UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

PREDICCIÓN DEL PRECIO DE BILLETES DE AVIÓN A PARTIR DE UNA RED NEURONAL LSTM

David García Ballester

Tutora: Valery Naranjo Ornedo

Cotutor: Adrián Colomer Granero

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2019-20

Valencia, 24 de julio de 2018

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada

Agradecimientos

Agradecer a Valery y Adrián su ayuda y orientación durante todo el proceso, así como por brindarme la oportunidad de adentrarme en el mundo de la inteligencia artificial y aprendizaje automático, que me ha parecido fascinante.

Agradecer también a mis padres por todo el apoyo y comprensión que he recibido durante la carrera, sin ellos nada de esto hubiera sido posible.

Resumen

El presente TFG versa sobre la predicción de precios de vuelos mediante técnicas de aprendizaje profundo, más concretamente mediante una propuesta de arquitectura de red neuronal recurrente con unidades LSTM o *Long Short Term Memory* que pertenece a una de las sub-ramas de este campo. Dicha red neuronal busca ser capaz de predecir las fluctuaciones de precio que presentan los vuelos y comprobar si en efecto existe una correlación entre el precio y el tiempo.

Para conseguir el objetivo propuesto se ha recopilado información de manera fiable y eficiente durante más de un año, con diversas técnicas de *Web Scraping*, ya que sin datos con los que nutrir la red LSTM ésta no podrá tener un rendimiento y precisión adecuada en su predicción.

La elección de redes LSTM ha sido deliberada, y no trivial, ya que este tipo de redes se adapta muy satisfactoriamente a problemas con un horizonte temporal y logran retener mejor la información a largo plazo (además de emplear la información a corto plazo), de ahí su nombre, cosa que otras técnicas no consiguen con tanta eficacia y se limitan más a trabajar con la información que reciben únicamente a corto plazo. Además, se proponen distintos modelos LSTM, con ciertas variaciones, con el fin de compararlos y determinar cuál brinda una mejor predicción. Estos modelos son: LSTM convencional, LSTM bidireccional y LSTM convolucional.

Finalmente, se presentan los resultados con distintas métricas que ayudan a evaluar el rendimiento de cada uno de los modelos.

Palabras clave: *inteligencia artificial, aprendizaje automático, aprendizaje profundo, red neuronal recurrente, LSTM, precio, vuelo, clasificación, predicción, bidireccional, convolucional, entrenamiento, recopilación, Web Scraping.*

Resum

El present TFG versa sobre la predicció de preus de vols mitjançant tècniques d'aprenentatge profund, més concretament mitjançant una proposta d'arquitectura de xarxa neuronal recurrent amb unitats LSTM o *Long Short Term Memory* que pertany a una de les sub-branques d'aquest camp. Aquesta xarxa neuronal busca ser capaç de predir les fluctuacions de preu que presenten els vols i comprovar si en efecte hi ha una correlació entre el preu i el temps.

Per aconseguir l'objectiu proposat s'ha recopilat informació de manera fiable i eficient durant més d'un any, amb diverses tècniques de Web scraping, ja que sense dades amb què nodrir la xarxa LSTM aquesta no podrà tenir un rendiment i precisió adequada en la seua predicció .

L'elecció de xarxes LSTM ha estat deliberada, i no trivial, ja que aquest tipus de xarxes s'adapta molt satisfactòriament a problemes amb un horitzó temporal i aconsegueixen retenir millor la informació a llarg termini (a més d'emprar la informació a curt termini), de aquí el seu nom, cosa que altres tècniques no aconsegueixen amb tanta eficàcia i es limiten més a treballar amb la informació que reben únicament a curt termini. A més, es proposen diferents models LSTM, amb certes variacions, per tal de comparar-los i determinar quin ofereix una millor predicció. Aquests models són: LSTM convencional, LSTM bidireccional i LSTM convolucional.

Finalment, es presenten els resultats amb diferents mètriques que ajuden a avaluar el rendiment de cada un dels models.

Paraules clau: *intel·ligència artificial, aprenentatge automàtic, aprenentatge profund, xarxa neuronal recurrent, LSTM, preu, vol, classificació, predicció, bidireccional, convolucional, entrenament, recopilació, Web scraping.*

Abstract

The present TFG deals with the prediction of prices of flights by means of deep learning techniques, more specifically through a proposed recurrent neural network architecture with LSTM or *Long Short Term Memory* networks that belong to one of the sub-branches of this field. This neural network seeks to be able to predict the price fluctuations presented by flights and check whether there is indeed a correlation between price and time.

In order to achieve the proposed objective, information has been collected in a reliable and efficient way during more than a year, with different techniques of Web Scraping, since without data with which to feed the LSTM network it will not be able to have an adequate performance and precision in its prediction.

The election of LSTM networks has been deliberate, and not trivial, since this type of networks adapts very satisfactorily to problems with a temporary horizon and they manage to retain better the information in the long term (besides using the information in the short term), from there its name, something that other techniques don't achieve with so much efficiency and they are limited more to work with the information that they receive only in the short term. In addition, different LSTM models are proposed, with some variations, in order to compare them and determine which one provides a better prediction. These models are: vanilla LSTM, bidirectional LSTM and convolutional LSTM.

Finally, the results are presented with different metrics that help to evaluate the performance of each one of the models.

Keywords: *artificial intelligence, automatic learning, deep learning, recurrent neural network, SLMT, price, flight, classification, prediction, bidirectional, convolutional, training, collection, Web Scraping.*

Tabla de contenido

Índice de figuras

Índice de gráficas

Índice de tablas

1. Introducción

1.1 Motivación

Con la creciente demanda de vuelos y un sector aéreo que genera una cantidad de ingresos sin precedentes (en torno a 612 mil millones de euros a nivel mundial en 2019 [1]), no cabe duda de que el turismo está en auge. Esto ha despertado un gran interés en los clientes o compradores de vuelos de encontrar la oferta más barata y ha propiciado en una reestructuración y renovación total del sector de las agencias de viaje. Hoy en día, el usuario medio ya no acude presencialmente a una agencia de viajes convencional para programar sus vacaciones, sino que accede a cualquiera de las múltiples páginas web que compara los distintos vuelos que pueden ser de su interés.

La motivación y lo que se busca con este proyecto es ir un paso más allá. Proporcionarle a los clientes la opción de no solo saber cuanto cuesta un vuelo en el momento actual, sino además poder predecir, con cierto grado de confianza, cuál va a ser el precio de ese vuelo en un intervalo de tiempo futuro y de esta manera brindarle la posibilidad de que pueda decidir cuál es el momento que más le interesa a él para comprar.

La idea de dicho proyecto surgió tras identificar que gran cantidad de personas carecían de un claro entendimiento de cómo fluctúan los precios de los vuelos a lo largo del tiempo y de cómo aprovechar las temporadas e intervalos de tiempo en los que dichos precios van a estar más asequibles. Se quiso comprobar si en efecto el factor tiempo tiene una estricta correlación con el precio de un determinado vuelo.

Además, este trabajo ha permitido al alumno adentrarse en el mundo de la inteligencia artificial, que es algo que llevaba tiempo queriendo, ya que el análisis de dichos datos se lleva a cabo entrenando una red neuronal. El problema abordado en este proyecto se piensa que, a priori, es perfecto para modelar en una red neuronal, ya que estas son excepcionalmente buenas en detectar y replicar, en su caso, patrones en los datos. La inteligencia artificial es una de las tecnologías que más está creciendo en los últimos años (un estudio de *The Business Research Company* [2] pronostica que el mercado de IA experimentará un aumento del 43,39% hasta final de año, debido en gran parte a la automatización de tareas propiciada por el COVID-19), y se cree que el presente trabajo final de grado es la oportunidad perfecta para adentrarse en este campo.

Con este proyecto se pretende poder contestar a muchas de las preguntas que una persona puede llegar a tener a la hora de comprar un vuelo: *¿Es más conveniente comprar entre semana o en fin de semana? ¿Tengo que comprar el vuelo con mucha antelación o con poca? ¿Cuándo empezará a subir el precio de un vuelo?*

1.2 Objetivos

El presente trabajo fin de grado tiene como objetivo general la predicción del precio de billetes de avión a partir de una red neuronal LSTM. Durante la consecución de este surgen una serie de objetivos específicos que se detallan a continuación:

- Introducción y perfeccionamiento en el lenguaje de programación Python: la realización del siguiente trabajo se va a llevar a cabo en su totalidad, al menos la parte tecnológica, mediante el lenguaje de programación Python, más concretamente la versión *3.7* del mismo. Con él se realizarán las diversas tareas, desde la recopilación y procesamiento de datos, hasta la creación de una red neuronal y su posterior entrenamiento. Para ello es necesario conocer los conceptos básicos de Python y su funcionamiento, además de la integración y uso de las distintas librerías que están disponibles, lo cual facilitarán en gran medida el trabajo.

- Conocimiento general de inteligencia artificial y aprendizaje automático: para llevar a cabo el presente trabajo se creyó que era de vital importancia tener una buena base de la inteligencia artificial en general, y del aprendizaje automático en específico. Es por ello que se realizó, entre otras tareas, un curso de *Machine Learning* de la Universidad de Stanford, para entender como funcionaba todo el proceso de aprendizaje automático y los fundamentos teóricos involucrados, antes de ponerse a programar y modelar la red neuronal.

- Conocimiento específico del aprendizaje profundo, redes neuronales, redes recurrentes y redes LSTM: dentro de la inmensidad de la rama del aprendizaje automático se fue concretando cada vez más hasta llegar a las redes neuronales recurrentes con unidades LSTM, que se creyó que eran el perfecto método para la modelación del presente trabajo.

- Creación de una base de datos cuantiosa y fiable: para el presente trabajo resulta trascendental tener a nuestra disposición una gran cantidad de datos, y para su obtención se recurre al *Web Scraping*. Es por ello que se requiere un cierto conocimiento en este campo, y más teniendo en cuenta lo ya comentado, que la página a la que se quería acceder está escrita en JavaScript, por lo que un *scraper* sencillo y tradicional no bastaría.

1.3 Estado del arte

La inteligencia artificial (IA) es la simulación de inteligencia humana en máquinas, centrándose en el aprendizaje y entrenamiento para la posterior resolución de problemas, frecuentemente a gran escala y con grandes cantidades de datos, que incluso un humano tendría dificultad para resolver.

Los comienzos de la inteligencia artificial se remontan al año 1956, con la primera definición aceptada mundialmente de la IA, aunque para aquel entonces ya se había estado trabajando con ella desde hacía unos años.

Hoy en día contamos con diversas subcategorías, entre las cuales destaca el aprendizaje automático o *machine learning*, que es la más conocida y la rama en la que se centrará este proyecto.

La inteligencia artificial ha dado grandes saltos en los últimos años y se empieza a utilizar de manera cotidiana a omisión de la mayoría de la población. Tiene gran cantidad de usos, desde algo tan conocido y tangible como la conducción autónoma de vehículos de transporte o trascendental como la detección de enfermedades, a algo más imperceptible como ayuda en fotografía móvil o personalización de anuncios según los gustos del usuario.

Centrándonos más en el aprendizaje automático, que como ya he comentado es una de las ramas de la inteligencia artificial, Tomás José Fontalvo Herrera [3] relaciona su desarrollo con el de las PYMES. En su artículo *Aprendizaje automático y PYMES: Oportunidades para el mejoramiento del proceso de toma de decisiones* explica como el aprendizaje automático esta siendo utilizado por grandes empresas, pero su adopción está siendo mucho más paulatina en las pequeñas y medianas empresas, cosa que les pone en una posición incluso más desfavorable en relación a sus hermanas mayores. El autor identifica las innumerables ventajas que ofrece el machine learning y explica lo mucho que se podrían beneficiar las PYMES en caso de que éstas lo incorporaran en su modelo de negocio.

El aprendizaje profundo se emplea en una gran variedad de campos, desde el de la agricultura hasta el médico. El neuro-científico y biólogo singapurense Justin Ker [4] relata las diversas aplicaciones que tiene el aprendizaje automático en el ámbito médico, sobre todo en el de prevención y detección de tumores, en su artículo *Deep Learning Applications in Medical Image Analysis.* En él describe el increíble avance en los últimos años tanto del aprendizaje automático, como el acentuado aumento en el uso de registro médicos electrónicos y las imágenes de diagnóstico. La unión de ambas peculiaridades es una oportunidad increíble para la detección y diagnosis de enfermedades mediante imágenes e inteligencia artificial. Se tratan temas médicos como la clasificación médica de imágenes, segmentación, registro, detección y localización. También examina asuntos relacionados como los obstáculos que aparecen en la investigación, tendencias emergentes y futuras posibles direcciones.

Otra de las increíbles aplicaciones del aprendizaje profundo es su uso en el entrenamiento de robots diseñados para la búsqueda y rescate de humanos en entornos desconocidos. Farzad Niroui [5] explica en su artículo *Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments* la extraordinaria importancia que tiene el aprendizaje profundo para los robots de rescate. Los entornos en los que se deben mover dichos robots son la mayor parte de veces caóticos e impredecibles, por lo que gracias a esta herramienta son capaces de aprender y adaptarse a la situación específica a la que se encuentren. Gracias a esto, no solo consiguen adaptarse mejor a medida que pasa más tiempo, sino que también son capaces de cubrir más terreno con mayor rapidez, por lo que logran alcanzar antes a las personas en necesidad de ayuda, lo cual es crítico en este tipo de operaciones.

Como ya he comentado anteriormente, las grandes empresas parecen haber adoptado con los brazos abiertos el aprendizaje automático y profundo, y haberlo incorporado en el día a día de la empresa.

Twitter, por ejemplo, emplea el aprendizaje automático para evaluar y puntuar virtualmente todos los tweets publicados, y elige los más adecuados para enseñar y publicar a cada usuario en su *timeline.* Es decir, no solo evalúa y puntúa de manera objetiva cada tweet, si no que lo hace de manera subjetiva para cada usuario, mostrándole así lo que él cree que más le va a agradar o interesar. Esto sería imposible

de no ser por el aprendizaje automático. Shuai Zhang [6] explica en su artículo *Deep Learning Based Recommender System: A Survey and New Perspectives* justamente esto. El aprendizaje profundo es una herramienta excelente para filtrar contenido y mostrar al usuario solo las publicaciones que más le podrían interesar, y más teniendo en cuenta la gran cantidad de información que hay hoy en día en internet. Además, indica que estás clasificaciones van mejorando día a día cuanta más información ven, lo cual significa una gran oportunidad de prospecto de futuro a largo plazo.

El psicólogo Christoph Strauch [7] efectuó un estudio en 2019 dónde trataba de averiguar el grado de confianza que tienen las personas en la conducción autónoma y, por lo tanto indirectamente, en el aprendizaje automático. En su artículo *Real autonomous driving from a passenger's perspective: Two experimental investigations using gaze behaviour and trust ratings in field and simulator* explica los resultados que obtuvo y cómo realizó el estudio. El estudio quiso comprobar el grado de confianza que tenían distintas personas cuando les llevaba un coche mediante conducción asistida y lo hacía siguiendo la mirada de los ojos, observando por ejemplo cuantas veces la persona miraba fijamente el volante o la carretera, en comparación a cuando conducía un humano. Esto lo hizo variando el estilo de conducción, para que este no influyera en los resultados, y llegó a la conclusión de que las personas desconfían más en la conducción asistida que en la humana, incluso sabiendo que estadísticamente es mucho más segura la asistida.

Adentrándonos más en las tecnologías relativas al presente proyecto, Tiany Wang, profesor de la Universidad de Florida, expone un expermiento muy similar en el artículo *A Framework for Airfare Price Prediction: A Machine Learning Approach* [8]. En él describe la innegable correlación que existe entre el precio de un vuelo y diversos factores como la distancia que ha de recorrer, el momento de compra y el precio de la gasolina. Con ayuda de bases de datos públicas y teniendo en cuenta diversos indicadores macroeconómicos, ha sido capaz de predecir los precios con una precisión muy elevado, obteniendo resultados favorables, como un 0.869 en la métrica de R ajustado.

Juhar Ahmed Abdella, profesor en *College of Information Technology* (UAEU), presenta a su vez un estudio muy completo sobre la predicción y computación de precios de billetes de avión en su artículo *Airline ticket price and demand prediction: A survey* [9]. En él trata tanto la predicción de los vuelos desde el punto de vista del consumidor, para adquirir los vuelos en los mejores precios, como su computación por parte de las aerolíneas con objetivo de maximizar beneficio. Desde el punto de vista del consumidor sugiere dos modelos, uno que se centra en buscar el momento óptimo para comprar un vuelo y otro que busca el precio mínimo que tendrá un determinado vuelo. Explica también que su estudio se realiza teniendo en cuenta un número muy limitado de variables, como son el histórico de precios de los vuelos, día de compra de los vuelos y la salida de los mismos. Además, añade que la predicción mejoraría a gran escala de tener en cuenta otros factores, como las búsquedas en redes sociales.

El profesor de la Universidad de Washington Oren Etzioni fue un paso más allá y logró implementar un sistema totalmente funcional de predicción de precios que puso a prueba. Esto lo relata en su artículo *To buy or not to buy: mining airfare data to minimize ticket purchase price* [10]. Observó que las empresas cambiaban los precios de los vuelos basándose en algoritmos propios y quiso comprobar si recopilando suficiente información era capaz de predecir estos precios y recomendar a pasajeros que esperasen o comprasen cuanto antes un determinado vuelo. Tras minar los datos necesarios modeló un sistema de aprendizaje multidisciplinario, llamado por él Hamlet, que empleaba distintas técnicas como las series temporales (estadística), finanzas computacionales (aprendizaje por refuerzo) y aprendizaje automático convencional para afrontar el problema. Cada algoritmo se modificó y ajustó al problema y se unieron mediante técnicas de *stacking* para mejorar el rendimiento y precisión final. Realizó pruebas durante 41 días observando más de 12.000 vuelos y simuló 341 pasajeros, ahorrándoles un total de 198.074 dólares. Los resultados que obtuvo fueron prósperos, pues de saber con perfecta exactitud de la variación de los precios se podría haber ahorrado 320.572 dólares, con lo que obtuvo un rendimiento de más del 61%. Únicamente un 0,71% de las veces que *Hamlet* recomendaba esperar, el vuelo se agotaba y de media consiguió un 4,4% de ahorro a los pasajeros con respecto al precio que tenía en el momento de la predicción.

Diego Escobari, profesor del departamento de economía y finanzas de la Universidad de Téjas, realizó un estudio plasmado en su artículo *Estimating dynamic demand for airlines* [11] en el que intentó estimar la demanda dinámica de precios de determinados vuelos. Sus conclusiones tras el estudio se podrían resumir en dos. La primera es que los usuarios más importantes o *high-valuation consumers* tienden a comprar antes en el horizonte temporal, podría ser debido a que disponen de más capital y no valoran tanto la posibilidad de una rebaja futura. La segunda es que cuanto más se acerca la salida de un vuelo más consumidores activos hay interesados en dicho vuelo.

Por último, como aplicación real de un objetivo similar al que se quiere conseguir se tiene a *AirHint*. *AirHint* es una página web que intenta predecir, dentro de un intervalo de confianza, la probabilidad que hay de que un precio baje. Además, también le indican al cliente si es recomendable o no adquirir el vuelo en cuestión. En su página web no se indica en ninguna sección que para realizar las predicciones se estén empleando redes neuronales, aunque tampoco se especifica lo contrario. El único indicio que se da acerca de cómo se evalúan los precios es en la sección de *"Confiar o no confiar"* en el que se indica *"Proporcionamos recomendaciones estadísticas y hay un margen de error … por lo tanto, no es posible garantizar un 100% de precisión. Sin embargo, es posible dar pistas objetivas e informadas y esto es exactamente lo que ofrecemos"* [12]. Por lo tanto, se da a entender que el análisis y pronóstico que se realiza es, al menos en parte, llevado a cabo mediante modelos estadísticos.

2. Obtención de datos

La obtención de datos comenzó en junio de 2019, por lo que a día de entrega del presente trabajo se llevan descargando datos alrededor de un año.

La descarga que datos se realiza mediante un *Web Scraping* en Python, que accede a la página web de Ryanair y descarga los precios de determinados destinos del día actual a 150 días vista. Al tratarse dicha página de una web compleja, escrita parcialmente en JavaScript, un Web Scraper tradicional como *beautifulsoup4* no funciona, ya que este consigue captar únicamente la información escrita en html. Para descargar el resto del contenido, que en este caso era el que interesaba, se tuvo que emplear una librería llamada *Selenium Webdriver* que, mediante la ayuda de un *plug-in* llamado *chromedriver*, es capaz de simular un navegador virtual y así poder descargar la información incluida en el código escrito en JavaScript. La documentación oficial de *Selenium Webdriver* lo describe como "*un complemento que maneja un navegador de forma nativa, como lo haría un usuario, ya sea localmente o en una máquina remota que utiliza el servidor de Selenio, marcando un salto adelante en términos de automatización del navegador. Selenium Webdriver se refiere tanto a los enlaces de lenguaje como a las implementaciones del código de control del navegador individual. Esto se conoce comúnmente como sólo Webdriver*" [13].

El *scraping* es tanto vertical como horizontal. El *scraping* horizontal se realiza cambiando la url dinámicamente. Esto se hace de una forma muy sencilla. La url de los vuelos contiene en el propio link la fecha en la que saldrá el vuelo, por lo que seleccionando esta fecha dentro del link y modificándola, aumentando un día en cada iteración, se puede hacer un *scraping* de tantos días como se quiera, que en este caso son 150. En cuanto al *scraping* vertical, simplemente es dentro de una misma página buscar verticalmente los elementos que se quieren descargar. Explicado de manera más intuitiva (ver Figura 1), sería como ir moviéndose dentro de una matriz de 2 dimensiones con 2 *bucles for*, siendo el *eje x* las páginas de los vuelos y el *eje y* cada uno de los elementos dentro de esa página:

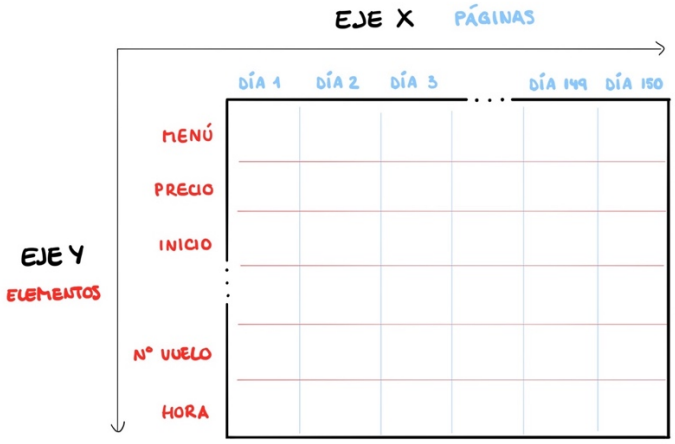


Figura 1: Matriz Web Scraping horizontal y vertical. Elaboración propia.

Una vez se ha "montado" la url con la fecha adecuada el script se asegura de que hay conexión a internet, y en caso de que así sea, se lanza la página web de dicho link. En caso de no haber conexión a internet se esperaría 60 segundos y volvería a intentarlo. Una vez dada la orden de abrir la página, el script está programado para esperar 15 segundos o a que cargue el contenido de la página, la primera condición que se cumpla. En caso de que pasaran 15 segundos y no cargara ningún contenido, bien sea por que la página web está teniendo problemas (podría estar bajo mantenimiento) o porque no exista un vuelo ese día, pasaría al siguiente día.

Para poder seleccionar los datos que se desea dentro de la página se filtrará por selector de CSS o Hoja de estilos en cascada. Como ejemplo (ver Figura 2), el precio tendrá un selector de CSS atribuido que será exclusivo a él y de saberlo se podrán encontrar los distintos precios que haya en la página. Para

poder encontrar dicho selector será tan fácil como entrar a una de las páginas desde el navegador Chrome, pinchar en uno de los precios con el click derecho y seleccionar *"inspeccionar elemento"*:

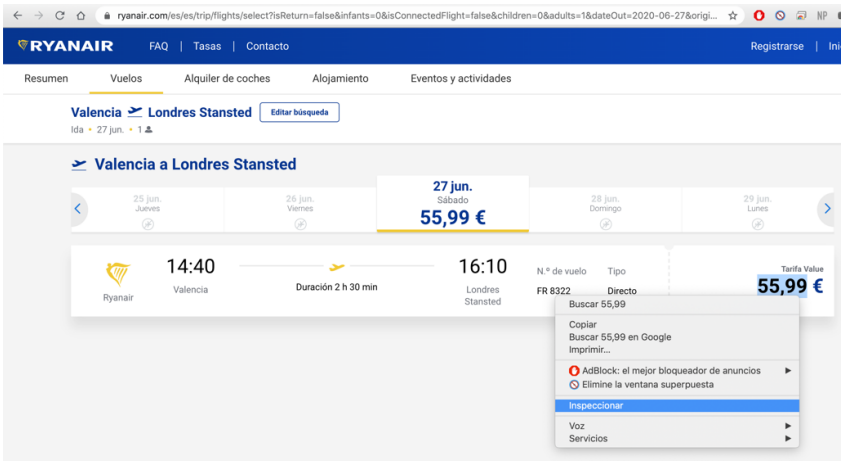


Figura 2: Selección elementos con inspeccionar elemento. Elaboración propia.

Una vez seleccionada esta opción se desplegará una ventana del lado derecho del navegador con una gran cantidad de información, como los elementos con los que cuenta la página, la estructura, estilos, propiedades, etc. Lo que se busca estará dentro del menú de elementos, que cuenta con el cuerpo completo de la página. Como la acción de inspeccionar elemento se ha efectuado estando seleccionado el precio del vuelo se abrirá la ventana de elementos por la parte del cuerpo en la que está dicho precio, por lo que es sumamente fácil de encontrar. Una vez encontrado se hace click derecho de nuevo y se selecciona *copiar*, y dentro de copiar se elige *copiar selector*, como se ve en la Figura 3:

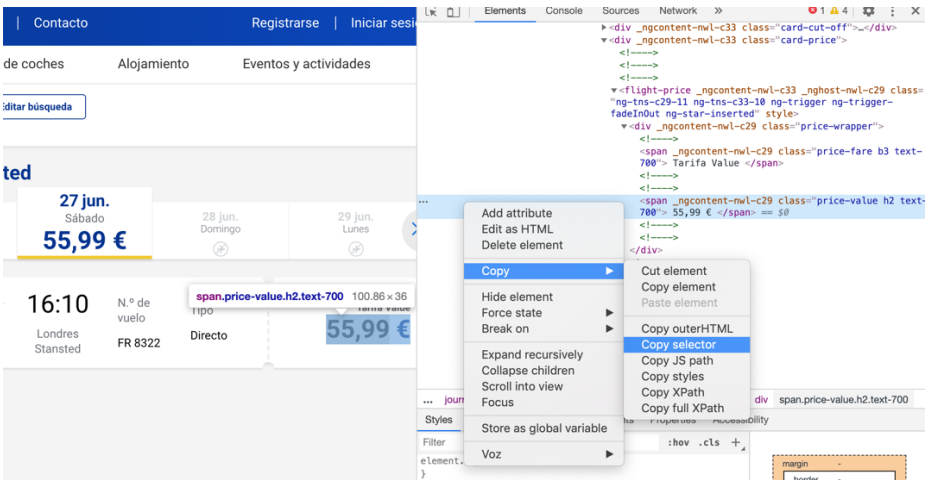


Figura 3: Copiar selector CSS. Elaboración propia.

Con la dirección de este selector de CSS copiado en el portapapeles se le puede indicar fácilmente al *scraper* el elemento que se quiere descargar. De este modo se seleccionan solo los datos que se quieren de la página, que para este caso sería el precio y el número del vuelo.

Tras haber descargado los datos, en ocasiones estos vienen con símbolos incrustados. Para limpiarlos y obtener únicamente los datos que queremos, se compara con una *lista blanca* (es un string que contiene las letras del abecedario, los números, el punto y la coma) y si hay algún carácter que no está en dicha lista se elimina, quedando únicamente los caracteres deseados.

Los datos que se obtienen son el destino, el día en el que sale el vuelo, el identificador de vuelo (para diferenciarlo en caso de que salga más de uno ese día) y su precio. Todo esto se guarda en un excel mediante la librería *openpyxl*, aunque más adelante se convierten a formato *csv* para facilitar la integración con la red neuronal. Una vez se ha descargado el precio, se utiliza la librería para comprobar si ya existe algún archivo excel asignado para ese vuelo. En caso de que sí exista lo carga y en caso de que no, lo crea. Una vez dentro del archivo excel, se incluye el destino, el identificador del vuelo y la fecha de despegue en la cabecera del archivo, y en las consiguientes celdas se incluye la fecha (columna A) y el precio (columna B) del día natural en el que se realiza el *scraping*.

Para intentar clasificar la recopilación de datos realizada vamos a centrarnos en un vuelo concreto (Tabla 1), el FR8324, que es un vuelo Valencia – Londres que sale diariamente.

El primer día que empezó la recopilación, digamos un día D, se obtuvieron 150 Excel para los vuelos que FR8324 que salían desde el día D hasta el día D+149. cada uno de estos Excel tenía un único dato, el recolectado ese día, es decir, el precio que tenían los vuelos FR8324 (del día D al D+149) si comprabas el día D.

Cada día, la base de datos se iba ampliando de dos formas:

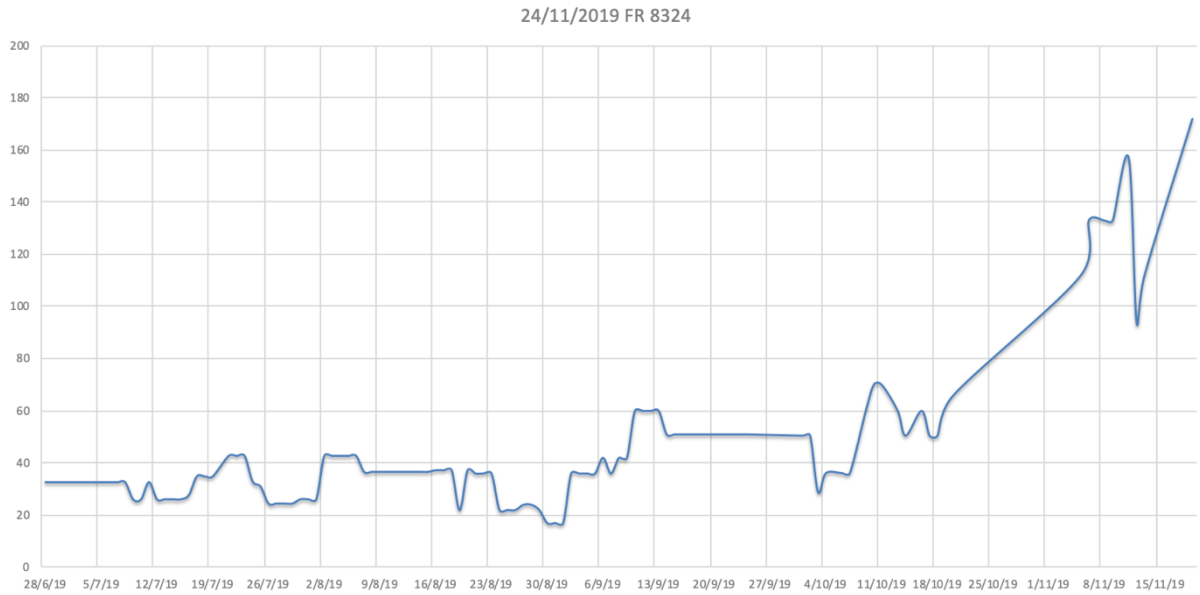
1. Añadiendo en cada excel el precio del vuelo del día "x" si se compra ese día.
2. Un nuevo excel con el vuelo que sal el día que entra en la ventana temporal de 150 días.

El ejemplo de la Tabla 1 muestra el excel obtenido a fecha de 26/07/2019 para el vuelo FR8324 que sale el día 24/11/2019 y que empezó a recolectarse 150 días antes de su salida, es decir, el 28/06/2019.

Londres	24/11/2019 FR 8324
28/06/2019	32,63
29/06/2019	32,63
30/06/2019	32,63
01/07/2019	32,63
02/07/2019	32,63
03/07/2019	32,63
04/07/2019	32,63
05/07/2019	32,63
06/07/2019	32,63
07/07/2019	32,63
08/07/2019	32,63
09/07/2019	26,1
10/07/2019	26,1
11/07/2019	32,63
12/07/2019	26,1
13/07/2019	26,1
14/07/2019	26,1
15/07/2019	26,1
16/07/2019	27,73
17/07/2019	34,87
18/07/2019	34,87
19/07/2019	34,87
21/07/2019	42,75
22/07/2019	42,75
23/07/2019	42,75
24/07/2019	32,94
25/07/2019	31
26/07/2019	24,47

Tabla 1: Precios vuelo día 24 de noviembre de 2019. Elaboración propia.

En la gráfica 1 se tiene este mismo vuelo. De esta manera se pueden apreciar mejor las fluctuaciones y la tendencia que sigue:



Gráfica 1: Precios para el vuelo FR 8324 del día 24/11/2019. Elaboración propia.

En la gráfica anterior se aprecia claramente una tendencia ascendente del precio a medida que se va acercando el día en el que está programado que despegue el vuelo, aunque este hecho se comentará en profundidad en apartados posteriores.

Adicionalmente, se implementó una función que, si todo el proceso iba bien y los datos se descargaban correctamente, enviaría un e-mail con el número de datos descargados y el tiempo total empleado para hacerlo (ver Figura 4). Esto, aunque a primera vista parece algo sin mucha importancia, resulta trascendental para el correcto funcionamiento a largo plazo de todo este sistema, ya que, en caso de no recibirse el mail, o recibirse con datos anómalos (como excesivo tiempo de descarga o escasez de datos descargados), permitiría inmediatamente darse cuenta de que algo no funciona bien y corregir el problema. Teniendo en cuenta que el proceso de descarga estaba automatizado, es decir, se realizaba de manera automática a una hora del día determinada, esto cobra aún más importancia.

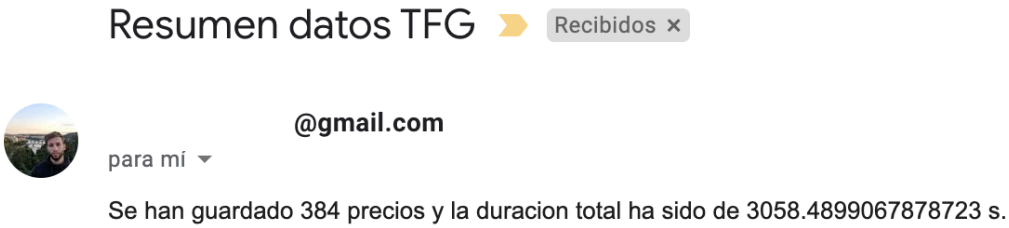


Figura 4: Mail con información de descarga de datos. Elaboración propia.

3. Marco conceptual

Para entender la tecnología que se ha elegido para este proyecto, se cree necesario un marco conceptual. Se empezará de afuera hacia dentro, es decir, de más general a más específico. Así se conseguirá detallar todos los campos y categorías que envuelven a la técnica empleada, lo cual facilitará al lector situarse y entender el por qué de esta decisión.

3.1 Inteligencia artificial

En 1956, John McCarthy [14], considerado uno de los fundadores de la inteligencia artificial, la definió como *"la ciencia e ingeniería de hacer máquinas que se comporten de una forma que llamaríamos inteligente si el humano tuviese ese comportamiento"*. Hoy en día se ha pasado a definir la inteligencia artificial como *"la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos. Estos procesos incluyen el aprendizaje, la adquisición de información y reglas para el uso de la información, el razonamiento usando las reglas para llegar a conclusiones aproximadas o definitivas y la autocorrección"* [15].

Lo que ambas definiciones tienen en común es que son bastante generales, si bien la segunda es algo más extensa, y por una buena razón. La inteligencia artificial es un campo inmenso, que contiene a día de hoy gran cantidad de subcampos, que a su vez tienen diversas ramificaciones, y no hace más que crecer.

Adicionalmente, la inteligencia artificial es, una de las ramas de las ciencias de la computación. Su objetivo es la creación de sistemas que puedan funcionar de manera independiente e inteligente y que intenten emular la inteligencia humana, que es capaz de aprender, mejorar y adaptarse a distintos entornos.

Sus aplicaciones son diversas y variadas, con lo cual resultaría prácticamente imposible enumerarlas todas, pero estas son algunas de las más importantes:

- Sector financiero: gran cantidad de entidades bancarias emplean la inteligencia artificial para multitud de funciones, entre las que se encuentran la gestión y adquisición de acciones, la organización de propiedades, la administración de operaciones, etc.

- Sector sanitario: la IA también es usada en este sector para automatizar varias de sus actividades. Destaca su uso para la prevención y detección de enfermedades. También se emplean sin operaciones, asistiendo a cirujanos para facilitar la intervención. Uno de los robots más conocidos en este campo es el *Da Vinci*.

- Atención al cliente: Los popularmente conocidos como *chatbots*, son asistentes virtuales, qué mediante diversas técnicas de inteligencia artificial, son capaces de asistir al usuario de manera automática. También se encuentran, por ejemplo, en contestadores de voz, detectando y reconociendo las palabras del cliente para así redirigirlo a la sección correspondiente.

- Sector de la música: en este sector también se está empezando a utilizar la inteligencia artificial para la edición y procesamiento automatizado de canciones. Además, se está avanzando bastante en tecnologías basadas en la inteligencia artificial que son capaces de crear canciones de cero.

- Sector de la industria: los robots potenciados por inteligencia artificial son comunes en este sector para tareas repetitivas, como por ejemplo el transporte de mercancías dentro de una

misma nave, trabajos peligrosos para humanos, control de calidad o que requieran fuerza sobrehumana.

- Sector automovilístico: muchas empresas del sector automovilístico están invirtiendo en el desarrollo de técnicas de conducción autónoma.

Para medir los avances de la IA y saber si en efecto era capaz de simular la inteligencia humana, o al menos parecerse a ella, se desarrolló un test. Dicho test es relativamente famoso y conocido por muchas personas, y es el Test de Turing [16].

Consiste en colocar a un usuario frente a distintas máquinas y hacer que converse con ellas. Algunas de estas máquinas están controladas por otros humanos y otras por la inteligencia artificial. El usuario que conversa con todas estas máquinas es consciente de que al menos una de ellas está siendo controlada por inteligencia artificial y su objetivo es descubrir cuál es. En caso de que el usuario no consiguiera detectar qué máquina estaba siendo controlada mediante la inteligencia artificial se diría que dicha máquina ha superado el Test de Turing.

Otra forma de describir este procedimiento sería como se puede observar en la figura 5. El test de Turing mide la capacidad que tiene un sistema de inteligencia artificial en emular comportamiento tanto humano como inteligente.

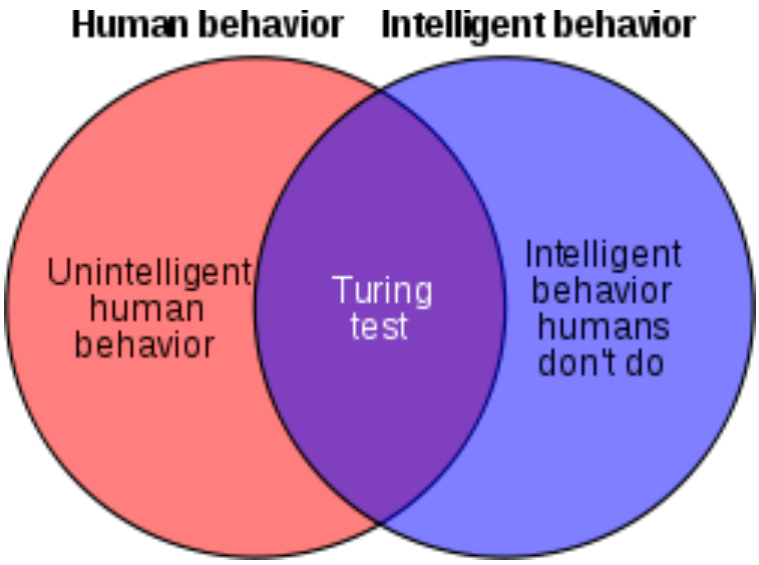


Figura 5: Comportamiento en el test de Turing [17]

3.2 Aprendizaje automático

Fernando Sancho Caparrini, profesor de la Universidad de Sevilla, define el aprendizaje automático como *"la rama de la Inteligencia Artificial que tiene como objetivo desarrollar técnicas que permitan a las computadoras aprender. De forma más concreta, se trata de crear algoritmos capaces de generalizar comportamientos y reconocer patrones a partir de una información suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento, es decir, un método que permite obtener por generalización un enunciado general a partir de enunciados que describen casos particulares"* [18]. Viendo esta definición, queda claro que el aprendizaje automático es la ciencia, dentro de la inteligencia artificial, que emplea datos para aprender y ser capaz de detectar patrones y

realizar predicciones o, dicho en otras palabras, es capaz de en base a modelos pasados predecir información futura, que es justamente lo que se busca para este proyecto.

El aprendizaje automático surgió en 1959 de la mano de Arthur Samuel, un programador y diseñador de videojuegos en la gigante americana *IBM*, y se empleó inicialmente para la clasificación de patrones. Hoy en día su uso se ha desplegado a diversas actividades, tales como: detección y filtrado de SPAM, personalización y depuración de contenido en redes sociales, video vigilancia, asistentes personales (Siri, Google Home, Alexa, etc.), predicción de tráfico y asistencia en los desplazamientos (p. ej. Google Maps), atención al cliente online, detección de fraudes online y personalización de publicidad.

Como se puede observar, sus aplicaciones son muy extensas y variadas, pero todas tienen algo en común, emplean la información de los usuarios y aprenden de ella para poder así ofrecer la mejor experiencia posible de manera personalizada.

En el aprendizaje automático, las dos ramas más ampliamente conocidas son el aprendizaje supervisado y aprendizaje no supervisado.

El aprendizaje automático supervisado y no supervisado se asemejan bastante. La diferencia más sustancial es que en el aprendizaje supervisado los datos que se le proporcionan a la red para su aprendizaje están etiquetados y clasificados, de manera que el sistema puede saber a posteriori cual es la clasificación correcta y, en base a esto, optimizando para una determinada función como por ejemplo MSE o *mean squared error,* corrige su algoritmo actualizando los pesos de su ecuación. En el aprendizaje no supervisado esta clasificación correcta y etiquetada de los datos no se tiene, por lo que la red ha de averiguar y saber distinguir de manera individual cual es la correcta clasificación y debe intentar aprender los patrones inherentes que presentan los datos.

Dentro de las distintas ramas del aprendizaje automático, cabe destacar el aprendizaje profundo, que es el campo del aprendizaje automático que se empleará para la resolución de este proyecto, y se explicará a continuación.

3.3 Aprendizaje profundo

El aprendizaje profundo, como ya se ha comentado es uno de los subcampos del aprendizaje automático. La figura 6 representa de una manera bastante intuitiva lo que hasta ahora se ha ido explicando:

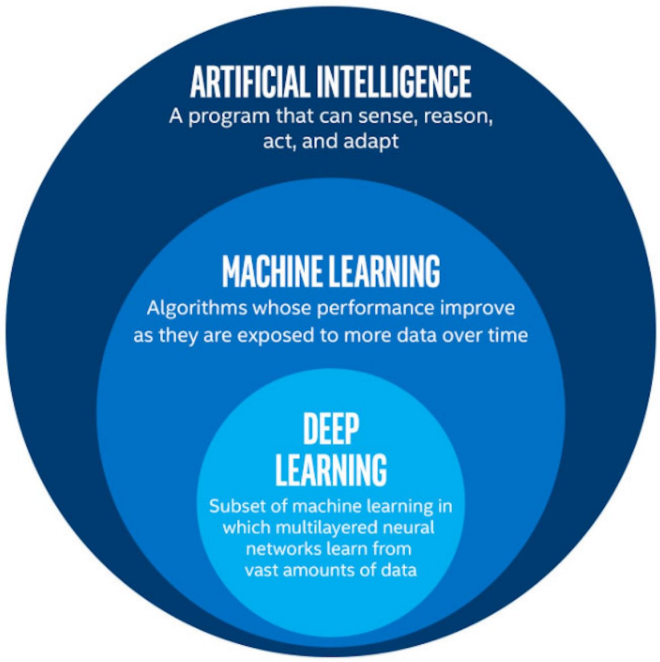


Figura 6: Explicación visual de aprendizaje profundo, aprendizaje automático e inteligencia artificial [19]

Como se puede observar en la figura 6, el aprendizaje automático clásico o *hand-crafted learning* abarca en su totalidad el aprendizaje profundo, siendo este a su vez una de las ramas de la inteligencia artificial. La IA se define como algo más genérico, un programa que tenga en cierta medida capacidad de raciocinio, mientras que el aprendizaje automático, y consecuentemente el aprendizaje profundo, se especifican más en la ciencia de predicción mediante algoritmos que mejoran y aprenden tras ser expuestos a grandes cantidades de datos.

Las diferencias más notables entre el aprendizaje automático clásico y el aprendizaje profundo son las siguientes:

- El aprendizaje automático clásico puede funcionar con muchos menos datos que el profundo.

- El tiempo de entrenamiento necesario es más elevado en el aprendizaje profundo que en el automático. En general, el proceso de entrenamiento y predicción en el aprendizaje automático se lleva a cabo mediante el procesador (CPU), mientras que en el profundo se hace mayoritariamente con la tarjeta gráfica (GPU), por lo que es importante tener una tarjeta gráfica relativamente potente. Todos los ordenadores tienen procesadores, aunque no todos tienen una tarjeta gráfica dedicada, por lo que para poder llevar acabo predicciones con aprendizaje profundo muchos optan por realizar este proceso remotamente, es decir, computarlo en algún servidor como por ejemplo en Colab de Google.

- En el aprendizaje automático la etapa de extracción de características ha de estar explícitamente señalada por un humano y codificada, mientras que en el aprendizaje profundo esta etapa de extracción la hace automáticamente la red.

Quizá a priori no quede muy clara la distinción entre el aprendizaje automático y el profundo, pero con el sencillo ejemplo de la figura 7 se puede entender mucho mejor:

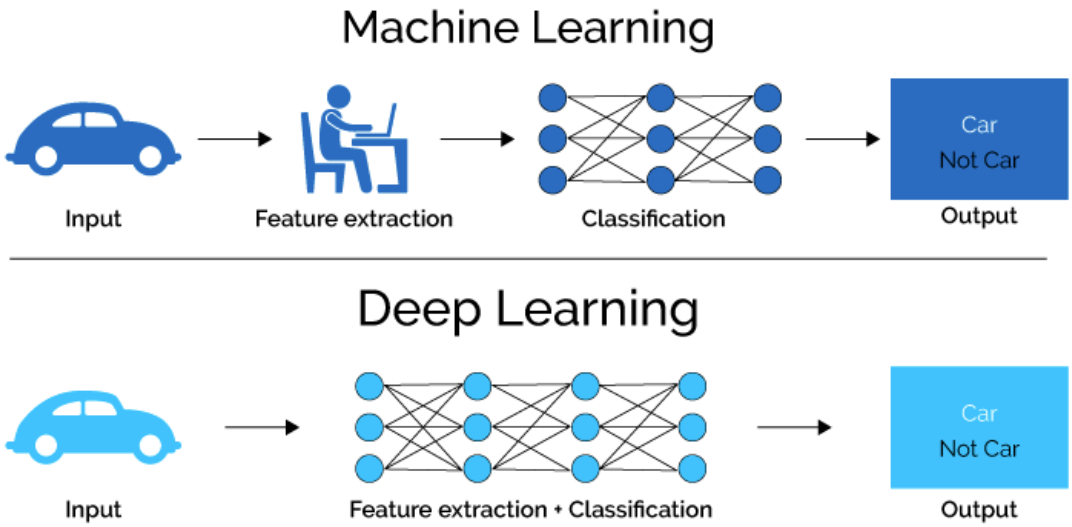


Figura 7: Diferencia entre aprendizaje profundo y automático [20]

Como bien se puede observar, en este caso la red ha de clasificar una serie de imágenes como coche o no coche, y para ello tiene distintas imágenes de coches. Pues bien, en el caso del aprendizaje automático un humando tendría que indicar de manera manual cuales son las características relevantes, que para este ejemplo serían las partes del coche, como las ruedas, ventanas, faros, etc. En el caso del aprendizaje profundo, la propia red sería capaz de hacer esta distinción y podría clasificar estas imágenes sin la necesidad de una introducción manual en el código de distintos parámetros que cuantifiquen estas características.

3.4 Redes neuronales artificiales o RNAs

M. H. Hassoun define las redes neuronales artificiales en su libro *Fundamentals of Artificial Neural Networks* como *"sistemas inspirados en la computación distribuida y paralela en el cerebro que le permite tener tanto éxito en tareas complejas de control y reconocimiento. La red neuronal biológica que logra esto puede ser modelada matemáticamentecaricaturizada por un gráfico ponderado y dirigido de nodos (neuronas) altamente interconectados. Los nodos artificiales son casi siempre simples funciones trascendentales cuyos argumentos son la suma ponderada de las entradas del nodo."* [21].

Las redes neuronales tienen siempre una capa de entrada y una capa de salida, y en la mayoría de los casos una o varias capas internas llamadas capas ocultas. En la figura 8 se ve de manera muy gráfica e intuitiva la organización de una RNA sencilla:

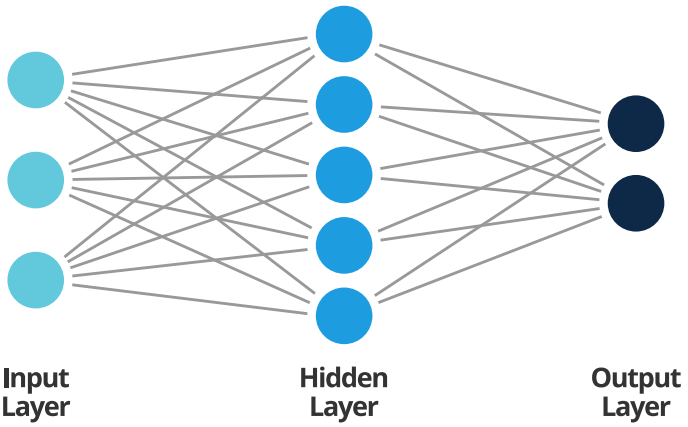
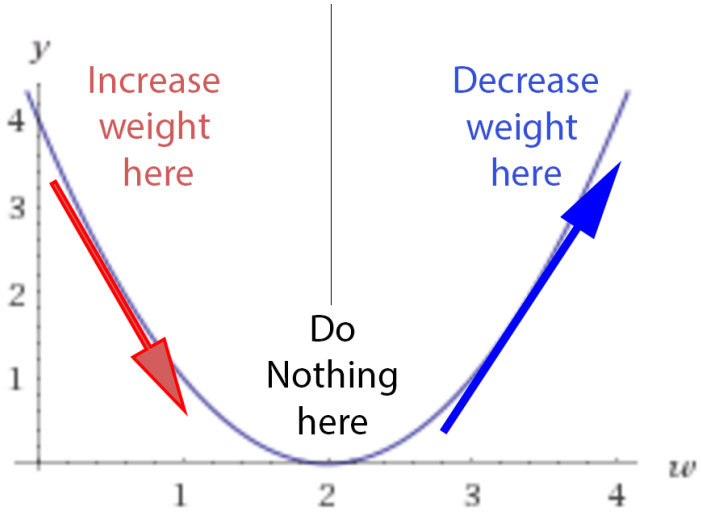


Figura 8: Forma de red neuronal artificial [22]

Cada circulo representa una neurona y está conectada a las subsiguientes neuronas mediante enlaces. Dichos enlaces tienen un peso atribuido que multiplica el valor de salida de la neurona, dando más o menos valor a la activación de las neuronas subsecuentes. A grandes rasgos, le red va jugando con estos pesos de manera automática y sin ser manualmente controlada por un ser humano, para poder obtener la mejor predicción posible, modificando dichos pesos para optimizar la función de pérdida (diferencia entre la predicción y la etiqueta). Gráficamente, una función pérdida podría ser como la gráfica 2:



Gráfica 2: Función de pérdidas en función del valor del peso [23]

En el eje *y* se representa el coste de la función y en el eje x el valor de un determinado peso. Como se puede observar existe un mínimo global cuando el valor del peso es 2, por lo que la red buscaría este punto. Mediante el gradiente de la función con respecto a los pesos la red sabría de manera autónoma en que sentido modificar el valor del peso para acercarse cada vez más a este mínimo.

Para calcular el gradiente del coste de la función, las redes neuronales emplean un método llamado *back-propagation* o retro-propagación en español, que consiste en tras haber realizado un proceso de *forward-propagation* (o en otras palabras, tras haber realizado un proceso de predicción), se calcula el error entre la predicción y la etiqueta y como bien indica el nombre se propaga de atrás a adelante, empezando por la salida, y se le indica a cada neurona individual el porcentaje de error que ha contribuido, con la idea de que de esta manera se corrija adecuadamente para minimizar dicho error. Visualmente (figura 9) el proceso se vería de la siguiente manera:

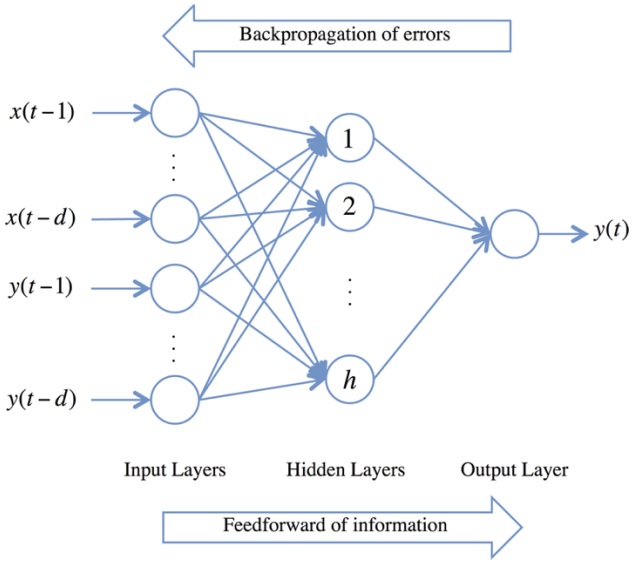


Figura 9: Propagación hacia adelante y hacia atrás [24]

Además de las operaciones lineales que se hace en cada neurona, empleando funciones de activación, como la activación *sigmoid*, pueden tratar con funciones no lineales obteniendo siempre algún tipo de relación entre la entrada y la salida.

3.5 Red neuronal recurrente

Hasta este momento todas las ejemplificaciones se han llevado a cabo mediante una red neuronal artificial tradicional o simple. Para el estudio de este proyecto se va a trabajar con un tipo específico de redes neuronales que permitirán obtener una mejor predicción, las redes neuronales recurrentes.

DeepAI define las redes recurrentes como *"un tipo de red neuronal que contiene bucles, lo que permite almacenar información dentro de la red. Las Redes Neuronales Recurrentes utilizan su razonamiento de experiencias anteriores para informar sobre los próximos eventos. Los modelos recurrentes son valiosos por su capacidad de secuenciar vectores, lo que abre la API para realizar tareas más complicadas"* [25].

Con esta definición queda bastante claro que estas redes emplean retroalimentación para aprender en el horizonte temporal de los datos pasados. Son capaces de mantener información entre épocas (la época es un parámetro que define el número de veces que el algoritmo de aprendizaje funcionará a través de todo el conjunto de datos de entrenamiento), lo cual permite una mejor predicción si los datos están ordenados de manera temporal.

Esto es justamente lo que se busca para este modelo, ya que los datos que se disponen están correlados temporalmente.

Una de las redes recurrentes más importantes, y la que se va a usar para la resolución de este proyecto, es la LSTM o *Long Short Term Memory*.

3.6 LSTM o Long Short Term Memory

Como ya se ha indicado anteriormente, las redes LSTM son un tipo de redes neuronales recurrentes. La diferencia más importante entre una red LSTM y una RNR tradicional la explica Felix A. Gers [26] en su libro *Learning to Forget: Continual Prediction with LSTM.* En él postula la siguiente idea: *"Las RNR estándar no aprenden en presencia de desfases temporales superiores a 5 - 10 pasos de tiempo*

discretos entre los eventos de entrada relevantes y las señales del objetivo. El problema del error que desaparece arroja dudas sobre si los RNR estándar pueden efectivamente exhibir ventajas prácticas significativas en comparación con las redes de alimentación basadas en ventanas de tiempo. Un modelo reciente, LSTM, no se ve afectado por este problema. La LSTM puede aprender a superar retrasos mínimos de más de 1000 pasos de tiempo discreto, forzando un flujo de error constante a través de "carruseles de error constante" dentro de unidades especiales, llamadas células." A la vista de esto, queda claro que las redes neuronales recurrentes tienen claras limitaciones a la hora de trabajar con horizontes temporales muy extensos, cosa que las redes LSTM buscan remediar.

Alex Graves [27] explica esto mismo también en su artículo *Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures,* indicando que *"La arquitectura LSTM fue motivada por un análisis del flujo de errores en los RNNs existentes que encontró que los retrasos de largo plazo eran inaccesibles para las arquitecturas existentes, porque el error retropropagado o bien explota o decae exponencialmente. Una capa de LSTM consiste en un conjunto de bloques conectados recurrentemente, conocidos como bloques de memoria. Estos bloques pueden ser pensados como una versión diferenciable de los chips de memoria en una computadora digital. Cada uno de ellos contiene una o más células de memoria conectadas de forma recurrente y tres unidades multiplicadoras - las puertas de entrada, salida y olvido - que proporcionan análogos continuos de operaciones de escritura, lectura y restablecimiento de las células."*

Es por esto por lo que se creyó conveniente emplear redes LSTM, pues el horizonte temporal de los datos que se tienen, si bien no es extremadamente extenso, si que supera los 100 días por set de datos.

Las redes recurrentes convencionales tienen la siguiente forma (figura 10):

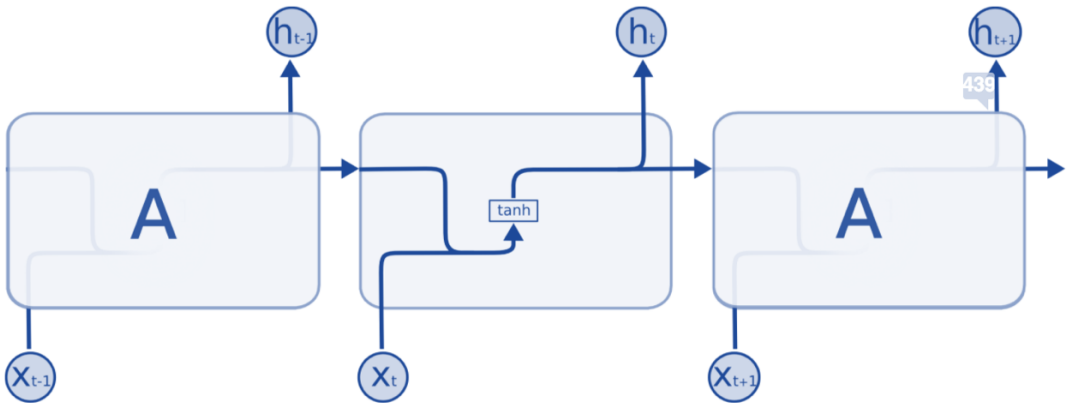


Figura 10: Red recurrente convencional [28]

Aquí se ve la estructura de "cadena" que tienen las redes recurrentes, que les permiten emplear información pasada para realizar mejores predicciones futuras. Como se puede observar tiene una estructura muy sencilla con una única capa (la de la tangente). Esta estructura se repite en cada uno de los componentes de la cadena, como se ve en la figura con los elementos señalados con la letra "A". Tras realizar la predicción, le pasan además información a la siguiente iteración. Como ya se ha comentado anteriormente este tipo de redes realizan buenas predicciones teniendo en cuenta información a corto plazo, pero si se requiere mantener la información a largo plazo resultan inadecuadas. Para ello se desarrollaron las LSTM, que son excelentes para este tipo de problemas, de ahí su nombre.

Una red LSTM *vanilla,* o tradicional, presenta la estructura interna que se puede encontrar en la figura 11:

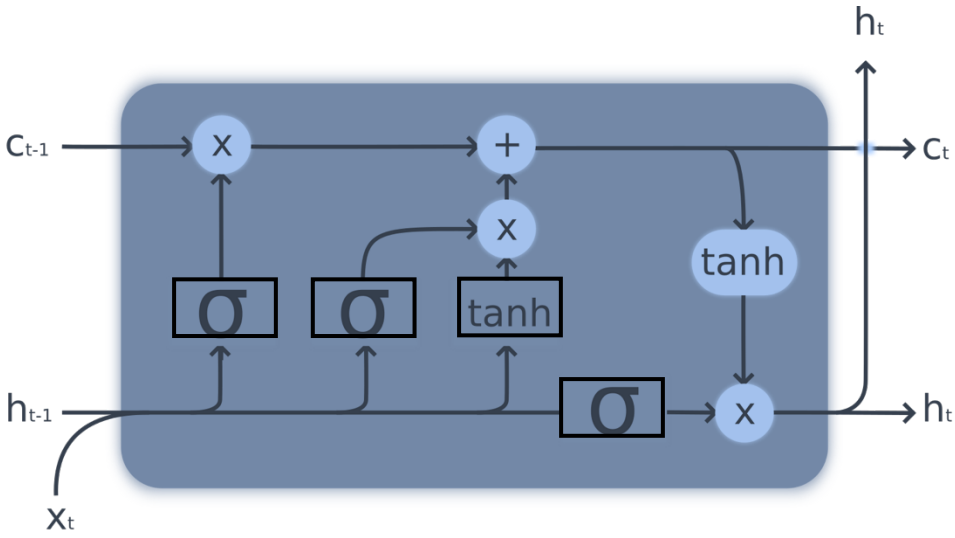


Figura 11: Interior capa LSTM [28]

La información fluye por dentro de las LSTM por la línea horizontal que atraviesa el sistema en lo alto de la imagen, que entra en el sistema como C_{t-1} y sale como C_t, también llamada célula. Dicha información es modificada y alterada mediante las distintas puertas que desembocan en esta línea principal. A su vez, x_t serían los valores de entrada, mientras que h_t representaría la predicción de la red en el momento "t".

Las figuras con forma redonda son operaciones que se efectúan sobre la información, mientras que las de forma rectangular son capas en la red. Las figuras con el símbolo σ son capas sigmoide, es decir, se basan en la función sigmoide:

$$y = 1⁄1 + e⁄ - x⁄⁄$$

Esta función puede tener como resultado cualquier número entre 0 y uno. Es de interés, ya que con ella las capas de las que se ha hablado anteriormente pueden decidir la cantidad de información que se quiere incluir, siendo en valores porcentuales 0% si la función sigmoid da como resultado 0, o un 100% si la función sigmoid da como resultado 1.

En las redes LSTM también se sigue en patrón del resto de las redes recurrentes, presentando la forma de "cadena" que se ha hablado anteriormente, realizando los cálculos en cada iteración y proporcionando la información a la siguiente iteración. Esto se observa bien la figura 12:

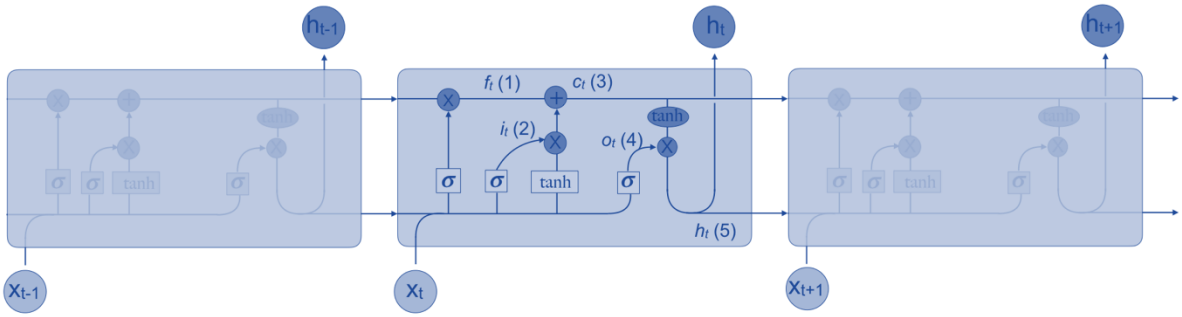


Figura 12: Cadena LSTM convencional [28]

Como se puede observar, cada una de estas iteraciones presenta la estructura interna de las redes LSTM que ya hemos comentado, con las 3 capas sigmoide, la capa tangente y las diversas operaciones que tienen lugar a lo largo de la línea principal o célula.

Para entender mejor y con más profundidad lo que ocurre en la red LSTM en cada iteración se comentará por partes su recorrido.

La primera parte es la que se encarga de concretar que información pasada se empleará. Se le conoce también como la *puerta del olvido (véase figura 13)*:

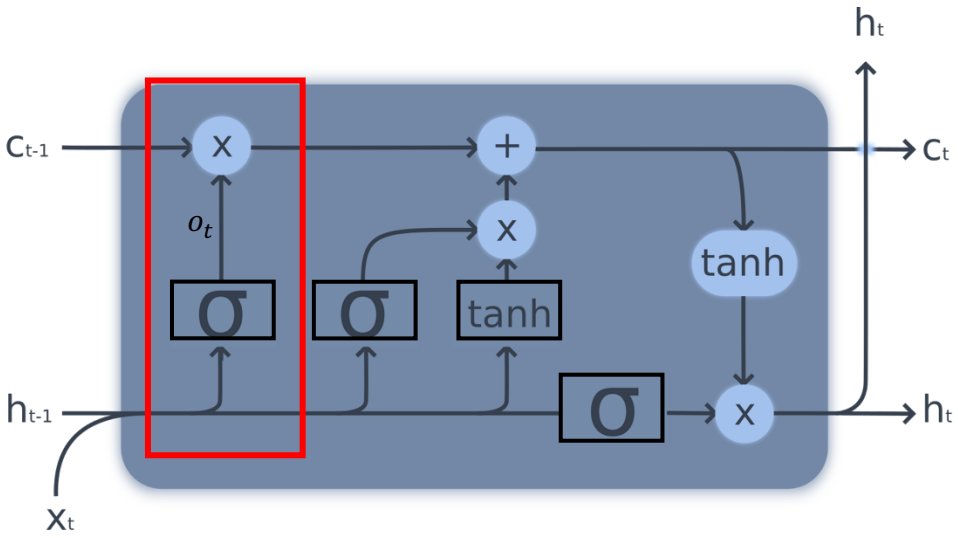


Figura 13: Puerta del olvido LSTM [28]

En la imagen se puede observar que elementos constituyen la puerta del olvido. En ella se observa que la información atraviesa una capa sigmoide, que como ya se ha comentado anteriormente, dará un valor entre 0 y 1 que determinará la cantidad de información que atravesará dicha puerta. La ecuación que proporcionaría dichos resultados sería:

$$o_t = \sigma * (W_f \cdot h_{t-1}, x_t + b_f)$$

El siguiente paso cuenta con 2 partes (figura 14):

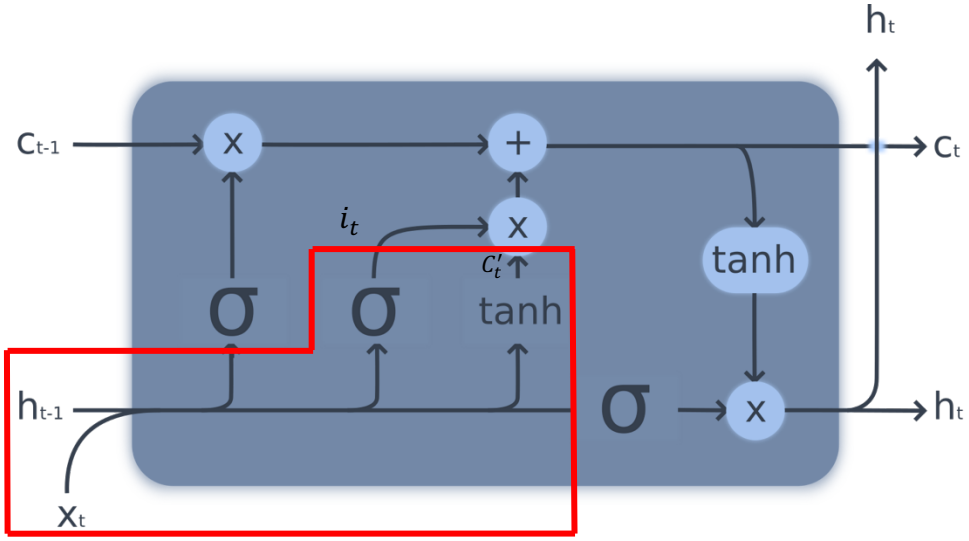


Figura 14: Puerta capa de entrada LSTM [28]

La primera de ellas cuenta con una capa sigmoide como en el anterior paso, solo que esta vez esta función sigmoide elige que valores se van a actualizar. Esta capa también es conocida como la puerta de capa de entrada. La segunda de estas dos partes cuenta con una capa tangente que crea un vector con las posibles entradas que se podrían añadir. Las ecuaciones para estas 2 partes son las siguientes:

$$i[t] = \sigma * \ W[i] \cdot \ h[t-1], x[t]] + \ b[i]$$

$$C[t]' = \tanh[] * (W[C] \cdot [\ h[t-1], x[t]] + \ b[C]$$

La siguiente parte (véase figura 15) busca actualizar C_{t-1}, con los valores y funciones que se han calculado en los anteriores pasos, convirtiéndolo en C_t:

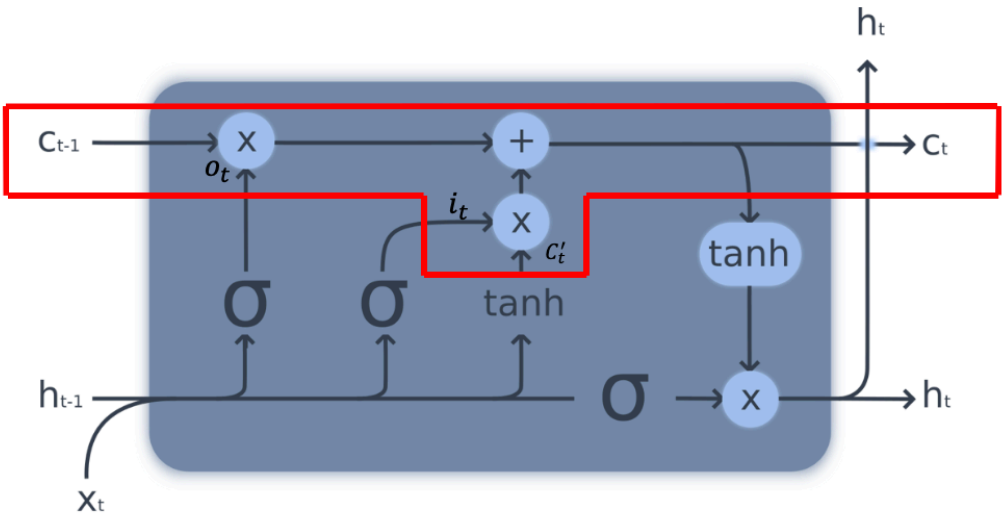


Figura 15: Capa actualización célula LSTM [28]

Como se ve en la imagen, se multiplica C_{t-1} por ot. Simultáneamente, se multiplican $C[t]'$ e $i[t]$, que son los nuevos valores que se desean incluir en la célula y sus ponderaciones (cuánto se desean actualizar los valores) respectivamente. Seguidamente se suma esta multiplicación al resultado de la multiplicación de C_{t-1} por ot y se obtiene C_t. La ecuación de este proceso sería:

$$C[t] = \ o[t] * \ C[t-1] + \ i[t] * \ C[t]'$$

La última parte (figura 16) del proceso es la de obtener los valores de salida de la célula. Para ello se van a emplear algunas capas que modificaran y filtrarán dichos valores:

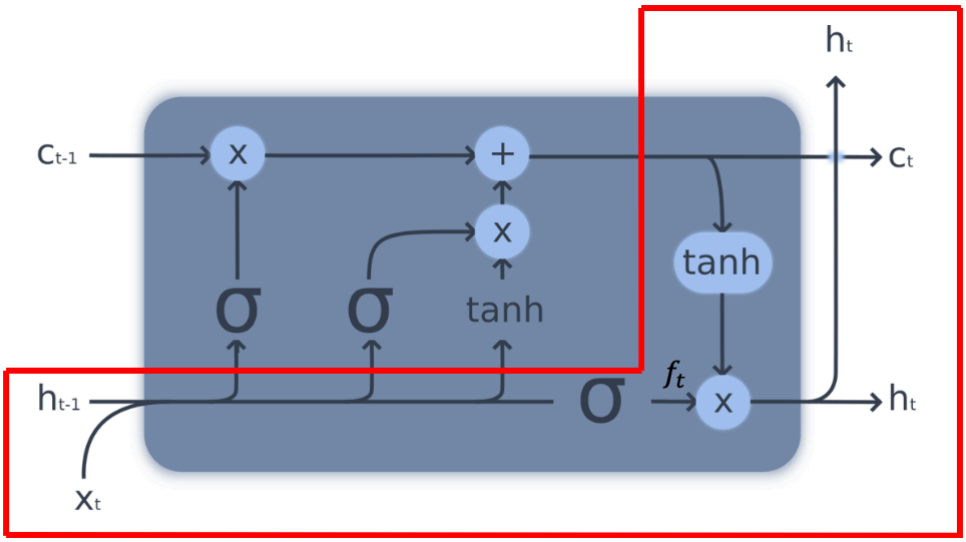


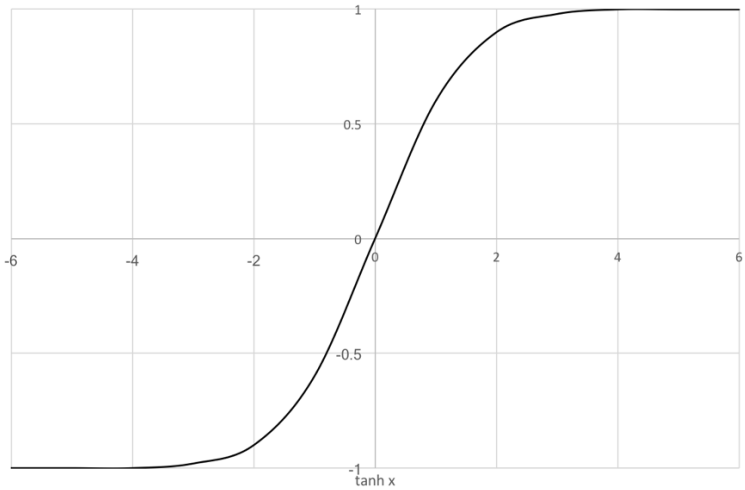
Figura 16: Obtención valores de salida LSTM [28]

En la imagen se observa como parte de C_t se ramifica y entra en una capa de tangente hiperbólica. Esto lo que hará es situar los valores entre 1 y -1. Seguidamente se multiplicará por el resultado de otra capa sigmoide que será la encargada de seleccionar únicamente los valores deseados de salida. Las ecuaciones en este paso serían:

$$f(t) = \sigma * \ W(f) \cdot \ h(t-1), x(t)) + \ b(f))$$

$$h(t) = \ f(t) * \tanh(C(t))$$

La función tanh está centrada en 0 y su forma se puede observar en la gráfica 3:

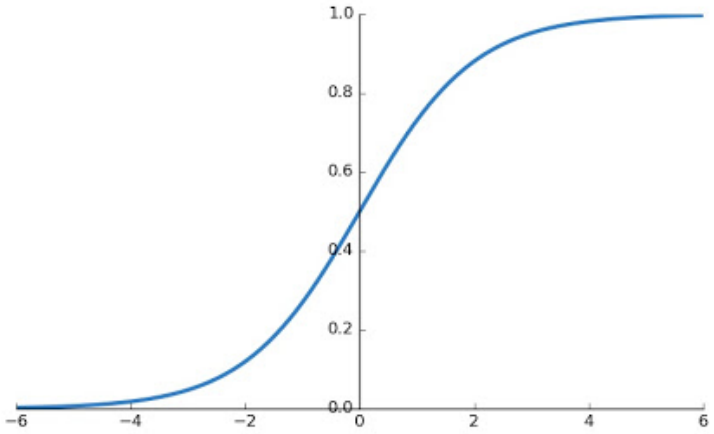


Gráfica 3: Función tanh. Elaboración propia.

Como se puede observar en la gráfica, toma valores entre -1 y 1 cortando los ejes en el eje de coordenadas. Su ecuación es:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

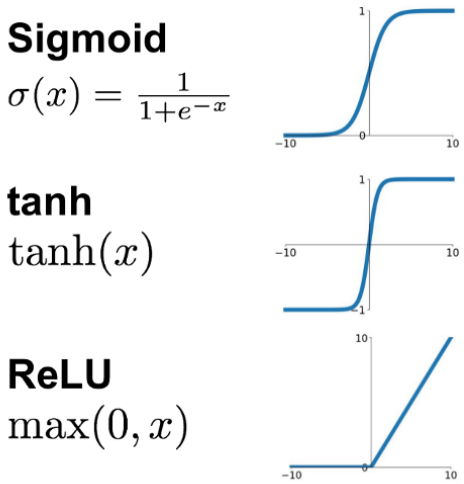
Por otro lado, la función sigmoide (véase gráfica 4) tiene la siguiente forma:



Gráfica 4: Función sigmoide. [29]

Gráficamente es muy parecida a la tanh, aunque en este caso corta el eje de ordenadas en 0,5.

Cuando se habla de la función de activación, las funciones más comunes para este tipo de red son la activación sigmoid, tanh y ReLU. Las funciones de sigmoid y tanh ya han sido comentadas con anterioridad, pero la ReLU o *Rectified Linear Activation Function* aún no se ha visto. En la gráfica 5 se tiene una comparación de la forma que toman estas tres funciones:



Gráfica 5: Comparación funciones sigmoide, tanh y ReLU [30]

Como se puede observar, la función ReLU coloca los *inputs* negativos a 0, mientras que devuelvo los inputs positivos de manera lineal. Esto podría acelerar en gran medida el proceso para modelos muy complicados y con gran cantidad de información de entrada, en especial el *back-propagation* que necesita de una función lineal para funcionar, y no compromete su eficacia.

El hecho de emplear la función ReLU frente a la sigmoide o tanh busca corregir 2 de las grandes limitaciones que presentan estas funciones en activación para modelos complejos:

- Saturación: estas 2 funciones presentan problemas de saturación. Esto quiere decir que cuando hay valores más altos, estos tienden a saltar a 1, y cuando hay valores más bajos estos saltan a 0 o -1, para la sigmoide y tanh respectivamente.

- Son solo sensibles a cambios cuando el valor de entrada se sitúa alrededor del centro de la función, que sería 0.5 para la sigmoide y 0 para tanh.

Por último, otro punto importante a tener en cuenta en este tipo de redes es el de *overfitting* o sobreentrenamiento. El *overfitting* sucede cuando una red tiene demasiadas capas, neuronas o recursos para modificar su algoritmo de manera muy específica y, por lo tanto, en la fase de entrenamiento, mejora tanto su modelo para los datos de entrenamiento que deja de poder generalizar bien en la fase de test, o para otros datos que no ha visto aun.

4. Materiales y metodología

4.1 Materiales

Los materiales que se han empleado durante el transcurso del trabajo son varios y muy dispares. Para intentar clasificarlos de la mejor manera posible se dividirán en materiales dedicados a la recopilación de información y a la predicción de precios mediante redes LSTM.

4.1.1 Descarga de datos

El proceso de descarga de datos no supone una carga computacional alta para el dispositivo con el que se efectúe, por lo que con un ordenador de características técnicas medias serviría. Para este caso, se ha realizado la recopilación de información con un *Mac Mini* de 2012, con *macOS 10.14.16 Mojave*, que cuenta con un procesador i5, 8 GB de memoria DDR3 a 1.600 MHz y un disco duro SSD Samsung 860 de 1 TB.

Se ha elegido este dispositivo, puesto que su sistema operativo facilita en gran medida el proceso de automatización de la descarga de los datos. De manera nativa tiene la opción de encenderse y apagarse a horas determinadas y esto se puede elegir en los ajustes del sistema, véase figura 17, lo cual es extremadamente intuitivo.

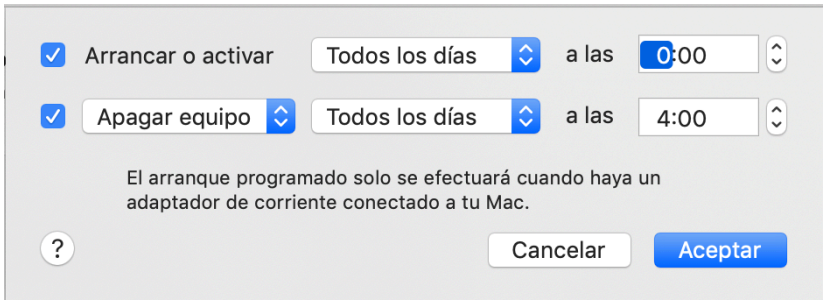


Figura 17: Automatización encendido y apagado ordenador. Elaboración propia.

De haberse elegido un PC se habría tenido que buscar en los ajustes la configuración adecuada de la *BIOS* para poder encender y apagar el ordenador a una hora determinada, y esto es en caso de que la *BIOS* con la que cuenta el ordenador tenga esta opción implementada, ya que la *BIOS* depende de la placa base del ordenador y por lo tanto varía de un PC a otro.

Una vez encendido el ordenador a la hora programada se emplea la aplicación *automator*, programa que viene de forma nativa también en *macOS*, que sirve para automatizar tareas. En este caso se ha creado un script ejecutable de Python que será el encargado de descargar todos los datos, y automator lo ejecutará 5 minutos tras haberse encendido el ordenador a la hora programada.

De esta manera se consigue que el hardware funcione de manera conjunta con el software para conseguir el objetivo deseado.

Además de todo esto, como ya se ha comentado con anterioridad, a nivel de software también se han utilizado diversas librerías en Python:

- Selenium: emula el navegador para descargar la información no embebida en html.

- Openpyxl: permite editar y guardar archivos excel.
- Time: para saber cuanto tarda en realizar todo el proceso de recopilación. Adicionalmente también se ha usado para pausar el script y esperar a que se cargase el contenido correctamente.
- Socket: se utiliza de manera conjunta con Smptlib para mandar los correos con la información oportuna tras la descarga de los datos.
- Smtplib: se emplea para iniciar sesión en la cuenta de e-mail, para poder enviar correos.
- Datetime: para obtener la fecha e ir modificándola en un bucle para obtener los vuelos de otros días posteriores.
- Os: se utiliza como medida de seguridad para tener permisos en caso de que sea necesario cerrar la aplicación de Excel si esta estuviese abierta mientras se efectúa la archivación de los datos, puesto que si está abierta mientras esto ocurre salta un error y no deja guardar los excels.

4.1.2 Predicción de precios mediante redes LSTM

El proceso de predicción de precios varía en gran medida con respecto al de descarga de datos. En este caso sí que se requiere de un ordenador relativamente potente, sobre todo en términos gráficos, ya que se trabaja con redes neuronales. Ya que no se dispone de ningún ordenador con una tarjeta gráfica dedicada, se empleará un servicio gratuito de computación remota llamado *Google Colaboratory*.

En la página oficial de Google Colab se autodefine como *"una herramienta que permite ejecutar y programar en Python en tu navegador, sin la necesidad de configuración, con acceso gratuito a GPU y que puedes compartir fácilmente"* [31]. Con esta definición queda claro que Google Colab es un entorno de programación en Python que permite ejecutar código de manera remota y haciendo uso de los servidores de Google.

Esto no es solo increíblemente útil por el hecho de que se pueden realizar tareas más exigentes (que un ordenador personal quizá tendría dificultad de realizar), sino que además permite trabajar desde cualquier dispositivo con una conexión a internet, haciéndolo mucho más cómodo, ya que se puede trabajar desde cualquier lugar y con distintos dispositivos. Esto también permite que se comparta el proyecto entre varios individuos, lo cual posibilita que varias personas trabajen conjuntamente.

Trabajar de manera colectiva es algo muy ventajoso para empresas y grupos de investigación, por no hablar de los beneficios económicos que conlleva poder verter la carga computacional a servidores remotos. Además de todo esto, Colab trae nativamente la mayoría de las librerías de aprendizaje profundo preinstaladas, por lo que para utilizarlas es tan fácil como importarlas directamente en el proyecto en el que se quieran utilizar. Por todas estas razones, Google Colab se ha convertido en la plataforma por antonomasia y más utilizada para este tipo de proyectos.

Además de estos materiales, también se han empleado diversas librerías para el diseño, entrenamiento y previsión de las redes neuronales:

- Google colab files: para subir los archivos csv al entorno de Google Colab.
- Numpy: para gestionar las bases de datos y la estructura de los mismos para su correcta utilización por parte de la red neuronal.
- Pandas: para leer e importar correctamente los archivos *csv* donde se encuentran los datos.

- Random: se utiliza para darle aleatoriedad programada al sistema, aunque siempre con un *seed* especificado para poder obtener resultados reproducibles.

- Tensorflow: es una librería *open-source* para facilitar la implementación de aprendizaje automático en un proyecto. Usa Python para el *front-end*, donde se pueden construir y configurar los distintos bloques que formaran la red neuronal, y C++ para el *back-end* para ejecutar estos bloques de la manera más eficiente y rápida posible. En el siguiente apartado se hablará mucho de está librería y se profundizará más.

- Matplotlib: sirve para poder ilustrar de manera gráfica los resultados obtenidos, como el *loss* o la predicción.

- Keras: es una librería de redes neuronales que trabaja por encima de Tensorflow y cuya finalidad es facilitar la modelación y prototipado de las redes neuronales.

4.2 Metodología

4.2.1 Preparación de los datos de entrenamiento

El proceso de entrenamiento de la red para la posterior predicción de los precios comienza con la preparación de los datos, para darles el formato adecuado y que sean correctamente incorporados en la fase de entrenamiento. Por ello, el primer paso que se pretende abarcar en este apartado es este, la preparación de los datos.

4.2.1.1 Normalización

Tras el proceso de obtención de datos se dispone de una gran cantidad de archivos excel. El primer paso es el de filtrado de dichos archivos, ya que únicamente interesarían los archivos con más de 100 días de precios. Para hacer este filtrado se hace uso de un script en Python que itera sobre todos los excels y descarta los que tienen menos de 100 filas de precios. En el momento de la entrega del TFG existen en torno a 214 archivos que cuentan con la información del precio de más de 100 días distintos. Esto quiere decir, en otras palabras, que se tiene información de la fluctuación de precios a 100 días (o más) de 214 vuelos.

Además, también se han de convertir los archivos excel a csv, ya que las librerías que se van a emplear tienen plena compatibilidad con esta extensión. Para ello se hace uso de la anteriormente mencionada librería Pandas, convirtiendo todos los archivos que han pasado el filtrado a csv.

Tras hacer esta preparación preliminar, se han de subir los archivos seleccionados a la plataforma de Colab e introducirse en listas. Una vez realizado esto se convierten los valores a *float32* para tener disponibles varios decimales.

Tras haber hecho esto se procede a normalizar los valores de los precios a números entre el rango de -1 y 1. La normalización de los datos es necesaria ya que optimiza y acelera notablemente el proceso de entrenamiento, haciendo que converja antes [32].

Es importante que se normalicen los datos principalmente por 2 razones:

- Todos los datos de entrada estarán en la misma escala. Si hubiese datos mucho más grandes que otros, estos harían que la ecuación de actualización de los pesos se actualizase mucho más rápido, ya que la velocidad de aprendizaje es proporcional a la magnitud de los valores. Por ello, habría pesos asociados a determinados valores que se actualizarían mucho más rápido que otros, y esto podría ir en detrimento del aprendizaje. Más adelante se hablará más a fondo de la ecuación de actualización de los pesos, con lo que esta explicación cobrará de más sentido.

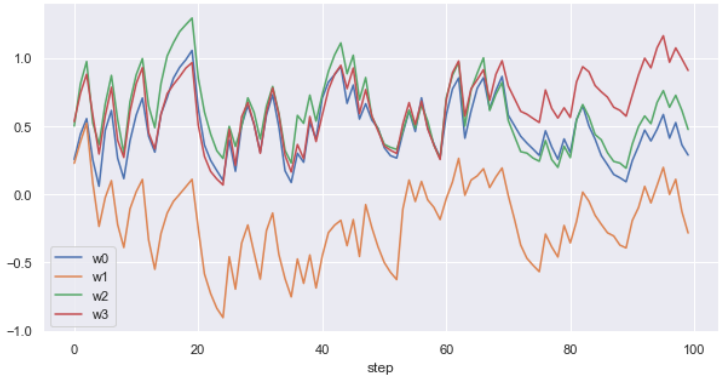
- Habrá valores tanto negativos como positivos, lo cual hará que mejore el aprendizaje para los nodos consiguientes. Esto también se explicará a continuación más a fondo.

Existen varias técnicas de normalización, pero las más ampliamente usadas en redes neuronales artificiales son la sigmoide y la tanh, funciones ya explicadas en el último apartado del marco conceptual.

La derivada parcial de la función de pérdidas se ve afectada de manera directa por el valor de entrada de los datos. Es por esto por lo que si todos los valores de entrada son del mismo signo, como es el caso de la normalización sigmoide, todos los pesos del nodo tendrán que incrementar o decrecer, no es posible que haya discrepancia en este sentido. Esto es muy ineficiente, ya que si en el proceso de descenso por gradiente el vector ha de cambiar de dirección lo tendrá que hacer haciendo *zig-zag*.

En caso de que los valores de entrada estuviesen normalizados por tanh, y por lo tanto pudiesen tomar valores positivos y negativos, los pesos también se podrían actualizar de manera independiente unos de otros, no siendo necesario que todos se moviesen en la misma dirección. Esto hace que el vector pueda cambiar más rápido de dirección y por lo tanto acelera y hace mucho más eficiente todo este proceso.

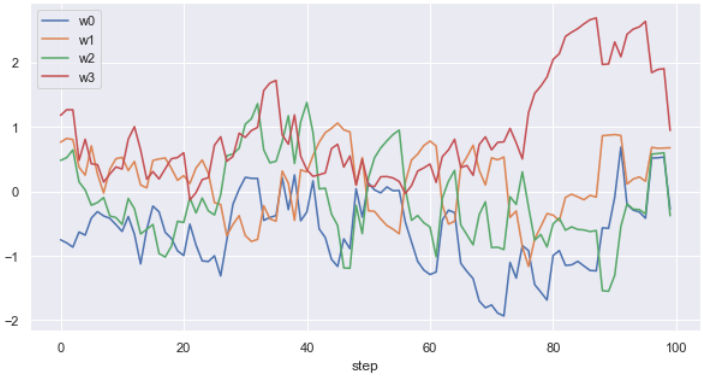
De manera más gráfica, en la gráfica 6, se pueden observar los cambios de determinados pesos en cada paso para datos normalizados con la función sigmoide:



Gráfica 6: Actualización pesos para una función sigmoide

Como se puede observar, en cada paso los pesos han de actualizarse en la misma dirección. El sistema ha de "jugar" con la magnitud del paso de cada peso para poder llevar a los pesos a sus respectivos valores óptimos. Aquí se observa claramente el movimiento de *zigzag* del que se ha hablado anteriormente.

En la gráfica 7 se observa el mismo proceso, pero con datos normalizados con la función tanh:



Gráfica 7: Actualización pesos para la función tanh

En este caso sí que se puede ver como el sistema ajusta el movimiento de los distintos pesos a lo que considera más oportuno, sin necesidad de actualizarlos todos en el mismo sentido.

Es por esto que se ha considerado más oportuno la normalización mediante la función tanh para el presente proyecto.

4.2.1.2 División de los datos

Para el correcto entrenamiento de la red y la posterior predicción de los datos es importante dividir los datos de manera concisa para una optimización del proceso.

En este caso se ha decidido seccionar los datos en 80% entrenamiento y 20% test, ya que es la proporción más comúnmente aceptada y que proporciona los mejores resultados de manera general. Esto quiere decir que de los 214 archivos excels con los que se cuenta, un 80% de los archivos se usará en su totalidad para el proceso de entrenamiento y un 20% se empleará en el test para la predicción.

Dentro de los archivos asignados al proceso de entrenamiento habrá a su vez una subdivisión de 80%-20% para entrenamiento y validación respectivamente. La parte de validación se emplea para evaluar el rendimiento de la red con el aprendizaje que ha llevado a cabo en la fase de entrenamiento. Sirve para poder configurar y mejorar los parámetros de entrenamiento así como para monitorizar el indeseable efecto del *overfitting*. Gareth James explica la validación en su libro *An Introduction to Statistical Learning: with Applications in R* de la siguiente manera: *"Supongamos que quisiéramos estimar el error de prueba asociado con el ajuste de un método de aprendizaje estadístico particular en un conjunto de observaciones. El enfoque del conjunto de validación [...] es una estrategia muy simple para esta tarea. Implica dividir aleatoriamente el conjunto de observaciones disponibles en dos partes, un conjunto de entrenamiento y un conjunto de validación o conjunto de retención. El modelo se ajusta al conjunto de entrenamiento, y el modelo ajustado se utiliza para predecir las respuestas de las observaciones del conjunto de validación. La tasa de error resultante del conjunto de validación -típicamente evaluada mediante la EME en el caso de una respuesta cuantitativa- proporciona una estimación de la tasa de error de la prueba."* [33]

Y por lo tanto, en la figura 18 se muestra la distribución final:

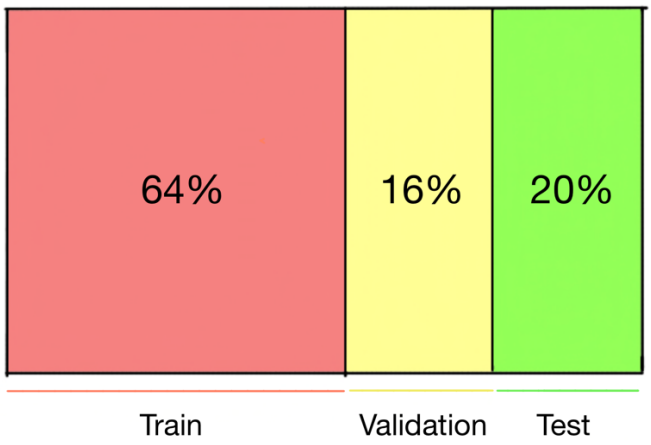


Figura 18: Distribución final de datos. Elaboración propia.

4.2.1.3 Flight Generator

Los *generator* o generadores son funciones en Python que se emplean para iterar. Su implementación es extremadamente fácil, ya que se construyen como funciones normales y corrientes con la única diferencia que en vez de usar el *return* para devolver la variable, se usa un *yield*.

Al usar *return* en un método dicha función se para completamente y es terminada, mientras que al emplearse *yield* la función es pausada hasta la siguiente vez que se le llame, guardando así su estado actual para próximas iteraciones.

Por lo tanto, para el presente proyecto el uso de un *generator* resulta trascendental, tanto para estructurar los valores de entrada de la red, como para alimentarla en el proceso de entrenamiento de manera progresiva.

El proceso de normalización se realiza dentro de este generador, al igual que otros diversos procesos de entre los cuales destaca la función *create_dataset*.

Para explicar esta función, primero se ha de hablar de la forma que tienen los datos al introducirse en la red neuronal. La manera en la que se trabajará será mediante un proceso de enventanado. En cada iteración de entrenamiento y validación, al sistema se le proporcionará una ventana con un número determinado de precios anteriores al que ha de averiguar. La cantidad de vuelos pasados que se le proporciona, que sería a su vez el tamaño de la ventana, viene determinado por la variable *look_back*. Pues este es el cometido de la función *create_dataset*. Se le proporciona un valor para *look_back* y esta actualiza la ventana en función de dicho valor. Esta ventana seguidamente se empleará en los procesos de entrenamiento o validación, dependiendo de la fase.

Para poder entenderlo mejor de manera visual, en la tabla 2 se muestra un ejemplo del enventanado para un *look_back* igual a 4:

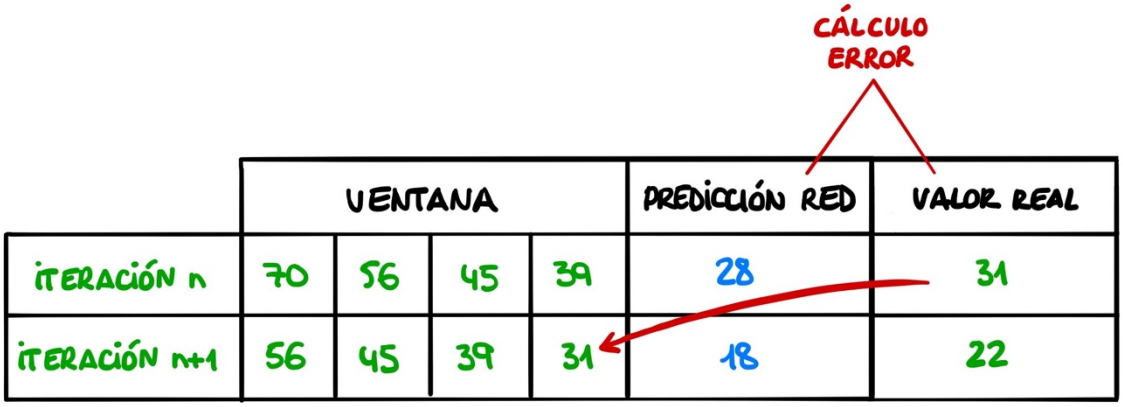


Tabla 2: Distribución de valores de entrada en la fase de entrenamiento. Elaboración propia.

Como se puede observar, la función *create_dataset* no actualiza los datos de entrada con los que la red ha predicho, sino que usa los reales. Esto como se verá más adelante difiere en gran medida de la manera de preparar los datos para la predicción en el apartado de test. Aquí, a grandes rasgos, la red emplea los datos que se le proporciona e intenta predecir el precio futuro. Una vez hecho esto se le proporciona el valor real que tendría que haber calculado de haber hecho la predicción correcta. Con esto, conociendo el error, la red modifica los pesos de la arquitectura e intenta mejorar su predicción para posteriores iteraciones. He aquí el motivo de que se le denomine a esta fase "fase de entrenamiento".

Una vez creadas las ventanas, estas se convierten al formato *Numpy array*, que es con el que trabaja *tensorflow*, y se devuelven al sistema mediante la anteriormente mencionada declaración *yield*.

4.2.2 Modelos LSTM

Con motivo de obtener mayor variedad de resultados y poder comparar distintas técnicas de aprendizaje profundo por LSTM, se ha decidido no solo emplear una red LSTM convencional, sino además construir otras dos variaciones de la LSTM. Las otras dos variaciones son la red LSTM bidireccional y la

convolucional. Con esto se busca conocer la eficacia de la red LSTM convencional y ver si realmente existe mejoría al trabajar con estas variaciones.

4.2.2.1 Modelación red LSTM convencional

La modelación de la red LSTM convencional cuenta con diversas opciones y configuraciones posibles. Para este caso se ha decidido optar por una red LSTM con una capa de entrada de 32 nodos, 2 capas ocultas con 16 y 8 neuronas respectivamente, y una capa de salida, como se muestra en a figura 19.

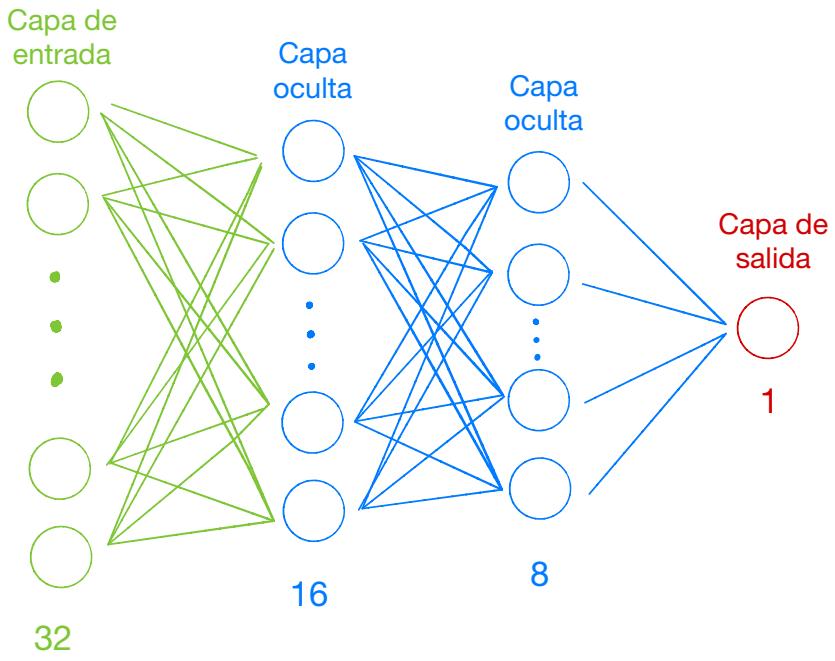


Figura 19: Modelo red LSTM convencional. Elaboración propia.

A primera vista puede parecer que se ha decidido emplear capas con relativamente pocas neuronas, pero esta decisión es totalmente deliberada. Como ya se verá posteriormente, a mayor número de neuronas no mejora el rendimiento y si se aumenta mucho comienza a haber *overfitting* o sobre entrenamiento.

Al tratarse este de un proyecto con datos de entrada unidimensionales, ya que los valores de entrada son precios distribuidos temporalmente, y no tratarse de otro tipo de información de mayor dimensionalidad como imágenes o video, los dos problemas descritos en el último apartado del marco conceptual relativos a la función tanh y sigmoide frente a la ReLU no son tan preocupantes, ya que las 3 funciones rinden bien. Tras probar con cada una de ellas, la que mejores resultados brinda es la función tanh para este modelo, lo cual es lógico habiendo normalizado los datos entre -1 y 1, por lo que será la que se utilice a lo largo del proyecto.

Los siguientes parámetros a configurar son el *batch_size*, el *look_back* y los *steps_per_epoch*.

El *batch_size* es un parámetro que indica el número de muestras con las que se entrena por iteración. De esta manera si el número de muestras es muy elevado y el ordenador no presenta de suficiente memoria para almacenarlas todas a la vez este problema quedaría solventado. Además, la red se entrena más rápido así, ya que de esta manera en cada iteración con cada *batch* se actualizan los pesos. Para el sistema se ha decidido que un *batch_size* de 128 es ideal, ya que acelera cuantiosamente el proceso y, como ya se verá posteriormente, mejora la predicción.

El *look_back,* como ya se ha indicado anteriormente, sería el tamaño de la ventana y se ha decidido darle un valor de 32, puesto que con esta ventana relativamente grande la red tiene oportunidad de

conocer gran cantidad de precios anteriores y tener una mejor idea relativa de la tendencia del precio y por lo tanto mejora en la fase de entrenamiento y en predicción.

Por último, los *steps_per_epoch* o pasos por época se calculan dividiendo el número total de datos de entrada entre el *batch_size*, de esta manera se asegura que la red ve todos los datos.

4.2.2.2 Modelación red LSTM bidireccional

Las redes LSTM bidireccionales son una variación de las LSTMs convencionales que en ciertas circunstancias pueden ayudar a obtener una mejor predicción. Se suelen utilizar para problemas donde se conocen todos los *timesteps* o pasos temporales. Este sistema permite entrenar 2 redes LSTM de manera conjunta, primero entrena una red de manera corriente con los datos de entrada y seguidamente entrena la segunda haciendo una copia de estos mismos datos de manera invertida. Esto podría ayudar al sistema a entender mejor el contexto y podría dar lugar a un entrenamiento y aprendizaje más completo. Mike Schuster explica esto mismo en su artículo *Bidirectional Recurrent Neural Networks* [34]. En él indica que *"para superar las limitaciones de una RNN regular [...] proponemos una red neuronal bidireccional recurrente que puede ser entrenada usando toda la información de entrada disponible en el pasado y el futuro de un marco de tiempo específico. […] La idea es dividir el estado de las neuronas de una RNN regular en una parte que es responsable de la dirección de tiempo positiva (estados hacia adelante) y una parte de la dirección de tiempo negativa (estados hacia atrás)."*

Por lo tanto, en un sistema de LSTM bidireccional se tendrían dos capas para de alguna manera poder afrontar el problema desde ambos lados y predecir el siguiente valor teniendo en cuenta tanto valores pasados como valores futuros. En la figura 20 se ve visualmente como funcionaría una red bidireccional LSTM:

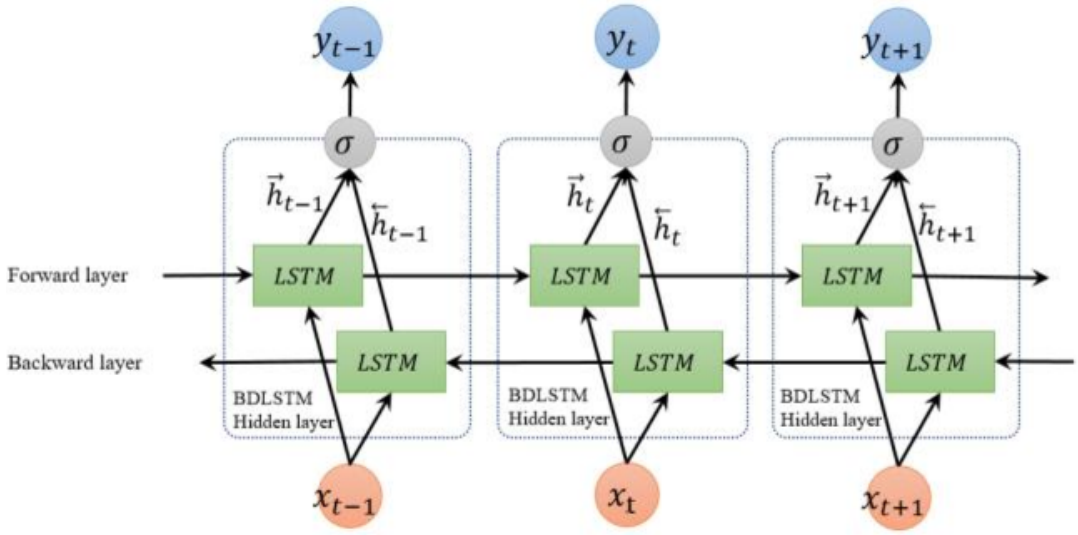


Figura 20: Funcionamiento LSTM bidireccional [35]

Aquí se pueden observar las 2 redes, una en cada dirección, que intentan predecir el mismo valor cada una desde una dirección. Las direcciones de estas redes se podrían decir que marcan la línea temporal que siguen. De esta manera las predicciones pueden tener información de ambas líneas temporales. Además, las BDLSTM pueden ser entrenadas de manera muy similar a las LSTM convencionales, ya que las neuronas de cada espacio temporal no tienen ninguna interacción entre si y a efectos de entrenamiento sería como entrenar dos redes independientes.

A modo de ejemplo, para captar mejor el concepto de cómo funcionan y cual puede ser el uso de estas redes, se tienen las siguientes frases con una incógnita *X*: "Tengo que ir a comprar a la *X*. El médico me ha recetado varios medicamentos". Para un humano queda bastante claro que la palabra incógnita es farmacia, puesto que lee las dos frases y con la ayuda de lo que viene antes y después de la palabra es capaz de descifrarla.

Pues bien, para una red LSTM convencional sería complicado predecir cual va a ser la palabra, ya que su única información sería "Tengo que ir a comprar a la". Solo tiene la información previa a la palabra que busca predecir. Las redes bidireccionales buscan solventar este problema teniendo la información tanto pasada como futura, o en este caso, tanto las palabras anteriores a la que se quiere predecir, como las posteriores. Con esto tienen mucha más información y contexto para poder predecir el elemento en cuestión.

Extrapolando todo esto al presente proyecto, en el que en vez de palabras y frases se tiene precios distribuidos en un horizonte temporal, queda bastante claro como esto podría mejorar en gran medida el proceso de predicción. El sistema puede predecir el precio del vuelo poniendo en contexto la información pasada y futura que le brindan las dos redes LSTM. La información que alimenta la red bidireccional (tanto la pasada como futura) la estructura de manera automática la librería *Bidirectional* de *Keras*.

Las BDLSTM vienen en la librería de *keras*, por lo que su implementación es muy sencilla (véase figura 21). Se establecen 2 capas de 32 y 16 neuronas respectivamente y 1 capa de salida, y se especifica la activación *tanh.* El resto de los parámetros y configuraciones son exactamente iguales a la LSTM convencional, así como la estructura de los datos que se introduce en el sistema.

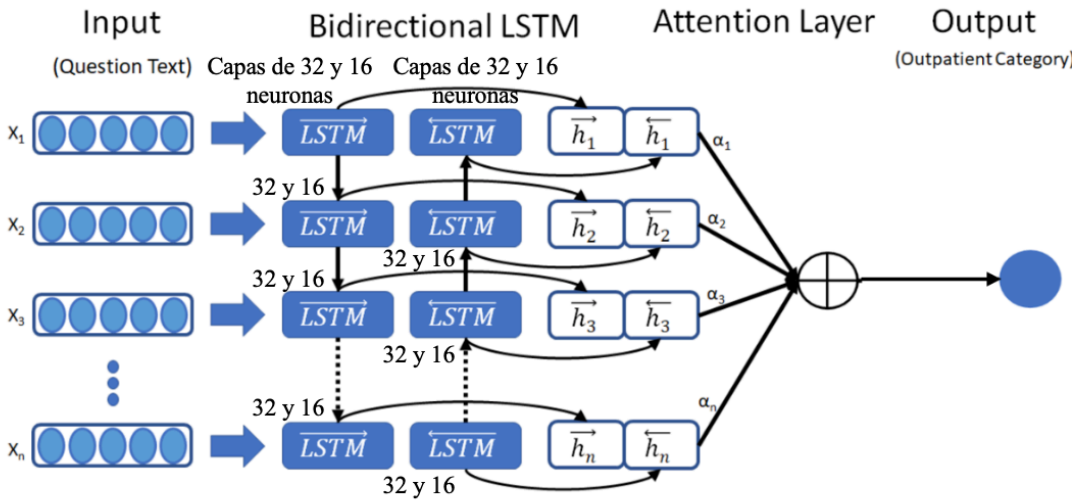


Figura 21: Modelo red LSTM bidireccional [36]

4.2.2.3 Modelación red LSTM convolucional

Las LSTM convolucionales o *CNN LSTM* son una variación redes neuronales LSTM convencionales que están específicamente diseñadas para problemas secuenciales. Se suelen usar para predicciones de imágenes o videos, aunque también se emplean en otros proyectos como el del presente trabajo cuyos datos y sistema de predicción tiene una correlación temporal secuencial [37].

Las *CNN LSTM* emplean una red *LSTM* "vanilla", como si de una red LSTM convencional se tratase, con la distinción añadida de una capa convolucional cuyo cometido es la extracción de las distintas características o *features* que presentan los datos de manera implícita. Esta funcionalidad resulta increíblemente útil para el presente proyecto, puesto que al tratarse de datos estructurados que siguen un horizonte temporal, cabría esperar que presentaran diversas características embedidas en los mismos. Algunas de las posibles características que podría detectar serían:

- Días que quedan para que salga el vuelo.
- Que día de la semana es.

- Si un vuelo está en temporada baja o alta.

Para comenzar a modelar la CNN LSTM es tan fácil como importar la librería convolucional que viene con *keras*. Para ello se tiene que tener claro los datos con los que se trabaja, puesto que dependiendo de los mismos se importara la *Conv1D*, *Conv2D* o *Conv3D*. Puesto que los valores de entrada del sistema tienen una dimensión, para este proyecto se elige la *Conv1D*. Si en vez de datos de una dimensión que están distribuidos en una línea temporal se tuviesen imágenes, por ejemplo, entonces sería más apropiada la *Conv2D* para modelar el problema.

A grandes rasgos, el funcionamiento de estas redes es muy similar independientemente de las dimensiones que tengan. La *Conv2D*, por ejemplo, siguen el mismo patrón que la *Conv1D* pero adaptando su funcionamiento a datos y modelos con una dimensión más.

La idea principal que hay detrás de las redes convolucionales es que hay un filtro, con un tamaño determinado, que se desplaza a través de los datos para intentar extraer las características implícitas de los mismos. En la figura 22 se observa una *CNN LSTM* de *Conv1D* que se desplaza a lo largo de varias datos que en este caso son palabras:

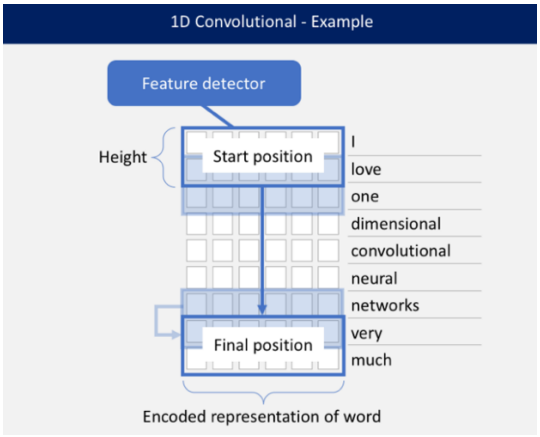


Figura 22: Funcionamiento LSTM convolucional 1D

Este es un ejemplo de *NLP* o procesamiento del lenguaje natural en el que las filas son vectores que representan las palabras que hay en la derecha. El filtro o *kernel* se desplaza a través de los vectores, en este caso tiene altura 2, lo que quiere decir que considera 2 palabras conjuntamente en el proceso de entrenamiento. Para este caso el filtro se desplazaría 8 veces hasta llegar al final de la frase.

Existen una serie de parámetros que han de configurar para el correcto funcionamiento de la red.

En primer lugar, se ha de configurar la cantidad de filtros que tiene el sistema, el cual se establece a 64. Esto quiere decir que la *CNN LSTM* es capaz de extraer hasta 64 características, ya que cada filtro es capaz de sacar una. Esta cantidad de filtros se cree que es más que suficiente para el rendimiento de la red y aumentarlo no parece dar un mejor rendimiento de la misma a partir de este número.

El siguiente parámetro es el *kernel size* o tamaño de *kernel*. Este nos da información de lo grande que es el filtro o, en otras palabras, la cantidad de valores que toma de manera conjunta. En este caso se ha optado por un *kernel size* de 1, con lo que la ventana cogerá los datos de uno en uno y se desplazará consecutivamente de esta manera también.

Tras está capa se añaden 2 capas bidireccionales LSTM con 16 y 8 neuronas respectivamente para completar el entrenamiento (véase figura 23).

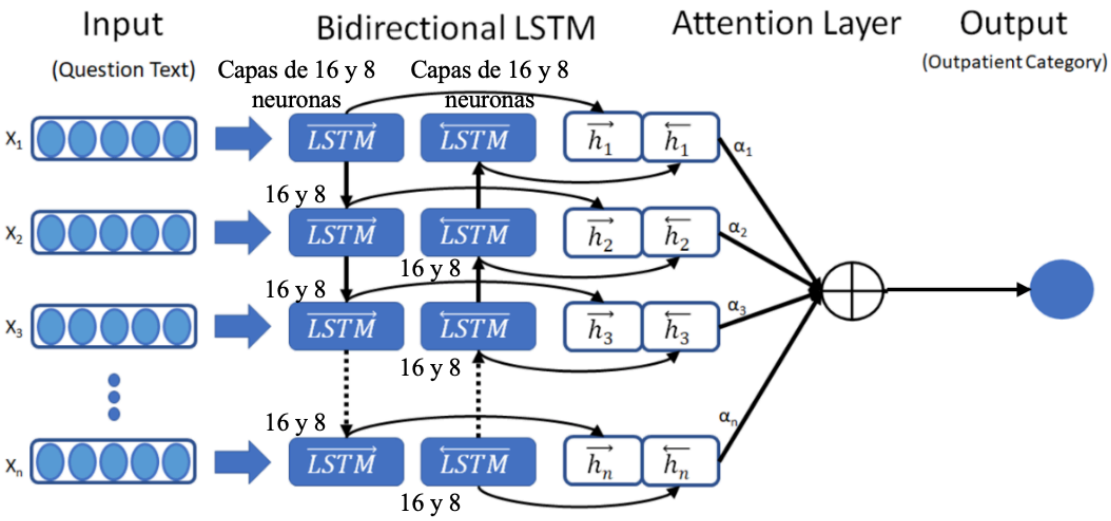


Figura 23: Parte bidireccional del modelo de LSTM convolucional

Estas vienen configuradas con la función de activación *tanh* y con un *dropout* de 0,5. El *dropout* hace que una parte de las neuronas que están en la capa aleatoriamente deje de funcionar en la iteración, reduciendo así considerablemente el *overfitting*. En este caso, al ser el *dropout* de 0.5, el 50% de las neuronas aleatoriamente dejará de funcionar.

El resto de los parámetros están configurados de la misma manera que en la red LSTM convencional.

4.2.3 Preparación datos test

Para el apartado de entrenamiento y validación, como ya se ha indicado con anterioridad, se emplea el 80% de los archivos csv, por lo tanto, para la fase de test quedaría el 20%. En esta fase lo más importante es la correcta preparación de los datos de entrada.

Si se recuerda bien, en la fase de entrenamiento se realizaba una predicción, y esa predicción se comparaba con el valor real que debería tener para calcular el error y mejorar el algoritmo. En las siguientes iteraciones, por lo tanto, no se usa como valor de entrada el predicho con anterioridad, sino el real. En la fase de test esto cambia, ya que una vez llega a los valores de test la red empieza a emplear como valores de entrada los predichos por ella misma y no los reales.

La fase de test se subdivide a su vez en dos partes. La primera sería de predicción con valores reales, y ocuparía el 80% de cada dataset o csv. En esta fase se realizan predicciones, contándose siempre con los valores de entrada reales, lo cual es muy similar al proceso de entrenamiento. Una vez hecha la predicción no se calcularía el error ni se mejoraría el algoritmo, puesto que ya no se está en la fase de entrenamiento.

La segunda fase es muy parecida a la primera, aunque difiere de esta en que una vez se ha hecho la predicción, en la siguiente iteración se emplearía el valor predicho anteriormente como valor de entrada y no el real, pues este le estaría oculto. Visualmente este proceso se entiende mejor con la tabla 3:

	VENTANA				PREDICCIÓN RED	VALOR REAL
PREDICCIÓN CON VALORES REALES	70	56	45	39	28	31
PREDICCIÓN TEST 1	56	45	39	28	18	22
PREDICCIÓN TEST 2	45	39	28	18	17	20
PREDICCIÓN TEST 3	39	28	18	17	11	21
PREDICCIÓN TEST 4	28	18	17	11	15	17

Tabla 3: Distribución de valores de entrada en la segunda fase de test. Elaboración propia.

En la imagen se muestra como ejemplo una red cuya ventana tiene tamaño 4. La primera fila correspondería a la última iteración de la primera fase que se ha comentado, la de predicción con valores reales. En esta fila todos los valores de entrada son números que están en verde, pues son valores reales. En la siguiente fila comienza la segunda fase de test. Como se puede observar cuenta con 3 valores de entrada reales, pero el último de estos sería el predicho por la red en la iteración anterior. Con estos valores de entrada ha de predecir el siguiente precio. Este proceso se repite y llega un punto en el que todos los valores de entrada son valores predichos por la propia red y ya no son valores reales. A este punto se llega en el número de la iteración que coincide con el tamaño de la ventana. Este proceso continua hasta el final del *dataset*.

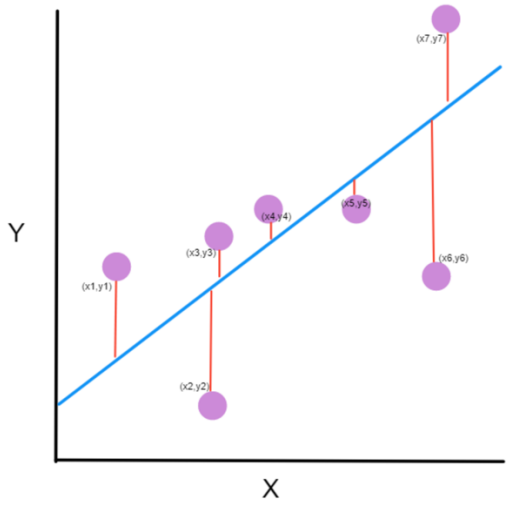
5. Resultados

Como se ha ido viendo durante el proyecto, para afrontar el problema de predicción de precios de vuelos se han construido 3 modelos distintos LSTM, que son: LSTM convencional, LSTM bidireccional y LSTM convolucional. Para poder comparar los resultados y su rendimiento se necesita tener unas métricas que puedan medir correctamente dichos parámetros.

5.1 Métricas empleadas

Las métricas que se van a usar son las siguientes MSE, RMSE, MAE y R^2. Como se puede observar, son todo métricas de regresión y cada una proporciona algo de información adicional al resto que permitirá juzgar cada modelo con exactitud y saber cuales son sus puntos fuertes y débiles.

La técnica MSE o *mean squared error* mide el promedio del error de manera cuadrática. La gráfica 8 muestra una serie de puntos y una línea que busca "acoplarse" lo mejor posible a estos puntos.



Gráfica 8: Explicación mean squared error [38]

Esta es una gráfica genérica de regresión lineal, aunque se acopla perfectamente al caso del presente proyecto. Los puntos serían los precios del vuelo que fluctúan a lo largo del tiempo en el eje X. La línea azul, a su vez, sería la predicción del sistema que intenta predecir dichos puntos.

Como se puede observar, de los puntos salen líneas rojas que cortan la línea azul, que como ya se ha mencionado sería la de predicción. Estas líneas simbolizan la distancia que hay entre el valor real y el que ha predicho la red. A destacar de estas líneas rojas, que no son el camino más corto entre el punto y la línea azul, es decir, no forman una línea perpendicular, si no que son paralelas al eje Y. Esto es así ya que lo que busca comparar es el dato real con el predicho, en la misma línea temporal. Para nuestro caso, por ejemplo, si la línea fuese perpendicular estaría comparando precios de vuelos de dos instantes de tiempo distinto, y esto no tendría sentido. La ecuación de la MSE es:

$$MSE = 1/N \sum_{i=n} (y_i - y_{i\_2})^2$$

Esta ecuación es un sumatorio de todas las distancias al cuadrado, o líneas rojas como se han llamado con anterioridad, que es dividido entre el número de líneas para obtener el promedio cuadrático. El uso del cuadrado tiene su importancia, no es trivial. Se emplea para "castigar" más a las distancias más grandes, es decir, a las veces que el sistema hace una predicción muy mala, y penaliza menos cuando el error es más pequeño. De esta manera no solo se asegura que la predicción es buena, si no que también

se tiene muy en cuenta que no haya predicciones anómalas y que varíen mucho de lo que tendrían que haber sido.

La siguiente métrica es la RMSE o *root mean squared error*. Es simplemente la raíz cuadrada de la MSE:

$$RMSE = \sqrt{MSE}$$

En este caso no se tendría en cuenta la varianza de los errores. Es una medida más "real", ya que da en términos reales cual es el error promedio, y esto es directamente comparable con los datos con los que se trabaja, puesto que está en la misma escala. Al haber una correlación entre RMSE y MSE, se sabe que si un modelo tiene mejor MSE que otro, también tendrá mejor MSE.

La MAE o *mean absolute error*. Es muy parecido a la MSE, pero en vez de elevar las distancias o errores al cuadrado los calcula en valor absolute y de manera lineal. De esta manera no penaliza tanto a los errores de predicción anómalos como la MSE. Su formula es:

$$MSE = \frac{1}{N} \sum_{i=n} |y_i - y_{pi}|$$

El valor absoluto se usa para que el error siempre de positivo. Si no se pusiera, podría haber casos en los que la predicción fuese superior al valor real y por lo tanto tras hacer la resta diera un valor negativo, el cual mejoraría el promedio de la métrica cuando debería de empeorarlo.

Por último, se tiene la métrica R^2 o R al cuadrado. Hasta ahora, todas las métricas proporcionan una buena visión del modelo, pero únicamente comparándolo a otros modelos, pues sin tenerse un punto de referencia es difícil saber si una MSE de 15, por ejemplo, es un resultado bueno o malo. La métrica R al cuadrado busca solventar esto comparando la MSE del modelo propuesto con el modelo más sencillo posible, es decir, calculando el promedio de los datos. Busca ver si en efecto el modelo propuesto mejora en algo la predicción frente a si simplemente se hubiese calculado la media de los datos y se hubiese calculado de esta manera la MSE. Su ecuación es la siguiente:

$$R^2 = 1 - \frac{MSE_{modelo}}{MSE_{promedio}}$$

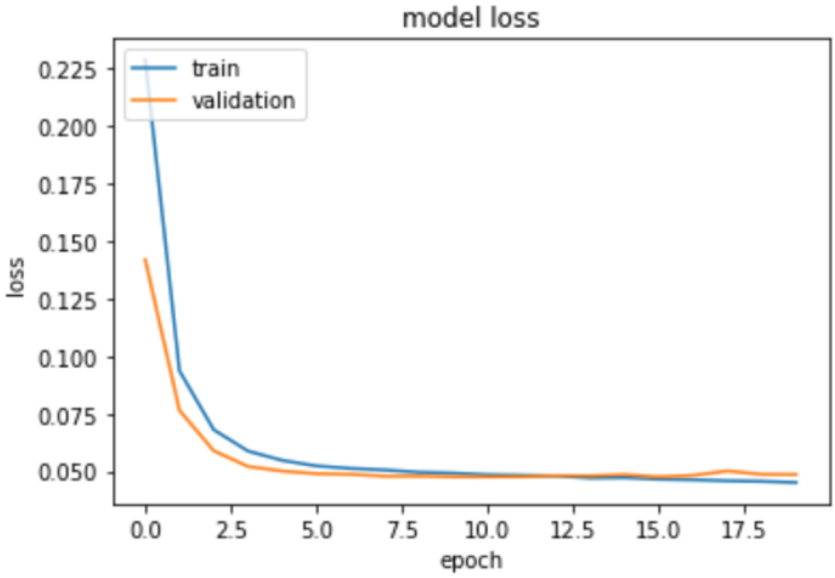
Los valores que puede tomar está ecuación van entre -∞ y 1. Cuanto menor es R^2 peor es la predicción. Si toma valor negativo significa que el modelo funciona peor que de haberse calculado la media del cuadrado de los datos. De esta manera se puede saber rápidamente si el modelo propuesto es por lo menos mínimamente valido de manera absoluta, sin necesidad de compararlo con otros.

Con las métricas ya claras se puede empezar a detallar los resultados para cada uno de los modelos.

5.2 Resultados entrenamiento

Para valorar el rendimiento de la LSTM convencional resulta indispensable empezar con la gráfica de pérdidas o *loss*. La evolución de la función de pérdidas es un método para evaluar como de bien se adapta a los datos un modelo. En el caso de que las predicciones se alejasen mucho de los valores reales, dicha función devolvería un número muy alto. De manera progresiva, con la asistencia de alguna función de optimización, que en este caso es *Adam* y como función de pérdidas el MSE, se consigue reducir el error de la predicción mediante saltos de gradiente.

La función de pérdidas para la red LSTM convencional tiene el aspecto que se muestra en la gráfica 9:

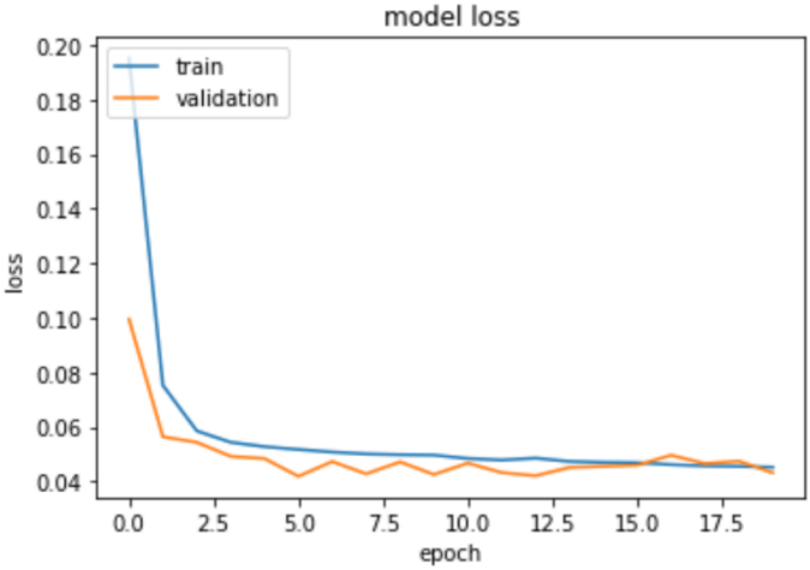


Gráfica 9: Función de pérdidas para la red LSTM convencional. Elaboración propia.

En la gráfica se ven dos funciones. En azul es la función de pérdidas de los datos de entrenamiento, mientras que en naranja la de validación. A primera vista, se observa una función de pérdidas con bastante buen aspecto. A medida que avanzan las épocas (iteraciones) disminuyen las pérdidas, tanto de entrenamiento como de validación. Esto quiere decir que la red en efecto aprende con cada iteración. El hecho de que también baje el *loss* en la validación, y que que no se separe mucho de la curva de *training,* es una clara señal de que el modelo no presenta problemas de sobre entrenamiento. En la primera época, las pérdidas son de 0,3 y 0,17 para entrenamiento y validación respectivamente.

Algo no tan bueno a destacar de las anteriores funciones de pérdidas es que a partir de la tercera o cuarta época se estabilizan las pérdidas y no continúan disminuyendo. Esto quiere decir que a partir de estas épocas el sistema no continua aprendiendo. Para intentar remediar esto se define un parámetro llamado *learning rate*. El *learning rate* regula la velocidad con la que aprende el sistema y puede hacer que aprenda más lentamente, en caso de que converja muy rápido (como es este caso) y encuentro una solución subóptima, o también puede acelerar el proceso de aprendizaje en caso de que este sea muy lento y la red se quede atascada. Cuanto menor es el *learning rate* más lentamente convergerá la función de pérdidas. En este caso se situa dicho parámetro en 1e-04. Tras observar los resultados se concluye que, si bien el *learning rate* ha mejorado algo la velocidad de convergencia, esta mejoría es prácticamente imperceptible y por tanto irrelevante para el rendimiento del sistema.

En el caso de la res LSTM bidireccional se obtiene la gráfica 10 de pérdidas:

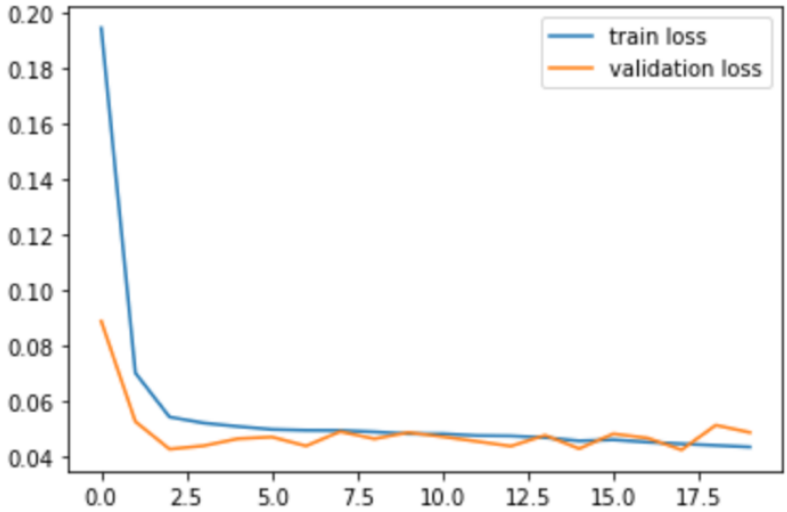


Gráfica 10: Función de pérdidas para la red LSTM bidireccional. Elaboración propia.

Como se puede observar, presente una disposición muy similar a la de la LSTM convencional. En este caso, sin embargo, los valores iniciales son algo más bajos, por lo que ya desde el comienzo se ve una cierta mejoría en este sentido. Los valores finales también son inferiores con respecto a la LSTM convencional, aunque no en la misma magnitud que los del comienzo. Viendo esto se podría concluir que este modelo, al menos a priori en el proceso de entrenamiento, presenta un mejor rendimiento. Lo que si comparte con la anterior función de pérdidas es su rápida convergencia. En este caso la función de pérdidas converge en torno a la quinta época, que es una pequeña mejoría con respecto a la anterior aunque dentro del margen de error, mientras que la de entrenamiento converge en la tercera o cuarta época.

Para intentar mejorar la rapidez de convergencia se emplea el *learning rate*, como en el caso anterior, aunque los resultados son similares, con lo que la mejoría, si existe, es prácticamente imperceptible.

Por último, en el caso de la red LSTM convolucional se obtiene el gráfico 11:



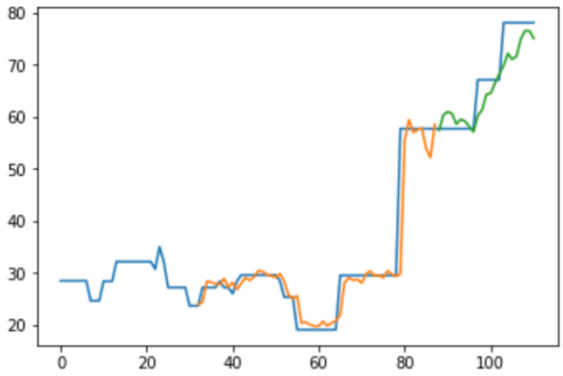
Gráfica 11: Función de pérdidas para la red LSTM convolucional. Elaboración propia.

Este caso sigue la tónica de lo que se lleva viendo hasta ahora, unas pérdidas relativamente buenas, que bajan en las primeras épocas, se estabilizan y que no presentan problemas muy graves como sobre entrenamiento o aumento de pérdida con las épocas.

5.3 Resultados de predicción

Tras ejecutar la predicción, se obtiene para cada vuelo las 4 métricas, los valores reales y predichos y al final de cada ejecución la media de las métricas para cada uno de los tres modelos. A modo de ejemplo, en la gráfica 12 se muestra lo que devolvería el sistema para la predicción del vuelo del día 6 de junio de 2020 con el modelo LSTM convencional:

```
Train Score: 4.21 RMSE
Test Score: 3.88 RMSE
Test Score: 15.06 MSE
Test Score: 3.08 MAE
Test Score: 0.80 R^2
```




```
Día = 1, Predicho = 57.332976, Real = 57.627161
Día = 2, Predicho = 60.281603, Real = 57.627161
Día = 3, Predicho = 60.890693, Real = 57.627161
Día = 4, Predicho = 60.525435, Real = 57.627161
Día = 5, Predicho = 58.513912, Real = 57.627161
Día = 6, Predicho = 59.402389, Real = 57.627161
Día = 7, Predicho = 59.106900, Real = 57.627161
Día = 8, Predicho = 58.060056, Real = 57.627161
Día = 9, Predicho = 57.012379, Real = 57.627161
Día = 10, Predicho = 60.164637, Real = 67.025388
Día = 11, Predicho = 61.226347, Real = 67.025388
Día = 12, Predicho = 64.213302, Real = 67.025388
Día = 13, Predicho = 64.436804, Real = 67.025388
Día = 14, Predicho = 66.362706, Real = 67.025388
Día = 15, Predicho = 68.153994, Real = 67.025388
Día = 16, Predicho = 69.765169, Real = 77.989990
Día = 17, Predicho = 72.089244, Real = 77.989990
Día = 18, Predicho = 70.959945, Real = 77.989990
Día = 19, Predicho = 71.567793, Real = 77.989990
Día = 20, Predicho = 74.889515, Real = 77.989990
Día = 21, Predicho = 76.466696, Real = 77.989990
Día = 22, Predicho = 76.436468, Real = 77.989990
Día = 23, Predicho = 75.006332, Real = 77.989990
```

Gráfica 12: Predicción del vuelo del día 6 de junio de 2020 con el modelo LSTM convencional. Elaboración propia.

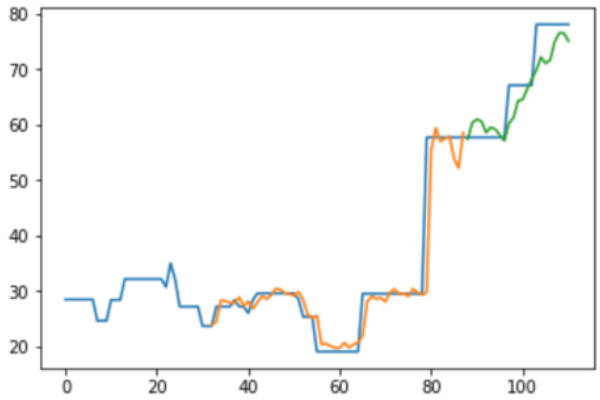
En este vuelo la predicción ha sido notablemente buena. Si nos fijamos en las métricas, salen todas con valores relativamente bajos, con lo que el error ha sido mínimo. A destacar que la métrica de R^2 sale positiva, con lo que se puede decir con total seguridad que la predicción para este vuelo es mejor que calculando la media de los valores.

Con respecto a la gráfica, en azul estarían los valores reales de los precios para cada día, en naranja los precios predichos por el sistema con valores de entrada reales y en verde los predichos por el sistema

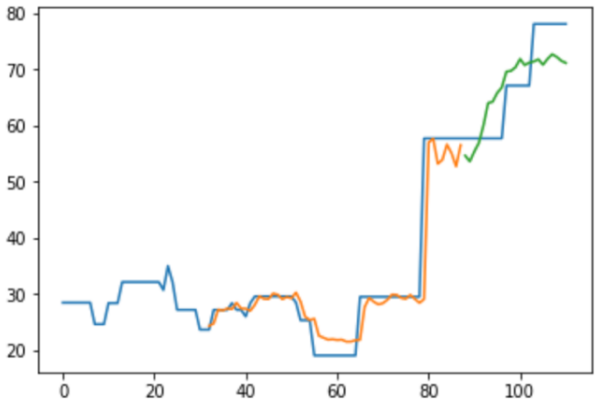
con valores de entrada calculados por la propia red. Al ver la predicción se observa que, en la parte de test, si bien no ha podido predecir con perfecta exactitud cada precio, si que ha predicho correctamente la tendencia del precio, así como su magnitud de variación.

Más abajo, se tienen los valores predichos y reales para cada uno de los días de la segunda fase test. Esta es una manera intuitiva de ver con exactitud los valores que ha predicho la red y los reales que tendría que haber calculado.

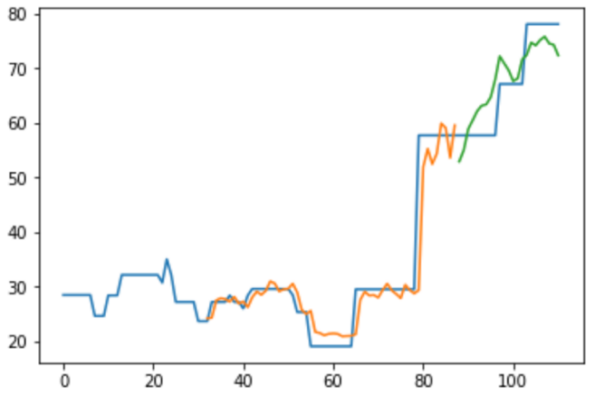
A modo de comparación, en las gráficas 13, 14 y 15 se encuentran las predicciones que han sacado cada uno de los modelos para ese mismo vuelo:



Gráfica 13: Predicción del vuelo del día 6 de junio de 2020 con el modelo LSTM convencional. Elaboración propia.



Gráfica 14: Predicción del vuelo del día 6 de junio de 2020 con el modelo LSTM bidireccional. Elaboración propia.



Gráfica 15: Predicción del vuelo del día 6 de junio de 2020 con el modelo LSTM convolucional. Elaboración propia.

Como se puede observar, ninguno de los 3 modelos ha tenido mucha dificultad en calcular la predicción para este vuelo. La mejor, aunque por poco, sería la de la LSTM convencional, aunque esto se debe contar como dentro del margen de error, ya que como se vera más adelante los 3 modelos presentan rendimientos muy similares y no hay un claro ganador.

Para comparar los resultados totales en la Tabla 4 se muestran los resultados medios (junto con la dispersión de los mismos) para la totalidad de muestras del conjunto de test. En dicha tabla se puede ver de manera conjunta cómo han rendido los distintos modelos con las métricas explicadas anteriormente.

	Convencional	Bidireccional	Convolucional
RMSE	14,46 ± 3,82	14,51 ± 3,47	14,18 ± 4,33
MSE	236,08 ± 93,68	237,94 ± 75,08	226,37 ± 112,26
MAE	11,30 ± 3,33	10,90 ± 3,09	11,36 ± 4,02
R^2	-0,69 ± 7,14	-0,51 ± 6,49	-0,98 ± 8,87

Tabla 4: Métricas RMSE, MSE, MAE y R^2 para los 3 modelos LSTM. Elaboración propia.

Salta a primera vista, como ya se ha comentado, que los 3 modelos tienen un rendimiento muy parecido y no hay uno mejor que otro.

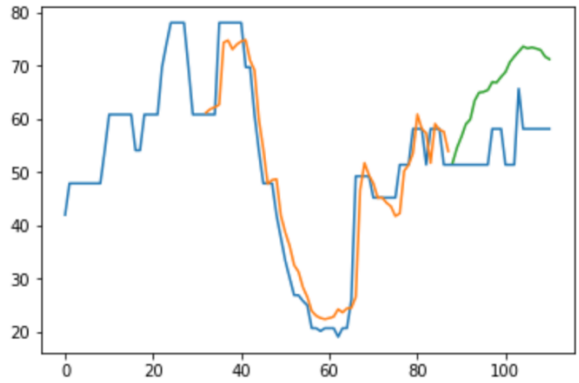
Los valores medios de MSE y RMSE que se obtienen para cada uno de los modelos son prácticamente idénticos. Siendo muy rigurosos, los mejores resultados se obtienen de la red convolucional, seguido de la convencional y finalmente la bidireccional, aunque es bastante aparente que la diferencia es tan mínima que está dentro del margen de error. Como ya se ha comentado, al existir una correlación directa entre RMSE y MSE, el modelo que mejores resultados tenga de uno también los tendrá en el otro, y lo mismo para el que peor resultado tenga.

En cuanto al valor media de la MAE, sigue la tónica que se lleva viendo durante todo este apartado. Son resultados semejantes, cuya diferencia se mide en decimales. Con esta métrica, al tratarse de la distancia media pura que hay entre los valores reales y los predichos, los valores que se obtienen también se podrían leer en euros. Es decir, de media en la red LSTM convencional la predicción ha diferido del valor real en 11,30 €. Lo mismo se puede decir de los otros dos modelos. Al pensarse así, es más intuitiva esta métrica y permite al lector darse cuenta de si en efecto la predicción es adecuada, ya que se puede medir en unidades reales y cuantificables. Al ver esta métrica en algo tan real como es el dinero, entra un poco en juego la percepción que tiene cada uno del dinero y como lo valora, por lo que la medición de esta métrica es algo subjetiva, aunque lo que sí está claro y se puede decir con bastante seguridad es que no es un error pequeño, pero tampoco colosal.

Por último, se tiene la métrica R^2. En los 3 casos sale negativa, por lo que recordando la ecuación de R^2, para que esto suceda la división de MSE_{modelo} entre $MSE_{promedio}$ ha de ser superior a 1. Esto quiere decir que la MSE del modelo tiene un error superior a la MSE de la media cuadrada de los valores reales. Esto, por lo tanto, nos daría a entender que ninguno de los 3 modelos proporciona una predicción que se adapte de manera constante a los datos reales, aunque ahora se verá que no es del todo así. Si fuese positivo el promedio, significaría que de media la predicción proporcionada por el sistema rinde mejor que la de una línea recta que atraviesa los datos y proporciona una media cuadrática de los valores reales, pero no es así.

Dicho esto, lo que también es cierto es que en la mayoría de los casos la R^2 es positiva y aceptable, pero hay determinadas predicciones que tienen en esta métrica un valor muy por debajo de lo habitual, como se puede observar en el valor de desviación típica tan elevado que presentan las métricas de R^2, y arrastran la media hacia abajo. Este es el caso del vuelo del 18/01/2020 (véase gráfica 16):

```
Train Score: 5.26 RMSE
Test Score: 12.80 RMSE
Test Score: 163.77 MSE
Test Score: 11.82 MAE
Test Score: -9.24 R^2
```



Gráfica 16: Predicción del vuelo del día 18 de enero de 2020 con el modelo LSTM convolucional. Elaboración propia.

En este caso, la predicción acierta la tendencia positiva, aunque no la magnitud de esta tendencia. El resto de las métricas están más o menos en la media (o incluso por debajo como es el caso de RMSE y MSE), pero la R^2 da un valor muy por debajo del promedio, haciendo que esta inevitablemente baje a valores muy inferiores. De hecho, de calcularse la media de R^2 en la LSTM convolucional sin tener en cuenta las anomalías, que es una técnica muy común en diversos modelos estadísticos, se obtendría un R^2 positivo, en torno a 0.13. Este comportamiento se repite también en los otros dos modelos LSTM con los que se cuenta.

6. Conclusión y líneas futuras

6.1 Conclusión

Tras el desarrollo, ejecución y finalización del presente proyecto hay varias conclusiones que se pueden extraer.

A la vista ha quedado que únicamente con el precio de los vuelos no ha habido suficiente información como para poder modelar y desarrollar un sistema de predicción fiable y constante. Esto no quiere decir únicamente que hayan faltado precios, sino que estos también tendrían que haber venido acompañados de otro tipo de información. Solo hay que pensar en las aerolíneas. Éstas no solo fijan su precio en base al histórico de precios, sino que tienen en cuenta muchos otros factores externos y que no se pueden reflejar, ni si quiera de manera implícita, en el precio.

En primer lugar, para un mejor funcionamiento de la red se tendrían que haber tenido en cuenta de manera más explícita ciertas características como: los días que faltan para que salga el vuelo, el día de la semana que es la compra, el día de la semana que es el vuelo, el mes que es el vuelo y la compra o si es temporada alta o baja. Es cierto que se podría argumentar que estas características vienen implícitas en los precios y que se podrían extraer. Y de hecho es lo que se ha intentado con la red LSTM convolucional, pero frente a los resultados que indican que esta red no proporciona ninguna mejoría frente a una red LSTM convencional, no queda más que aceptar que estas características no se han podido extraer de esta manera, al menos no de manera correcta o significativa. Una explicación posible de por qué no ha sido capaz de extraer dichas características es el efecto de la pandemia mundial (COVID-19) sobre los vuelos, que no solo ha propiciado variaciones anómalas en los precios, sino que también ha obligado a cancelar la mayoría de los mismos.

Por otro lado, también hay otras muchas características que sí que no se han tenido en cuenta, ni por activa ni por pasiva. Cabe esperar que uno de los factores más importantes a la hora de determinar el precio de un vuelo que tienen en cuenta las aerolíneas son las vacantes que tiene el vuelo. Cuantos más lleno está un avión, mayor es el precio. Y esto sí que es información a la que no se puede acceder de ninguna manera. Esta información únicamente la poseen las aerolíneas y no están a disposición del público, por lo que resultaría casi, por no decir completamente, imposible hacerse con ella. Esta es, a juicio del alumno, la mayor limitación que se ha tenido a la hora de desarrollar el proyecto.

Además, la elaboración de los 3 modelos distintos LSTM ha resultado poco fructífera, pues el rendimiento y los resultados obtenidos de cada una de estas es muy similar y las pocas diferencias que hay son tan pequeñas que se pueden atribuir al margen de error.

Algo que también ha influido en la falta de precisión del modelo es, sin lugar a duda, la falta de precios. Es cierto que la recopilación de datos lleva más de un año realizándose, pero aun así para que una predicción de esta índole funcione con un error relativamente bajo se necesita una gran cantidad de datos que sencillamente en un año y con un únicamente un ordenador no se puede hacer. Esto se ha ido viendo a medida que se realizaba la predicción a lo largo del año, a medida que se iba disponiendo de más información la predicción mejoraba gradualmente.

Pero no todas las conclusiones son negativas, también hay aspectos favorables. La fase de entrenamiento la realiza de manera adecuada, con valores en la función de pérdidas relativamente bajos y que muestran una tendencia negativa, con lo que se deduce que el sistema aprende con cada iteración, si bien esta mejoría queda algo estancada una vez converge la función de pérdidas para cada modelo.

El sistema, si bien no es capaz de predecir de manera exacta cada uno de los precios que constituye un vuelo, sí es capaz de aprender de manera positiva de los datos que se le proporciona en la fase de entrenamiento. De esto modo, de manera habitual, es capaz de predecir la tendencia de los vuelos y la tónica que van a seguir, incluso en muchos casos se acerca a las magnitudes de variación que van a presentar, salvo excepciones. Uno podría pensar, mirando las métricas que presentan los modelos, que los resultados no son excepcionalmente buenos, pero se debe recordar que hay ciertos vuelos en los que

de manera anómala predice un resultado totalmente dispar con la realidad y por este motivo las métricas se ven ancladas en valores muy por debajo del rendimiento que tendría la red de no ser por estos casos.

Para concluir, se ha diseñado e implementado un sistema capaz de predecir las tendencias que seguirán los precios de los vuelos, aunque sin poder predecir su precio exacto de manera constante. Estas predicciones, si bien interesantes y con potencial, carecen de una utilidad más allá del ámbito de la investigación. No se pueden emplear para calcular de manera exacta el precio un determinado día, únicamente se tendría una idea general de por donde rondará el precio, con un error de media, como ya se ha comentado anteriormente, de unos 11 euros.

Si el modelo resultante se quisiera llevar a una etapa de producción, se tendría que intentar solventar el cuello de botella de este proyecto, la falta de información adicional y complementaria al precio que se ha detallado más arriba. De tener en cuenta esta información se cree que se tendría una predicción mucho más potente y competente, capaz de predecir con mucha más exactitud las fluctuaciones y cambios en los precios.

6.2 Líneas futuras

El alumno, autor de este trabajo final de grado, pertenece al doble grado de *Ingeniería de Tecnologías y Servicios de Telecomunicación - Administración y Dirección de Empresas* y, por lo tanto, ha realizado un TFG también para el grado de ADE. Dicho TFG versa sobre la implementación de este mismo proyecto en el marco empresarial, y cómo se podría lanzar una empresa cuyo servicio principal fuese la predicción de precios y posterior asesoramiento a clientes con la herramienta elaborada en este TFG, para recomendarles el momento adecuado para adquirir un determinado vuelo.

En el transcurso de la realización de dicho TFG, se efectuó un estudio de mercado y se determinó que no existe ninguna empresa que pueda prestar los servicios que se intentan alcanzar en este proyecto. Por lo tanto, se llega a la siguiente conclusión: de mejorarse esta predicción y poder predecir los precios de los vuelos de manera relativamente exacta y fiable, existe un nicho de mercado que esta herramienta podría satisfacer.

Para poder, por lo tanto, mejorar la predicción se ha de solventar el cuello de botella que presenta este proyecto y del que se viene hablando en estos últimos apartados, la falta de información adicional y complementaria a los precios.

La primera línea de actuación sería añadir de manera explícita las características que se han intentado incluir en el modelo de LSTM convolucional de manera implícita, como son: los días que faltan para que salga el vuelo, el día de la semana que es la compra, el día de la semana que es el vuelo, el mes que es el vuelo y la compra o si es temporada alta o baja. Esto se podría implementar mediante la inclusión de un perceptrón multicapa o *MLP*, convirtiendo así el sistema en una red neuronal híbrida caracterizada por unidades LSTM y un perceptrón multicapa. Los *MLP* son excepcionalmente buenos en encontrar relación en datos lineales y no lineales, por lo que se adecuaría muy bien al presente modelo, además de tener soporte para multitud de variables de entrada, por lo que sería perfecto para implementar de manera categórica la información adicional que le falta al sistema.

De manera adicional, también se podría intentar hablar o llegar a un acuerdo con alguna aerolínea para que nos proporcionará la información del número de vacantes que hay en un vuelo. De tener estos datos e implementarlos en el modelo, con por ejemplo la *MLP* que se ha sugerido anteriormente, podría mejorar en gran medida el rendimiento y precisión de la red.

Bibliografía

[1] «Statista,» 2020. [En línea]. Available: https://es.statista.com/estadisticas/635149/trafico-aereo-ingresos-mundiales-por-pasajeros/. [Último acceso: 2004-2019].

[2] T. B. R. Company, Artificial Intelligence Global Market Report 2020-30: COVID-19 Growth and Change, 2020.

[3] T. J. F. Herrera, T. J. F. Herrera y A. A. M. Mendoza, Aprendizaje automático y PYMES: Oportunidades para el mejoramiento del proceso de toma de decisiones, 2020.

[4] J. Ker, L. Wang, J. Rao y T. Lim, «Deep Learning Applications in Medical Image Analysis,» 2018, pp. 9375-9389.

[5] F. Niroui, K. Z. K. y G. Nejat, «Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments,» de *IEEE Robotics and Automation Letters*, 2019, pp. 610-617.

[6] Z. Shuai, L. Yao, A. Sun y Y. Tay, Deep Learning Based Recommender System: A Survey and New Perspectives, 2019.

[7] C. Straucha, K. Mühlb, K. Patroa, C. Grabmaiera, S. Reithingera, M. Baumannb y A. Huckaufa, Real autonomous driving from a passenger's perspective: Two experimental investigations using gaze behaviour and trust ratings in field and simulator, 2019.

[8] T. Wang, S. Pouyanfar, H. Tian, Y. Tao, M. Alonso, S. Luis y S.-C. Chen, «A Framework for Airfare Price Prediction: A Machine Learning Approach,» de *IEEE 20th International Conference on Information Reuse and Integration for Data Science*, Los Angeles, CA, USA, IEEE, 2019, pp. 200-207.

[9] J. A. Abdellaa, N. Zakib, K. Shuaiba y F. Khanc, Airline ticket price and demand prediction: A survey, King Saud University, 2019.

[10] O. Etzioni, R. Tuchinda, C. A. Knoblock y A. Yates, «To buy or not to buy: mining airfare data to minimize ticket purchase price.,» de *n Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, 2003, p. 119–128.

[11] D. Escobari, «Estimating dynamic demand for airlines,» de *Economics Letters*, 2014, pp. 26-29.

[12] «AirHint,» Favencia Ltd., 2015. [En línea]. Available: https://www.airhint.com/.

[13] «Selenium,» Mayo 2020. [En línea]. Available: https://www.selenium.dev/documentation/en/webdriver/.

[14] J. McCarthy, «Science Daily,» 1956. [En línea]. Available: https://www.sciencedaily.com/terms/artificial_intelligence.htm#:~:text=John%20McCarthy%2C%20who%20coined%20the,synthetic%20intelligence%20or%20computational%20rationality..

[15] M. Rouse, «Search Data Center,» Abril 2017. [En línea]. Available: https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI.

[16] H. Baird, A. Coates y R. Fateman, «International Journal on Document Analysis and Recognition,» de *Pessimal Print: A Reverse Turing Test*, 2003, p. 158–163.

[17] J. Shabbir y T. Anwer, Artificial Intelligence and its Role in Near Future, JOURNAL OF LATEX CLASS FILES, 2015.

[18] F. S. Caparrini, «Fernando Sancho Caparini,» 23 Septiembre 2017. [En línea]. Available: http://www.cs.us.es/~fsancho/?e=75.

[19] V. Kote, Unsupervised-Learning Assisted Artificial Neural Network for Optimization., 2019.

[20] U. &. N. T. Odi, Geological Facies Prediction Using Computed Tomography in a Machine Learning and Deep Learning Environment., 2018.

[21] M. H. Hassoun, Fundamentals of Artificial Neural Networks, MIT Press, 1995.

[22] D. H. Chowdary, «Medium,» 2 Junio 2020. [En línea]. Available: https://medium.com/towards-artificial-intelligence/how-to-build-and-train-your-first-neural-network-9a07d020c4bb.

[23] K. T. Reddy y D. T. Swarnalatha, Neural Network and Deep Learning Analysis Using Backpropagation, International Journal of Engineering Research and Technology (IJERT), 2018.

[24] J. C. P. M'ng, Forecasting East Asian Indices Futures via a Novel Hybrid of Wavelet-PCA Denoising and Artificial Neural Network Models, 2016.

[25] DeepAI, Recurrent Neural Network.

[26] F. A. Gers, J. Schmidhuber y F. Cummins, «Learning to Forget: Continual Prediction with LSTM,» de *Ninth International Conference on Artificial Neural Networks*, Edinburgh, UK, 2006, pp. 850-855.

[27] A. Graves y J. Schmidhuber, «Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures,» 2005, pp. 602-610.

[28] C. Olah, «Github,» Understanding LSTM Networks, 2015. [En línea]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[29] R. Restrepo, «Ronny Rest,» 2017. [En línea]. Available: http://ronny.rest/blog/post_2017_08_10_sigmoid/.

[30] A. Oprea, *Machine Learning and Data Mining I. DS 4400,* Northeastern University, 2019.

[31] Google Colab, [En línea]. Available: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=5fCEDCU_qrC0.

[32] T. Stöttner, «Towards Data Science,» Why Data should be Normalized before Training a Neural Network, [En línea]. Available: https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-

c626b7f66c7d#:~:text=The%20tanh%20function%20is%20quite,therefore%20they%20are%20al ways%20positive..

[33] G. James, D. Witten, T. Hastie y R. Tibshirani, de *An Introduction to Statistical Learning: with Applications in R*, 2013, p. 176.

[34] M. Schuster y K. K. Paliwal, «Bidirectional Recurrent Neural Networks,» *IEEE,* vol. 45, nº 11, p. 9, 1997.

[35] G. Melli, «Gabor Melli's Research Knowledge Base,» 2019. [En línea]. Available: http://www.gabormelli.com/RKB/Bidirectional_LSTM_(biLSTM)_Model.

[36] C.-W. Chen, S.-P. Tseng, T.-W. Kuan y J.-F. Wang, Outpatient Text Classification Using Attention-Based Bidirectional LSTM for Robot-Assisted Servicing in Hospital, 2020.

[37] N. Ackermann, «Good Audience,» Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences, 4 Septiembre 2018. [En línea]. Available: https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf.

[38] M. Binieli, «Free Code Camp,» Machine learning: an introduction to mean squared error and regression lines, 16 Octubre 2018. [En línea]. Available: https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/.