diferentes tareas que conforman el proyecto y cuya consecución implica alcanzar los objetivos establecidos al inicio del mismo, así como el tiempo estimado necesario para realizar cada una de ellas. Por este motivo, se definirán una serie de tareas descriptivas (análogas a las historias de usuario) y se estimará el esfuerzo que habrá que dedicar a cada una de ellas utilizando la técnica del *planning poker* y los puntos de historia.

El *planning poker* es una de las técnicas más conocidas y efectivas utilizada por equipos ágiles de desarrollo software que permite caracterizar la complejidad de cada tarea (y estimar de forma aproximada el tiempo que requerirá basado en la experiencia del equipo). Su funcionamiento es el siguiente [107]:

- En primer lugar, cada miembro del equipo de desarrollo recibe un mazo de cartas que es idéntico para todos ellos. Aunque existen varios modelos de mazos, el más utilizado es el que está compuesto por 12 cartas y se basa en la sucesión de Fibonacci. Este set de cartas contiene nueve cartas diferentes con los valores 0, 1/2, 1, 2, 3, 5, 8, 13 y 21; y tres cartas extras: una con el símbolo de infinito (∞), otra con el símbolo de interrogación (?), y otra con un símbolo de un café o alguna bebida (Figura 5.1).

Figura 5.1: Cartas de estimación de Planning Poker. Fuente: [107].

- A continuación, se debe definir qué representan los valores de estas cartas, ya que pueden ser unidades de tiempo (días, semanas) o grado de dificultad (puntos de historia). Independientemente de lo que representen, el *0* simboliza el valor más bajo y el *21* el más alto; el ∞ se utiliza para demostrar imposibilidad (por lo que la tarea se debe descomponer); el *?* se emplea para indicar incertidumbre o desconocimiento; y el símbolo del café se usa cuando se quiere pedir un tiempo de descanso.

- Finalmente, para cada tarea, todos los miembros del equipo deben mostrar la carta que mejor define para ellos la complejidad de dicha actividad según la medida establecida. Si después de conocer todas las estimaciones no se llega a un consenso, se debe debatir las causas y repetir la estimación utilizando las cartas (hasta un máximo de tres veces). Si tras esto no se llega a un acuerdo, se tasa dicha tarea con la estimación más alta o con la media de las estimaciones.

Para este proyecto, el alumno y los tutores utilizarán esta técnica para estimar la complejidad de cada tarea. Cada tarea se dividirá en subtareas, y cada una de ellas será estimada utilizando los puntos de historia o *story points* (SP), que denotan el esfuerzo relativo que supone realizar cada tarea frente otras.

La lista de tareas y subtareas que componen este proyecto y la estimación de cada una de ellas es la siguiente:

- **T1. Estudio del dominio del problema.**
  - **T1.1.** Estudio problemática de la gestión de tráfico aéreo (1 SP).
  - **T1.2.** Estudio de efectos de los retrasos aéreos (1 SP).
  - **T1.3.** Estudio de propuestas similares utilizando redes neuronales LSTM (2 SP).
  - **T1.4.** Estudio de propuestas similares utilizando redes neuronales LSTM espacio-temporales (5 SP).

- **T2. Estudio de las redes neuronales.**
  - **T2.1.** Estudio funciones de activación (1/2 SP).
  - **T2.2.** Estudio aprendizaje de una neurona y de una red neuronal (1 SP).
  - **T2.3.** Estudio factores que afectan a la configuración de una red neuronal (1/2 SP).
  - **T2.4.** Estudio redes neuronales LSTM (3 SP).
  - **T2.5.** Estudio redes neuronales LSTM espacio-temporales (2 SP).
- **T3. Implementación de los modelos.**
  - **T3.1.** Carga e interpretación del conjunto de datos (1/2 SP).
  - **T3.2.** Implementación de una red LSTM simple (5 SP).
  - **T3.3.** Configurar red LSTM de forma óptima (2 SP).
  - **T3.4.** Implementar primera LSTM espacio-temporal (5 SP).
  - **T3.5.** Implementar red LSTM espacio-temporal específica para el problema (8 SP).
- **T4. Evaluación y análisis de resultados.**
  - **T4.1.** Estudio de los resultados obtenidos con redes LSTM (1 SP).
  - **T4.2.** Estudio de los resultados obtenidos con redes LSTM espacio-temporales (1 SP).
  - **T4.3.** Interpretar resultados de todos los conjuntos y modelos de datos (3 SP).
- **T5. Documentación del proyecto.**
  - **T5.1.** Contextualizar el proyecto (1 SP).
  - **T5.2.** Documentar metodología (2 SP).
  - **T5.3.** Documentar estimación, planificación, presupuestos iniciales (2 SP).
  - **T5.4.** Documentar balance final (1/2 SP).
  - **T5.5.** Documentar dominio del problema (2 SP).
  - **T5.6.** Documentar redes neuronales (3 SP).
  - **T5.7.** Documentar construcción de los modelos (3 SP).
  - **T5.8.** Documentar análisis de resultados (2 SP).
  - **T5.9.** Documentar conclusiones y trabajos futuros (1 SP).
  - **T5.10.** Revisar memoria al terminar el proyecto (3 SP).
  - **T5.11.** Realizar presentación final (2 SP).

Tras realizar este estudio, se ha obtenido un total de 63 puntos de historia que se deberán distribuir entre los 5 *sprints* que conforman el proyecto. Esta distribución de las tareas y subtareas entre los diferentes *sprints* se detallará en la Sección 5.2, aunque hay que tener en cuenta que no existe una equivalencia directa y estandarizada entre los puntos de historia y el número de horas necesarias para su realización.

5.2. Planificación del trabajo

Este proyecto se desarrollará como parte de la asignatura 'Trabajo Fin de Grado' del Grado de Ingeniería Informática de Servicios y Aplicaciones de Segovia (Universidad de Valladolid). Por esta razón, tanto el alcance como la duración de este estarán limitados por la carga de trabajo que establece dicha asignatura. Al tratarse de una asignatura de 12 créditos ECTS [108], dado que cada crédito equivale a 25 horas de trabajo [109], este proyecto se deberá desarrollar en 300 horas.

Como se ha explicado anteriormente, este proyecto se dividirá en 5 *sprints*, cada uno de tres semanas (lo que equivale a una carga de trabajo de 20 horas semanales). Se ha decidido que la reunión de *Inicio del Proyecto*, las *Retroplanning* y las *Weekly* (explicadas en la Subsección 4.2.2) se realizarán los miércoles, y por ello, el primer *sprint* comenzará el miércoles día 3 de febrero de 2021, y el último finalizará el martes día 13 de julio de 2021. Además hay que destacar que los fines de semana y los días festivos no se consideran días laborables, por lo que en ellos no se realizará ningún tipo de trabajo relacionado con este proyecto. Finalmente, aunque el calendario académico del curso 2020-2021 establece que las vacaciones de Semana Santa del año 2021 son del viernes 26 de marzo al lunes 5 de abril [110]; para este proyecto solo se considerará festiva la semana del miércoles 31 de marzo al martes 6 de abril.

Para poder realizar todas las tareas presentadas en la Sección 5.1 en este tiempo, se deberán distribuir entre los *sprints* de manera adecuada. Para ello, se repartirán de forma que en cada *sprint* se realicen un número similar de puntos de historia; y dado que en total se han obtenido 63 SP, aproximadamente se harán unos 12,6 SP por *sprint*.

La Tabla 5.1 detalla la distribución de tareas por *sprint* y la Figura 5.2 muestra el diagrama de Gantt esquematizado correspondiente a su planificación temporal por semanas. Sin embargo, para una mejor visualización de este diagrama, se indican los tramos por semanas (que van de miércoles a martes) en vez de por días, y no se han incluido ninguna de las reuniones que se deben celebrar en UVagile (Subsección 4.2.2).

Sprint	Fecha inicio	Fecha fin	Tareas	SP
#1	03/03/2021	23/03/2021	T1.1. / T1.2. / T1.3. / T2.1. / T2.2. / T2.3. / T2.4 / T3.1./ T5.6.	12.5
#2	24/03/2021	20/04/2021	T3.2. / T3.3. / T4.1. / T5.1. / T5.2. / T5.3.	13
#3	21/04/2021	11/05/2021	T1.4. / T2.5. / T3.4. / T5.5.	14
#4	12/05/2021	01/06/2021	T3.5. / T4.2. / T4.3.	12
#5	02/06/2021	22/06/2021	T5.4. / T5.7. / T5.8. / T5.9. / T5.10. / T5.11.	11.5

Tabla 5.1: Distribución de las tareas por *sprint*.

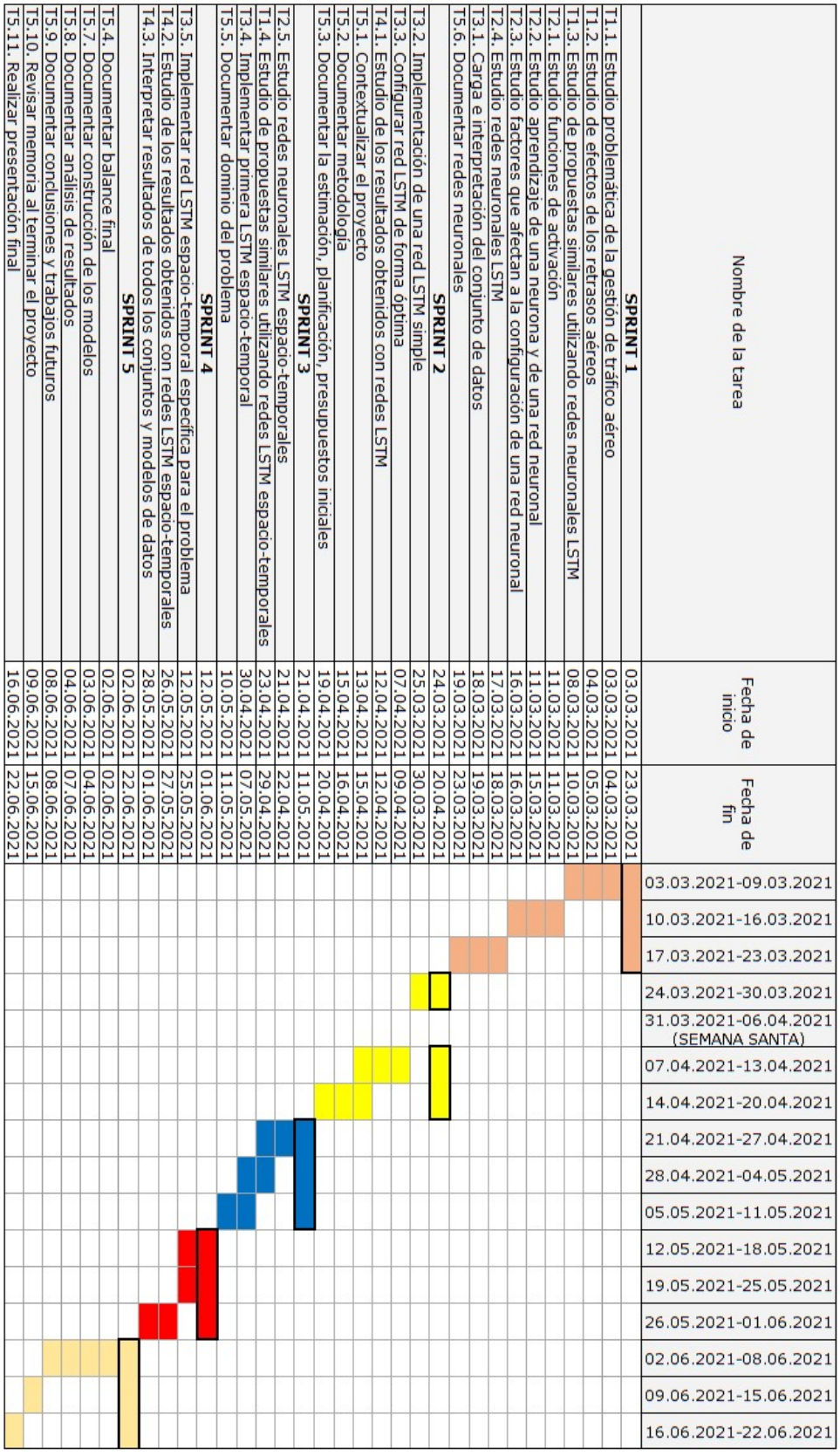


Figura 5.2: Planificación inicial de las tareas.

5.3. Presupuesto

Tras haber realizado la planificación temporal, es necesario estimar los costes del proyecto. Sin embargo, en un proyecto de investigación como este, es especialmente complicado realizar un presupuesto inicial que se corresponda realmente con los costes reales al final de este, principalmente porque no se saben exactamente las herramientas que se pueden llegar a necesitar durante el transcurso de este, o los contratiempos que pueden afectar directamente a las estimaciones iniciales (nuevas ideas surgidas tras el descubrimiento de algún dato, bloqueos por la inexistencia de documentación y por la ausencia de experiencia previa con la tarea, etc.).

Los presupuestos que se presentan a continuación se desglosan en tres categorías (*hardware*, *software* y costes de personal) y se corresponden con la estimación inicial realizada al comienzo del proyecto. Sin embargo, hay que destacar que para calcular el coste real asociado al uso del *hardware* y del *software*, se deben usar las fórmulas 5.1 que tienen en cuenta la frecuencia de uso, el tiempo utilizado, y el coste mensual de estas herramientas.

$$
\begin{aligned}
&Coste\_por\_mes\ (€/mes) = \frac{\text{Coste total (€)}}{\text{Vida útil (meses)}} \\
&Coste\_real\_por\_mes\ (€/mes) = \text{Coste\_por\_mes} \times \text{Porcentaje de uso} \\
&Coste\_real\ (€) = \text{Coste\_real\_por\_mes} \times \text{Meses de uso (meses)}
\end{aligned} \tag{5.1}
$$

- *Hardware*: el desarrollo del proyecto se llevará a cabo en un ordenador personal de gama media-alta cuyas especificaciones son: procesador i7-8550U, sistema operativo de 64 bits, 16 GB de RAM, 256 GB de memoria SSD, y 1 TB de memoria HDD. Además, para conseguir una correcta comunicación con el resto del equipo UVagile, poder consultar documentación, emplear herramientas *online*, etc., es necesario disponer de una conexión a Internet por lo que se incluirá este gasto en el presupuesto. El resto de gastos corrientes, como la electricidad para cargar el ordenador, no son dedicados, por lo que no se incluirán. Por todo ello, el coste total asociado al *hardware* es de **186 €** y se desglosa de la siguiente manera:

Herramienta	Coste total (€)	Vida útil	% uso	Meses uso	Coste real (€)
Ordenador personal	1300	5 años	80	4,5	78
Conexión a Internet	30 (por mes)	-	80	4,5	108

Tabla 5.2: Costes asociados al *hardware*.

- *Software*: El coste total asociado al *software* es de **0 €** ya que todas las herramientas utilizadas tienen licencia gratuita para estudiantes como se puede observar en la Tabla 5.3.

Herramienta	Coste por mes (€)	% uso	Meses uso	Coste real (€)
Google Colab	0	20	4,5	0
Jupyter Notebook	0	40	4,5	0
Overleaf	0	40	4,5	0
Microsoft Teams	0	60	4,5	0
Trello	0	20	4,5	0
EasyRetro	0	5	4,5	0

Tabla 5.3: Costes asociados al *software*.

- Costes de personal: aunque el trabajo va a ser realizado por una única persona, esta tomará diferentes roles a lo largo del proyecto (analista de datos junior y desarrollador Python junior). Por esta razón, su sueldo bruto deberá ser la suma del obtenido por cada rol adoptado, es decir, 4885,17 €.

  Nota: los sueldos de la Tabla 5.4 se han hallado teniendo en cuenta los datos proporcionados por [111] y que un trabajador medio realiza un total de aproximadamente 1800 horas al año.

Rol	Sueldo (€/hora)	Horas	Total (€)
Analista de datos junior	16,59	120	1990,87
Desarrollador Python junior	16,08	180	2894,3

Tabla 5.4: Sueldos según el rol contratado.

  Además, dado que es necesario dar de alta en la Seguridad Social a esta persona, se debe tener en cuenta este coste adicional, que será de 1568,14 €, ya que se corresponde con el 32,1 % del sueldo bruto de la persona (ya que se trata de un contrato temporal [112]). Por todo ello, el coste total asociado al personal es de **6453,31 €** y se desglosa de la siguiente manera:

Motivo	Coste real (€)
Sueldo analista de datos junior	1990,87
Sueldo desarrollador Python junior	2894,3
Seguridad Social	1568,14

Tabla 5.5: Costes asociados al personal.

A partir de este estudio se prevé que el coste total del proyecto sea de **6639,31 €**, de los cuales 186 € derivarán del uso de recursos *hardware*, y 6453,31 € de costes de personal.

5.4. Balance final del proyecto

En esta sección se muestra el análisis temporal y el balance de costes que se ha realizado una vez terminado el proyecto.

5.4.1. Análisis temporal

Al seguir la planificación temporal efectuada al comienzo del proyecto han surgido una serie de contratiempos que han supuesto un cambio en dicha planificación:

1. Como consecuencia de los exámenes finales de principios de junio y la necesidad de poder dedicar algo de tiempo al Trabajo Fin de Grado del Grado de Matemáticas se decidió realizar una pausa de tres semanas en este proyecto. Concretamente, desde el 12 de mayo de 2021 hasta el 2 de junio de 2021. Esto provocó que la fecha de finalización de este proyecto se viese retrasada hasta el 14 de julio de 2021.

2. Las redes LSTM espacio-temporales suponen un campo de investigación totalmente novedoso y reciente. Como consecuencia, apenas existe documentación sobre ellas y en el *sprint* 3 no se consiguió implementar con éxito una red de este tipo. Además, esto también implicaba la imposibilidad de construir con éxito una red LSTM espacio-temporal en un proyecto de tan corta duración como en el que nos encontrábamos sin alargar los plazos. Sin embargo, durante el estudio de las propuestas que planteaban este tipo de redes se abrió una nueva vía de investigación consistente en el uso combinado de redes convolucionales y redes LSTM (y que denominaremos redes ConvLSTM de ahora en adelante) ya que este tipo de redes también permitían explotar las dependencias temporales y espaciales de los datos [6, 33].

Tras la reunión de *Retroplanning* del final del *sprint* 3 en la que se observó que las redes ConvLSTM descubiertas también permitían alcanzar el objetivo OBJ-3.2, e impulsados por la falta de documentación de las redes LSTM espacio-temporales, el equipo UVagile decidió cambiar la organización de los siguientes *sprints* para centrar sus esfuerzos en buscar más información acerca de estas nuevas redes y probarlas en este estudio para analizar si permitían realizar mejores predicciones que las redes LSTM. Todo esto derivó en la eliminación de las tareas T3.5. y T4.2. para evitar seguir estancados con las redes LSTM espacio-temporales, y en la aparición de cinco nuevas tareas relacionadas con las redes ConvLSTM:

- **T2.6.** Estudio redes convolucionales (1 SP).
- **T3.5.** Implementar red ConvLSTM específica para nuestro problema (8 SP).
- **T4.2.** Estudio de los resultados obtenidos con redes ConvLSTM (1 SP).
- **T5.12.** Documentar redes convolucionales (2 SP).
- **T5.13.** Actualizar dominio del problema (1 SP).

Además, durante el transcurso del proyecto se logró disponer de más datos para realizar pruebas de los que se tenían al inicio de este, lo que dio lugar a otras dos nuevas tareas:

- **T4.4.** Prueba de los modelos obtenidos en los nuevos conjuntos de prueba (1 SP).
- **T5.14.** Documentar evaluación de resultados de los nuevos conjuntos de prueba (1 SP).

Con el fin de no posponer la fecha de finalización del proyecto, se decidió incorporar estas nuevas tareas a los dos sprints restantes. En la Tabla 5.6 se puede observar cómo se distribuyeron estas nuevas tareas y la modificación del calendario debida al parón realizado, y en la Figura 5.3 se muestra el diagrama de Gantt real tras la finalización del proyecto.

Sprint	Fecha inicio	Fecha fin	Tareas	SP
#1	03/03/2021	23/03/2021	T1.1. / T1.2. / T1.3. / T2.1. / T2.2. / T2.3. / T2.4 / T3.1./ T5.6.	12.5
#2	24/03/2021	20/04/2021	T3.2. / T3.3. / T4.1. / T5.1. / T5.2. / T5.3.	13
#3	21/04/2021	11/05/2021	T1.4. / T2.5. / T3.4. / T5.5.	14
#4	12/05/2021	01/06/2021	T2.6./ T3.5. / T4.2. / T4.3./ T5.12. / T5.13.	16
#5	02/06/2021	22/06/2021	T4.4. / T5.4. / T5.7. / T5.8. / T5.9. / T5.10. / T5.11. / T5.14.	13.5

Tabla 5.6: Distribución de las tareas por *sprint*.

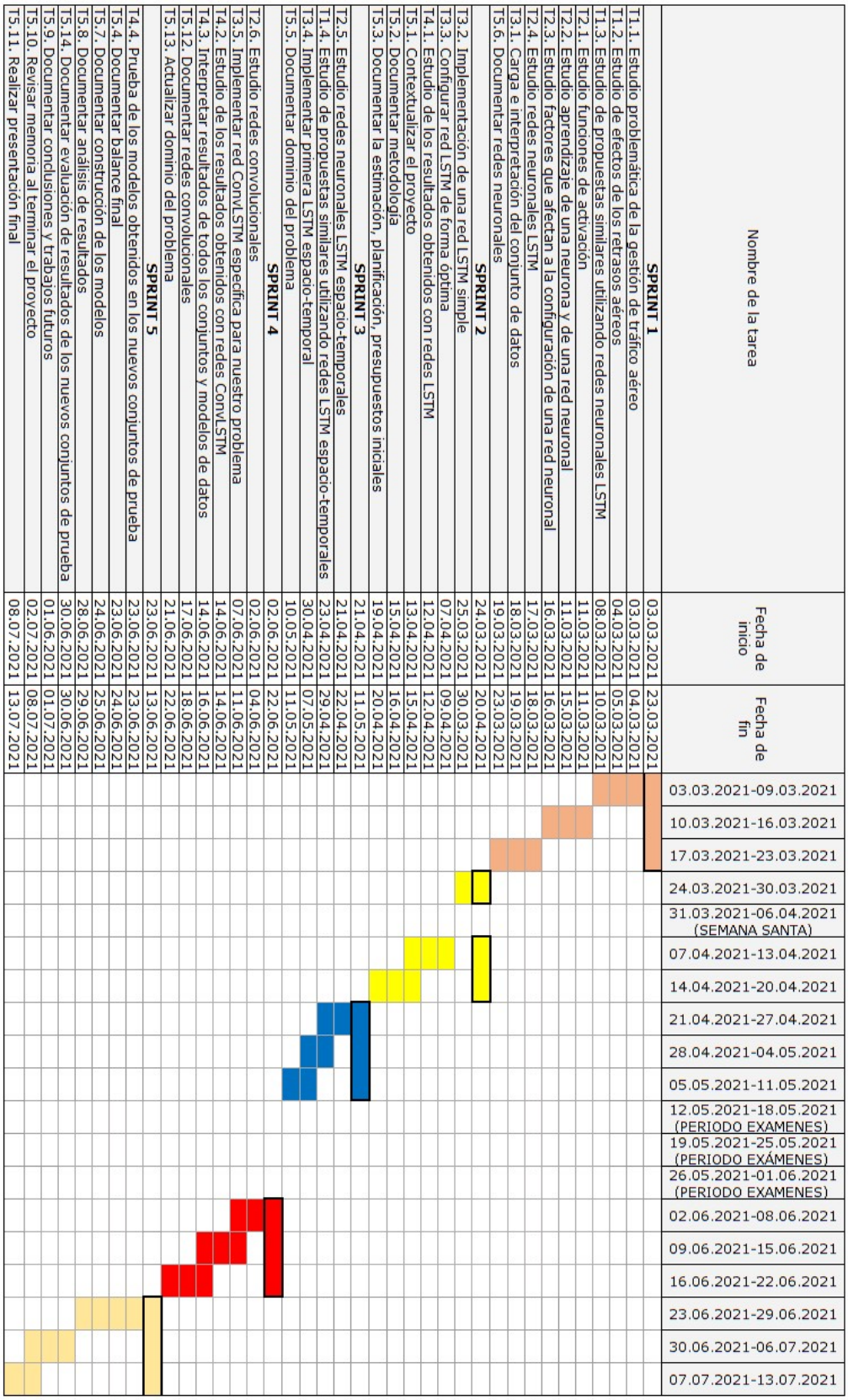


Figura 5.3: Planificación final del proyecto.

5.4.2. Balance de costes

Los costes iniciales de *hardware* y *software* no han sufrido modificaciones a pesar de las nuevas tareas surgidas. Sin embargo, el número de horas analizando datos y programando sí que se ha visto modificado y con ello los costes de personal.

En la Tabla 5.7 se puede observar los sueldos al finalizar el proyecto, y en la Tabla 5.8 cómo ha afectado estos cambios a los costes de personal (teniendo en cuenta que la Seguridad Social supone un 32,1 % del sueldo bruto de la persona).

Rol	Sueldo (€/hora)	Horas	Total (€)
Analista de datos junior	16,59	130	2156,7
Desarrollador Python junior	16,08	190	3055,2

Tabla 5.7: Sueldos según el rol contratado al terminar el proyecto.

Motivo	Coste real (€)
Sueldo analista de datos junior	2156,7
Sueldo desarrollador Python junior	3055,2
Seguridad Social	1673,02

Tabla 5.8: Costes asociados al personal al terminar el proyecto.

Finalmente, el coste total del proyecto es de **7070,92 €** frente a los 6639,31 € estimados al comienzo del proyecto, y se puede ver un desglose de este en la Tabla 5.9.

Tipo de coste	Total (€)	% del total	Incremento coste inicial (€)
Hardware	186	2,63	0
Software	0	0	0
Personal	6884,92	97,37	431,61
TOTAL	7070,92	100	431,61

Tabla 5.9: Desglose final del coste total.

Capítulo 6

Implementación de los modelos

En este capítulo se detallan los conjuntos de datos disponibles (Sección 6.1), y cómo se deben cargar y preparar para utilizarlos en el entrenamiento y prueba de los modelos que se construyan (Sección 6.2).

Asimismo, se explica el proceso seguido para construir los diferentes modelos de aprendizaje (Sección 6.3), y los análisis realizados para comparar su precisión (Sección 6.4).

6.1. Descripción de los conjuntos de datos

Para este proyecto disponemos de tres conjuntos de datos (o *datasets*) en formato CSV para entrenar y probar los modelos construidos:

- **unDiaFlightData.csv:** este archivo contiene 1.264.055 filas de datos correspondientes a 522 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados el viernes 2 de febrero de 2018.

- **tresDiasFlightData.csv:** este archivo contiene 1.194.007 filas de datos correspondientes a 1.171 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados entre el viernes 2 de febrero de 2018, y el domingo 4 de febrero de 2018 (incluidos).

- **sieteDiasFlightData.csv:** este archivo contiene 2.747.114 filas de datos correspondientes a 2.743 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados entre el viernes 2 de febrero de 2018, y el jueves 8 de febrero de 2018 (incluidos).

En el caso del archivo *unDiaFlightData.csv*, los datos proceden de la integración de tres fuentes: mensajes ADS-B en crudo (de Frambuesa y de ADSBHub), y vectores reconstruidos a partir de mensajes ADS-B (de OpenSky). Los datos de los otros dos archivos están todos extraídos de OpenSky y enriquecidos con información de planes de vuelo procedentes de Network Manager, propiedad de Eurocontrol [113].

Cada fila de datos en estos archivos incluye la siguiente información:

Nombre característica	Tipo	Descripción
timestamp	int	Marca temporal en la que se realizó la medición.
latitude	float	Latitud de la aeronave en el momento de la obtención del dato (en grados).
longitude	float	Longitud de la aeronave en el momento de la obtención del dato (en grados).
altitude	float	Altitud de la aeronave en el momento de la obtención del dato (en metros).
speed	float	Velocidad horizontal de la aeronave en el momento de la obtención del dato (en millas por hora).
vspeed	float	Velocidad vertical de la aeronave en el momento de la obtención del dato (en millas por hora).
ground	bool	Indica si la aeronave está en tierra o volando.
leg	string	Identificador único del vuelo generado artificialmente por la unión de las características callsign, hexident, leg_tbegin y leg_tend.
flight_operator	string	Nombre del operador al que pertenece la aeronave (no disponible en los *datasets tresDiasFlightData.csv* y *sieteDiasFlightData.csv*).
leg_tbegin	int	Marca temporal en la que comenzó el vuelo.
leg_tend	int	Marca temporal en la que finalizó el vuelo.
estimated_departure_time	int	Marca temporal con el momento estimado de salida del aeropuerto de origen de la aeronave.
estimated_arrival_time	int	Marca temporal con el momento estimado de llegada al aeropuerto de destino de la aeronave.
airport_origin	string	Aeropuerto de origen.
airport_destination	string	Aeropuerto de destino.
aircraft_model	string	Modelo de la aeronave.
Takeoff	int	Marca temporal con el momento de despegue de la aeronave.
Touchdown	int	Marca temporal con el momento de aterrizaje de la aeronave.
fpId	string	Identificador único del plan de vuelo seguido por la aeronave (no disponible en el *dataset unDiaFlightData.csv*).
hexident	string	Código único del transpondedor que envía las señales en la aeronave (no disponible en el *dataset unDiaFlightData.csv*).
callsign	string	Identificador del vuelo (no disponible en el *dataset unDiaFlightData.csv*).
flightDate	string	Fecha en la que se realizó el vuelo en formato año-mes-día (no disponible en el *dataset unDiaFlightData.csv*).
RTA	int	Tiempo restante (en segundos) hasta el aterrizaje en el momento de la obtención del dato. Calculado como Touchdown-timestamp.

Tabla 6.1: Información incluida en cada fila de datos.

El objetivo de los modelos que se van a construir es calcular a partir de una serie de características el tiempo restante hasta el aterrizaje, es decir, predecir el valor del RTA (Remaining Time of Arrival) en un momento dado. Para el entrenamiento (y prueba) de los modelos, como se utilizarán vuelos ya finalizados, se dispondrá del valor del RTA y se podrá utilizar para compararlo con el valor predicho por los modelos.

Sin embargo, en un vuelo en curso, el valor Touchdown, y por lo tanto del RTA, se desconocerán hasta que se produzca el aterrizaje. No obstante, al aplicar en el futuro estos modelos a un trayecto en curso, se obtendrá un valor de RTA estimado para la posición de la aeronave en el aire.

Además de todos estos datos, en los análisis que sean necesarios se añadirán las siguientes columnas:

Nombre característica	Tipo	Descripción
distanciaHaversine	float	Distancia de la aeronave hasta el aeropuerto en el momento de la obtención del dato (en kilómetros).
day_of_week	int	Día de la semana en la que se realizó el vuelo.

Tabla 6.2: Características calculadas a partir de la información inicial.

Los valores de la columna *distanciaHaversine* se calcularán haciendo uso de la función *haversine* de la librería *Haversine*, y teniendo en cuenta que las coordenadas del aeropuerto de Madrid-Barajas son longitud $-3,5694800°$, y latitud $40,4918100°$ [114].

Por su parte, los valores de la columna *day_of_week* se calcularán a partir de los registros de la columna *flightDate* y haciendo uso de las funciones *to_datetime* (convierte el argumento que se le pasa a un objeto de tipo *datetime*) y *dayofweek* (devuelve el día de la semana con el lunes igual a 0 y el domingo igual a 6) de la librería *Pandas* [99].

6.2. Carga y preparación de los conjuntos de datos

Antes de utilizar cualquiera de los conjuntos de datos disponibles, se deberá cargar y preparar de la siguiente manera.

En primer lugar, se deben cargar todas las librerías descritas en la Sección 4.4 que vayan a ser necesarias en el resto del código. Una vez realizado esto, se puede proceder a la carga del conjunto de datos que se vaya a utilizar con la función *read_csv* de la librería *Pandas*, y a la ordenación de las columnas de dicho *dataset* para lograr una rápida visualización de las columnas que nos interesen.

En función del conjunto de datos utilizado, se hace necesario ahora realizar las siguientes operaciones:

- En el caso del archivo *unDiaFlightData.csv*, tras su carga es necesario eliminar los registros cuyo identificador de vuelo es "SWT102_34210D_1517596837_1517597270" debido a que se trata de un vuelo que tiene solamente 10 registros y causa problemas al entrenar y/o probar los diferentes modelos.

- En los otros dos archivos, el campo 'altitude' contiene valores nulos, lo que ocasiona problemas a la hora de entrenar los modelos. Por ello, se debe ordenar los registros por vuelo y por *timestamp* para que la trayectoria esté ordenada, y después llenar los nulos en la altitud replicando el valor no nulo inmediatamente superior. Esto también es necesario replicarlo para el campo 'speed' si se utiliza dicha característica.

El siguiente paso consiste en eliminar aquellos registros en los que el campo 'ground' es verdadero, es decir, los mensajes enviados por las aeronaves cuando se encuentran en tierra (independientemente de si aún no han despegado o si acaban de aterrizar). El motivo de esto se debe a que las aeronaves en muchas ocasiones envían gran cantidad de mensajes de forma intermitente mientras están dando vueltas por las pistas del aeropuerto, pudiendo derivar en un peor entrenamiento de los modelos.

El siguiente problema surge con los vuelos transatlánticos ya que no existen receptores ADS-B sobre el océano Atlántico y su seguimiento solo se lleva a cabo por los informes de posición emitidos por los pilotos como se explicó en la Sección 2.2. Por esta razón, no se dispone de mensajes de vuelos que están sobrevolando el océano, y como se puede ver en la Figura 6.1, el RTA desciende bruscamente cuando se pasa del último mensaje recibido a un lado del océano al primer mensaje recibido al otro lado del océano. Este salto también es perceptible en muchas otras características que abordan nuestro estudio como pueden ser la latitud y la longitud.

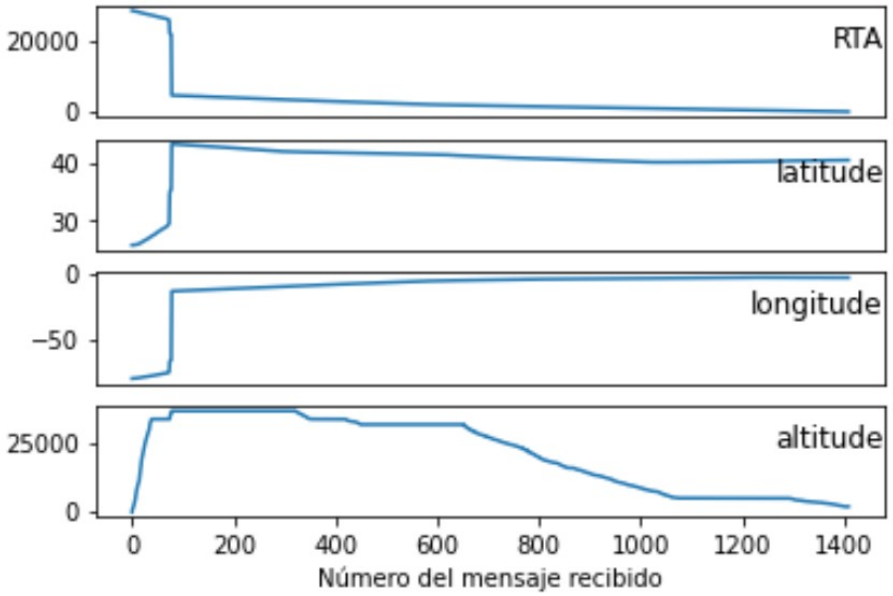


Figura 6.1: Perfil de los mensajes de un vuelo transatlántico.

Debido a la falta de estos mensajes, estos cambios tan bruscos provocan un peor entrenamiento, y en consecuencia, peores predicciones de los modelos. Para evitarlo, nos vamos a quedar con los mensajes recibidos por las aeronaves cuyo RTA sea inferior a 7200 segundos (es decir, menos de 2 horas).

Otro problema que surge al entrenar con redes neuronales, es que los valores utilizados por ellas, en general, necesitan ser normalizados o escalados. Esto se debe a que los algoritmos de aprendizaje automático tratan los números sin saber qué representan, por lo que una gran diferencia en el rango de valores puede derivar en que los números que alcancen mayores valores jueguen un papel más importante en el proceso de entrenamiento del modelo. Solo los modelos cuyos algoritmos están basados en árboles o grafos no requieren normalizar los datos [115].

Se hace por lo tanto imprescindible realizar este escalado en nuestro estudio para evitar estos problemas. Para ello, se creará una copia del *dataset*, y en ella se aplicará un proceso de normalización a todos las características que vayan a usar los modelos para que estas se encuentren en el rango [0,1]. Para cada característica a escalar se aplicará la fórmula:

$$\tilde{x}_t = \frac{x_t - x_{min}}{x_{max} - x_{min}}$$

donde x_{max} y x_{min} representan el valor máximo y mínimo de los datos respectivamente. Para realizar esto de forma sencilla en Python, se utilizará la función *MinMaxScaler* proporcionada por la librería *Sklearn*.

A continuación, se debe dividir el conjunto de datos en dos subconjuntos: el primero contendrá el 80 % de los datos y se utilizará para entrenar las redes neuronales; mientras que el segundo abarcará el 20 % de los datos restantes y se empleará para probar los modelos entrenados.

Para realizar esta división sobre nuestros *datasets*, en vez de llevarlo a cabo sobre el conjunto total de datos (mensajes individuales), se agruparán por vuelos todos los registros (para trabajar con trayectorias completas), y una vez hecho esto, se utilizará aproximadamente el 80 % de los vuelos como entrenamiento, y el 20 % como prueba. Para evitar posibles sesgos en la elección de los vuelos que forman cada conjunto, se deberá escoger estos vuelos de forma aleatoria, por lo que se hará uso de la función *random* de la librería *NumPy*. Además, para poder replicar los resultados obtenidos en un futuro, se utilizará una semilla a partir de la cual se realizará esta división aleatoria.

La división escogida de vuelos de entrenamiento y vuelos de prueba de los diferentes conjuntos de datos de partida es la siguiente:

- **unDiaFlightData.csv:** de los 521 vuelos válidos en este archivo, 400 se utilizarán para entrenamiento y 121 para pruebas.
- **tresDiasFlightData.csv:** de los 1171 vuelos válidos en este archivo, 850 se utilizarán para entrenamiento y 321 para pruebas.
- **sieteDiasFlightData.csv:** de los 2743 vuelos válidos en este archivo, 2200 se utilizarán para entrenamiento y 543 para pruebas.

El siguiente paso consiste en crear una función para formatear los datos y ajustarlos a la entrada requerida por la red neuronal. La función *create_dataset* que se puede ver en la Figura 6.2 se encarga de realizar esta operación ya que devuelve dos conjuntos de datos en forma matricial:

- dataX: valores de las características utilizadas para predecir el RTA.
- dataY: valores del RTA a predecir.

```
#Formatear los datos
def create_dataset(df, lookback=1):
    dataX, dataY = [], []
    for i in range(len(df)-lookback-1):
        a = df[i:(i+lookback), 0:numerocaracteristicas]
        dataX.append(a)
        dataY.append(df[i + lookback, numerocaracteristicas])
    return np.array(dataX), np.array(dataY)
```

Figura 6.2: Función create_dataset.

La variable *numerocaracteristicas* indica el número de características que se van a utilizar para predecir el RTA.

Por su parte, la variable *lookback*, que por defecto se establece a uno, indica el número de registros hacia atrás que se necesitan para definir uno hacia delante. Este valor debe ser lo suficientemente grande para que si hay un registro con datos erróneos, no afecte en gran medida a la predicción del siguiente valor; pero lo suficientemente pequeño para que los registros muy antiguos no tengan un gran peso en la predicción del siguiente valor. Por este motivo, y teniendo en cuenta el estudio realizado en [11], en todos los modelos que se construyan se utilizará un valor de *lookback* igual a 50.

Además, en cada iteración de esta función, el mensaje más antiguo de los que se deberían tener en cuenta para predecir el siguiente registro (según el tamaño del *lookback* establecido) se deja de tener en cuenta y se añade el más reciente. Este proceso de ventana deslizante se puede observar en la Figura 6.3.

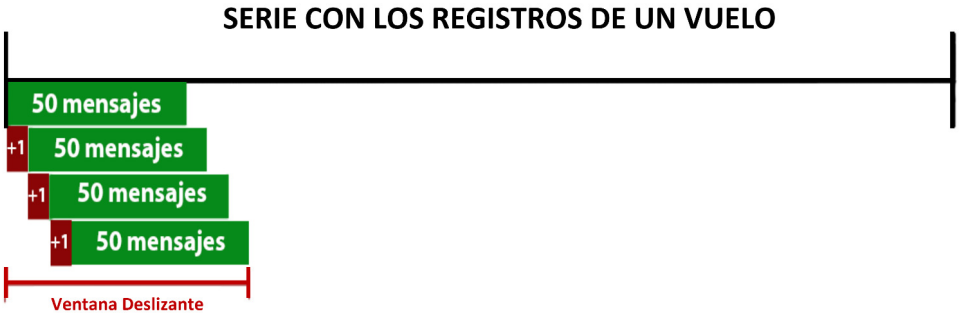


Figura 6.3: Ventana deslizante.

Finalmente, para cada vuelo de entrenamiento se escogerán las características que se necesiten para el estudio, se formatearán los datos con la función *create_dataset*, y los conjuntos obtenidos se almacenarán junto con los datos formateados del resto de los vuelos de entrenamiento. Esta unión de todos los datos formateados de todos los vuelos de entrenamiento se debe realizar porque si se usa cada vuelo de forma independiente para entrenar los modelos, los últimos vuelos que se empleen como entrada a los modelos tendrán una mayor influencia en el entrenamiento que los primeros vuelos utilizados.

6.3. Construcción de los modelos

Con la librería *Keras* se puede construye un modelo de red neuronal mediante una secuencia de capas que se añaden con la función *add* y que se erigen sobre un objeto de la clase *Sequential*. Para la realización de este proyecto se han construido tres modelos: un modelo LSTM, un modelo ConvLSTM que efectúa una convolución sencilla y un modelo ConvLSTM que realiza una convolución más compleja. La arquitectura de cada uno de ellos se detalla en las Subsecciones 6.3.1, 6.3.2 y 6.3.3 respectivamente.

Una vez construidos, cada modelo se debe compilar haciendo uso de la función *compile* del objeto *Sequential*, y para ello se debe especificar el estimador de error y el optimizador que se empleará. Tras realizar varias pruebas con diferentes configuraciones, finalmente se decidió que todos los modelos utilizarán como estimador de error el error cuadrático medio (MSE) y el optimizador Adam.

Además, tras la compilación del modelo se puede ejecutar la función *summary* del objeto *Sequential* sobre él con el fin de obtener un resumen de las capas que lo conforman.

Finalmente, se debe realizar el entrenamiento de los modelos. Para esta tarea *Keras* hace uso de la función *fit* a la que se le debe pasar por argumento el conjunto de valores utilizados para hacer la predicción, el conjunto de valores a predecir, la fracción del conjunto de entrenamiento utilizado para evaluar las pérdidas y métricas definidas al compilar el modelo, el número de épocas de entrenamiento (en inglés *epoch*) y el tamaño de los lotes (en inglés *batch*). Además, también se suele incluir un argumento denominado *verbose* para modificar la barra de progreso del entrenamiento que se visualiza durante este.

El número de épocas es el número de iteraciones sobre la totalidad de los datos que el modelo va a realizar para su entrenamiento, mientras que el tamaño de los lotes es la cantidad de registros que procesa el modelo antes de actualizar los pesos siguiendo el algoritmo de actualización de parámetros previamente establecido. Aunque en [11] se estableció el numero de épocas en 25 y el tamaño de los lotes en 1000, tras realizar varios experimentos, y teniendo en cuenta el tamaño de todos los conjuntos de datos iniciales, se fijó en 30 el número de épocas y en 1500 el tamaño de los lotes. Dichos experimentos se encuentran detallados en el Apéndice B.

6.3.1. Modelo LSTM

En la Subsección 3.4.1 se explicó que las redes LSTM se utilizan principalmente para la clasificación de series temporales. Esto junto con los buenos resultados obtenidos en trabajos previos como [5] o [11] (explicados en la Sección 2.4) que utilizan este tipo de redes, hacen que el primer modelo que se vaya a construir y evaluar sea una red LSTM, en concreto, una red LSTM *Vanilla*. Es decir, este modelo estará formado por una capa LSTM y una capa densa, y para construirlas se utilizarán las clases *LSTM* [116] y *Dense* [117] de la librería *Keras*.

La clase *LSTM* define como función de activación de las capas LSTM la función tangente hiperbólica. Por esta razón, bastará con indicar el número de neuronas que compondrán dicha capa, que estableceremos en 10 para poder comparar nuestros resultados con los obtenidos por [11], y el tamaño de la entrada (argumento *input_shape*) por ser la primera capa de este modelo, que en nuestro estudio serán vectores de tamaño $lookback \times numerocaracteristicas$. Por su parte, el argumento de la capa densa indica el tamaño de la salida del modelo, por lo que en este caso será igual a 1 ya que solo se quiere predecir un único valor para cada secuencia de mensajes.

Nota: este modelo se denominará **soloLSTM** para podernos referir a él durante el resto del documento.

6.3.2. Modelo convolucional sencillo

Este modelo busca aprovechar las ventajas que supone el uso combinado de las redes convolucionales y las redes LSTM como pusieron de manifiesto los artículos [6] y [33] (explicados en la Sección 2.4). Como se puede ver en la Figura 6.4, que se corresponde con el modelo propuesto por [33], las dependencias espaciales pueden ser capturadas por las redes convolucionales, y las temporales pueden ser aprendidas por las redes LSTM.

Para ello, se va a adoptar una arquitectura similar a la propuesta en [33], y por esta razón, este modelo estará formado por una capa convolucional que utilizará una convolución de una dimensión, una capa LSTM, una capa aplanadora y una capa densa, construidas a partir de las clases *Conv1D* [118], *LSTM* [116], *Flatten* [119] y *Dense* [117] de la librería *Keras* respectivamente.

La capa *Conv1D* necesita como argumentos el número de filtros que se deben emplear (argumento *filters*), la longitud de la ventana de la convolución (argumento *kernel_size*), la función de activación a utilizar (argumento *activation*) y el tamaño de la entrada (argumento *input_shape*) por ser la primera capa de este modelo, que en nuestro estudio serán vectores de tamaño $lookback \times numerocaracteristicas$. Tras realizar varias pruebas, se llegó a la conclusión que se alcanzaban los mejores resultados utilizando 64 filtros, un tamaño de ventana igual a 4, y la función de activación ReLU. Dichos experimentos se encuentran detallados en el Apéndice B.

A continuación de esta capa se encontrará la capa LSTM que estará compuesta por 10 neuronas al igual que en el modelo soloLSTM. Después se hallará la capa aplanadora que no necesita ningún argumento y que se utilizará porque se va a obtener una salida

multidimensional y se desea que sea lineal para pasarla a la capa densa. Y finalmente, se ubicará la capa densa, que al igual que en el modelo soloLSTM, su argumento va a ser igual a uno ya que solo se quiere predecir un único valor para cada secuencia de mensajes.

Nota: este modelo se denominará **ConvSencillo** para podernos referir a él durante el resto del documento.

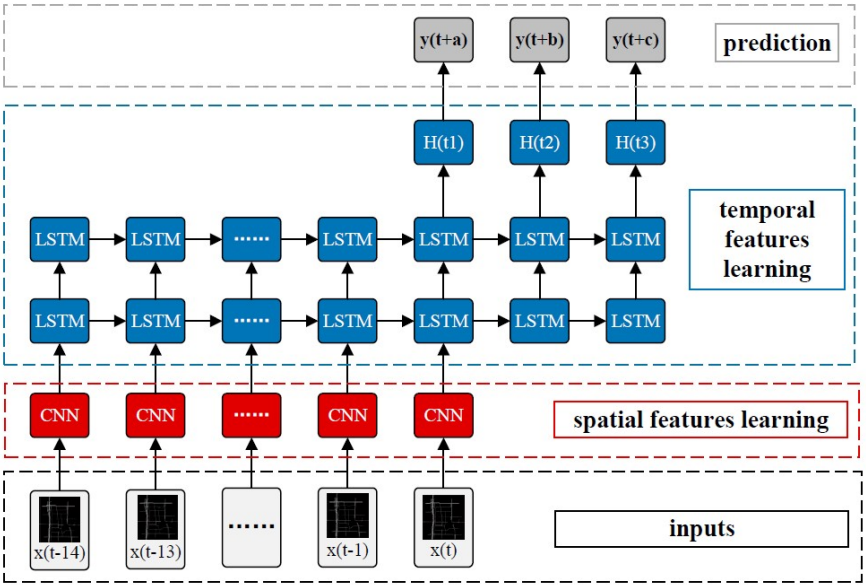


Figura 6.4: Modelo red convolucional recurrente espacio-temporal. Fuente: [33].

6.3.3. Modelo convolucional complejo

Este último modelo se basa en la misma idea que el modelo ConvSencillo, pero en esta ocasión se van a añadir tres capas convolucionales *Conv1D* antes de la capa LSTM para estudiar si se produce una mejora en las predicciones como consecuencia de añadir una mayor complejidad a la arquitectura del modelo. El resto del modelo permanece idéntico al explicado en la Subsección 6.3.2.

En esta ocasión, se decidió que cada una de las tres capas convolucionales tuviera un número de filtros diferentes para evaluar su desempeño, por lo que se fijó una de 32 filtros, otra de 64 y finalmente otra de 128. Respecto a la longitud de la ventana de la convolución en cada una de las capas, se estableció en 4 debido a que con este valor el modelo ConvSencillo realizaba las mejores predicciones independientemente del número de filtros de su capa (como se pone de manifiesto en los experimentos detallados en el Apéndice B).

Los argumentos de la primera capa *Conv1D* serán *filters*=32, *kernel_size*=4, *activation*=ReLU, y el tamaño de la entrada (argumento *input_shape*) serán vectores de tamaño *lookback* × *numerocaracteristicas*; los argumentos de la segunda capa *Conv1D* serán *filters*=64, *kernel_size*=4 y *activation*=ReLU; y los argumentos de la tercera capa *Conv1D* serán *filters*=128, *kernel_size*=4 y *activation*=ReLU.

Nota: este modelo se denominará **ConvComplejo** para podernos referir a él durante el resto del documento.

Finalmente, para una mejor comparación de los tres modelos construidos, se recoge en la Tabla 6.3 las capas y argumentos de todos ellos.

Capa	soloLSTM	ConvSencillo	ConvComplejo
Primera convolucional	—	-Número filtros=64 -Longitud de la ventana convolución=4 -Activación=ReLU -Tamaño entrada= lookback × numerocaracteristicas	-Número filtros=32 -Longitud de la ventana convolución=4 -Activación=ReLU -Tamaño entrada= lookback × numerocaracteristicas
Segunda convolucional	—	—	-Número filtros=64 -Longitud de la ventana convolución=4 -Activación=ReLU
Tercera convolucional	—	—	-Número filtros=128 -Longitud de la ventana convolución=4 -Activación=ReLU
LSTM	-Número neuronas=10 -Tamaño entrada= lookback × numerocaracteristicas	-Número neuronas=10	-Número neuronas=10
Aplanadora	—	-Sí, sin argumentos	-Sí, sin argumentos
Densa	-Tamaño salida=1	-Tamaño salida=1	-Tamaño salida=1

Tabla 6.3: Resumen de las capas y argumentos de los modelos construidos.

6.4. Análisis de los modelos construidos

Para los diferentes análisis que se muestran en esta sección, se ha escogido un conjunto de datos, y después se han entrenado los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** con los vuelos de entrenamiento de dicho conjunto. A continuación, estos tres modelos han realizado predicciones para los vuelos de pruebas de dicho *dataset* (que se habían separado antes de realizar el entrenamiento, como se explicó en la Sección 6.2, y que por lo tanto nunca han sido vistos por los modelos). Finalmente, al tratarse de vuelos ya completados, se puede evaluar la precisión de las predicciones realizadas por los modelos con el RTA real en cada instante.

El proceso detallado es el siguiente: en primer lugar, para cada vuelo de prueba se separan las características que se hayan escogido para el estudio y se formatean con la función *create_dataset* que se había creado previamente. Con esto se logra separar por un

lado el conjunto de valores utilizados para hacer la predicción, y por otro lado, el conjunto de valores a predecir. El primer conjunto devuelto por esta función será utilizado por los modelos ya entrenados para generar una predicción de salida haciendo uso de la función *predict*. El segundo conjunto, al tratarse de los valores a predecir, permitirá evaluar cómo de precisa es la predicción realizada por los modelos. Sin embargo, para realizar esta evaluación, primero se deben desescalar todos los valores obtenidos para devolverlos a su magnitud original. Esto se realizará con la función *inverse_transform* proporcionada por la librería *Sklearn*. A continuación, se deben unificar las predicciones realizadas con el valor de RTA esperado para poder compararlos, y finalmente, solo en caso de que el vuelo tenga una duración superior a media hora, se evaluará la precisión de las predicciones (haciendo uso del error absoluto medio o MAE ya que indicará los segundos de diferencia existentes entre la predicción y el valor esperado). Para ello, se valorarán tres aspectos:

1. MAE TOTAL: representará el error absoluto medio entre todas las predicciones del RTA realizadas y los RTA esperados. Este valor nos permitirá saber si durante todo el transcurso del vuelo se ha obtenido un RTA cercano al esperado o no.

2. MAE 30 MIN: representará el error absoluto medio calculado a falta de 30 minutos para la llegada al aeropuerto de destino. Para realizar este cálculo, se utilizarán los últimos 10 valores predichos y los últimos 10 valores esperados antes de que el RTA esperado fuera inferior a 1800 segundos (30 minutos).

3. MAE 15 MIN: representará el error absoluto medio calculado a falta de 15 minutos para la llegada al aeropuerto de destino. Para realizar este cálculo, se utilizarán los últimos 10 valores predichos y los últimos 10 valores esperados antes de que el RTA esperado fuera inferior a 900 segundos (15 minutos).

Por lo general, aunque los modelos deberían poder generalizar la mayoría de vuelos, estos aspectos pueden variar mucho de uno a otro, ya que habrá vuelos para los que los modelos realicen muy buenas predicciones (por ejemplo, porque hayan sido entrenados con un vuelo muy similar), mientras que habrá otros en los que no será así (por ejemplo, porque un vuelo cuente con muy pocos registros o esté siguiendo una ruta muy poco frecuentada y de la que los modelos no tengan ninguna referencia). Por este motivo, para cada vuelo se calculará su MAE TOTAL, su MAE 30 MIN, y su MAE 15 MIN, y estos valores se sumarán al MAE TOTAL, al MAE 30 MIN y al MAE 15 MIN del total de los vuelos. Finalmente, estos tres últimos valores se dividirán entre el número de vuelos cuya duración sea superior a media hora, y estos serán los valores que se muestren al finalizar la ejecución de los programas y que se analizarán en las siguientes subsecciones.

Nota: este análisis también se puede realizar sobre el conjunto de vuelos de entrenamiento a pesar de que sus registros ya hayan sido vistos previamente por los modelos. Sin embargo, esto no es muy frecuente ya que como han sido entrenado con ellos, puede existir un sesgo (y se obtendrían valores que es poco probable que se generalizasen a los nuevos datos), por lo que solo se realiza si se quiere comprobar si existe un sobreentrenamiento u *overfitting* en los modelos (es decir, para visualizar si estos aprenden detalles excesivos y

demasiado ruido en los datos de entrenamiento, lo que afecta negativamente al rendimiento de los modelos sobre nuevos datos) [120]. Por esta razón, en las siguientes subsecciones solo se analizarán los resultados obtenidos en los conjuntos de vuelos de prueba sobre los diferentes modelos vistos en la Sección 6.3.

6.4.1. Análisis con datos de 1 día

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **unDiaFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*) y *altitud* (*altitude*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). La precisión de estas predicciones se recoge en la Tabla 6.4 y en la Figura 6.5.

Nota: las columnas 'Estudio previo' de la Tabla 6.4 y de la Figura 6.5 muestran los resultados que obtuvo el modelo LSTM de Petar Georgiev en [11]. Esta comparación es equiparable dado que el conjunto de vuelos que utilizó en dicho estudio es el mismo que el utilizado en este. Sin embargo, no calculó el MAE de todo el vuelo de ahí que no se incluya este valor. Además, esta comparación solo podrá realizarse para este análisis ya que en [11] solo se evaluaron los resultados obtenidos por este *dataset*.

	Estudio previo	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	-	232,290	253,745	233,686
MAE 30 MIN	271	216,013	230,155	233,315
MAE 15 MIN	146	140,615	136,918	174,410

Tabla 6.4: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de un día utilizando las características latitud, longitud y altitud.

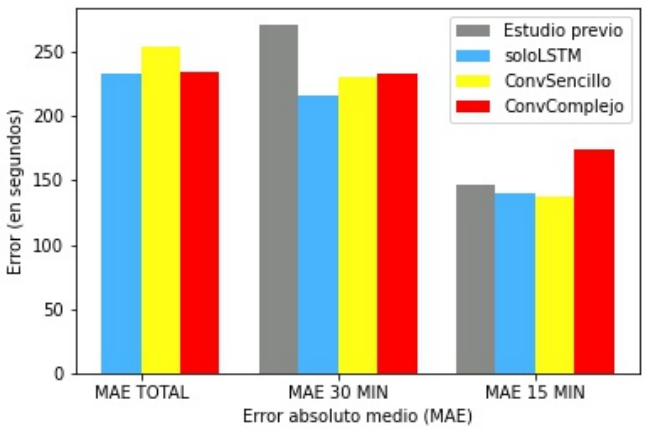


Figura 6.5: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de un día utilizando las características latitud, longitud y altitud.

Los tres modelos propuestos en este documento realizan mejores predicciones a 30 minutos para la llegada que la obtenida por el modelo LSTM propuesto en [11], llegando a reducirla en casi 60 segundos en el caso del modelo soloLSTM. No obstante, esta mejora no es tan significativa a 15 minutos para la llegada (de hecho, el modelo ConvComplejo empeora las predicciones realizadas por el modelo LSTM propuesto en [11]).

No obstante, tras este estudio no podemos arrojar conclusiones definitivas sobre cuál de los tres modelos propuestos (soloLSTM, ConvSencillo y ConvComplejo) es el mejor, ni tampoco si resulta conveniente utilizar estos modelos entrenados sobre otros conjuntos de prueba más grandes, ya que la limitación del volumen de datos del *dataset* utilizado no permite explotar todo el potencial de las redes LSTM por la posible falta de estacionalidad y/o patrones de los datos. Por este motivo, repetiremos el proceso con datos de múltiples días para favorecer la aparición de patrones.

6.4.2. Análisis con datos de 3 días

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **tresDiasFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*) y *altitud* (*altitude*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). La precisión de estas predicciones se recoge en la Tabla 6.5 y en la Figura 6.6.

	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	163,163	155,853	155,772
MAE 30 MIN	185,581	177,257	176,013
MAE 15 MIN	117,507	105,600	96,508

Tabla 6.5: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 3 días utilizando las características latitud, longitud y altitud.

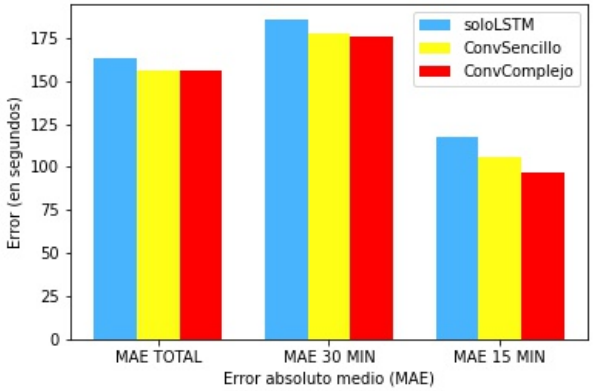


Figura 6.6: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 3 días utilizando las características latitud, longitud y altitud.

Como consecuencia de una mayor variedad en el conjunto de vuelos se puede observar una mejora significativa de las predicciones de todos los modelos con respecto a las obtenidas en el estudio anterior (entre todos los modelos se reduce en casi 47 segundos de media el MAE a 30 minutos para la llegada y en unos 44 segundos el MAE a 15 minutos). Además, aunque los predicciones realizadas por el modelo soloLSTM son bastantes buenas, se aprecia que el modelo ConvSencillo mejora aún más estas predicciones (reduciendo en unos 10 segundos todas las predicciones realizadas). Por su parte, el modelo ConvComplejo, a pesar de realizar una predicción similar al modelo ConvSencillo a 30 minutos para la llegada, presenta una mayor precisión a 15 minutos (para la llegada), donde el error cometido es de tan solo 96,5 segundos (9 segundos menos que el modelo ConvSencillo).

Como se puede observar en la Figura 6.7, los viernes es el día de la semana que generalmente operan más vuelos, y los sábados y domingos los días que menos. Además, en estos tres días es habitual la existencia de vuelos totalmente diferentes al resto de la semana. Por todo esto, dado que el dataset *tresDiasFlightData.csv* está formado por vuelos del viernes al domingo, puede existir un sesgo en los modelos entrenados, por lo que no resulta conveniente utilizar estos modelos entrenados sobre otros conjuntos de prueba más grandes y que abarquen otros días de la semana.

Travel Period	Mon	Tue	Wed	Thu	Fri	Sat	Sun
Jan 2018	98,584	94,161	96,060	98,063	100,757	86,532	94,545
Feb 2018	99,581	96,030	97,316	98,845	101,954	89,232	95,502
Mar 2018	101,959	98,197	99,571	101,227	103,747	90,498	98,025
Apr 2018	105,530	101,803	103,295	104,997	107,015	94,005	101,527
May 2018	106,434	102,649	104,056	105,012	107,364	94,834	101,488
Jun 2018	109,296	106,990	108,338	109,833	111,881	100,518	106,118
Jul 2018	112,835	110,243	111,251	112,326	113,754	103,766	109,271
Aug 2018	112,182	109,397	110,535	111,742	113,233	102,274	108,605
Sep 2018	109,972	106,450	107,955	109,756	111,383	98,306	105,591
Oct 2018	108,432	104,578	105,714	107,703	109,714	96,295	103,784
Nov 2018	105,039	100,692	102,423	104,234	106,444	92,349	99,746
Dec 2018	104,517	100,081	102,073	104,118	106,830	92,503	100,348

Figura 6.7: Número de vuelos por día de la semana en el año 2018. Fuente: [121].

6.4.3. Análisis con datos de 7 días

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **sieteDiasFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*) y *altitud* (*altitude*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). La precisión de estas predicciones se recoge en la Tabla 6.6 y en la Figura 6.8.

	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	202,169	184,111	181,960
MAE 30 MIN	199,017	185,186	187,277
MAE 15 MIN	148,650	127,725	110,018

Tabla 6.6: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud y altitud.

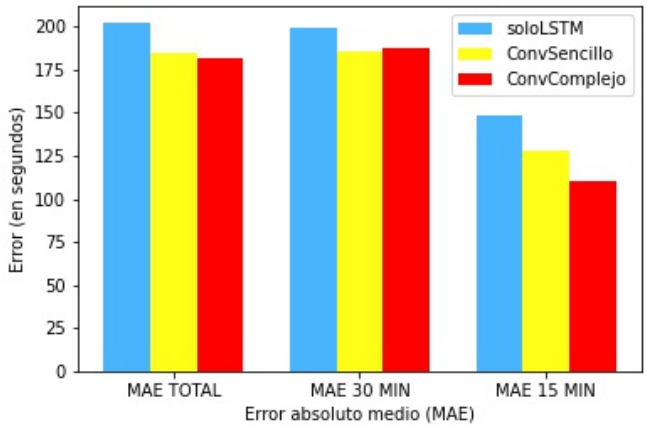


Figura 6.8: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud y altitud.

En este caso se observa que aunque los tres modelos realizan buenas predicciones, el modelo ConvComplejo es el mejor de todos ellos ya que reduce en casi 35 segundos la predicción a 15 minutos para la llegada con respecto a la realizada por el modelo soloLSTM.

Las predicciones realizadas por estos modelos resultan más realistas que las obtenidas en los estudios previos ya que se utilizan datos de una semana entera, por lo que se puede decir que hemos encontrado un modelo que predice mejor que un modelo compuesto únicamente por neuronas LSTM. No obstante, se sigue sin poder explotar todo el potencial de las redes LSTM por la limitación del volumen de datos, por lo que sería necesario realizar más pruebas sobre conjuntos de datos más grandes para corroborar esta afirmación.

6.4.4. Análisis de los modelos LSTM

En este estudio se van a comparar las predicciones obtenidas hasta el momento por el modelo **soloLSTM**. Las columnas *UnDia*, *TresDias* y *SieteDias* de la Tabla 6.7 y la Figura 6.9 recogen las predicciones obtenidas por el modelo soloLSTM en las Subsecciones 6.4.1, 6.4.2 y 6.4.3 respectivamente.

Además, en este estudio se quiere determinar si la configuración del modelo soloLSTM base y los parámetros escogidos para su entrenamiento (que resultaban óptimos para el conjunto de vuelos del dataset *unDiaFlightData.csv*), siguen siendo los mejores cuando

se utilizan conjuntos de datos más grande (en concreto, se analizará para el conjunto de vuelos del archivo *sieteDiasFlightData.csv*).

Debido a que existe un mayor número de vuelos, y por tanto, una mayor variabilidad en este conjunto de datos, se ha decidido incrementar el número de neuronas y de *batch* con respecto a la configuración base del modelo soloLSTM para ver si se producen mejores significativas. Tras realizar varios ensayos, se ha escogido un nuevo modelo soloLSTM con una nueva arquitectura, en concreto, formado por una capa LSTM que cuenta con 50 neuronas LSTM, y una capa densa; y además, en su entrenamiento se ha establecido el tamaño de *batch* en 3000.

Este nuevo modelo soloLSTM se ha entrenado utilizando el archivo **sieteDiasFlightData.csv**, y después se ha realizado las predicciones del RTA para el conjunto de vuelos de pruebas (de dicho *dataset*) utilizando las características *latitud* (*latitude*), *longitud* (*longitude*) y *altitud* (*altitude*). Las predicciones realizadas por él se recogen en las columnas *SieteDiasModificado* de la Tabla 6.7 y la Figura 6.9.

	UnDia	TresDias	SieteDias	SieteDiasModificado
MAE TOTAL	232,290	163,163	202,169	189,787
MAE 30 MIN	216,013	185,581	199,017	194,614
MAE 15 MIN	140,615	117,507	148,650	113,459

Tabla 6.7: MAE de las predicciones realizadas por los modelos soloLSTM utilizando las características latitud, longitud y altitud.

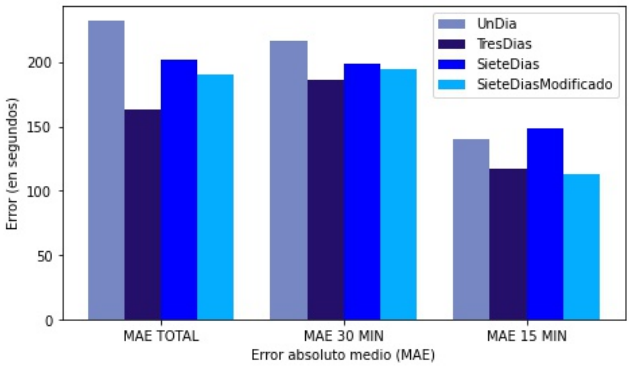


Figura 6.9: MAE de las predicciones realizadas por los modelos soloLSTM utilizando las características latitud, longitud y altitud.

Dado que se van a estudiar las predicciones obtenidas por el modelo soloLSTM en las subsecciones anteriores, en cada columna, el modelo ha sido entrenado con los vuelos de entrenamiento de un *dataset*, y ha realizado las predicciones sobre los vuelos de prueba del mismo *dataset*. Por ello, hay que tener cuidado al sacar conclusiones de la Tabla 6.7 y la Figura 6.9, ya que son modelos entrenados con distintos conjuntos de datos, por lo que no son directamente comparables.

Teniendo esto en cuenta, se observa que las predicciones realizadas sobre el conjunto de vuelos de tres días (columna *TresDias*) son mejores que sobre el de un día (columna *UnDia*) ya que se reducen en unos 30 segundos todas las medidas. Esto se puede deber a que el archivo de tres días cuenta con una mayor variedad de vuelos que el de un día.

Por esta misma razón, se esperaba que las predicciones realizadas sobre el conjunto de vuelos de siete días (columna *SieteDias*) fueran mejores que sobre el de tres días. Sin embargo, se obtuvieron resultados bastante peores, sobre todo en las predicciones a 15 minutos de la llegada, llegando incluso a empeorar las predicciones sobre el conjunto de vuelos de un día. Esto se puede deber a dos motivos:

1. Por el *dataset* utilizado en cada caso, ya que cuando el modelo utiliza el archivo *tresDiasFlightData.csv*, al solo contener vuelos que han operado de un viernes a un domingo y que generalmente son similares unos a otros, se puede estar beneficiando de estas repeticiones logrando muy buenos resultados. Esto no ocurre cuando utiliza el archivo *sieteDiasFlightData.csv*, ya que este *dataset* incluye vuelos de toda una semana donde existe una mayor variedad en las trayectorias y en el número de vuelos.

2. A que con el *dataset sieteDiasFlightData.csv* el modelo soloLSTM no esté siendo capaz de aprender la gran variedad de datos que existen a lo largo de una semana, quizá por una pobre configuración del modelo, o por realizar un entrenamiento inadecuado.

Las predicciones realizadas por el nuevo modelo soloLSTM (columna *SieteDiasModificado*) corroboran que la configuración del modelo soloLSTM base, o los parámetros escogidos para su entrenamiento, no son los mejores para el conjunto de vuelos del archivo *sieteDiasFlightData.csv*, ya que se podrían alcanzar mejores resultados con otras configuraciones. A pesar de ello, no se realizarán estas modificaciones para lo que resta del proyecto para mantener la configuración de partida y poder comparar adecuadamente las predicciones del resto de experimentos con las obtenidas en la Sección 6.4.3.

No obstante, el primero de los motivos planteados no debería ser descartado, ya que no se posee suficiente información y no se han realizado suficientes pruebas como para corroborarlo o desecharlo.

6.4.5. Análisis incluyendo la distancia al aeropuerto

Cuando una aeronave se aproxima a un aeropuerto queriendo aterrizar debe ajustar su rumbo para poder realizarlo en la pista que tenga asignada. Esto implica que en ocasiones las aeronaves tengan que dar pequeños rodeos como el que se puede observar en la Figura 6.10. Además, esto provoca que dos vuelos que se encuentren a la misma distancia de un aeropuerto no tienen porqué tardar el mismo tiempo en aterrizar.

Figura 6.10: Aterrizaje por el sur en el aeropuerto de Menorca. Fuente: [122].

Las aeronaves que van a aterrizar en el aeropuerto Adolfo Suárez Madrid-Barajas también deben corregir su rumbo en ocasiones para poder aterrizar en la pista que tienen asignada, por lo que el objetivo de este estudio será determinar si el uso de la distancia de la aeronave al aeropuerto para predecir el RTA permite realizar mejores predicciones que las obtenidas en la Subsección 6.4.3.

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **sieteDiasFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*), *altitud* (*altitude*) y *distancia al aeropuerto* (*distanciaHaversine*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). La precisión de estas predicciones se recoge en la Tabla 6.8 y en la Figura 6.11.

	soloLSTM	**ConvSencillo**	**ConvComplejo**
MAE TOTAL	187,820	180,325	181,755
MAE 30 MIN	179,016	180,938	193,074
MAE 15 MIN	137,105	120,359	105,005

Tabla 6.8: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud y distancia al aeropuerto.

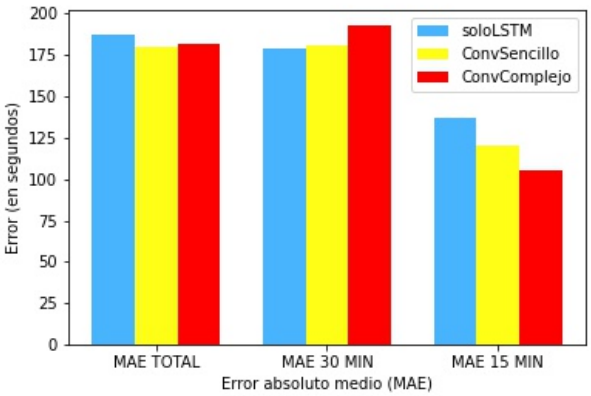


Figura 6.11: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud y distancia al aeropuerto.

Las predicciones realizadas confirman que utilizar también la distancia al aeropuerto resulta beneficioso para predecir el RTA, ya que el MAE a 30 minutos para la llegada se reduce de media 9 segundos, y el MAE a 15 minutos disminuye de media 8 segundos. Asimismo, en esta ocasión el modelo ConvComplejo es el que vuelve a realizar las mejores predicciones a 15 minutos de la llegada, llegando a reducir en 22 segundos la predicción realizada por el modelo soloLSTM.

6.4.6. Análisis incluyendo la distancia al aeropuerto y velocidad horizontal

Generalmente, en un vuelo la velocidad (horizontal) de la aeronave solo sufre variaciones en el momento del despegue o cuando está próximo a aterrizar, mientras que mantiene una velocidad más o menos constante durante la mayor parte de su trayecto (Figura 6.12).

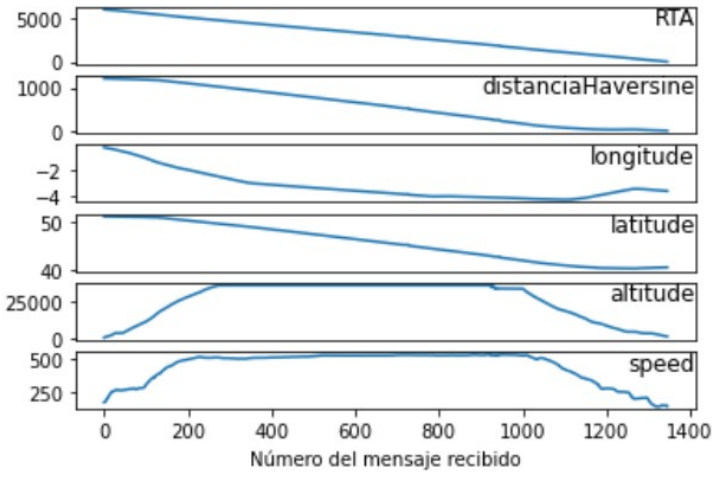


Figura 6.12: Perfil completo de los mensajes de un vuelo.

Aunque una aeronave no suele comenzar las maniobras de descenso hasta que está próxima al aeropuerto de destino, la necesidad de aproximarse de forma adecuada al aeropuerto puede provocar posibles virajes en su trayectoria que conllevan cambios en su velocidad. Por este motivo, el objetivo de este estudio será determinar si el uso de la velocidad horizontal para predecir el RTA permite realizar mejores predicciones que las obtenidas en la Subsección 6.4.5.

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **sieteDiasFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*), *altitud* (*altitude*), *distancia* (*distanciaHaversine*) y *velocidad horizontal* (*speed*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). La precisión de estas predicciones se recoge en la Tabla 6.9 y en la Figura 6.13.

	soloLSTM	**ConvSencillo**	**ConvComplejo**
MAE TOTAL	181,299	170,875	179,324
MAE 30 MIN	176,292	165,216	187,488
MAE 15 MIN	117,329	114,356	90,688

Tabla 6.9: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud, distancia al aeropuerto y velocidad horizontal.

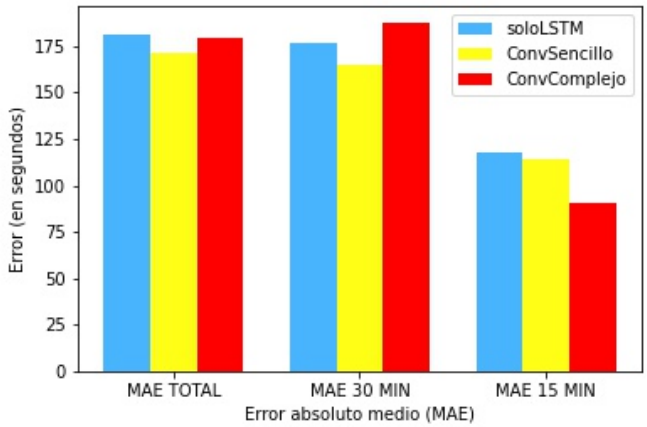


Figura 6.13: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud, distancia al aeropuerto y velocidad horizontal.

Las predicciones realizadas presentan mejoras significativas, sobre todo en el caso del modelo soloLSTM, ya que gracias al uso de la velocidad horizontal, este modelo logra reducir en unos 20 segundos la predicción a 15 minutos de la llegada con respecto a la realizada sin emplear dicha velocidad. Además, vuelve a quedar patente que el modelo ConvComplejo es el que mejores predicciones realiza a 15 minutos de la llegada, aunque no logra realizar tan buenas predicciones como los otros modelos a 30 minutos de la llegada.

6.4.7. Análisis incluyendo la distancia al aeropuerto, velocidad horizontal y día de la semana

Generalmente, los vuelos que operan en un aeropuerto son los mismos semana tras semana, existiendo cierta periodicidad que puede ser aprovechada por las redes LSTM, y solo se alteran con los cambios de temporada o en fechas especiales (Navidad, fiestas nacionales, etc.). Por ejemplo, en la Figura 6.14 se observa que el vuelo Turín-Madrid durante los meses de noviembre y diciembre de 2021 solo operará los lunes y los viernes.

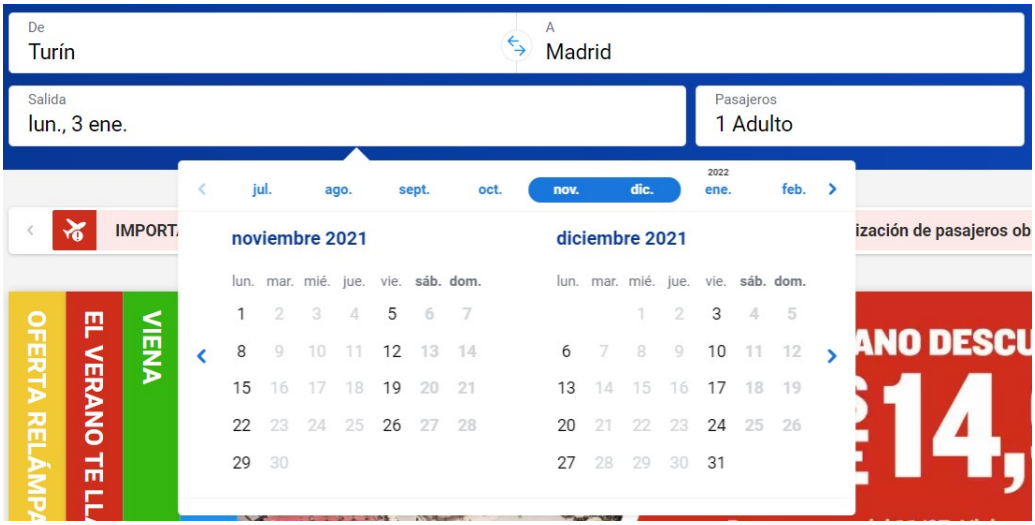


Figura 6.14: Pantalla de búsqueda de vuelos en la compañía Ryanair. Fuente: [123].

Asimismo, aunque un vuelo solo opere en determinados días de la semana, puede que se realice siempre siguiendo la misma ruta y/o a la misma hora (Figura 6.15), estableciendo una cierta regularidad que puede ser aprovechada por las redes LSTM.

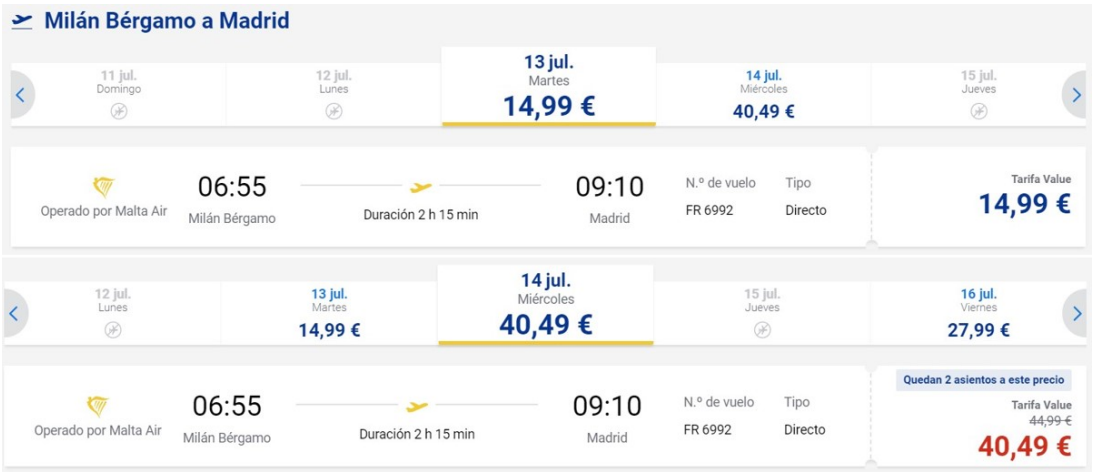


Figura 6.15: Ejemplo de la regularidad de un vuelo. Fuente: [123].

El objetivo de este estudio será determinar si el uso del día de la semana en la que se realiza el vuelo para predecir el RTA permite realizar mejores predicciones que las obtenidas en la Subsección 6.4.6.

En este estudio se va a comparar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** sobre el conjunto de vuelos del archivo **sieteDiasFlightData.csv**. Además, para realizar las predicciones del RTA se utilizarán las características *latitud* (*latitude*), *longitud* (*longitude*), *altitud* (*altitude*), *distancia* (*distanciaHaversine*), *velocidad horizontal* (*speed*) y *día de la semana* (*day_of_week*). Para ello, se entrenarán estos modelos con el conjunto de vuelos de entrenamiento (de dicho *dataset*), y después se realizarán las predicciones para el conjunto de vuelos de pruebas (de dicho *dataset*). En esta ocasión, la característica *día de la semana* (*day_of_week*) no se escalará, ya que queremos que esta variable juegue un papel más importante en el proceso de predicción del RTA con el fin de aprovechar la periodicidad de los vuelos por semanas que se ha explicado previamente. La precisión de estas predicciones se recoge en la Tabla 6.10 y en la Figura 6.16.

	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	185,695	186,300	195,819
MAE 30 MIN	194,985	191,584	206,840
MAE 15 MIN	134,260	127,796	121,219

Tabla 6.10: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

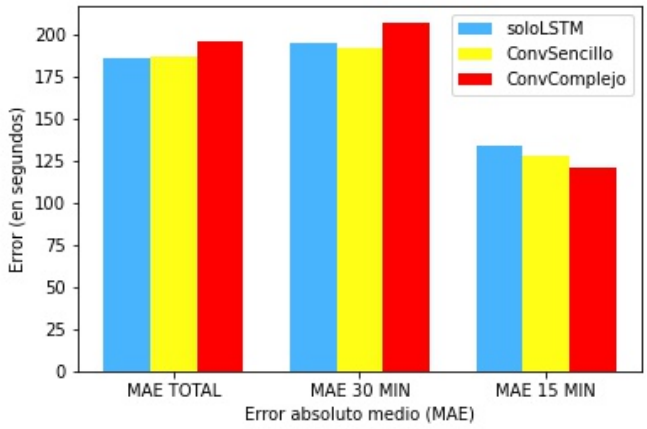


Figura 6.16: MAE de las predicciones realizadas sobre los vuelos de prueba del conjunto de datos de 7 días utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

Las predicciones realizadas utilizando el día de la semana empeoran las obtenidas sin ese dato (en más de 40 segundos de media el MAE a 30 minutos para la llegada, y en unos 20 segundos el MAE a 15 minutos). Esto se puede deber a la limitación del volumen de datos del *dataset* utilizado, ya que con solo una semana de datos es imposible que estos

modelos encuentren patrones semanales que les ayuden a su aprendizaje. Entrenar estos modelos con un nuevo conjunto de datos más amplio permitiría seguramente encontrar patrones en los mismos días de la semana como los comentados al inicio de este análisis, logrando así unas mejores predicciones.

A pesar de esto, todos los modelos son capaces de encontrar ciertos patrones en los datos ya que las predicciones realizadas son relativamente buenas. En esta ocasión, los modelos ConvSencillo y ConvComplejo vuelven a realizar mejores predicciones que el modelo soloLSTM, al igual que en el resto de análisis realizados en este proyecto.

Capítulo 7

Evaluación de los modelos

En este capítulo se va a evaluar la precisión de los modelos **soloLSTM**, **ConvSencillo** y **ConvComplejo** que fueron entrenados con el dataset **sieteDiasFlightData.csv** y que utilizan las características *latitud* (*latitude*), *longitud* (*longitude*), *altitud* (*altitude*), *distancia* (*distanciaHaversine*), *velocidad horizontal* (*speed*) y *día de la semana* (*day_of_week*) para predecir el RTA en conjuntos de datos diferentes a los usados en el entrenamiento, es decir, como si fuese una aplicación en la realidad.

Observación: se recuerda que la característica *día de la semana* se utilizará sin escalar, ya que se quiere que esta variable juegue un papel más importante en el proceso de predicción del RTA con el fin de aprovechar la periodicidad de los vuelos por semanas que se ha explicado anteriormente.

Para realizar esta evaluación, se probará la precisión de estos modelos sobre tres nuevos conjuntos de prueba que contienen vuelos realizados en otras fechas diferentes a las de la semana del 2 al 8 de febrero de 2018, y que por lo tanto, no tienen vuelos en común con ninguno de los conjuntos de datos utilizados en la Sección 6.1. Es decir, los vuelos contenidos en estos nuevos *datasets* no han sido previamente visualizados por los modelos.

Los conjuntos de prueba que se utilizarán son los siguientes:

- **flightData2018-02-09.csv:** este archivo contiene 316.061 filas de datos correspondientes a 331 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados el viernes 9 de febrero de 2018.
- **flightData2018-02-24.csv:** este archivo contiene 363.582 filas de datos correspondientes a 343 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados el sábado 24 de febrero de 2018.
- **flightData2018-02-26.csv:** este archivo contiene 430.640 filas de datos correspondientes a 407 vuelos con destino el Aeropuerto Adolfo Suárez Madrid-Barajas realizados el lunes 26 de febrero de 2018.

Los datos de todos los archivos están extraídos de OpenSky y enriquecidos con información de planes de vuelo procedentes de Network Manager, propiedad de Eurocontrol.

Para llevar a cabo la evaluación de estos modelos, se debe realizar el mismo procedimiento que el explicado en la Sección 6.4, pero en esta ocasión sobre estos conjuntos de prueba.

Sin embargo, se debe tener en cuenta que el lunes día 5 de febrero de 2018 hubo un temporal de nieve en Madrid que provocó el retraso de numerosos vuelos, así como el desvío y la cancelación de muchos otros. Además, durante algunas horas dos pistas tuvieron que ser cerradas para quitar la nieve que se acumulaba sobre ellas y los vuelos programados se espaciaron por seguridad [124, 125]. Dado que nuestros tres modelos han sido entrenados, entre otros datos, con vuelos de este día, y que el día es la característica que tiene una mayor importancia a la hora de predecir el RTA al no estar escalada, este temporal seguramente afecte a la predicción realizada para los vuelos del día 26 por tratarse de vuelos efectuados un lunes.

Por esta razón, en primer lugar se evaluarán las predicciones obtenidas al probar los tres modelos sobre los días 9 y 24 (Sección 7.1), y después, sobre el día 26 (Sección 7.2).

7.1. Evaluación en los días 9 y 24

Los predicciones realizadas por los tres modelos sobre el conjunto de vuelos del día 9 (resultados en la Tabla 7.1 y en la Figura 7.1) y del día 24 (resultados en la Tabla 7.2 y en la Figura 7.2) revelan que nuestros modelos no son tan buenos prediciendo como esperábamos que fuesen tras los resultados que se obtuvieron en la Subsección 6.4.7.

A 30 minutos para la llegada al aeropuerto, parece que el modelo soloLSTM realiza mejores predicciones que los otros dos modelos. Sin embargo, no presenta un gran desempeño a 15 minutos para la llegada, donde los otros dos modelos realizan predicciones bastante mejores.

Por otro lado, el modelo ConvComplejo parece ser el que realiza mejores predicciones en general dado su error absoluto medio para el total del vuelo.

Finalmente, tras todos los análisis realizados se podría decir que este modelo ConvComplejo es el más adecuado para predecir el tiempo de llegada de las aeronaves al aeropuerto (aunque sería necesario hacer muchas más pruebas sobre otros conjuntos de datos para confirmar este hecho).

	soloLSTM	**ConvSencillo**	**ConvComplejo**
MAE TOTAL	424,372	500,956	256,084
MAE 30 MIN	254,297	445,684	236,023
MAE 15 MIN	291,083	320,023	83,118

Tabla 7.1: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 9 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

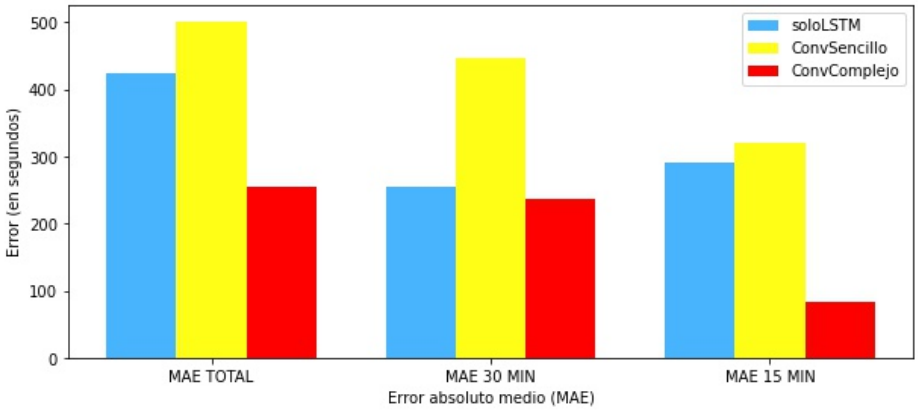


Figura 7.1: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 9 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	408,898	337,628	347,503
MAE 30 MIN	259,888	302,116	391,941
MAE 15 MIN	361,501	162,514	216,213

Tabla 7.2: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 24 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

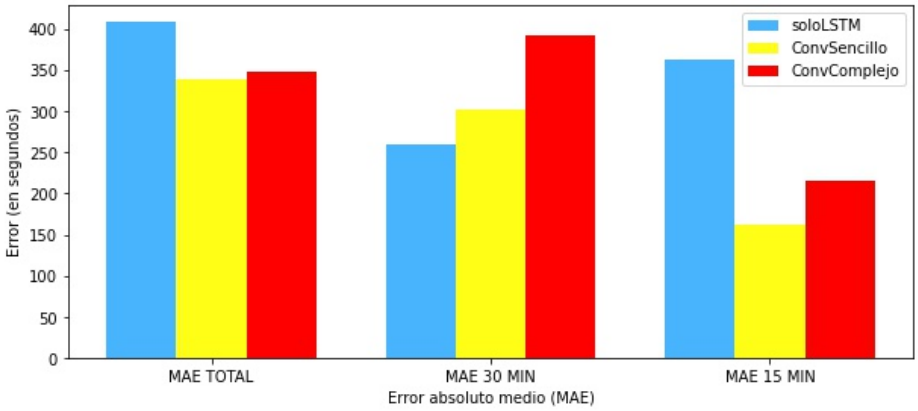


Figura 7.2: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 24 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

7.2. Evaluación en el día 26

A pesar de que los modelos realizan predicciones especialmente distantes de las esperadas sobre el conjunto de vuelos del día 26 (resultados en la Tabla 7.3 y en la Figura 7.3), estos resultados nos permiten afirmar que el temporal de nieve en Madrid afectó en gran medida al entrenamiento de los modelos, ya que en caso contrario, estos habrían realizado unas predicciones similares a las de los días 9 y 24 de la Sección 7.1.

Por lo tanto, si se hubiese escogido otro conjunto de vuelos de entrenamiento para entrenar a los modelos, seguramente se podrían haber realizado mejores predicciones sobre este conjunto de vuelos del día 26.

No obstante, observando las predicciones realizadas, se puede apreciar que los modelos que utilizan al menos una capa de red convolucional, en especial el modelo ConvComplejo, mejoran en gran medida las predicciones hechas por el modelo soloLSTM.

Esto está en consonancia con todos los resultados obtenidos en este proyecto, ya que a pesar de los cambios en las trayectorias de los vuelos como consecuencia del temporal, las redes convolucionales de los modelos ConvSencillo y ConvComplejo están siendo capaces de extraer las dependencias espaciales de estas trayectorias, logrando así mejores resultados que el modelo soloLSTM que no utiliza estas redes.

	soloLSTM	ConvSencillo	ConvComplejo
MAE TOTAL	1831,748	1032,380	662,175
MAE 30 MIN	2094,939	931,637	576,953
MAE 15 MIN	997,244	430,919	225,123

Tabla 7.3: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 26 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

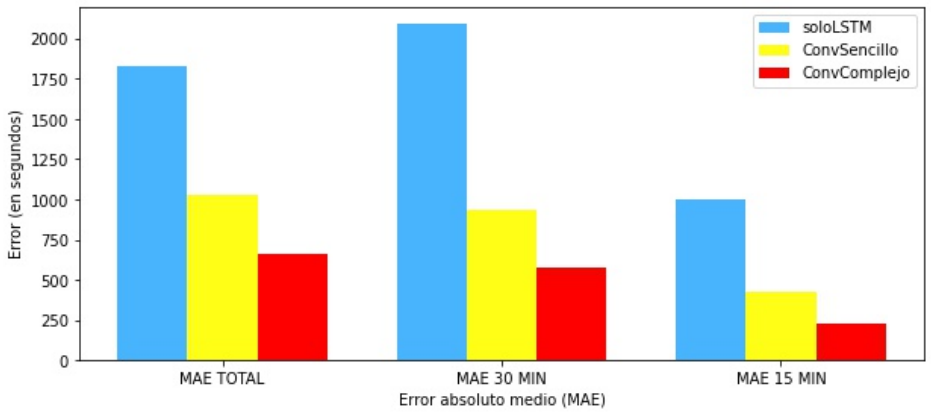


Figura 7.3: MAE de las predicciones realizadas sobre el conjunto de vuelos del día 26 de febrero de 2018 utilizando las características latitud, longitud, altitud, distancia al aeropuerto, velocidad horizontal y día de la semana.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

Tras la finalización del proyecto se puede afirmar que los objetivos planteados al inicio de este se han alcanzado completamente. En esta sección se va a hacer un repaso sobre su consecución.

- En primer lugar, pese a que el virus *SARS-CoV-2* provocó el desplome del tráfico aéreo en el año 2020, en el Capítulo 2 se pudo examinar cómo se realiza la gestión aérea en España y los aspectos más importantes del aeropuerto Adolfo Suárez Madrid-Barajas (OBJ-1).

- A continuación, en el Capítulo 3 se estudió en profundidad en qué consisten las redes neuronales, los diferentes parámetros e hiperparámetros que se deben establecer para construir estas redes y realizar su entrenamiento, y cómo se lleva a cabo este proceso de entrenamiento (OBJ-2).

- En el Capítulo 6 se logró construir un modelo de red LSTM (OBJ-3.1) que realizaba mejores predicciones que el modelo LSTM implementado por Petar Georgiev en [11]. También fue posible desarrollar dos nuevos modelos ConvLSTM que hacían un uso combinado de las redes LSTM y de las redes convolucionales, con el fin de explotar las dependencias temporales y espaciales de los datos respectivamente (OBJ-3.2).

  Tras analizar los predicciones obtenidas a lo largo de este capítulo, se observó que a pesar de que los dos modelos ConvLSTM no lograban realizar buenas predicciones del tiempo restante para la llegada al aeropuerto para los vuelos de prueba del conjunto de datos de un día, en el resto de estudios conseguían realizar mejores predicciones que el modelo basado únicamente en redes LSTM. Esto refleja que en la práctica, los modelos ConvLSTM construidos también son capaces de explotar las dependencias temporales y espaciales de los datos (OBJ-3.2 y OBJ-4).

Asimismo, en este mismo capítulo también se probaron los modelos construidos sobre diferentes conjuntos de vuelos y utilizando diferentes características de datos para predecir el tiempo de llegada de las aeronaves. Como consecuencia de estas pruebas, destaca que el uso de conjuntos de datos más grandes de datos, así como el empleo combinado de la terna de valores (latitud, longitud, altitud) junto a otros parámetros como la distancia al aeropuerto o la velocidad horizontal, juegan un papel fundamental para lograr mejores predicciones (OBJ-3.3 y OBJ-4).

- Finalmente, las evaluaciones efectuadas en el Capítulo 7 nos permiten, por un lado, corroborar que los modelos ConvLSTM realizan mejores pronósticos que los modelos basados únicamente en redes LSTM, y por otro, comprobar de primera mano que el tiempo meteorológico puede afectar en gran medida al entrenamiento de los modelos y a las predicciones que efectuarán después sobre otros conjuntos de datos (OBJ-4).

8.2. Trabajo futuro

A partir de este proyecto existen diferentes líneas de trabajo futuro que no se han podido llevar a cabo por la limitación de tiempo del mismo, pero que podrían suponer grandes mejoras de tiempo en las predicciones a realizar. En particular se destacan:

1. Ampliar el conjunto de vuelos de entrenamiento y entrenar a los modelos con él: disponer de un conjunto histórico de vuelos más amplio permitiría explotar todo el potencial de las redes LSTM y de los modelos construidos, ya que serían capaces de encontrar y explotar los patrones en los datos, y por lo tanto, realizar predicciones más exactas.

2. Probar nuevas configuraciones de entrenamiento y/o arquitectura en los modelos: la elección de algunos de los parámetros para entrenar o construir los modelos realizados permitía obtener las mejores predicciones posibles para el conjunto de vuelos de un día. Sin embargo, esta configuración no era la más idónea para el conjunto de vuelos de siete días como se comprobó en la Sección 6.4.4. Experimentar con otros valores de los parámetros y/o configuraciones podrían derivar en la obtención de mejores predicciones para conjuntos de entrenamiento más amplios.

3. Incorporar datos meteorológicos para realizar las predicciones: como se ha podido comprobar en la Sección 7.2, el tiempo meteorológico es un factor muy importante en el campo de la navegación aérea y puede afectar a la organización de aeropuertos, a la ruta de vuelos en curso, etc. Incorporar esta información a lo largo de todos los puntos de cada trayecto de un vuelo mejoraría significativamente las predicciones.

8.3. Aprendizaje personal

La realización de este Trabajo Fin de Grado me ha supuesto una gran experiencia y aprendizaje en todos los ámbitos por varios motivos, entre los que destacan:

- Trabajar bajo el marco de trabajo UVagile, ya que me ha permitido avanzar de forma progresiva en el proyecto, visualizando dichos avances cada poco tiempo. Esto ha favorecido una continuada revisión y mejora del trabajo, evitando así la falta de calidad y el estrés personal asociado durante la ejecución del mismo que hubiera supuesto la ausencia de dicha herramienta.
- Conseguir gestionar mejor el tiempo necesario que debía emplear en cada tarea gracias al uso de Trello donde recogía todas las tareas que debía realizar.
- Poder trabajar por primera vez con el lenguaje de programación Python y con sus principales librerías matemáticas (NumPy, Pandas, etc.) que, como matemático e informático que seré, pueden ser especialmente útiles en mi futuro laboral.
- Poder trabajar, también por primera vez, con técnicas de *Machine Learning* y redes neuronales, y aprender qué son, cómo trabajar con ellas, y desarrollar mis primeros modelos neuronales completos.
- Mejorar mis habilidades de razonamiento y la capacidad de sacar conclusiones relacionadas con los análisis realizados gracias al uso de diferentes modelos neuronales y la gran variedad de resultados obtenidos.
- Perfeccionar mi dominio del lenguaje LaTeX gracias a la elaboración de este documento, que además ha supuesto una mejora de la calidad de esta memoria y del Trabajo Fin de Grado del Grado de Matemáticas.
- Darme cuenta que tengo iniciativa propia para buscar soluciones alternativas cuando surgen problemas o contratiempos inesperados.

Bibliografía

[1] Curiosfera-Historia (última visita el 12-07-2021). *Historia de la aviación y del avión.* Accesible en: `https://curiosfera-historia.com/historia-de-la-aviacion/`

[2] Rumbo (última visita el 12-07-2021). *El avión es el medio de transporte más seguro del mundo.* Accesible en: `https://www.rumbo.es/ideas-para-viajar/cultura/miedo-a-volar`

[3] Grupo Banco Mundial (última visita el 12-07-2021). *Transporte aéreo, pasajeros transportados.* Accesible en: `https://datos.bancomundial.org/indicator/IS.AIR.PSGR`

[4] Aena (última visita el 12-07-2021). *Significant magnitudes of Aena airports.* Accesible en: `https://portal.aena.es/en/corporate/relevant-figures.html`

[5] Kai Zhang et al. (2020): *Spatio-Temporal Data Mining for Aviation Delay Prediction.* En: *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC).* Págs. 1-7

[6] Hao Zhang et al. (2021): *A multi-step airport delay prediction model based on spatial-temporal correlation and auxiliary features.* En: *IET Intelligent Transport Systems, 15.7.* Págs. 916-928.

[7] Samet Ayhan et al. (2018): *Predicting Estimated Time of Arrival for Commercial Flights.* En: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18).* Págs. 33–42.

[8] Flightradar24 (última visita el 12-07-2021). *Live Flight Tracker - Real-Time Flight Tracker Map.* Accesible en: `https://www.flightradar24.com/`

[9] Wen-Bo Du et al. (2016): *Analysis of the Chinese Airline Network as multi-layer networks.* En: *Transportation Research Part E: Logistics and Transportation Review, 89.* Págs. 108-116.

[10] Aena (última visita el 12-07-2021). *El transporte aéreo y el turismo en España.* Accesible en: `https://portal.aena.es/es/corporativa/transporte-aereo-y-turismo.html`

[11] Petar Georgiev Aleksandrov. (2020): *Una propuesta basada en aprendizaje automático para la mejora de la predicción de tiempos de llegada.* Universidad de Valladolid, 128 páginas. Accesible en: https://uvadoc.uva.es/handle/10324/41514

[12] Wikipedia (última visita el 12-07-2021). *Air traffic management.* Accesible en: https://en.wikipedia.org/wiki/Air_traffic_management

[13] Abigail Tabor (última visita el 12-07-2021). *What is Air Traffic Management Technology Demonstration-1?* Accesible en: https://www.nasa.gov/ames/atd1

[14] Enaire (última visita el 12-07-2021). *Quiénes somos.* Accesible en: https://www.enaire.es/sobre_enaire/conoce_enaire/quienes_somos_ENAIRE

[15] Enaire (última visita el 12-07-2021). *ENAIRE incorpora una nueva versión avanzada del sistema de control de tráfico aéreo.* Accesible en: https://www.enaire.es/es_ES/2021_04_30/ndp_enaire_incorpora_nueva_version_avanzada_sacta

[16] Enaire (última visita el 12-07-2021). *SACTA.* Accesible en: https://www.enaire.es/servicios/atm/sistemas_de_gestion_del_transito_aereo_atm/sacta

[17] Enaire (última visita el 12-07-2021). *ENAIRE incorpora una nueva versión avanzada del sistema de control de tráfico aéreo.* Accesible en: https://www.enaire.es/es_ES/2021_04_30/ndp_enaire_incorpora_nueva_version_avanzada_sacta

[18] Escudo Digital (última visita el 12-07-2021). *El cielo de España, más digitalizado y seguro gracias a Indra.* Accesible en: https://escudodigital.com/tecnologia/indra-sacta-itec-4-0-enaire-navegacion-aerea-controladores-aereos-medioambiente-cielo/

[19] Marco Meides (última visita el 12-07-2021). *The OpenSky Network - Free ADS-B and Mode S data for Research.* Accesible en: https://opensky-network.org/

[20] ADSBHub (última visita el 12-07-2021). *ADSBHub - Free ADS-B Data Exchange and Plane Tracking.* Accesible en: https://www.adsbhub.org/

[21] Miguel Ángel Martínez Prieto et al. (2017): *Integrating flight-related information into a (Big) data lake.* En: *IEEE/AIAA 36th Digital Avionics Systems Conference (DASC).*

[22] ADS-B Technologies (última visita el 12-07-2021). *ADS-B Technologies Website.* Accesible en: http://www.ads-b.com/

[23] Isle of Man Aircraft Registry (última visita el 12-07-2021). *What is ADS-B OUT?* Accesible en: https://www.iomaircraftregistry.com/flight-operations/flight-operations/ads-b-out/

[24] Daniel Jambrina (última visita el 12-07-2021). *Cómo funciona el ADS-B. La tecnología que viene ya está aquí.* Accesible en: `http://www.aviacionglobal.com/articulos-tecnicos-de-aviacion/como-funciona-el-ads-b-la-tecnologia-que-viene-ya-esta-aqui/`

[25] Andrés Marina (última visita el 12-07-2021). *Comienzan las pruebas del ADS-B en el Atlántico Norte.* Accesible en: `https://aviaciondigital.com/comienzan-pruebas-ads-b-atlantico-norte/`

[26] ADS-B Technologies (última visita el 12-07-2021). *ADS-B Technologies Website.* Accesible en: `http://www.aeroservicio.com/adsb-in-out/`

[27] Aeropuerto de Madrid-Barajas (última visita el 12-07-2021). *Bienvenido al aeropuerto Madrid-Barajas.* Accesible en: `https://www.aeropuertomadrid-barajas.com/`

[28] Secretaría de Estado de Transportes, Movilidad y Agenda Urbana. (2020-07): *Informe de seguimiento anual. Plan de acción en materia de contaminación acústica. Año 2019 - Aeropuerto Adolfo Suárez Madrid-Barajas.* Accesible en: `https://www.mitma.gob.es/recursos_mfom/paginabasica/recursos/informe_de_seguimiento_2019_-_aeropuerto_as_madrid-barajas.pdf`

[29] Redacción la Vanguardia (última visita el 12-07-2021). *¿Qué significan los números en las pistas de aterrizaje de los aeropuertos?* Accesible en: `https://www.lavanguardia.com/ocio/viajes/20181211/453472279873/que-significan-numeros-pistas-aterrizaje.html`

[30] EuropaPress (última visita el 12-07-2021). *Enaire gestiona un 66,5 % menos de vuelos en el primer semestre que hasta junio de 2019.* Accesible en: `https://www.europapress.es/turismo/transportes/aeropuertos/noticia-enaire-gestiona-665-menos-vuelos-primer-semestre-junio-2019-20210709125803.html`

[31] Secretaría de Estado de Transportes, Movilidad y Agenda Urbana (última visita el 12-07-2021). *Eurocontrol.* Accesible en: `https://www.mitma.gob.es/areas-de-actividad/aviacion-civil/organismos-internacionales/eurocontrol`

[32] Eurocontrol (última visita el 12-07-2021). *COVID-19 Impact on EUROCONTROL Airports.* Accesible en: `https://www.eurocontrol.int/sites/default/files/2021-07/eurocontrol-brief-on-covid19-impact-madrid-apt-20210712.pdf`

[33] Haiyang Yu et al. (2017): *Spatiotemporal Recurrent Convolutional Networks for Traffic Prediction in Transportation Networks.* En: *Sensors, 17.7, 1501.*

[34] Fernando Sancho Caparrini (última visita el 12-07-2021). *Introducción al Aprendizaje Automático.* Accesible en: `http://www.cs.us.es/~fsancho/?e=75`

[35] RecluiT (última visita el 12-07-2021). *Historia y evolución del Machine Learning.* Accesible en: https://recluit.com/historia-y-evolucion-del-machine-learning/

[36] Paloma Recuero de los Santos (última visita el 12-07-2021). *Tipos de aprendizaje en Machine Learning: supervisado y no supervisado.* Accesible en: https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/

[37] BBVA (última visita el 12-07-2021). *'Machine learning': ¿qué es y cómo funciona?* Accesible en: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/

[38] Sas (última visita el 12-07-2021). *Aprendizaje automático: Qué es y por qué es importante.* Accesible en: https://www.sas.com/es_mx/insights/analytics/machine-learning.html

[39] Fernando Sancho Caparrini (última visita el 12-07-2021). *Aprendizaje Supervisado y No Supervisado.* Accesible en: http://www.cs.us.es/~fsancho/?e=77

[40] Luis Alberto García (última visita el 12-07-2021). *Aprendizaje por refuerzos; ¿Qué es? ¿Cómo se usa?* Accesible en: https://www.kabel.es/aprendizaje-refuerzos/

[41] Dell Technologies (última visita el 12-07-2021). *Inteligencia Artificial:¿Qué es el aprendizaje de refuerzo? Una Explicación Simple y Ejemplos Prácticos.* Accesible en: https://www.delltechnologies.com/es-es/blog/inteligencia-artificial-aprendizaje-refuerzo-explicacion-y-ejemplos/

[42] Héctor Garbisu Arocha (última visita el 12-07-2021). *Utilización de Redes Neuronales Recurrentes para el análisis de firmas.* Accesible en: https://accedacris.ulpgc.es/bitstream/10553/18830/4/0728346_00000_0000.pdf

[43] N. Sáenz y M.Álvaro (última visita el 12-07-2021). *Redes neuronales: concepto, aplicaciones y utilidad en medicina.* Accesible en: https://www.elsevier.es/es-revista-atencion-primaria-27-articulo-redes-neuronales-concepto-aplicaciones-utilidad-13033737

[44] Artyco (última visita el 12-07-2021). *Qué son las redes neuronales y cuál es su aplicación en el marketing.* Accesible en: https://artyco.com/que-son-las-redes-neuronales-y-cual-es-su-aplicacion-en-el-marketing/

[45] Michael Hurtado (última visita el 12-07-2021). *Introducción a las redes neuronales.* Accesible en: http://slides.com/michaelhurtado/neuralnet

[46] Aníbal Bregón. (2020): *Sistemas inteligentes.* En: *Universidad de Valladolid.*

[47] Grupo us (última visita el 12-07-2021). *Conceptos básicos sobre redes neuronales.* Accesible en: http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm

[48] Pedro Larranaga et al. (última visita el 12-07-2021). *Tema 8. Redes Neuronales.* Accesible en: `http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf`

[49] Mariano Rivera (última visita el 12-07-2021). *Red Neuronal Multicapa en Keras.* Accesible en: `http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/nn_multicapa/nn_multicapa.html`

[50] IBM (última visita el 12-07-2021). *El modelo de redes neuronales.* Accesible en: `https://www.ibm.com/support/knowledgecenter/es/SS3RA7_sub/modeler_mainhelp_client_ddita/components/neuralnet/neuralnet_model.html`

[51] Industria 4.0, Inteligencia Artificial (última visita el 12-07-2021). *Qué son las redes neuronales y sus funciones.* Accesible en: `https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/`

[52] Diego Calvo (última visita el 12-07-2021). *Función de activación – Redes neuronales.* Accesible en: `https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/`

[53] Sebastian Raschka and Vahid Mirjalili. (2019): *Python Machine Learning: Aprendizaje automático y aprendizaje profundo con Python, scikit-learn y TensorFlow.* Marcombo S.L. ISBN: 978-84-267-2720-6.

[54] Developers Google (última visita el 12-07-2021). *Reducción de la pérdida: Tasa de aprendizaje.* Accesible en: `https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?hl=es-419`

[55] Miguel Díaz Kusztrich (última visita el 12-07-2021). *Algoritmo de retro propagación.* Accesible en: `http://software-tecnico-libre.es/es/articulo-por-tema/todas-las-secciones/todos-los-temas/todos-los-articulos/algoritmo-de-retro-propagacion`

[56] Diego Calvo (última visita el 12-07-2021). *Función de coste – Redes neuronales.* Accesible en: `https://www.diegocalvo.es/funcion-de-coste-redes-neuronales/`

[57] Jesús Martínez (última visita el 12-07-2021). *¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?* Accesible en: `https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/`

[58] Luis Velasco (última visita el 12-07-2021). *Optimizadores en redes neuronales profundas: un enfoque práctico.* Accesible en: `https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5`

[59] Robbins herbert & Monro Sutton. (1951): *A Stochastic Approximation Method.* En: *The Annals of Mathematical Statistics, 22.3.* Págs. 400-407.

[60] Coding Compiler (última visita el 12-07-2021). *Machine Learning vs. Deep Learning.* Accesible en: https://codingcompiler.com/machine-learning-vs-deep-learning/

[61] Ian Goodfellow. (2016): *Deep Learning.* MIT Press. Accesible en: https://www.deeplearningbook.org/

[62] Jochem Grietens (última visita el 12-07-2021). *The difference between Machine & Deep Learning.* Accesible en: https://verhaert.com/difference-machine-learning-deep-learning/

[63] Carlos García Moreno (última visita el 12-07-2021). *¿Qué es el Deep Learning y para qué sirve?* Accesible en: https://www.indracompany.com/es/blogneo/deep-learning-sirve

[64] M. Tim Jones (última visita el 12-07-2021). *Arquitecturas de aprendizaje profundo.* Accesible en: https://developer.ibm.com/es/articles/cc-machine-learning-deep-learning-architectures/

[65] SPRH Labs (última visita el 12-07-2021). *Understanding Deep Learning: DNN, RNN, LSTM, CNN and R-CNN.* Accesible en: https://medium.com/@sprhlabs/understanding-deep-learning-dnn-rnn-lstm-cnn-and-r-cnn-6602ed94dbff

[66] David E. Rumelhart et al. (1986): *Learning representations by backpropagating errors.* En: *Nature, 323 (6088).* Págs. 533–536.

[67] Paul J.Werbos. (1988): *Generalization of backpropagation with application to a recurrent gas market model.* En: *Neural Networks, 1.4.* Págs. 339-356.

[68] Jeffrey L. Elman. (1990): *Finding structure in time.* En: *Cognitive Science, 14.2.* Págs. 179–211.

[69] Diego Calvo (última visita el 12-07-2021). *Red Neuronal Recurrente – RNN.* Accesible en: https://www.diegocalvo.es/red-neuronal-recurrente/

[70] Roger Strukhoff (última visita el 12-07-2021). *Introduction to Neural Networks and Metaframeworks with TensorFlow.* Accesible en: https://www.altoros.com/blog/introduction-to-neural-networks-and-metaframeworks-with-tensorflow/

[71] Jakub Kvita (última visita el 12-07-2021). *Visualizations of RNN units.* Accesible en: https://kvitajakub.github.io/2016/04/14/rnn-diagrams/

[72] Wikipedia (última visita el 12-07-2021). *Vanishing gradient problem.* Accesible en: https://en.wikipedia.org/wiki/Vanishing_gradient_problem

[73] Meta-learning Lab (última visita el 12-07-2021). *Problema del desvanecimiento del gradiente (vanishing gradient problem).* Accesible en: `https://mlearninglab.com/2018/05/06/problema-de-desvanecimiento-del-gradiente-vanishing-gradient-problem/`

[74] Sepp Hochreiter & Jürgen Schmidhuber. (1997): *Long Short-Term Memory.* En: *Neural Computation, 9.8.* Págs. 1735-1780.

[75] Scott Sun et al. (última visita el 12-07-2021). *Lecture 16: Building Blocks of Deep Learning.* Accesible en: `https://sailinglab.github.io/pgm-spring-2019/notes/lecture-16/`

[76] Oinkina (última visita el 12-07-2021). *Understanding LSTM Networks.* Accesible en: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[77] Jason Brownlee (última visita el 12-07-2021). *Gentle Introduction to Generative Long Short-Term Memory Networks.* Accesible en: `https://machinelearningmastery.com/gentle-introduction-generative-long-short-term-memory-networks/`

[78] Jason Brownlee (última visita el 12-07-2021). *Encoder-Decoder Long Short-Term Memory Networks.* Accesible en: `https://machinelearningmastery.com/encoder-decoder-long-short-term-memory-networks/`

[79] Simeon Kostadinov (última visita el 12-07-2021). *Understanding Encoder-Decoder Sequence to Sequence Model.* Accesible en: `https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346`

[80] Kunihiko Fukushima. (1980): *Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position.* En: *Biological Cybernetics, 36.4.* Págs. 193–202.

[81] Asifullah Khan et al. (2020): *A Survey of the Recent Architectures of Deep Convolutional Neural Networks.* En: *Artif Intell Rev, 53.* Págs. 5455-5516.

[82] Wikipedia (última visita el 12-07-2021). *Red neuronal convolucional.* Accesible en: `https://es.wikipedia.org/wiki/Red_neuronal_convolucional`

[83] Diego Calvo (última visita el 12-07-2021). *Red Neuronal Convolucional CNN.* Accesible en: `https://www.diegocalvo.es/red-neuronal-convolucional/`

[84] Pilar Cornieles (última visita el 12-07-2021). *Entendiendo las redes neuronales: De la neurona a RNN, CNN y Deep Learning.* Accesible en: `https://ia-latam.com/2019/02/06/entendiendo-las-redes-neuronales-de-la-neurona-a-rnn-cnn-y-deep-learning/`

[85] Ken Schwaber and Jeff Sutherland. (2020): *The Scrum Guide.* Accesible en: https://www.scrum.org/resources/scrum-guide

[86] Scrum.org (última visita el 12-07-2021): *The Scrum Framework Poster.* Accesible en: https://www.scrum.org/resources/scrum-framework-poster

[87] Miguel Ángel Martínez Prieto et al. (2020): *Hacia la consolidación de las aulas ágiles..* En: *Actas de las XXVI Jornadas sobre Enseñanza Universitaria de la Informática, vol. 5.* Págs. 21-28.

[88] Miguel Ángel Martínez Prieto et al. (2019): *¿Puede ser Agile la Docencia Universitaria? (UVagile).* Accesible en: http://uvadoc.uva.es/handle/10324/37210

[89] Miguel Ángel Martínez Prieto et al. (2021): *Activando el Aula Ágil (UVagile).* Accesible en: https://uvadoc.uva.es/handle/10324/47451

[90] Jupyter (última visita el 12-07-2021). *Project Jupyter.* Accesible en: https://jupyter.org/

[91] Overleaf (última visita el 12-07-2021). *LaTeX,Evolved.* Accesible en: https://www.overleaf.com/

[92] Google (última visita el 12-07-2021). *¿Qué es Colaboratory?* Accesible en: https://colab.research.google.com/notebooks/intro.ipynb

[93] Microsoft (última visita el 12-07-2021). *¿Qué es Microsoft Teams?* Accesible en: https://support.microsoft.com/es-es/office/v%C3%ADdeo-%C2%BFqu%C3%A9-es-microsoft-teams-422bf3aa-9ae8-46f1-83a2-e65720e1a34d

[94] Kanbanize (última visita el 12-07-2021). *Qué es un tablero Kanban: Fundamentos.* Accesible en: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban

[95] Trello (última visita el 12-07-2021). *¿Qué es Trello?* Accesible en: https://help.trello.com/article/708-what-is-trello

[96] EasyRetro (última visita el 12-07-2021). *Improve with Fun Sprint Retrospectives.* Accesible en: https://easyretro.io/

[97] Wikipedia (última visita el 12-07-2021). *Python.* Accesible en: https://es.wikipedia.org/wiki/Python

[98] NumPy.org (última visita el 12-07-2021). *NumPy.* Accesible en: https://numpy.org/

[99] Pandas (última visita el 12-07-2021). *Pandas documentation.* Accesible en: https://pandas.pydata.org/pandas-docs/stable/index.html

[100] Matplotlib development team. (última visita el 12-07-2021). *Matplotlib.* Accesible en: `https://matplotlib.org/`

[101] Michael Waskom (última visita el 12-07-2021). *Seaborn: statistical data visualization.* Accesible en: `https://seaborn.pydata.org/`

[102] Martín Abadi et al. (última visita el 12-07-2021). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Accesible en: `https://www.tensorflow.org/`

[103] Keras Team (última visita el 12-07-2021). *Keras: the Python deep learning API.* Accesible en: `https://keras.io/`

[104] Python Software Foundation. (última visita el 12-07-2021). *Math — Mathematical functions.* Accesible en: `https://docs.python.org/3/library/math.html`

[105] Jérémie du Boisberranger et al. (última visita el 12-07-2021). *Scikit-learn: machine learning in Python.* Accesible en: `https://scikit-learn.org/stable/index.html`

[106] Balthazar Rouberol (última visita el 12-07-2021). *Haversine - PyPI.* Accesible en: `https://pypi.org/project/haversine/`

[107] Scrum Manager (última visita el 12-07-2021). *Estimación de póquer.* Accesible en: `https://www.scrummanager.net/bok/index.php?title=Estimaci%C3%B3n_de_p%C3%B3quer`

[108] Universidad de Valladolid (última visita el 12-07-2021). *Guía docente de la asignatura Trabajo Fin de Grado.* Accesible en: `https://www.dropbox.com/s/vytz6i00vm8ls0x/TFG_2020_2021_GuiaDocente.pdf`

[109] Universidad de Valladolid (última visita el 12-07-2021). *¿Qué es el crédito E.C.T.S ?.* Accesible en: `https://www.uva.es/export/sites/uva/2.docencia/2.02.mastersoficiales/2.02.13.preguntasfrecuentes/`

[110] Universidad de Valladolid (última visita el 12-07-2021). *Calendario académico 2020-2021.* Accesible en: `https://www.uva.es/export/sites/uva/7.comunidaduniversitaria/7.06.calendarioacademico/_documentos/Calendario-20-21.pdf`

[111] Glassdoor (última visita el 12-07-2021). *Buscador de sueldos y remuneración.* Accesible en: `https://www.glassdoor.es/Sueldos/index.htm`

[112] Javier Santos Pascualena (última visita el 12-07-2021). *¿Cuánto cuesta contratar un trabajador?.* Accesible en: `https://www.infoautonomos.com/blog/cuanto-cuesta-contratar-un-trabajador/`

[113] Eurocontrol (última visita el 12-07-2021). *Network operations — EUROCONTROL.* Accesible en: `https://www.eurocontrol.int/network-operations`

[114] Antipodas.net (última visita el 12-07-2021). *Coordenadas de Aeropuerto de Madrid-Barajas (España).* Accesible en: https://www.antipodas.net/coordenadaspais/espana/adolfo-suarez-madrid-barajas-airport.php

[115] Baijayanta Roi (última visita el 12-07-2021). *All about Feature Scaling.* Accesible en: https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35

[116] Keras Team (última visita el 12-07-2021). *LSTM layer.* Accesible en: https://keras.io/api/layers/recurrent_layers/lstm/

[117] Keras Team (última visita el 12-07-2021). *Dense layer.* Accesible en: https://keras.io/api/layers/core_layers/dense/

[118] Keras Team (última visita el 12-07-2021). *Conv1D layer.* Accesible en: https://keras.io/api/layers/convolution_layers/convolution1d/

[119] Keras Team (última visita el 12-07-2021). *Flatten layer.* Accesible en: https://keras.io/api/layers/reshaping_layers/flatten/

[120] Jason Brownlee (última visita el 12-07-2021). *Overfitting and Underfitting With Machine Learning Algorithms.* Accesible en: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[121] Stack Exchange (última visita el 12-07-2021). *Which day of the week is typically the least busy at an airport?* Accesible en: https://travel.stackexchange.com/questions/123852/which-day-of-the-week-is-typically-the-least-busy-at-an-airport

[122] Javier Gilabert (última visita el 12-07-2021). *Fomento tramita el desvío de un tercio del tráfico aéreo sobre poblaciones del interior de Menorca.* Accesible en: https://www.menorca.info/menorca/local/2019/10/05/667873/fomento-tramita-desvio-tercio-del-trafico-aereo-sobre-poblaciones-del-interior-menorca.html

[123] Ryanair (última visita el 12-07-2021). *Página Oficial de Ryanair.* Accesible en: https://www.ryanair.com/es/es

[124] Silvia Fernández (última visita el 12-07-2021). *Retrasos en Barajas y 70 vuelos cancelados por el temporal de nieve.* Accesible en: https://www.elmundo.es/economia/2018/02/05/5a784c66268e3e706b8b45b9.html

[125] Antena 3 (última visita el 12-07-2021). *El temporal de nieve obliga a cancelar 70 vuelos en Barajas y a desviar seis.* Accesible en: https://www.antena3.com/noticias/sociedad/la-nieve-obliga-a-cerrar-pistas-del-aeropuerto-de-madrid-barajas-decenas-de-vuelos-retrasados_201802055a7849130cf25b59e2e90578.html

Apéndices

Apéndice A

Resultados de las retrospectivas

En esta sección del apéndice se adjuntan los resultados de las reuniones de Retrospectiva (realizadas como parte de la reunión *Retroplanning* de UVagile). Las figuras A.1, A.2, A.3, A.4 y A.5 que se muestran en las páginas sucesivas corresponden al tablero de la herramienta EasyRetro al finalizar cada retrospectiva después de cada *sprint*.

El resultado de estas sesiones resulta muy enriquecedor para el equipo de desarrolladores (que en este caso es el alumno) ya que le permite ver y poner en valor todo aquello que ha ido haciendo bien a lo largo de las semanas. Además le ayuda a darse cuenta de los problemas que han ido surgiendo durante el desarrollo del proyecto y cómo puede solucionarlos.

En ellas se destaca la buena adaptación del alumno al proyecto, ya que desconocía por completo el campo de investigación y la metodología UVagile. También destaca su iniciativa para resolver los problemas y aportar nuevas ideas para alcanzar los objetivos fijados en cada *sprint*, así como su organización dentro de este.

Por otra parte, como objetivos a mejorar tras cada *sprint*, en general se han señalado los conocimientos del lenguaje de programación Python, puesto que no contar con experiencia previa de este lenguaje al comienzo del proyecto ha supuesto en ocasiones una complejidad añadida.

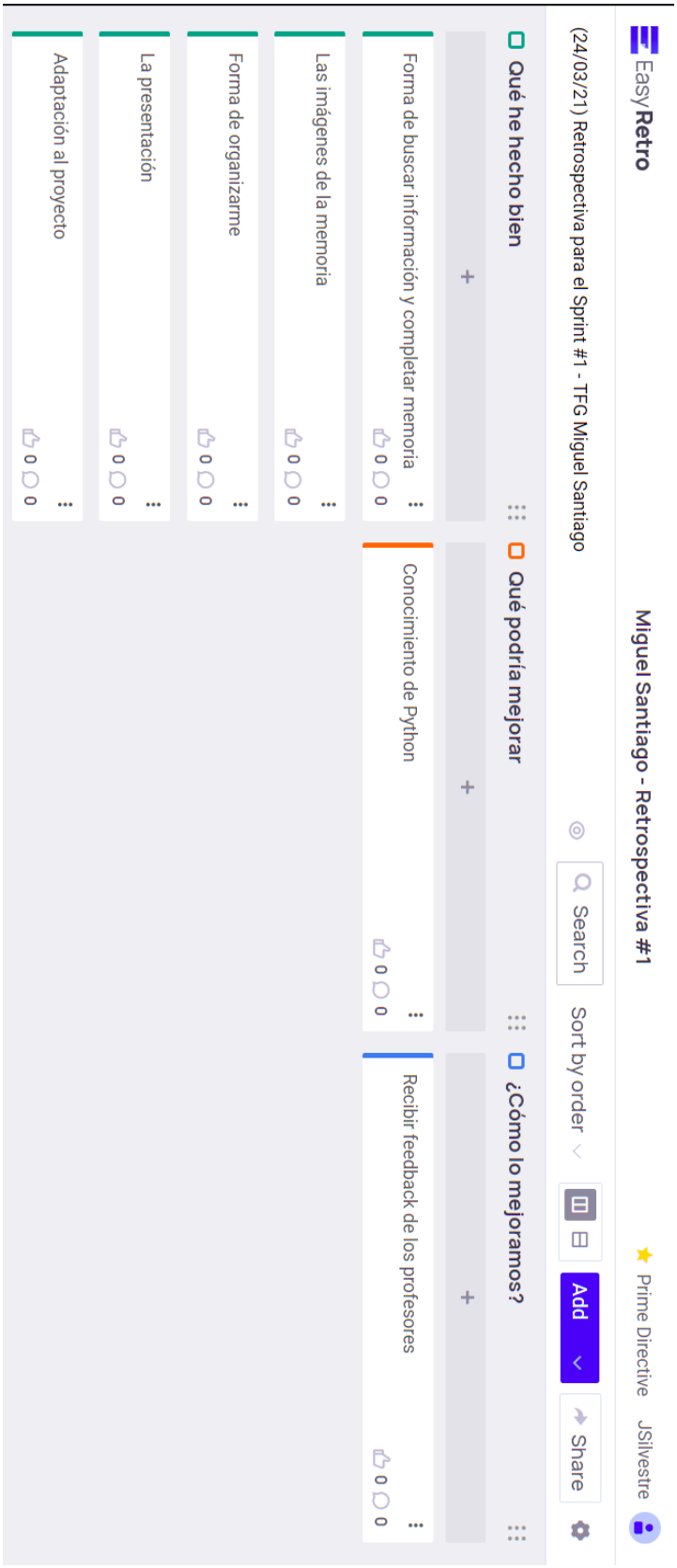


Figura A.1: Resultados retrospectiva #1 (2021-03-24).

Figura A.2: Resultados retrospectiva #2 (2021-04-21).

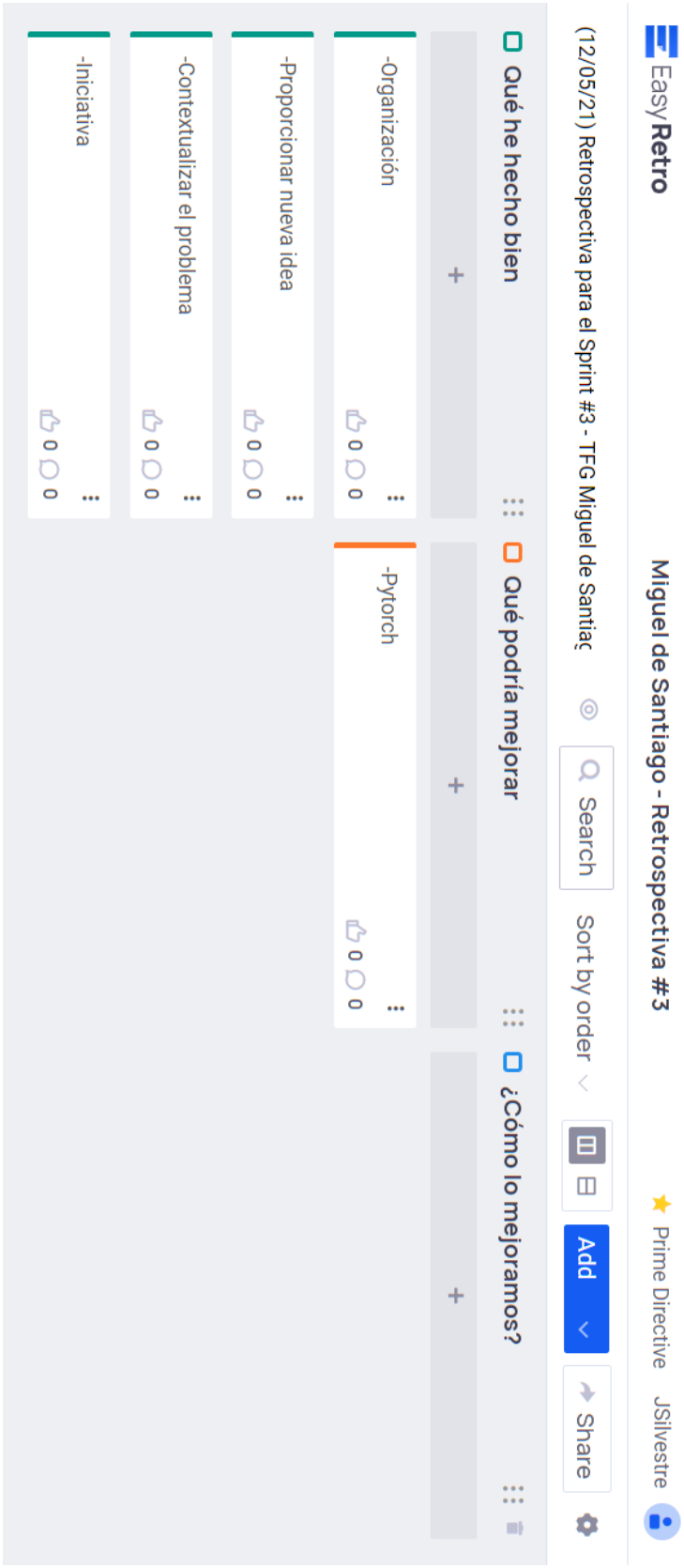


Figura A.3: Resultados retrospectiva #3 (2021-05-12).

Figura A.4: Resultados retrospectiva #4 (2021-06-23).

Figura A.5: Resultados retrospectiva #5 (2021-07-13).

Apéndice B

Experimentos para la optimización de los modelos

Optimización del número de épocas y el tamaño de los lotes

Para establecer un número de épocas y un tamaño de los lotes fijo para todos los modelos construidos (soloLSTM, ConvSencillo y ConvComplejo) se realizaron varias pruebas sobre el *dataset unDiaFlightData.csv*. Para estos experimentos se emplearon las métricas explicadas en la Sección 6.4 para comparar las predicciones obtenidas. Hay que destacar que aunque estos valores se usarán en todos los modelos, estas pruebas consistieron en optimizarlos para el modelo soloLSTM, ya que fue el primero del que se disponía y permitía comparar las predicciones obtenidas con las del estudio [11].

Partiendo de los estudios realizados en [11], se realizaron cuatro experimentos para los valores 20, 25, 30 y 35 en el número de épocas. Además, para cada posible valor en el número de épocas, se realizaron cuatro pruebas, empleando en cada una de ellas los valores 500, 1000, 1500 y 2000 para el tamaño de los lotes (*batch*).

Las predicciones obtenidas en todos estos experimentos se recogen agrupadas por el número de épocas en las Tablas B.1 (20 épocas), B.2 (25 épocas), B.3 (30 épocas) y B.4 (35 épocas). Tras analizar los resultados obtenidos en dichas tablas, se observa que las mejores predicciones se alcanzan cuando se fija el número de épocas en 30. Además, en los cuatro experimentos se advierte que cuando el tamaño de los lotes se fija en 1500 se logran mejores predicciones.

Por estos dos motivos, se fijó en 30 el número de épocas y en 1500 el tamaño de los lotes para todos los análisis realizados en la Sección 6.4.

Épocas = 20	***Batch* = 500**	***Batch* = 1000**	***Batch* = 1500**	***Batch* = 2000**
MAE TOTAL	246,676	251,079	249,426	254,307
MAE 30 MIN	248,886	252,840	240,551	275,330
MAE 15 MIN	203,673	193,683	188,258	188,810

Tabla B.1: Primer experimento con el número de épocas.

Épocas = 25	***Batch* = 500**	***Batch* = 1000**	***Batch* = 1500**	***Batch* = 2000**
MAE TOTAL	258,844	238,891	228,150	252,750
MAE 30 MIN	240,923	235,585	223,807	251,289
MAE 15 MIN	236,384	182,084	169,997	218,003

Tabla B.2: Segundo experimento con el número de épocas.

Épocas = 30	***Batch* = 500**	***Batch* = 1000**	***Batch* = 1500**	***Batch* = 2000**
MAE TOTAL	253,925	225,694	232,290	228,534
MAE 30 MIN	252,266	229,993	216,013	221,154
MAE 15 MIN	198,834	153,815	140,615	183,653

Tabla B.3: Tercer experimento con el número de épocas.

Épocas = 35	***Batch* = 500**	***Batch* = 1000**	***Batch* = 1500**	***Batch* = 2000**
MAE TOTAL	229,679	245,507	235,358	248,913
MAE 30 MIN	236,429	239,337	242,071	241,731
MAE 15 MIN	201,902	192,281	165,278	190,593

Tabla B.4: Cuarto experimento con el número de épocas.

Optimización del número de filtros y la longitud de la ventana de convolución

Para establecer un número de filtros y la longitud de la ventana de convolución en el modelo ConvSencillo, se realizaron varios experimentos sobre el *dataset unDiaFlightData.csv*. Para ello, se usó dicho modelo y se emplearon las métricas explicadas en la Sección 6.4 para comparar las predicciones obtenidas.

Este proceso consistió en realizar tres experimentos para los valores 32, 64 y 128 en el número de filtros. Además, para cada posible valor en el número de filtros, se realizaron tres pruebas, empleando en cada una de ellas los valores 2, 4 y 6 para la longitud de la ventana de convolución (*kernel size*).

Las predicciones obtenidas en todos estos experimentos se recogen agrupadas por el número de filtros en las Tablas B.5 (32 filtros), B.6 (64 filtros) y B.7 (128 filtros). Tras analizar los resultados obtenidos en dichas tablas, se observa que las mejores predicciones se alcanzan cuando se fija el número de filtros en 64. Además, en los tres experimentos se advierte que cuando la longitud de la ventana de convolución se fija en 4 se logran mejores predicciones.

Por estos motivos, se fijó en 64 el número de filtros y en 4 la longitud de la ventana de convolución para todos los análisis del modelo ConvSencillo realizados en la Sección 6.4.

Filtros = 32	***Kernel size* = 2**	***Kernel size* = 4**	***Kernel size* = 6**
MAE TOTAL	239,085	224,511	238,262
MAE 30 MIN	223,859	226,827	237,979
MAE 15 MIN	206,804	184,498	201,300

Tabla B.5: Primer experimento con el número de filtros.

Filtros = 64	***Kernel size* = 2**	***Kernel size* = 4**	***Kernel size* = 6**
MAE TOTAL	245,949	253,745	244,882
MAE 30 MIN	238,006	230,155	248,857
MAE 15 MIN	169,498	136,918	197,572

Tabla B.6: Segundo experimento con el número de filtros.

Filtros = 128	***Kernel size* = 2**	***Kernel size* = 4**	***Kernel size* = 6**
MAE TOTAL	255,144	248,203	270,995
MAE 30 MIN	239,776	262,625	262,159
MAE 15 MIN	223,517	186,962	230,616

Tabla B.7: Tercer experimento con el número de filtros.

Apéndice C

Contenido adjunto

En esta sección del apéndice se detalla el contenido anexado junto a este documento en la entrega del Trabajo Fin de Grado. Este contenido está formado por tres carpetas: *Capítulo 6* y *Capítulo7*, que recogen los programas escritos en el lenguaje de programación *Python* necesarios para realizar todas las operaciones que se han descrito a lo largo de los capítulos 6 y 7 de esta memoria; y *Experimentos*, que reúne los programas escritos en *Python* con todos los experimentos que se describen en el Apéndice B.

Dentro de la carpeta *Capítulo 6* se encuentra una carpeta por cada análisis realizado en la Sección 6.4, y cada una se llama igual que la sección a la que hace referencia. En el interior de cada carpeta se encuentran los programas que se han utilizado para cada análisis, y cada programa se denomina igual que el modelo que representa. En cada programa se carga y prepara el conjunto de datos necesario, y se construye, se entrena y se prueba el modelo correspondiente.

Observación: los programas *UnDia.ipynb*, *TresDias.ipynb* y *SieteDias.ipynb* de la carpeta *Subsección 6.3.4. Análisis de los modelos LSTM* son los mismos que los programas *soloLSTM.ipynb* de las carpetas *Subsección 6.3.1. Análisis con datos de 1 día*, *Subsección 6.3.2. Análisis con datos de 3 días* y *Subsección 6.3.3. Análisis con datos de 7 días* pero con otro nombre diferente para poder diferenciarlos.

Además, dentro de la carpeta *Capítulo 6* también se halla un programa denominado *Resumen análisis realizados.ipynb* que recoge todas las predicciones realizadas por los modelos que se hallan dentro de esta carpeta.

Por otro lado, dentro de la carpeta *Capítulo 7* se encuentran los tres programas realizados para llevar a cabo la evaluación de los modelos sobre los vuelos de los días 9, 24 y 26 de febrero de 2018. Estos programas se denominan con el nombre del día utilizado en la evaluación.

Finalmente, dentro de la carpeta *Experimentos* se encuentran dos carpetas: *Optimización épocas y lotes* y *Optimización filtros y ventana convolución*. En la primera de estas carpetas se encuentran todos los programas realizados para optimizar el número de épocas y el tamaño de los lotes (donde cada uno de ellos se denomina con la prueba efectuada),

así como un programa resumen con todos los experimentos que se han llevado a cabo. En la segunda carpeta se encuentran todos los programas realizados para optimizar el número de filtros y la longitud de la ventana de convolución (donde cada uno de ellos se denomina con la prueba realizada), e igualmente, se halla un programa resumen con todos los experimentos efectuados.

El árbol de directorios y programas adjuntos es el siguiente:

```
|
+---Capítulo 6
|   |   Resumen análisis realizados.ipynb
|   |
|   +---Subsección 6.3.1. Análisis con datos de 1 día
|   |       ConvComplejo.ipynb
|   |       ConvSencillo.ipynb
|   |       soloLSTM.ipynb
|   |
|   +---Subsección 6.3.2. Análisis con datos de 3 días
|   |       ConvComplejo.ipynb
|   |       ConvSencillo.ipynb
|   |       soloLSTM.ipynb
|   |
|   +---Subsección 6.3.3. Análisis con datos de 7 días
|   |       ConvComplejo.ipynb
|   |       ConvSencillo.ipynb
|   |       soloLSTM.ipynb
|   |
|   +---Subsección 6.3.4. Análisis de los modelos LSTM
|   |       SieteDias.ipynb
|   |       SieteDiasModificado.ipynb
|   |       TresDias.ipynb
|   |       UnDia.ipynb
|   |
|   +---Subsección 6.3.5. Análisis incluyendo la distancia al aeropuerto
|   |       ConvComplejo.ipynb
|   |       ConvSencillo.ipynb
|   |       soloLSTM.ipynb
|   |
|   +---Subsección 6.3.6. Análisis incluyendo la distancia al aeropuerto
y velocidad horizontal
|   |       ConvComplejo.ipynb
|   |       ConvSencillo.ipynb
|   |       soloLSTM.ipynb
|   |
|   \---Subsección 6.3.7. Análisis incluyendo la distancia al aeropuerto,
velocidad horizontal y día de la semana
|           ConvComplejo.ipynb
|           ConvSencillo.ipynb
|           soloLSTM.ipynb
```

```
|
\---Capítulo 7
|       Evaluación Día 09.ipynb
|       Evaluación Día 24.ipynb
|       Evaluación Día 26.ipynb
|
\---Experimentos
    |
    +---Optimización épocas y lotes
    |       Épocas=20, batch=1000.ipynb
    |       Épocas=20, batch=1500.ipynb
    |       Épocas=20, batch=2000.ipynb
    |       Épocas=20, batch=500.ipynb
    |       Épocas=25, batch=1000.ipynb
    |       Épocas=25, batch=1500.ipynb
    |       Épocas=25, batch=2000.ipynb
    |       Épocas=25, batch=500.ipynb
    |       Épocas=30, batch=1000.ipynb
    |       Épocas=30, batch=1500.ipynb
    |       Épocas=30, batch=2000.ipynb
    |       Épocas=30, batch=500.ipynb
    |       Épocas=35, batch=1000.ipynb
    |       Épocas=35, batch=1500.ipynb
    |       Épocas=35, batch=2000.ipynb
    |       Épocas=35, batch=500.ipynb
    |       Resumen experimentos realizados.ipynb
    |
    \---Optimización filtros y ventana convolución
            Filters=128, kernel_size=2.ipynb
            Filters=128, kernel_size=4.ipynb
            Filters=128, kernel_size=6.ipynb
            Filters=32, kernel_size=2.ipynb
            Filters=32, kernel_size=4.ipynb
            Filters=32, kernel_size=6.ipynb
            Filters=64, kernel_size=2.ipynb
            Filters=64, kernel_size=4.ipynb
            Filters=64, kernel_size=6.ipynb
            Resumen experimentos realizados.ipynb
```

Apéndice D

Acrónimos y siglas

- Adagrad: *Adaptative Gradient Algorithm* (Algoritmo de Gradiente Adaptativo).
- Adam: *Adaptative Moment Estimation* (Estimación del Momento Adaptativo).
- ADS-B: *Automatic Dependent Surveillance - Broadcast* (Transmisión de Vigilancia Dependiente Automática).
- ATM: *Air Traffic Management* (Gestión del Tráfico Aéreo).
- ATS: *Air Traffic Services* (Servicio de Tráfico Aéreo).
- CNN: *Convolutional Neural Networks* (Redes convolucionales).
- ConvComplejo: modelo convolucional complejo.
- ConvLSTM: uso combinado de redes convolucionales y redes LSTM.
- ConvSencillo: modelo convolucional sencillo.
- LSTM: *Long Short-Term Memory Networks* (Redes de memoria a largo plazo).
- MAE: *Mean Absolute Error* (Error Absoluto Medio).
- MSE: *Mean Squared Error* (Error Cuadrático Medio).
- ReLU: *Rectified Linear Unit* (Unidad lineal rectificada).
- RNN: *Recurrent Neural Networks* (Redes neuronales recurrentes).
- RTA: Remaining Time of Arrival (Tiempo restante de llegada).
- SACTA: Sistema Automatizado de Control de Tránsito Aéreo.
- SGD: *Stochastic Gradient Descent* (Descenso del Gradiente Estocástico).
- soloLSTM: modelo LSTM.
- SP: *Story Points* (Puntos de historia).