Machine learning: Redes neuronales aplicadas a la Fórmula 1

Alejandro Izquierdo Bonilla

Trabajo de fin de grado de Matemáticas
Universidad de Zaragoza

Director del trabajo: Ricardo López Ruiz
14 de junio de 2023

Abstract

In recent years, data science - the storage, processing and study of vast amounts of data - has gained great importance, due to the important conclusions that can be drawn.

In the sports field, particularly in the automotive industry, the difference between victory and defeat is tiny, and any slight advantage is of crucial importance. Thus, in the last two decades, it has been possible to appreciate a large investment in the storage and treatment of the data generated in the competition to try to obtain competitive advantages thanks to them.

Automatic learning, machine learning in English, is the fundamental pillar around which all current research revolves. Thanks to it, systems are trained to act, if not better, at least in the same way as a human would, drawing conclusions from data that they have not faced before.

The objective of this work is to expose, through the use of machine learning methods, how to improve decision-making in a Formula 1 race. In particular, a neural network is used to analyze the change in tire compound happened during the race.

In chapter 1, a general explanation is made about the different aspects of machine learning and the problems they solve. Subsequently, in chapter 2, the concept of neural networks and the multiple parts that compose them are deepened, from the number of elements within them to the algorithms that govern their internal functioning and decision making.

Finally, in chapter 3, a neural network is implemented on a Formula 1 data set and its results are analyzed. The strategy of the careers that make up the data set can be found in Annex I, while the code for data processing and the structure of the neural network is found in Annex II and the training process of the neural network in Annex III.

Resumen

En los últimos años, la ciencia de datos - el almacenamiento, procesamiento y estudio de ingentes cantidades de datos - ha cobrado una gran relevancia, debido a las importantes conclusiones que se pueden extraer.

En el ámbito deportivo, en particular en la automoción, la diferencia entre la victoria y la derrota es ínfima y, cualquier mínima ventaja, es de crucial importancia. Así, en las últimas dos décadas, se ha podido apreciar una gran inversión para el almacenamiento y tratamiento de los datos generados en la competición para tratar de obtener ventajas competitivas gracias a ellos.

El aprendizaje automático, machine learning en inglés, es el pilar fundamental alrededor del cuál pivotan todas las investigaciones actuales. Gracias a él, se entrenan sistemas para que actúen, si no mejor, al menos igual que lo haría un humano, extrayendo conclusiones de datos a los que no se ha enfrentado con anterioridad.

El objetivo del presente trabajo consiste en exponer, mediante la utilización de métodos de aprendizaje automático, cómo mejorar la toma de decisiones en una carrera de Fórmula 1. En particular, se hace uso de una red neuronal para analizar el cambio de compuesto de los neumáticos sucedido durante la carrera.

En el capítulo 1, se realiza una explicación general sobre las distintas vertientes del aprendizaje automático y las problemáticas que solucionan. Posteriormente, en el capítulo 2, se profundiza en el concepto de redes neuronales y las múltiples partes que las componen, desde el número de elementos dentro de las mismas hasta los algoritmos que rigen su funcionamiento interno y toma de decisiones.

Finalmente, en el capítulo 3, se implementa una red neuronal sobre un conjunto de datos de Fórmula 1 y se analizan sus resultados. La estrategia de las carreras que conforman el conjunto de datos puede encontrarse en el Anexo I, mientras que el código del tratamiento de los datos y estructura de la red neuronal se encuentra en el Anexo II y el proceso de entrenamiento de la red neuronal en el Anexo III.

Índice general

Capítulo 1

Inteligencia artificial: Machine learning

1.1. Introducción

Actualmente, dar una definición de la inteligencia artificial (IA) que no resulte polémica no es sencillo, principalmente por la falta de consenso existente sobre qué es la inteligencia. Una de las definiciones más aceptadas dice que es «la ciencia e ingeniería de hacer máquinas inteligentes, estando relacionado con el estudio de la inteligencia humana, pero sin limitarse a métodos biológicamente observables» [1]; es decir, el estudio y desarrollo de mecanismos para lograr, de manera artificial, realizar procesamientos lógicos propios de los seres humanos.

Se considera que 1950 es la fecha de partida en la investigación en inteligencia artificial, con la publicación de «Computing Machinery and Intelligence» [2] por parte del matemático inglés Alan Turing. Turing se plantea una sencilla pregunta: «¿Pueden las máquinas pensar?». Ofrece un test, conocido como Test de Turing, para discriminar entre una respuesta de un humano o de una máquina. Aunque ante los avances actuales, el test ha quedado desfasado, sigue gozando de especial relevancia debido a su importancia histórica y conceptual, siendo aún objeto de debate y de investigación.

En nuestros días, las aplicaciones de la inteligencia artificial son muy diversas. En los últimos años han cobrado una gran relevancia, con la proliferación de inteligencias artificiales dedicadas a la creación de imágenes artificiales, identificación de autores de obras anónimas, redacción de textos a través de una idea inicial, reconocimiento facial... Así mismo, son empleadas en el sector sanitario, integradas en herramientas dedicadas a la detección de tumores, o en el sector armamentístico, en sistemas de control de vehículos autónomos. Cada capa del tejido productivo es susceptible de ver la parcial o total implementación de sus labores por parte de una IA.

Actualmente el estudio de la inteligencia artificial está diversificado: machine learning, deep learning, procesamiento del lenguaje natural... En el presente trabajo vamos a centrarnos en la rama del **machine learning**, es decir, el aprendizaje automático. Esta disciplina se enfoca en el uso de datos y algoritmos para imitar el proceso de aprendizaje, mejorando gradualmente su precisión. A día de hoy, el machine learning es uno de los principales pilares de la ciencia de datos. Las empresas son conscientes del potencial de sus datos, razón por la cual diseñan y entrenan algoritmos para hacer clasificaciones o predicciones y descubrir información clave en proyectos de *minería de datos*. Sobre la base de esta información se toman decisiones que eventualmente contribuyen al crecimiento de las capacidades productivas de las empresas interesadas. Dentro del aprendizaje automático se distinguen tres tipos: el aprendizaje supervisado, el aprendizaje no supervisado y el aprendizaje por refuerzo.

1.2. El aprendizaje supervisado

El aprendizaje supervisado constituye una rama dentro del aprendizaje automático; un método de análisis de datos en el que, utilizando algoritmos que aprenden iterativamente de los datos, se permite que los propios algoritmos detecten información oculta sin tener que haber sido programado explícitamente el qué y dónde buscarla. Se emplea para resolver problemas cuya solución es previamente conocida,

utilizándose un conjunto de datos etiquetados para entrenar un algoritmo que realice tareas específicas. Se denomina supervisado debido a que cada dato de entrada tiene asociado un dato de salida debidamente etiquetado. A través del aprendizaje supervisado se permite que los algoritmos «aprendan» de los datos y los apliquen a nuevas entradas desconocidas para obtener el resultado correcto. El entrenamiento de estos algoritmos precisa de una gran muestra de datos de entrada y de salida correctamente etiquetados.

1.2.1. Problemas resueltos mediante el aprendizaje supervisado

Los problemas que pueden ser resueltos mediante aprendizaje supervisado se descomponen fundamentalmente en dos bloques claramente diferenciados:

- Problemas de **clasificación**: En este tipo de problemas se entrena un algoritmo para clasificar los datos de entrada en variables discretas. Por ejemplo, la clasificación de un correo electrónico recién recibido como deseado o no deseado.
- Problemas de **regresión**: El objetivo es encontrar correlaciones entre las variables predictoras que permitan predecir la variable respuesta de tipo continuo. Por ejemplo, encontrar la función que sigue la distribución de una serie de puntos en el plano y, a partir de nuevos datos, predecir correctamente sus valores respuesta.

1.2.2. Algoritmos empleados en aprendizaje supervisado

Los siguientes algoritmos, que únicamente vamos a enumerar debido a que un desarrollo de los mismos excedería los requisitos de longitud del trabajo, son algunos de los más destacados dentro del aprendizaje supervisado: regresión lineal, regresión logística, algoritmo del vecino más próximo, Naïve Bayes, SVM, árboles de decisión y bosque aleatorio.

1.3. El aprendizaje no supervisado

El aprendizaje no supervisado constituye una rama dentro del aprendizaje automático donde el modelo se entrena ajustándose a las observaciones. En ocasiones, el algoritmo tendrá datos sin etiquetar a los cuales deberá dar sentido por sí mismo. Este tipo de algoritmos crean patrones de similitud entre datos de entrada, siendo su principal ventaja respecto al aprendizaje supervisado que la muestra puede crearse con muy poco esfuerzo y no se precisa del etiquetado los datos. Este tipo de algoritmos están cobrando una mayor relevancia [3] .

1.3.1. Problemas resueltos mediante el aprendizaje no supervisado

Los problemas que pueden ser resueltos mediante aprendizaje no supervisado son fundamentalmente de dos tipos:

- Problemas de **clustering** (agrupamiento)

  Pueden ser considerados los problemas más representativos del aprendizaje no supervisado. El objetivo es agrupar los datos según su similitud en *clusters*: colecciones de datos «similares» y diferentes de los datos contenidos en el resto de particiones. Por ejemplo: la segmentación de clientes en torno a los cuales se desarrollan campañas de marketing.

- Problemas de **reducción de la dimensionalidad**

  La reducción de la dimensionalidad es una de las técnicas clave dentro del aprendizaje no supervisado. Su propósito es comprimir la información aportada por los datos originales, encontrando un conjunto de variables diferente y más pequeño, que almacene la información más relevante de los datos originales, al tiempo que minimiza la pérdida de información propiciada por la transformación. Una de las principales ventajas de esta técnica es que ayuda a mitigar los problemas

asociados con datos de una alta dimensionalidad y permite la visualización de aspectos destacados de los mismos que, de otro modo, serían difíciles de explorar. Por ejemplo, el cálculo de distancias en un espacio euclideo N-dimensional, a un espacio 2-dimensional, de forma que las distancias se preservasen lo máximo posible.

1.3.2. Algoritmos empleados en aprendizaje no supervisado

Al igual que con los algoritmos de aprendizaje supervisado, los siguientes algoritmos son algunos de los más destacados dentro del aprendizaje no supervisado: K-medias, agrupación jerárquica, detección de anomalías, análisis de componentes principales y algoritmo a priori.

1.4. El aprendizaje por refuerzo

El aprendizaje por refuerzo es un método de entrenamiento de aprendizaje automático. En él se dirige el aprendizaje automático no supervisado mediante la recompensa de los comportamientos deseados y la penalización de los no deseados. Para ello, se desarrolla un método que asigne valores positivos a las acciones deseadas, con el fin de que en iteraciones sucesivas el algoritmo adopte esas acciones y, de manera análoga, se asocian valores negativos a los comportamientos a evitar. De este modo, se incentiva al algoritmo a que aprenda a rechazar las decisiones negativas y a buscar aquellas que resultan positivas. Eventualmente, el algoritmo obtiene la mayor recompensa posible, logrando alcanzar a largo plazo una solución óptima.

De acuerdo con investigadores de DeepMind, el aprendizaje por refuerzo puede conducir al desarrollo de la inteligencia artificial general: una inteligencia artificial cuyas capacidades cognitivas estén al mismo nivel que las del ser humano [4] .

1.4.1. Problemas resueltos mediante el aprendizaje por refuerzo

Dentro de la implementación de algoritmos de aprendizaje por refuerzo se distinguen, dependiendo de la política de recompensa, dos enfoques.

Por un lado, el **refuerzo positivo** que alienta una acción deseada que el algoritmo haya ejecutado, con el objetivo de aumentar la probabilidad de que vuelva a repetir el mismo comportamiento. En la vida cotidiana, el uso de premios en el adiestramiento de animales sería un claro ejemplo de esta política de aprendizaje.

Por otro lado, el **refuerzo negativo**. En él, se elimina una condición negativa, con el objetivo de que las posibilidades de la acción deseada aumenten. En el día a día, la decisión de una persona de salir antes del punto A para llegar puntual al punto B sería un exponente de aprendizaje por refuerzo negativo.

1.4.2. Algoritmos empleados en aprendizaje por refuerzo

Al igual que en los aprendizajes automáticos detallados anteriormente, algunos de los algoritmos más relevantes dentro del aprendizaje por refuerzo, son los siguientes: policy optimization, Q-learning, SARSA, temporal difference y MARL.

Capítulo 2

Redes neuronales

Una **red neuronal** es una herramienta dentro del campo de la inteligencia artificial a través de la cual las computadoras procesan datos siguiendo un proceso inspirado en cómo lo hace el propio cerebro humano. Este tipo de procesos dentro del machine learning se conocen como **aprendizaje profundo** *(deep learning)*. Para ello, se utilizan una serie de nodos interconectados en una estructura de capas similar al cerebro humano. Se crea así, un sistema adaptable bajo el cual el modelo puede aprender de sus errores y mejorar continuamente.

2.1. Funcionamiento del cerebro humano

El premio Nobel Ramón y Cajal descubrió a finales del siglo XIX la estructura celular del sistema nervioso y describió el funcionamiento de las neuronas. Gracias a él, y a los avances posteriores, sabemos que el cerebro humano está compuesto por alrededor de 10^{11} elementos, cada uno con aproximadamente 10^4 conexiones: las **neuronas**. A grandes rasgos, las neuronas están formadas por tres partes: las dendritas, el cuerpo celular y el axón. Las dendritas son estructuras nerviosas, distribuidas formando un patrón ramificado, encargadas de captar y transportar los impulsos eléctricos al cuerpo celular. El cuerpo celular es el responsable del procesamiento de estos impulsos eléctricos y de la elaboración de una respuesta acorde. Por último, el axón es una estructura nerviosa que sale del cuerpo de la neurona, con la finalidad de transmitir el impulso nervioso generado por el cuerpo celular a la siguiente neurona. Así mismo, el punto de contacto entre el axón de una neurona y las dendritas de la siguiente se llama sinapsis. La disposición de las neuronas y la fuerza de las sinapsis entre ellas, dadas por un complejo proceso químico, determinan la estructura y función de la red neuronal. La siguiente figura corresponde a un diagrama de dos neuronas biológicas.

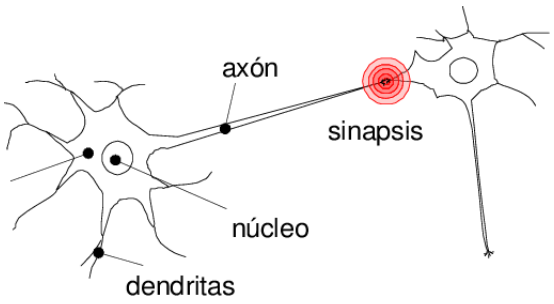


Figura 2.1: Conexión dos neuronas

Fuente: https://www.researchgate.net/figure/Figura-101-Modelo-biologico-que-representa-la-conexion-entre-dos-neuronas_fig18 _31759521

Parte de la estructura neuronal está predefinida al nacer; mientras que otras partes se desarrollan con el paso del tiempo, mediante el aprendizaje, a medida que se generan, y destruyen, conexiones entre neuronas. Este desarrollo es más perceptible en etapas tempranas de desarrollo. Las estructuras

neuronales se mantienen en perpetuo cambio a lo largo de la vida, consistiendo fundamentalmente en el fortalecimiento y/o debilitamiento de las uniones sinápticas. Los recuerdos se forman siguiendo este patrón [5] .

Las redes neuronales artificiales no se acercan a la complejidad del cerebro humano. No obstante, hay dos similitudes clave entre las redes neuronales biológicas y las artificiales.

- En primer lugar, el componente fundacional de ambas redes son elementos computacionales simples (aunque las neuronas artificiales son mucho más simples que las neuronas biológicas) que están interconectados.
- En segundo lugar, las conexiones sinápticas entre las neuronas determinan el objetivo de la red.

Cabe destacar que, aunque las neuronas biológicas son muy lentas si atendemos a sus impulsos eléctricos (10^{-3} segundos en el caso de la neurona biológica frente a 10^{-10} segundos en el caso de la neurona artificial), el cerebro humano es capaz de realizar un amplio abánico de tareas mucho más rápido que cualquier procesador convencional. Esto es así debido a las grandes estructuras en paralelo de las redes neuronales biológica; todas las neuronas operan al mismo tiempo. Las redes neuronales artificiales comparten esta misma arquitectura.

2.2. Neurona artificial

En 1943, el neurofisiólogo Warren McCulloch de la Universidad de Illinois y el matemático Walter Pitts de la Universidad de Chicago, publicaron un artículo en el que establecieron un procesador elemental como modelo de neurona, replicando las características clave de las neuronas biológicas. La neurona, a partir de un vector de entrada procedente bien del exterior o bien de otra neurona, proporciona una única respuesta o salida. Posteriormente, el 1949, el neuropsicólogo Donald Hebb definió un algoritmo para calcular el peso de la conexión neuronal dentro de la red. Con estas bases teóricas establecidas, en 1957, Frank Rosenblatt desarrolló el primer modelo computacional inspirado en el cerebro humano: el perceptrón.

2.2.1. Modelo genérico de neurona artificial

Los elementos que constituyen una neurona son los siguientes:

- **Entradas**: $x_j(t)$
- **Pesos sinápticos**: w_{ij}

  Representan la intensidad de la interacción entre la neurona presináptica j y la neurona postsináptica i.
- **Regla de propagación**: $\sigma(w_{ij}, x_j(t))$

  Proporciona el valor del potencial postsináptico, $h_i(t)$ de la neurona i en función de sus pesos y entradas. Es decir, $h_i(t) = \sigma(w_{ij}, x_j(t))$. La función de propagación más habitual es de tipo lineal, y se basa en una suma ponderada de las entradas con los pesos sinápticos, es decir, $h_i(t) = \sum_j w_{ij} x_j$.
- **Función de activación o de transferencia**: $f_i(a_i(t-1), h_i(t))$

  Proporciona el estado de activación actual $a_i(t)$ de la neurona i en función de su estado anterior, $a_i(t-1)$ y de su potencial postsináptico actual $h_i(t)$. En la mayor parte de los modelos de redes neuronales, se considera que el estado de activación actual $a_i(t)$ depende únicamente de su potencial postsináptico actual $h_i(t)$. La función de activación se suele considerar determinista y en la mayoría de los modelos es monótona, creciente y continua.

- **Función de salida**: $F_i(a_i(t))$

  Proporciona la salida actual $y_i(t)$ de la neurona i en función de su estado de activación actual $a_i(t)$. En la mayoría de los casos, se toma simplemente la identidad, de forma que el estado de activación de la neurona es la propia salida.

De esta forma, la operación realizada por la neurona i puede expresarse como:

$$y_i(t) = F_i(f_i(a_i(t-1), \sigma(w_{ij}, x_j(t))))$$

2.2.2. Modelo estándar de neurona artificial

Si se considera que la regla de propagación sea la suma ponderada y que la función de salida sea la identidad, la neurona estándar está formada entonces por:

- Un conjunto de entradas $x_j(t)$
- Unos pesos sinápticos w_{ij} asociados a las entradas
- Una regla de propagación $h_i(t) = \sum_j w_{ij} x_j$
- Una función de activación $y_i(t) = f_i(h_i(t))$ que representa simultáneamente la salida de la neurona y su estado de activación.

En ocasiones, junto con el conjunto de pesos sinápticos se aporta un parámetro adicional θ_i, el umbral, que será restado del potencial postsináptico. De este modo, el argumento de la función de activación resulta:

$$\sum_j w_{ij} x_j - \theta_i$$

La siguiente figura ilustra el modelo en cuestión:

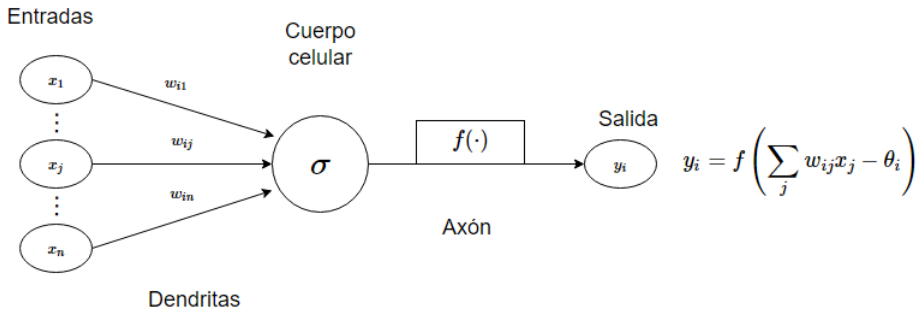
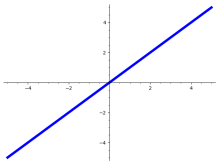
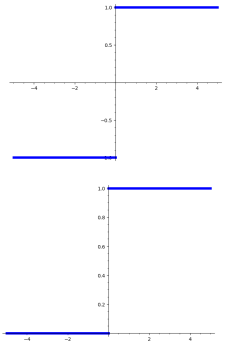
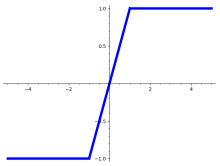
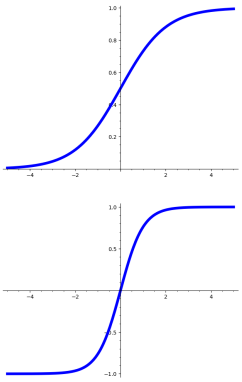
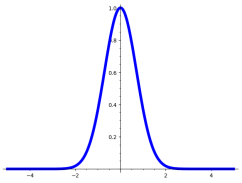
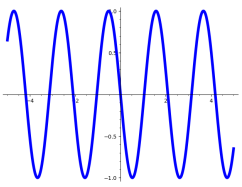


Figura 2.2: Modelo neurona artificial

2.2.3. Funciones de activación más habituales

Las funciones de activación $f(\cdot)$ más habituales son las siguientes:

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty,+\infty]$	
Escalón	y=sign(x) y=H(x)	{-1,1} {0,1}	
Lineal a trozos	$y = \begin{cases} -1, \text{si } x < -1 \\ x, \text{ si } -1 \leq x \leq 1 \\ 1, \text{ si } x > 1 \end{cases}$	$[-1,1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = \tanh(x)$	[0,1] [-1,1]	
Gaussiana	$y = Ae^{-Bx^2}$	[0,1]	
Sinusoidal	$y = \sin(\omega x + \psi)$	$[-1,1]$	

2.3. Arquitectura de redes neuronales

Se denomina arquitectura de la red neuronal a la estructura bajo la cual están asociadas las múltiples neuronas que constituyen la red. En las redes neuronales artificiales, los nodos se conectan por medio de sinapsis, siendo esta estructura de conexiones la que determinará el comportamiento de la red. Las conexiones sinápticas son **unidireccionales**, es decir, la información sólo fluye en un sentido, desde la neurona presináptica a la neurona postsináptica. Además, el tipo de conexión entre neuronas puede ser una **conexión excitatoria**, si el peso sináptico asociado es positivo, o una **conexión inhibitoria**, si el peso sináptico es negativo.

Las **capas** son las unidades estructurales bajo las cuales se agrupan las neuronas.En ellas, las neuronas que la conforman trabajan en paralelo. Dentro de una misma capa las neuronas pueden encontrarse agrupadas formando grupos neuronales, normalmente de neuronas del mismo tipo. El conjunto de una o más capas constituye una **red neuronal**.

Las capas, dependiendo de su posicionamiento dentro de la red, pueden ser clasificadas en los siguientes tres grupos:

- **Capa de entrada**: Está compuesta por aquellas neuronas que reciben datos de entrada.
- **Capa de salida**: Está compuesta por aquellas neuronas que proporcionan los datos de salida de la red.
- **Capa oculta**: Esta capa está situada entre las capas de entrada y de salida. Por lo tanto, está compuesta por aquellas neuronas que no tienen una conexión directa con el entorno, es decir, aquellas que ni reciben los datos iniciales ni transmiten los datos finales.

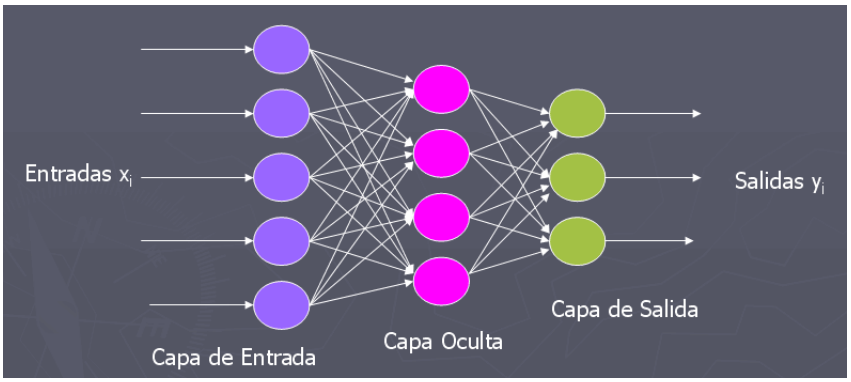


Figura 2.3: Diagrama red neuronal

Fuente: https://rodas5.us.es/file/7352f327-eeab-83c2-d362-8e7f2be2170f/1/redes_neuronales_artificiales_SCORM.zip/page_08.htm

A su vez, las conexiones entre neuronas también pueden clasificarse atendiendo a si pertenecen a una misma capa o no. En caso de ser conexiones entre neuronas de una misma capa, estas se denominan **intracapa**, mientras que si son entre neuronas de diferentes capas se denominan **intercapa**.

2.3.1. ¿Qué es una red neuronal?

Una **red neuronal** es un grafo con las siguientes propiedades:

- Cada nodo i tiene asociado un estado x_i
- Cada conexión de una pareja de nodos i, j tiene asociado un peso $w_{ij} \in \mathbb{R}$
- Cada nodo i tiene asociado un umbral θ_i
- Cada nodo i tiene asociada una función $f_i(x_j, w_{ij}, \theta_i)$ que depende de los pesos de las conexiones, del umbral y del estado de los nodos j conectados al nodo i. Esta función es la encargada de proporcionar el nuevo estado del nodo.

2.3.2. Número de neuronas en las capas

A la hora de diseñar una red neuronal, es necesario indicar el número de neuronas que ha de tener cada capa. Tanto en la capa de entrada como en la capa de salida es sencillo: la capa de entrada tendrá tantas neuronas como variables y, en la capa de salida, dependerá de si se trata de un problema de regresión o de clasificación. Si se trata de un problema de regresión, la capa de salida tendrá una única neurona; mientras que si se trata de un problema de clasificación, tendrá tantas neuronas como etiquetas diferentes contenga dicho problema.

Sin embargo, la elección del número de capas ocultas, así como de su número de neuronas, resulta más complejo.

De acuerdo con Jeff Heaton [6], el número de capas ocultas tiende a oscilar entre 0 y 2 para los problemas más comunes, resultando:

- **0 capas ocultas**: La red sólo será capaz de representar funciones o decisiones lineales separables.
- **1 capa oculta**: La red podrá aproximar cualquier función que contenga una aplicación continua de un espacio finito a otro.
- **2 capas ocultas**: La red podrá representar una decisión límite arbitraria con precisión arbitraria con funciones de activación racionales y será capaz de aproximar cualquier aplicación diferenciable con cualquier precisión.

En cuanto al número de neuronas por capa oculta, no existe un criterio unificado para discernir cual es el número idóneo. Estos son algunos de los criterios más seguidos:

- El número de neuronas ocultas debe de estar entre el número de neuronas de la capa de entrada y el número de neuronas de la capa de salida.
- El número de neuronas ocultas debe ser $\frac{2}{3}$ del número de neuronas de la capa de entrada, más el número de neuronas de la capa de salida.
- El número de neuronas ocultas debe de ser menor o igual que el doble de neuronas de la capa de entrada.

Aparte de estos criterios generales, existe cierto consenso relativo a la inferencia de los propios datos sobre la disposición de las capas intermedias. Por lo tanto, la cantidad de neuronas de las capas ocultas, así como el número de capas, también dependen de los datos de entrenamiento, la cantidad de valores atípicos, la complejidad de los datos a aprender y del tipo de funciones de activación empleadas.

Como criterio de partida, la mayoría de problemas pueden ser resueltos mediante una única capa oculta, formada por un número de neuronas igual a la media entre el número de neuronas de la capa de entrada y el número de salida. Un número de neuronas menor producirá un ajuste insuficiente y con un alto sesgo; mientras que un número demasiado elevado de neuronas podría provocar un sobreajuste, aumentando la varianza de la red y su tiempos de entrenamiento.

2.3.3. Tipos de redes neuronales

Dependiendo de la forma en la que las neuronas se encuentran asociadas, se determinan diferentes tipos de redes neuronales:

- **Redes neuronales monocapa**: Como su nombre indica, es una red constituida por una única capa de neuronas, que proyecta los datos de entrada a una capa de neuronas de salida donde se realizan los cálculos pertinentes. Es el modelo más sencillo de red neuronal.
- **Redes neuronales multicapa**: Son una generalización del concepto anterior, obtenidas añadiendo capas a la red neuronal. En ellas, puede existir un conjunto de capas ocultas, es decir, una serie de capas intermedias entre la capa de entrada y la capa de salida. En este tipo de red, las neuronas pueden estar total o parcialemente conectadas con el resto de integrantes de la red.

- **Redes neuronales recurrentes**: Su diferencia fundamental con los tipos anteriores radica en la existencia de lazos de realimentación en la red. Estos lazos pueden ser entre neuronas de distintas capas, de una misma capa o incluso de una neurona consigo misma. Destacan en el estudio de sistemas no lineales.

Atendiendo a los modelos de neurona que se utilicen, a la arquitectura o topología de la red e incluso al propio algoritmo de aprendizaje escogido, surgen distintos modelos de redes neuronales.

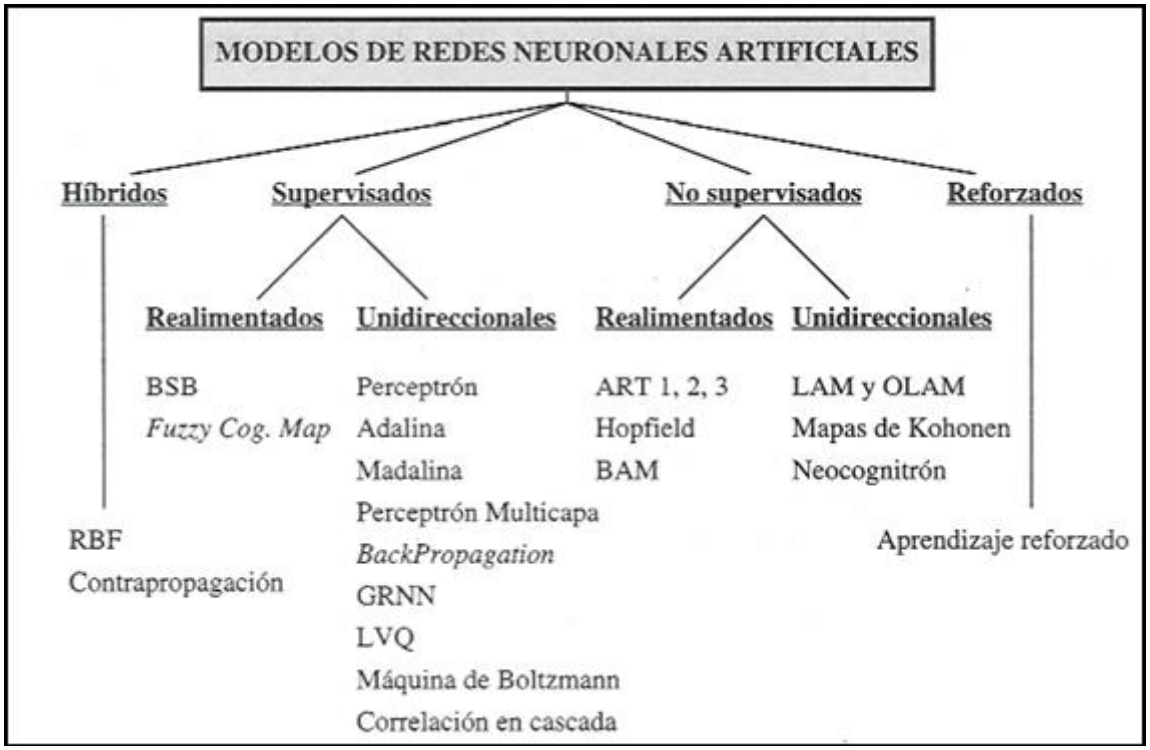


Figura 2.4: Principales modelos de redes neuronales

Fuente: http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm#estructuraANS

2.4. Funciones de pérdida

A la hora de entrenar una red neuronal surge la siguiente pregunta: ¿Cómo elegir de entre distintas redes neuronales cuál es mejor? La forma más habitual de obtener una respuesta a esta cuestión es a través de minimizar una función objetivo, llamada **función de pérdida** o **función de coste**, siendo su resultado la **pérdida** o **error**. A través de esta función se puede comprobar qué tan bien la red neuronal modela los datos dados. La elección de la función de pérdida no es baladí; hay varios factores que influyen en su elección, como la naturaleza del problema de aprendizaje automático, la facilidad de cálculo de derivadas o la distribución de datos atípicos, entre otros. Idealmente la función ha de cumplir las siguientes propiedades:

- Debe reflejar el objetivo que el modelo está tratando de lograr. Así, en problemas de regresión, el objetivo es minimizar las diferencias entre las predicciones y los valores objetivo; mientras que en los problemas de clasificación el objetivo es minimizar el número de errores en la clasificación.

- Debe ser continua e infinitamente diferenciable. Esto es debido a la naturaleza de los optimizadores, que se verán más adelante.

- Debe ser convexa. Una función convexa sólo tiene un mínimo global, lo que garantiza que los optimizadores hallen la solución óptima global. En la práctica esta propiedad es muy difícil de lograr.
- Debe ser simétrica. El error por encima del valor objetivo debería de causar la misma pérdida que el error por debajo del objetivo.
- Debe ser rápida de calcular.

A continuación expondremos algunas de las funciones de pérdida más empleadas, dependiendo de si se trata de problemas de regresión o de clasificación.

2.4.1. Error cuadrático medio

El **error cuadratico medio** o **MSE** (*Mean Squared Error*) es una de las funciones de pérdida más habituales en problemas de regresión. Consiste en calcular la media de las diferencias al cuadrado entre los valores respuesta teóricos y obtenidos, es decir:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

donde hay n muestras, $y^{(i)}$ son los valores respuesta teóricos e $\hat{y}^{(i)}$ son los valores respuesta obtenidos.

El hecho de calcular así las diferencias anula la problemática de si el valor obtenido está por encima o por debajo del valor teórico. Sin embargo, los valores con un gran error son penalizados. El error cuadrático medio también es una función convexa, con un mínimo global claramente definido.

La principal desventaja de esta función de pérdida es su extrema sensibilidad a los valores atípicos; un valor obtenido muy alejado del valor teórico aumentará ostensiblemente el error.

2.4.2. Error absoluto medio

El **error absoluto medio** o **MAE** (*Mean Absolute Error*) es una función de pérdida alternativa al error cuadrático medio, también empleada en problemas de regresión. Calcula la diferencia en valor absoluto entre los valores respuesta teóricos y los obtenidos, es decir:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y^{(i)} - \hat{y}^{(i)}|$$

donde las variables son las mismas que en la función anterior.

El error absoluto medio es menos susceptible a los valores atípicos, dado que, a diferencia de en el error cuadrático medio, el factor es lineal, no cuadrático.

Sin embargo, presenta otras desventajas, destacando sus limitaciones de derivabilidad.

2.4.3. Pérdida de Huber

La **función de pérdida de Huber** surge como una alternativa al error cuadrático medio. Combina las ventajas tanto del MSE como del MAE definiéndose de la siguiente manera:

$$\begin{cases} \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 & |y^{(i)} - \hat{y}^{(i)}| \leq \delta \\ \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y^{(i)} - \hat{y}^{(i)}|\right) - \frac{1}{2}\delta^2 & |y^{(i)} - \hat{y}^{(i)}| > \delta \end{cases}$$

donde δ es un valor umbral previamente establecido, crítico debido a que determina que es considerado un valor atípico.

Mediante esta definición, si la diferencia en valor absoluto es menor que el valor umbral, se emplea el error cuadrático medio. En caso contrario, la función empleada tiene un comportamiento similar al error absoluto medio. De esta forma se corrigen los principales problemas de ambas funciones anteriormente

mencionadas. Así mismo, previa elección de δ, la función de pérdida de Huber puede ser menos sensible a los valores atípicos que el MSE. Debido a esto, si los datos son propensos a contener valores atípicos, la función de pérdida de Huber resulta una mejor elección.

2.4.4. Entropía cruzada binaria

Dentro de los problemas de clasificación destaca la **entropía cruzada binaria** (*Binary Cross Entropy*). Esta función se usa cuando sólo hay dos posibles resultados en la clasificación. Viene dada por:

$$-\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)}\log\hat{y}^{(i)}+\left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right)$$

Es habitual emplearla en combinación con la función de activación 'Sigmoide' en la capa de salida de la red neuronal.

2.4.5. Entropía cruzada categórica

En caso de haber más de dos posibles resultados en la clasificación se emplea la **entropía cruzada categórica** (*Categorical Cross Entropy*). Su fórmula viene dada por:

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m}y_{ij}\log\hat{y}^{(ij)}$$

Es habitual emplearla en combinación con la función de activación 'SoftMax' en la capa de salida de la red neuronal.

2.5. Algoritmos de aprendizaje

Desde una perspectiva matemática, entrenar una red neuronal consiste en hallar una serie de valores que permitan minimizar una función: la función de pérdida de la sección anterior. El proceso de obtener los valores ideales del modelo neuronal es abrumador, debido a la gran cantidad de parámetros susceptibles de ser ajustados, y ajustar los valores de forma manual supone una tarea titánica. Para ello, se recurre a una serie de algoritmos encargados de seleccionar los mejores pesos, conocidos como **optimizador**. Este algoritmo optimizador es un método de optimización encargado de mejorar el rendimiento de la red neuronal. Durante el entrenamiento de la red, es el encargado de modificar los pesos cada época en su intento de minimizar la función de pérdida. Así, relacionan la función de pérdida con los nuevos valores de los pesos, actualizando el modelo en función de los resultados obtenidos por la función de pérdida. A través de la función de pérdida el optimizador puede discernir si el entrenamiento está mejorando o empeorando y actuar en consecuencia. A continuación detallaremos algunos de los algoritmos de aprendizaje automático más habituales.

2.5.1. SGD

Recordemos el **descenso por gradiente** o GD (*Gradient Descent*). Es un algoritmo que estima numéricamente donde se encuentran los mínimos locales de una función. Para ello se selecciona un peso inicial w_0 y, en cada iteración, actualizamos:

$$w_{i+1} = w_i - \eta_i \nabla f(w_i)$$

Los pesos son actualizados usando el valor inmediatamente anterior. La elección del peso inicial determina el mínimo local que será encontrado. También es necesario establecer un criterio de parada. El criterio más habitual es situarlo en $\nabla f(w_i)$, de manera que el algortimo pare cuando $\nabla f(w_i)$ sea muy pequeño y el valor w_i apenas esté variando. El factor η_i es conocido como ratio de aprendizaje (*learning rate*). No existe un criterio exacto para su elección. Sin embargo, valores muy grandes pueden provocar que nunca

se alcance una solución, mientras que valores muy pequeños pueden ocasionar que la convergencia sea extremadamente lenta.

Por otra parte, los **gradientes estocásticos** son gradientes imprecisos, es decir, son valores aproximados del verdadero gradiente $\nabla f(\cdot)$. En redes neuronales, el gradiente estocástico empleado es el gradiente resultante de computar el gradiente sobre un subconjunto de los datos, en vez de sobre el conjunto total. Se emplean debido a su eficiencia tanto computacional como en términos de memoria necesaria.

En estas condiciones, se introduce el **descenso por gradiente estocástico** o **SGD** (*Stochastic Gradient Descent*). Formalmente, es un método iterativo para optimizar una función objetivo con propiedades de suavidad adecuadas (por ejemplo ser diferenciable). Puede ser considerada una aproximación estocástica de la optimización basada en descenso del gradiente, ya que reemplaza el gradiente real (calculado a partir de todo el conjunto de datos), por una estimación del mismo (calculado a partir de un subconjunto de datos seleccionados al azar). En problemas de optimización de alta dimensión, esto reduce enormemente la carga computacional, logrando iteraciones más rápidas a cambio de una tasa de convergencia más baja.

La idea básica detrás de la aproximación estocástica corresponde al algoritmo Robbins-Monro de los años 50. No obstante, y a pesar de su antigüedad, el descenso estocástico por gradiente sigue siendo un método de optimización relevante dentro del aprendizaje automático. Cabe destacar que a diferencia de un descenso por gradiente, el descenso estocástico por gradiente no reduce el valor de la pérdida en cada iteración.

Por lo tanto, fijado un peso inicial w_0, en cada iteración resulta:

$$w_{i+1} = w_i - \eta \nabla_{mb} f(w_i)$$

donde η es el ratio de aprendizaje y $\nabla_{mb} f(\cdot)$ es el gradiente estocástico sobre un subconjunto de datos elegidos al azar (*mini-batch*). El algoritmo recorre el conjunto de datos, realizando la actualización de los valores correspondiente. Puede ser necesario que recorra el conjunto de datos varias veces hasta obtenerse la convergencia. En esa situación, los datos pueden ser barajados para evitar la formación de ciclos. Cabe destacar que en SGD la convergencia puede resultar más suave que en GD, debido a que el gradiente se promedia en cada paso sobre más muestras de entrenamiento.

En cuanto a la convergencia del SGD, ésta ha sido estudiada mediante teorías de minimación convexa y de aproximación estocástica. En líneas generales, cuando los ratios de aprendizaje η decrecen a un ritmo apropiado, y bajo unas hipótesis generales, el descenso por gradiente estocástico converge casi seguro a un mínimo global, siempre que la función objetivo sea convexa o pseudoconvexa. De lo contrario converge casi seguro a un mínimo local. (Teorema de Robbins-Siegmund).

2.5.2. Momentum

El descenso por gradiente estocástico con momento o **Momentum**, es una extensión del SGD. Está diseñado para acelerar el proceso de optimización, disminuyendo el número de evaluaciones de funciones requeridas para alcanzar los óptimos, y para mejorar la capacidad del algoritmo de optimización, alcanzando mejores resultados finales.

La naturaleza del SGD permite la posibilidad de que la búsqueda del óptimo «rebote» por todo el espacio de búsqueda en función del gradiente. Por ejemplo, la búsqueda puede progresar en dirección al mínimo, pero verse desviada en una dirección diferente debido al gradiente de puntos específicos encontrados durante el proceso de búsqueda. Esto puede relantizar la velocidad de convergencia. Una solución a esta problemática es la ofrecida por Momentum: incluir un historial de actualizaciones de los parámetros en función del gradiente encontrado en actualizaciones pasadas

El algoritmo consta de dos fases. Primero se calcula el cambio de posición, $\Delta(w_i)$:

$$\Delta(w_{i+1}) = \alpha \Delta(w_i) - \eta \nabla f(w_i)$$

.

A continuación, se actualiza la posición anterior como una combinación lineal entre el gradiente y el estado previo:

$$w_{i+1} = w_i + \Delta(w_i)$$

Juntando ambas fases se obtiene la siguiente fórmula:

$$w_{i+1} = w_i - \eta \nabla f(w_i) + \alpha \Delta(w_i)$$

donde el parámetro w que minimiza $f(w)$ ha de ser estimado, η es el ratio de aprendizaje y α es el factor de momento, un factor de decaimiento exponencial entre 0 y 1, por lo general muy cercano a 1, que determina la contribución relativa del gradiente actual y los gradientes anteriores a la variación del peso.

Este algoritmo cobra una especial importancia en problemas donde la función objetivo tiene una gran curvatura (el gradiente puede cambiar mucho en regiones relativamente pequeñas del espacio de búsqueda). También resulta de utilidad cuando el espacio de búsqueda es «plano», es decir, el gradiente es prácticamente nulo en su totalidad. La naturaleza del algoritmo permite que la búsqueda progrese en la dirección anterior a ingresar en la región del espacio plana, siendo capaz de cruzar la región en la búsqueda del óptimo.

El nombre *momentum* surge de una analogía con la física. El vector de peso w viaja por el espacio de parámetros. A diferencia del SGD, éste tiende a seguir viajando en la misma dirección, evitando oscilaciones. El método está íntimamente relacionado con la dinámica de Langevin subamortiguada y puede ser combinado con el recocido simulado.

2.5.3. AdaGrad

El algoritmo de gradiente adaptativo o **AdaGrad** (*Adaptative Gradient Algorithm*) es una modificación del SGD, con un ratio de aprendizaje por parámetro. Fue publicado en el artículo «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization» por Duchi, Hazan y Singer en el año 2010. A grandes rasgos, aumenta el ratio de aprendizaje de los parámetros más dispersos y reduce el ratio de aprendizaje de los parámetros con valores más próximos entre sí. A menudo, esta estrategia aumenta el rendimiento de la convergencia en comparación con el SGD, en las situaciones en las que el número de datos es reducido. Se parte de un ratio de aprendizaje fijo η, pero para cada componente es multiplicado por los elementos de $\{G_{j,j}\}$, los cuales conforman la diagonal de la matriz:

$$G = \sum_{i=1}^{n} g_i g_i^T$$

donde $g_i = \nabla f(w)$ es el gradiente en la iteración i-ésima. Así, los elementos de la diagonal están dados por :

$$G_{j,j} = \sum_{i=1}^{n} g_{i,j}^2$$

La fórmula de la actualización resulta:

$$w_j^{(t+1)} = w_j^{(t)} - \frac{\eta}{\sqrt{G_{j,j}}} g_j$$

Cada $G_{j,j}$ da lugar a un factor de escala para la ratio de aprendizaje que se aplica sobre un único parámetro w_j. Mediante esta fórmula, las actualizaciones extremas de parámetros quedan amortiguadas, mientras que los parámetros que reciben menos actualizaciones gozan de ratios de aprendizaje más elevados.

Entre otras ventajas del algoritmo, destaca su nulo requerimiento de un ratio de aprendizaje óptimo inicial y que, en términos generales, su convergencia es más rápida que la de su predecesor, el algoritmo SGD.

Su principal desventaja es que, en ocasiones, los ratios de aprendizaje descienden drásticamente debido a la acumulación de los gradientes desde el principio del entrenamiento. Debido a esto, se alcanza un instante a partir del cual el modelo deja de ser capaz de aprender, pues los ratios de aprendizaje son prácticamente nulos.

2.5.4. RMSProp

El algoritmo **RMSProp** (*Root Mean Square Propagation*) es una variación del AdaGrad propuesta por Geoffrey Hinton. El objetivo es reducir la «agresividad» de AdaGrad en su política de variación de ratios de aprendizaje. Para ello, en lugar de mantener el acumulado de los gradientes, se utiliza el concepto de «ventana» para considerar sólo los gradientes más recientes. Sobre estos gradientes se aplica una media exponencial ponderada para suavizar los cambios aplicados a los parámetros, a diferencia de la suma acumulativa de gradientes al cuadrado empleada en AdaGrad. Mediante este enfoque, en ocasiones, se puede evitar que los ratios de aprendizaje se vuelvan excesivamente pequeños. Al igual que en el AdaGrad, el ratio de aprendizaje está adaptado a cada parámetro.

En primer lugar, se calculan las medias exponenciales de los gradientes como:

$$v(w,t) \leftarrow \gamma v(w,t-1) + (1-\gamma)(\nabla L_i(w))^2$$

donde γ es el coeficiente de olvido y $L_i(\cdot)$ es la función de pérdida asociada a cada parámetro.

A continuación, se tiene que:

$$\Delta(w_t) = -\frac{\eta}{\sqrt{v(w,t)}}\nabla L_i(w)$$

Finalmente,

$$w_{t+1} = w_t + \Delta(w_t)$$

Así, la fórmula de actualización es de la forma:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v(w,t)}}\nabla L_i(w)$$

Como algoritmo derivado de AdaGrad, comparte sus principales ventajas. Además, no tiene una caída del aprendizaje tan brusca y es capaz de evitar el sobreajuste. No obstante, y a diferencia de AdaGrad, resulta un algoritmo más lento que SGD en su búsqueda del mínimo global.

2.5.5. Adam

El algoritmo **adam** (*Adaptative Moment Estimation*) es una mejora del RMSProp, introducido en la conferencia ICLR 2015. A diferencia del RMSProp, en lugar de adaptar los ratios de aprendizaje en función del primer momento promedio, se utiliza el promedio de los momentos de orden dos. Por lo tanto, se emplean tanto los gradientes como sus momentos de orden dos. En este algoritmo de optimización, dados parámetros $w^{(t)}$ y una función de pérdida $L^{(t)}$, donde t indexa la iteración de entrenamiento (comenzando en 0), la actualización del parámetro de Adam viene dada por:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2$$

$$m_w = \frac{m_w^{(t+1)}}{1-\beta_1^T}$$

$$v_w = \frac{v_w^{(t+1)}}{1-\beta_2^T}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{m_w}{\sqrt{v_w}+\varepsilon}$$

donde ε es un escalar positivo para evitar la división por cero y β_1 y β_2 son los coeficientes de olvido para los gradientes y sus segundos momentos respectivamente.

Capítulo 3

Aplicación de la IA en Fórmula 1

3.1. ¿Por qué Fórmula 1?

Fórmula 1, la cúspide de la automoción. Una serie de pilotos a los mandos de los monoplazas tecnológicamente más avanzados del mundo compitiendo entre ellos para alzarse con la victoria. Una competición milmillonaria, con una logística gigantesca y con legiones de seguidores alrededor de los cinco continentes.

Desde sus orígenes en 1950 hasta los años 80, los únicos indicadores que llevaba incorporado el monoplaza eran un velocímetro, un cuentarrevoluciones y un medidor del nivel de combustible o del aceite, siendo el piloto la única persona responsable de su control. Sin embargo, en 1980, Karl Kempf, ingeniero del equipo Tyrrell, diseñó un sistema formado por un ordenador y una serie de sensores para controlar la suspensión electrónica del monoplaza Tyrrell 010. Aunque no llegó a competir con el sistema a bordo, debido a su alta complejidad, en los siguientes años de esa década se perfeccionó el sistema, dando lugar a la pugna entre los equipos Williams y McClaren, en la década de los 90. En 1993, debido al auge de prestaciones de los coches, la Federación Internacional de Automovilismo (FIA), máximo organismo rector en competiciones automovilísticas, decidió prohibir las ayudas electrónicas dirigidas a facilitar la conducción, no así la recopilación de datos por los cada vez más numerosos sensores.

En 2001, de la mano de TAG Electronics, se introdujo la telemetría bidireccional, permitiendo a los ingenieros no sólo recibir información del coche desde el muro de control, sino también modificar la configuración de éste, en tiempo real, mientras se encuentra en pista. En 2002, se logró modificar el mapeado del motor y activar o desactivar determinados sensores desde el pit, siendo nuevamente prohibida esta injerencia externa al piloto en 2003, aunque permaneció en vigor la prohibición de ayudas electrónicas al piloto. En **2007**, cada monoplaza contaba con alrededor de **120 sensores**, recopilando unos **20 Gigabytes de datos** por carrera, acumulando los equipos 7 Terabytes de datos al finalizar la temporada [7].

En **2022**, cada monoplaza cuenta con **entre 150 y 300 sensores**, recopilando hasta 1.1 millones de datos por segundo [8], generándose unos **300 Gigabytes de datos** por carrera y coche. Estos datos, junto con el resto de datos recopilados por los equipos, ascienden a cifras de entre 40 y 50 Terabytes de datos semanales [9].

En la actualidad, el análisis e interpretación de estos datos por parte de los equipos supone la diferencia entre el éxito y el fracaso, en una industria tan competitiva. Los equipos destinan abundantes recursos, tanto humanos como económicos, a su análisis, con el objetivo de obtener una ventaja competitiva sobre sus rivales, ya sea a través de implementaciones técnicas o, como se pretende en el presente trabajo, mediante una mejor toma de decisiones en carrera, en concreto definiendo una mejor estrategia.

La estrategia de carrera se fundamenta en dar respuesta a **dos cuestiones clave**: **qué neumáticos iniciales ha de llevar el coche** y **cuándo deben de ser cambiados**. Debido al estrés al que están sometidos los mismos, durante la carrera sufren una degradación muy significativa, siendo preciso su cambio tanto por cuestiones de seguridad (posibilidad de un reventón o pérdida de control) como por cuestiones puramente competitivas (si un rival ha cambiado neumático y está con gomas más frescas, en cada vuel-

ta va recortar tiempo y se corre el riesgo de ser adelantados por él o, si está por delante, que aumente su ventaja). **La naturaleza de los datos de libre acceso de los que disponemos, obliga a centrarse en esta segunda situación**; es decir, tratar de encontrar las ventanas óptimas de cambio de neumáticos dependiendo del compuesto inicial, así como las distintas estrategias que se puedan seguir.

3.2. Red neuronal para el estudio de la elección de los neumáticos

Para responder a la cuestión formulada anteriormente, se programa una red neuronal en Python, mediante TensorFlow y Keras, dos librerías de código abierto especializadas en el aprendizaje automático y las redes neuronales respectivamente.

3.2.1. Creación del dataset

Los datos se obtienen a través de FastF1, un paquete de Python que permite acceder y analizar los resultados, horarios, datos de tiempo y telemetría de la Fórmula 1. Debido a la naturaleza de las APIS a través de las cuales se obtienen los resultados, las sesiones anteriores a 2018 se encuentran incompletas. Los formatos no coinciden, dado que hay parámetros que se registran en la actualidad que no se hacían con anterioridad. También hay numerosos errores de registro de tiempos por vuelta y velocidades, así como de compuesto de neumático empleado. Debido a ello, el dataset consta únicamente de las carreras transcurridas entre 2018 y 2022.

De esta forma, tras realizar un split 80-20 entre datos de entrenamiento y validación, quedan conformados dos conjuntos de datos: el bloque de entrenamiento, compuesto por las carreras de 2018, 2019, 2020 y 2021; y el bloque de validación, compuesto por la carrera de 2022.

Una vez definidos los conjuntos de datos, se realiza un análisis exploratorio de los datos. A la vista de los resultados del análisis, se decide que entre las 27 variables reflejadas en los datos y que conforman los conjutnos de datos, las variables más representativas son ocho: **'LapNumber'** (número de vuelta), **'SpeedI1'** (velocidad máxima alcanzada en la *speed trap* 1), **'SpeedI2'** (velocidad máxima alcanzada en la *speed trap* 2) , **'SpeedFL'** (velocidad máxima alcanzada en la línea de meta) , **'SpeedST'** (velocidad máxima alcanzada en la recta principal), **'TyreLife'** (número de vueltas de uso de los neumáticos), **'Stint'** (número de veces que se han cambiado los neumáticos) y **'Tiempo_en_segundos'** (tiempo por vuelta en segundos, obtenido previa tranformación del formato original del dataset).

Tras el análisis exploratorio y la elección de las variables más representativas, se procede al tratamiento de los valores nulos. Estos valores no están contemplados en la API, bien por un fallo de esta, bien por un error de algún sensor encargado de obtenerlos en la carrera. Su tratamiento resulta vital ya que, en caso de no tratarlos, se generarán problemas en el funcionamiento de la red y no será posible extraer ningún tipo de conclusiones relacionadas con la precisión o fiabilidad de la red.

Los datos nulos están concentrados únicamente en las cuatro variables de velocidades registradas ('SpeedI1', 'SpeedI2', 'SpeedFL', 'SpeedST'). Lo ideal sería encontrar los valores exactos de los datos no registrados. Sin embargo, estos datos no son de fácil acceso, ya que la FIA sólo publica las velocidades máximas registradas por cada piloto en toda la carrera. Por ello, y a riesgo de perder fidelidad del modelo, se opta por sustituir los valores nulos por el promedio de los valores inmediatamente anteriores y posteriores.

Una vez realizado todo este tratamiento previo sólo queda identificar las vueltas en las que se ha producido una parada en boxes para realizar un cambio de neumáticos y, una vez halladas, identificar tanto esa vuelta como las 2 vueltas anteriores y posteriores como cambio de neumáticos. La necesidad de realizar este «incremento» de los datos es debido a cuestiones de índole técnica; ya que, únicamente con los datos de la parada real, la red tiene grandes problemas a la hora de entender la mecánica del cambio de neumáticos. Gracias a considerar el entorno de la parada como cambio de neumáticos, la red, como se ha comprobado empíricamente, produce mejores resultados. Así, quedan conformadas cuatro etiquetas: **0** (no hacer nada), **1** (cambiar a neumático SOFT), **2** (cambiar a neumático MEDIUM) y **3** (cambiar a neumático HARD), dependiendo de las posibles decisiones a ser tomadas.

3.2.2. Arquitectura de la red

La naturaleza del dataset determina la estructura de la capa de entrada y de la capa de salida. La capa inicial está formada por 8 neuronas, una por cada variable del dataset, con función de activación 'ReLu' (*Rectified Lineal Unit*): función más habitual tanto en capas de entrada como ocultas, destacando que no activa todas las neuronas al mismo tiempo y que es computacionalmente más eficiente que sus contrapartes que activan todas las neuronas. A su vez, la capa final está compuesta por 4 neuronas, cada una de las posibles cuatro decisiones, con función de activación 'SoftMax' : función exponencial normalizada habitualmente utilizada en la capa final de los clasificadores basados en redes neuronales. En cuanto a la existencia de capas ocultas, se incluye una, compuesta por 16 neuronas y con función de activación 'ReLu'.

Por otra parte, como función de pérdida se ha empleado la función 'categorical-crossentropy' (entropía categórica cruzada, determinada por la naturaleza del problema) y como optimizador se ha seleccionado 'adam' (es el optimizador que mejores resultados produce dada la naturaleza de los datos y la evidencia empírica).

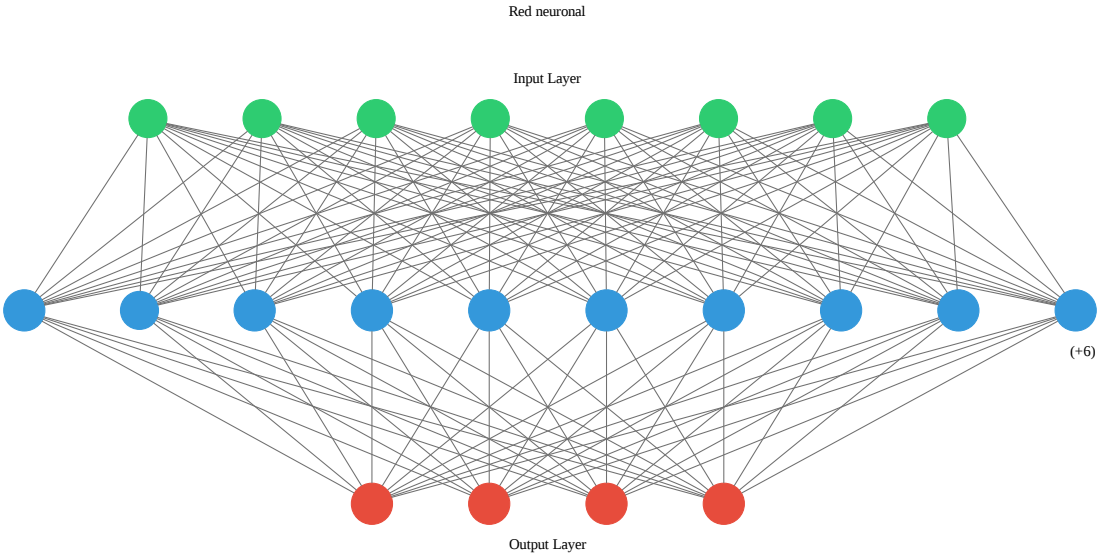


Figura 3.1: Arquitectura del modelo

3.2.3. Resultados de la red

La red se ha entrenado durante 100 épocas. Las métricas del resultado final dentro del entrenamiento son las siguientes:

```
 Epoch 100/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5573 -
    accuracy: 0.7231 - val_loss: 0.4762 - val_accuracy: 0.7558
```

Resulta notable la precisión mostrada dentro de los datos de entrenamiento, superior al 70 %.

Por su parte, al aplicar el modelo a los datos de validación obtenemos los siguientes resultados:

```
38/38 [==============================] - 0s 2ms/step - loss: 0.8886 - accuracy:
    0.5860
[0.8886395692825317, 0.5859504342079163]
```

La precisión mostrada (0.5860) indica que en el 58.60 % de los casos, la decisión tomada por la red concuerda con la decisión tomada en la competición.

La siguiente gráfica muestra cómo ha evolucionado la pérdida tanto de los valores de entrenamiento como de los valores de validación a lo largo de las 100 épocas:

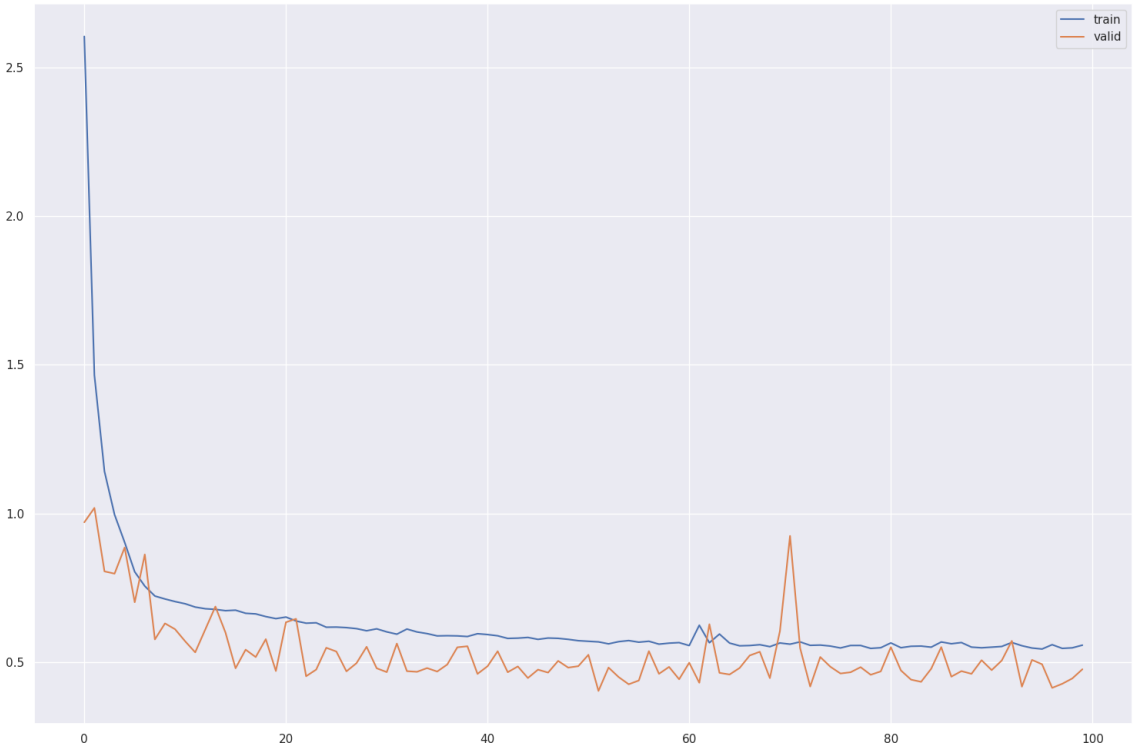


Figura 3.2: Evolución del aprendizaje del modelo

Finalmente, obtenemos la siguiente matriz de confusión:

```
38/38 [==============================] - 0s 1ms/step
=== Confusion Matrix ===
tf.Tensor(
[[  0   0   0   0]
 [  0 269 395   0]
 [  0  71 440   0]
 [  0   1  34   0]], shape=(4, 4), dtype=int32)
```

La interpretación de la matriz de confusión es la siguiente:

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 269 & 395 & 0 \\ 0 & 71 & 440 & 0 \\ 0 & 1 & 34 & 0 \end{pmatrix}$$

- El eje X representa la toma de decisiones de la red neuronal, mientras que el eje Y representa la toma de decisiones por parte de los equipos durante la carrera.
- De las $341 = 269 + 71 + 1$ veces en que la red ha escogido el neumático SOFT, en la realidad los equipos de Fórmula 1 eligieron 269 veces el neumático SOFT como el adecuado. En los otros 71 casos, 70 veces fue el neumático MEDIUM y 1 única vez el neumático HARD.
- De las $869 = 395 + 440 + 34$ veces en que la red ha escogido el neumático MEDIUM, en la realidadlos equipos de Fórmula 1 eligieron 440 veces el neumático MEDIUM como el adecuado. En los 429 casos restantes, fue empleado en 395 ocasiones el neumático SOFT como el neumático idóneo, mientras que en las otras 34 se empleo el neumático HARD.

- Finalmente, la red nunca se decanta por el uso de neumático HARD.

- Cabe destacar también que del total de 1210 ocasiones, la red ha elegido neumático SOFT en 341 de ellas y el neumático MEDIUM en otras 869. El neumático HARD nunca lo ha tomado en consideración.

3.2.4. Conclusiones de la implementación

A la vista de los resultados anteriores, podemos establecer las siguientes conclusiones:

- En el 78.89 % de los casos, el cambio a neumático SOFT, por parte de la red, coincide con el cambio ocurrido en la realidad.

- En el 50.63 % de los casos, el cambio a neumático MEDIUM, por parte de la red, coincide con el cambio ocurrido en la realidad.

- La red no es capaz de predecir correctamente el cambio a neumático HARD. Esto es debido a lo inusual de esa elección de neumáticos; sólo fueron elegidos por cuatro pilotos en 2019 y un piloto en 2020 (ver estrategias de carrera en esos años). A este respecto, cabe destacar que la elección en la carrera pudo deberse a razones ajenas al rendimiento puro. En caso de ser una carrera sin lluvia, como mínimo es obligatorio realizar un cambio de compuesto de neumático, habiendo tres disponibles: SOFT, MEDIUM y HARD. En determinadas ocasiones, los equipos pueden no disponer de neumáticos nuevos del compuesto idóneo, habiéndolo usado durante las tres sesiones de entrenamientos libres. Debido a ello, pueden verse obligados a optar por un neumático de un compuesto diferente para cumplir la regla impuesta. En otras ocasiones, puede haber habido un abandono o un accidente que haya forzado la salida a pista del *Safety Car*- un vehículo especial empleado para agrupar al pelotón de pilotos y dar tiempo a los comisarios a limpiar de escombros la sección del circuito afectada- que prohibe adelantar, mientras está circulando, y obliga a los monoplazas a circular a velocidades más reducidas que durante la competición. En consecuencia, los neumáticos sufren más, al circular en condiciones diferentes a aquellas para las que fueron diseñados, y pueden forzar un cambio de neumáticos antes de tiempo o bien alterar la estrategia de carrera de un equipo e intentar defender su posición actual, optando por no parar o reducir el número de paradas a realizar.

- La red tiene tendencias a asignar neumáticos MEDIUM cuando se seleccionó neumático SOFT y viceversa. Esto puede ser explicado por la extrema igualdad de los tiempos entre ambos compuestos. Su principal diferencia se encuentra en su composición y en el umbral en el que dejan de ser capaces de ofrecer tiempos por vuelta competitivos. Dado que los datos proceden de las propias carreras, los equipos siempre realizan el cambio de neumáticos dentro de la vida útil del mismo, ergo nunca se producen grandes diferencias entre ambos compuestos en sus respectivos tiempos y velocidades alcanzadas y, por lo tanto, resultan extremadamente similares para la red.

3.2.5. Propuestas de mejora en un futuro

Aunque la red neuronal desarrollada en esta misma sección cumple su objetivo, dista de ser perfecta. A continuación se presentan algunas propuestas de mejora para un futuro:

- En primer lugar, el hecho de que sólo se puedan usar las carreras a partir de 2018 ha sido un factor limitativo. En un futuro se podrían intentar solucionar los problemas de las APIS anteriores al 2018 y así aumentar el número de datos. Igualmente, conforme pasen los años, mayor número de carreras se acumularán con la API y formato de los datos actuales.

- En segundo lugar, los datos han sido un factor problemático. Al ser datos de índole pública, estos apenas contienen información relevante sobre los equipos, ni se asemejan a los datos empleados por las divisiones de estrategia de los equipos de Fórmula 1. Los datos empleados por los equipos incluyen variables tales como la presión de los neumáticos, flujos de combustible dentro del bloque motor, temperaturas en diversas partes: frenos, motor, neumáticos, asfalto... Estos datos gozan de una mayor riqueza y, por lo tanto, el acceso a los mismos resulta, en el mejor de los casos, limitado. Es posible que la red neuronal obtuviese mejores resultados, en cuanto a precisión se refiere, en caso de poder entrenarla con conjuntos de datos «profesionales».

- En tercer lugar, no hemos sido capaces de extraer información de las tres sesiones de entrenamientos libres que se realizan previas a cada carrera, al igual que de la clasificación de la misma. En caso de conseguir idear un sistema para aprovechar los datos de las cuatro sesiones previas a la disputa de la carrera, tal vez fuese posible gozar de mayor precisión e incluso ser capaces de aventurar estrategias de carrera viables, en vez de limitarnos a confirmar las estrategias de carrera seguidas por parte de los equipos.

- Finalmente, con más reflexión y estudio, sería posible aumentar el número de datos disponibles, realizando una simulación de carreras realista y comparar en ella diferentes estrategias para poder discernir si existe una estrategia superior a las demás. A la vista de las estrategias seguidas en las carreras, la estrategia idónea parece ser la siguiente: Comenzar la carrera con neumático SOFT y realizar un cambio de compuesto a MEDIUM. En ocasiones, si el neumático desfallece, finalizar con un cambio de compuesto adicional a SOFT. En cuanto al número de vueltas, parece oscilar entre las 10-20 vueltas para SOFT y las 20-30 vueltas para MEDIUM, en función de factores ambientales.

Anexos

Apéndice A

Anexo I: Estrategias de carrera

En el presente anexo se muestran las estrategias de carrera seguidas durante los 5 años contemplados en el estudio. El neumático SOFT se representa en rojo, el neumático MEDIUM en amarillo y el neumático HARD en blanco. Los pilotos aparecen en orden de finalización de la carrera.

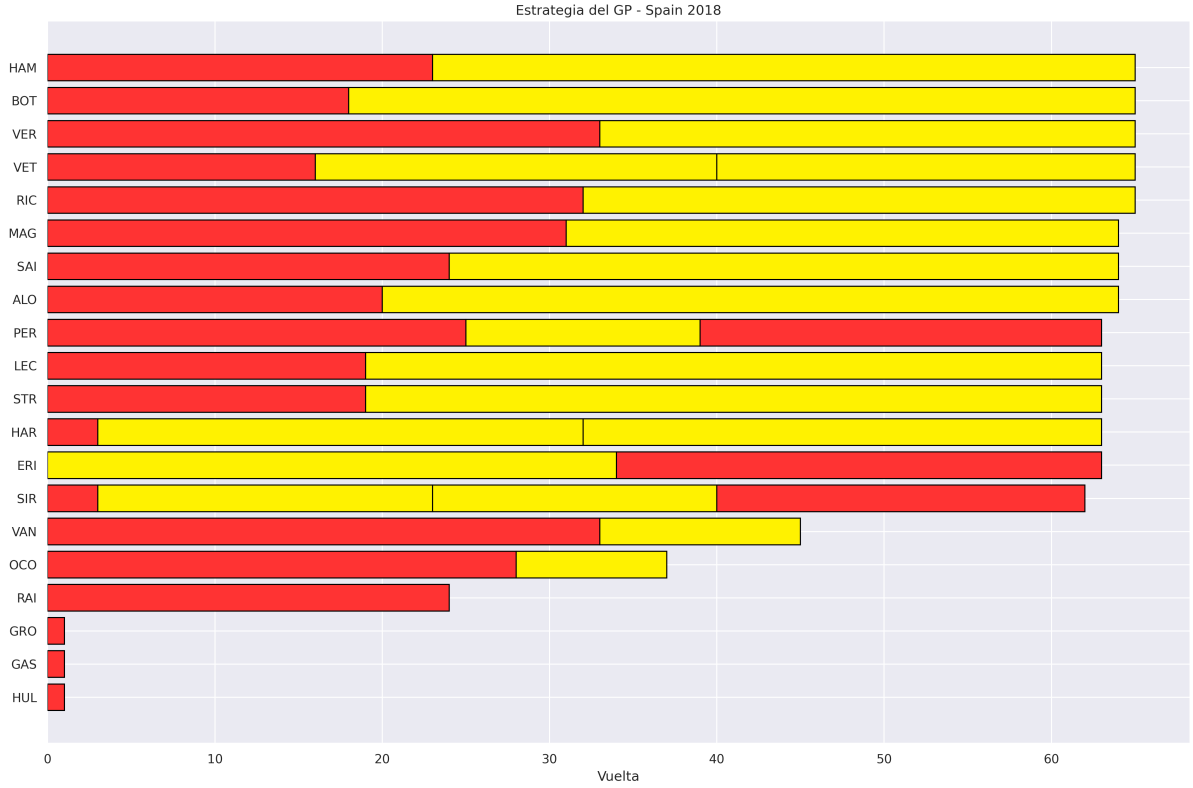


Figura A.1: Estrategia GP España 2018

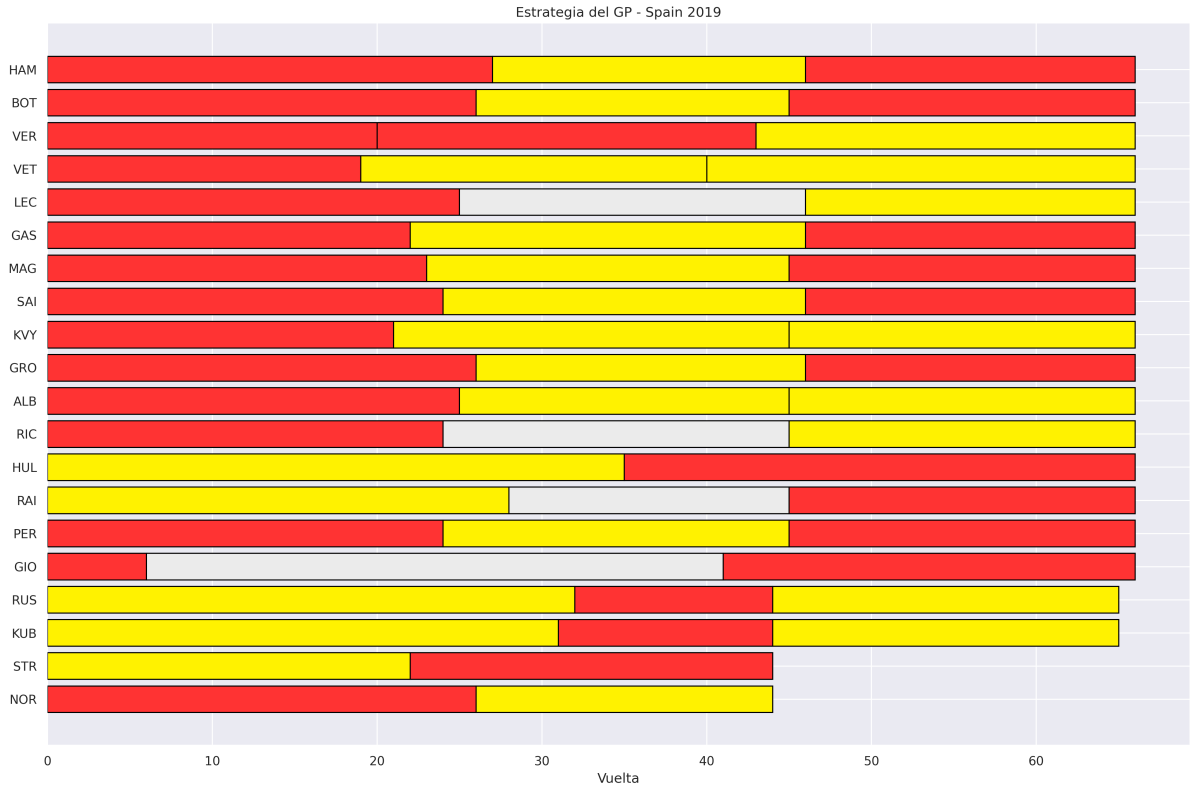


Figura A.2: Estrategia GP España 2019

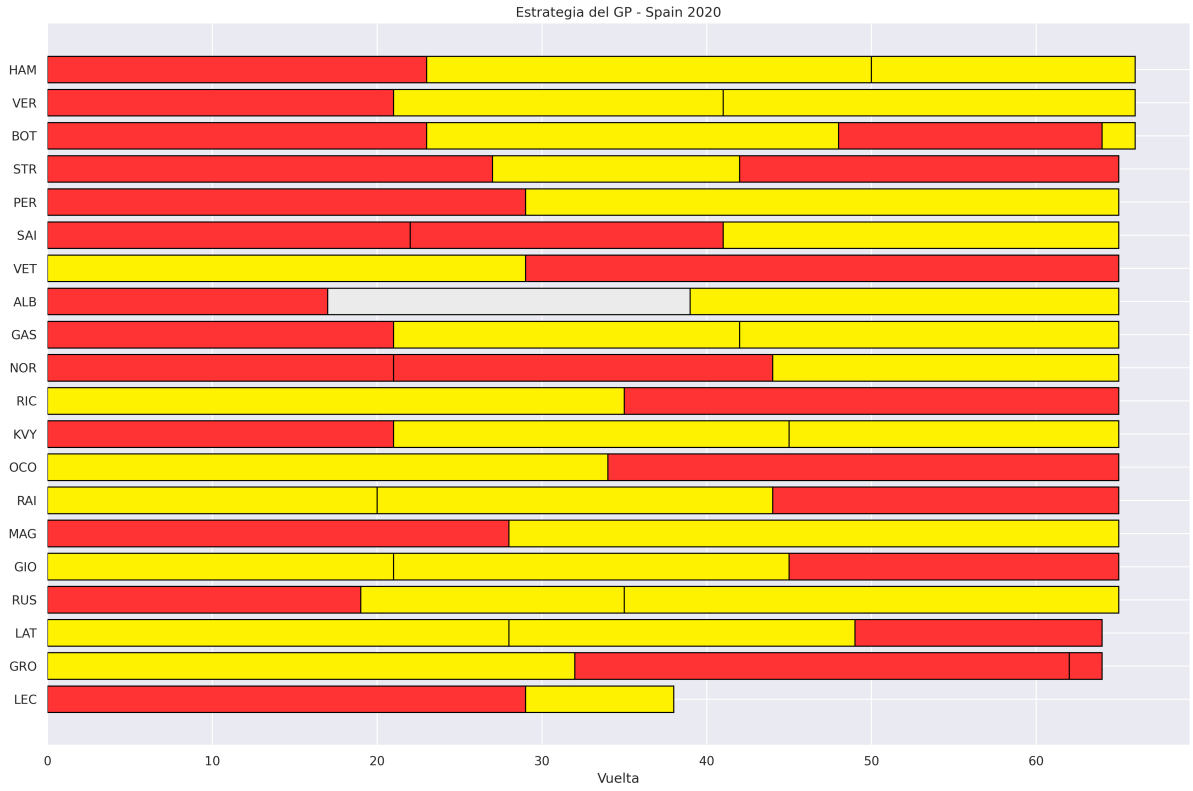


Figura A.3: Estrategia GP España 2020

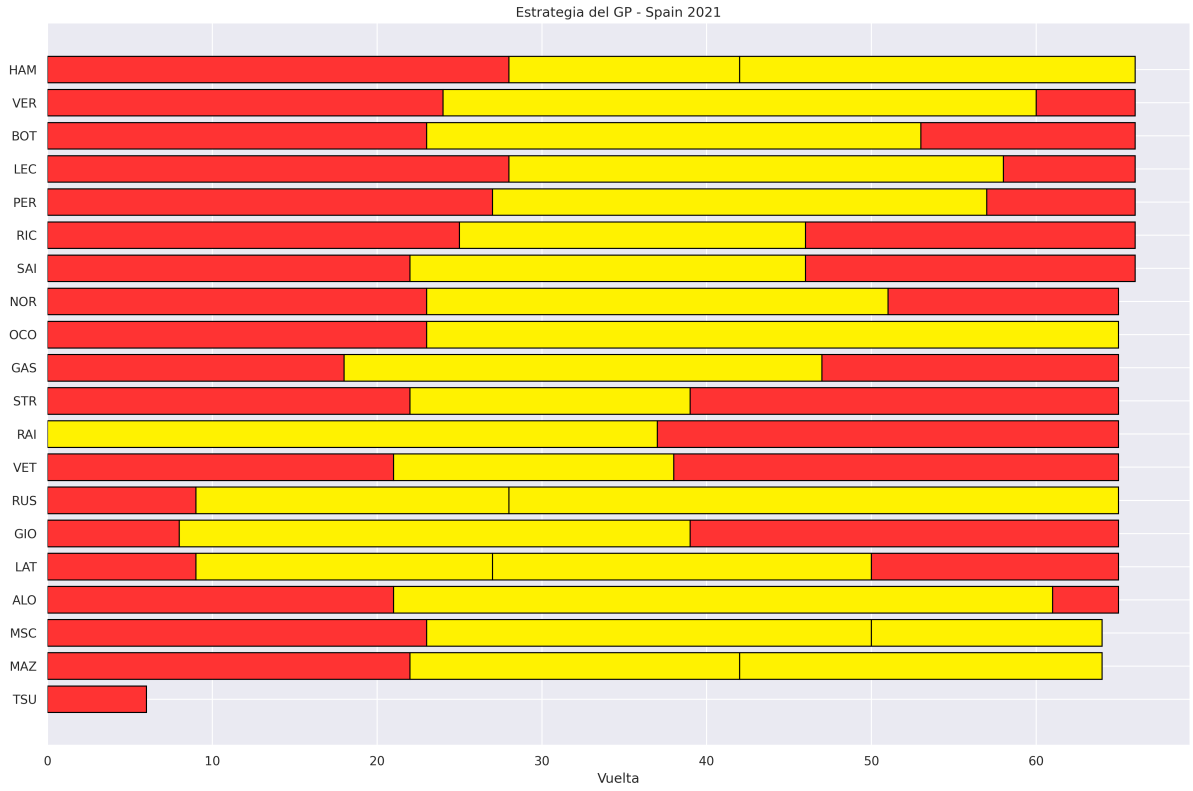


Figura A.4: Estrategia GP España 2021

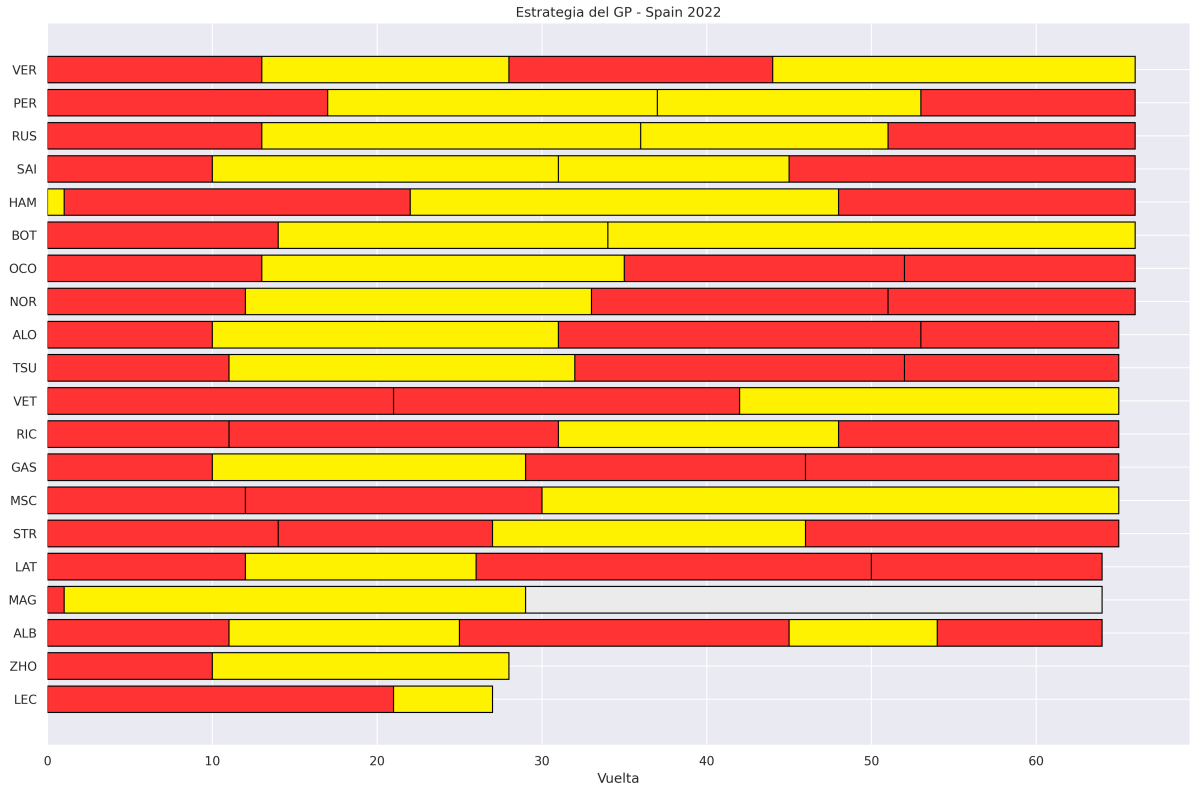


Figura A.5: Estrategia GP España 2022

Apéndice B

Anexo II: Código empleado

```
1
2   #Your google drive is made accesible to Colab.
3 try:
4     from google.colab import drive
5     drive.mount('/content/drive')
6     # The working directory
7     %cd /content/drive/MyDrive/TFG
8     %ls -lht
9     # To import own packages set local path in packages syspath
10    import sys
11    sys.path.insert(0,"./")
12 except ImportError:
13    print("You are not in google.colab!!")
14    pass
15
16 pip install fastf1
17
18 pip install Scikit-learn
19
20 pip install tensorflow
21
22 import fastf1 as ff1
23 from fastf1 import plotting
24 from matplotlib import pyplot as plt
25 from matplotlib.pyplot import figure
26 import numpy as np
27 import pandas as pd
28 %matplotlib inline
29 import seaborn as sns
30 sns.set_theme()
31
32 from sklearn import preprocessing
33 from sklearn.model_selection import train_test_split
34 from sklearn.decomposition import PCA
35 from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
36 from sklearn.metrics import recall_score, f1_score, precision_recall_curve, auc
37
38 import tensorflow as tf
39 from tensorflow import keras
40 from tensorflow.keras import layers
41 from keras.layers import Dense, Dropout, Flatten
42 from keras.utils import to_categorical
43
44 from tensorflow.keras import models
45 from tensorflow.keras import optimizers
46 import math as math
47
```

```
ff1.Cache.enable_cache("/content/drive/MyDrive/TFG")

circuit = 'Spain'

# Cargamos datos de la carrera
race2018 = ff1.get_event(2018, circuit)
race2019 = ff1.get_event(2019, circuit)
race2020 = ff1.get_event(2020, circuit)
race2021 = ff1.get_event(2021, circuit)
race2022 = ff1.get_event(2022, circuit)

carrera2018=race2018.get_session('R')
carrera2019=race2019.get_session('R')
carrera2020=race2020.get_session('R')
carrera2021=race2021.get_session('R')
carrera2022=race2022.get_session('R')

carrera2018.load()
carrera2019.load()
carrera2020.load()
carrera2021.load()
carrera2022.load()

vueltas2018=carrera2018.laps
vueltas2019=carrera2019.laps
vueltas2020=carrera2020.laps
vueltas2021=carrera2021.laps
vueltas2022=carrera2022.laps

vueltas=pd.concat([vueltas2018,vueltas2019,vueltas2020,vueltas2021,vueltas2022])

vueltas.drop(vueltas[vueltas['LapTime'].isna()].index, inplace = True)

vueltas['Tiempo_en_segundos']=vueltas[:5608]['LapTime'] / np.timedelta64(1, 's')
vueltas['Tiempo_sector1_en_segundos']=vueltas[:5608]['Sector1Time'] / np.
    timedelta64(1, 's')
vueltas['Tiempo_sector2_en_segundos']=vueltas[:5608]['Sector2Time'] / np.
    timedelta64(1, 's')
vueltas['Tiempo_sector3_en_segundos']=vueltas[:5608]['Sector3Time'] / np.
    timedelta64(1, 's')

vueltas["Valores"]=0

vueltas['Valores'] = np.where((vueltas['TyreLife'] == 1) & (vueltas['Compound']
    == 'sUPERSOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 1) & (vueltas['Compound']
    == 'SOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 1) & (vueltas['Compound']
    == 'MEDIUM'), 2, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 1) & (vueltas['Compound']
    == 'HARD'), 3, vueltas['Valores'])

vueltas['Valores'] = np.where((vueltas['TyreLife'] == 2) & (vueltas['Compound']
    == 'SUPERSOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 2) & (vueltas['Compound']
    == 'SOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 2) & (vueltas['Compound']
    == 'MEDIUM'), 2, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 2) & (vueltas['Compound']
    == 'HARD'), 3, vueltas['Valores'])

vueltas['Valores'] = np.where((vueltas['TyreLife'] == 3) & (vueltas['Compound']
    == 'SUPERSOFT'), 1, vueltas['Valores'])
```

```
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 3) & (vueltas['Compound'] == 'SOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 3) & (vueltas['Compound'] == 'MEDIUM'), 2, vueltas['Valores'])
vueltas['Valores'] = np.where((vueltas['TyreLife'] == 3) & (vueltas['Compound'] == 'HARD'), 3, vueltas['Valores'])

vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-1) & (vueltas['Compound'] == 'SUPERSOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-1) & (vueltas['Compound'] == 'SOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-1) & (vueltas['Compound'] == 'MEDIUM'), 2, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-1) & (vueltas['Compound'] == 'HARD'), 3, vueltas['Valores'])

vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-2) & (vueltas['Compound'] == 'SUPERSOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-2) & (vueltas['Compound'] == 'SOFT'), 1, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-2) & (vueltas['Compound'] == 'MEDIUM'), 2, vueltas['Valores'])
vueltas['Valores'] = np.where(((vueltas['TyreLife'] == 1)-2) & (vueltas['Compound'] == 'HARD'), 3, vueltas['Valores'])

vueltas.info()

# selecting numerical columns, in this case all the 5 existing columns
all_col = vueltas.select_dtypes(include=np.number).columns.tolist()
#num_plots = len(all_col)
plt.figure(figsize=(15,7))

sns.heatmap(vueltas[all_col].corr(),
            annot=True,
            linewidths=0.5,vmin=-1,vmax=1,
            center=0,
            cbar=True,)

plt.show()

#DIBUJOS ESTRATEGIAS (Desaparicion neumaticos SUPERSOFT y ULTRASOFT en 2018)

compound_colors = {
    #'SUPERSOFT': '#8C004B',
    'SUPERSOFT': '#FF3333',
    'SOFT': '#FF3333',
    'MEDIUM': '#FFF200',
    'HARD': '#EBEBEB',
    'INTERMEDIATE': '#39B54A',
    'WET': '#00AEEF',
}

driver_stints2018 = vueltas2018[['Driver', 'Stint', 'Compound', 'LapNumber']].groupby(['Driver', 'Stint', 'Compound']).count().reset_index()
driver_stints2018 = driver_stints2018.rename(columns={'LapNumber': 'StintLength'})

driver_stints2018 = driver_stints2018.sort_values(by=['Stint'])
#Iniciamos el dibujo
plt.rcParams["figure.figsize"] = [15, 10]
plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()
```

```

#Nos ordena por orden de finalizacion
for driver in carrera2018.results['Abbreviation']:
    #Seleccionamos los stints del piloto en cuestion
    stints = driver_stints2018.loc[driver_stints2018['Driver'] == driver]

    #Loopeamos todos los stints del piloto y generamos barra horizontal para cada stint
    previous_stint_end = 0
    for _, stint in stints.iterrows():
        plt.barh(
            [driver],
            stint['StintLength'],
            left=previous_stint_end,
            color=compound_colors[stint['Compound']],
            edgecolor = "black"
        )

        previous_stint_end = previous_stint_end + stint['StintLength']

# Titulo
plt.title(f'Estrategia del GP - {circuit} {2018}')

# Etiquetamos eje x
plt.xlabel('Vuelta')

# Invertimos eje y
plt.gca().invert_yaxis()

# Quitamos bordes
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

#Guardamos y mostramos
plt.savefig('strategy_spain_2018.png', dpi=300)

plt.show()

driver_stints2019 = vueltas2019[['Driver', 'Stint', 'Compound', 'LapNumber']].groupby(['Driver', 'Stint', 'Compound']).count().reset_index()
driver_stints2019 = driver_stints2019.rename(columns={'LapNumber': 'StintLength'})

driver_stints2019 = driver_stints2019.sort_values(by=['Stint'])
#Iniciamos el dibujo
plt.rcParams["figure.figsize"] = [15, 10]
plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()

#Nos ordena por orden de finalizacion
for driver in carrera2019.results['Abbreviation']:
    #Seleccionamos los stints del piloto en cuestion
    stints = driver_stints2019.loc[driver_stints2019['Driver'] == driver]

    #Loopeamos todos los stints del piloto y generamos barra horizontal para cada stint
    previous_stint_end = 0
    for _, stint in stints.iterrows():
        plt.barh(
            [driver],
            stint['StintLength'],
```

```
            left=previous_stint_end,
            color=compound_colors[stint['Compound']],
            edgecolor = "black"
        )

        previous_stint_end = previous_stint_end + stint['StintLength']

# Titulo
plt.title(f'Estrategia del GP - {circuit} {2019}')

# Etiquetamos eje x
plt.xlabel('Vuelta')

# Invertimos eje y
plt.gca().invert_yaxis()

# Quitamos bordes
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

#Guardamos y mostramos
plt.savefig('strategy_spain_2019.png', dpi=300)

plt.show()

driver_stints2020 = vueltas2020[['Driver', 'Stint', 'Compound', 'LapNumber']].
    groupby(['Driver', 'Stint', 'Compound']).count().reset_index()
driver_stints2020 = driver_stints2020.rename(columns={'LapNumber': 'StintLength'
    })

driver_stints2020 = driver_stints2020.sort_values(by=['Stint'])
#Iniciamos el dibujo
plt.rcParams["figure.figsize"] = [15, 10]
plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()

#Nos ordena por orden de finalizacion
for driver in carrera2020.results['Abbreviation']:
    #Seleccionamos los stints del piloto en cuestion
    stints = driver_stints2020.loc[driver_stints2020['Driver'] == driver]

    #Loopeamos todos los stints del piloto y generamos barra horizontal para
    cada stint
    previous_stint_end = 0
    for _, stint in stints.iterrows():
        plt.barh(
            [driver],
            stint['StintLength'],
            left=previous_stint_end,
            color=compound_colors[stint['Compound']],
            edgecolor = "black"
        )

        previous_stint_end = previous_stint_end + stint['StintLength']

# Titulo
plt.title(f'Estrategia del GP - {circuit} {2020}')

# Etiquetamos eje x
plt.xlabel('Vuelta')

```

```
# Invertimos eje y
plt.gca().invert_yaxis()

# Quitamos bordes
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

#Guardamos y mostramos
plt.savefig('strategy_spain_2020.png', dpi=300)

plt.show()

driver_stints2021 = vueltas2021[['Driver', 'Stint', 'Compound', 'LapNumber']].
    groupby(['Driver', 'Stint', 'Compound']).count().reset_index()
driver_stints2021 = driver_stints2021.rename(columns={'LapNumber': 'StintLength'
    })

driver_stints2021 = driver_stints2021.sort_values(by=['Stint'])
#Iniciamos el dibujo
plt.rcParams["figure.figsize"] = [15, 10]
plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()

#Nos ordena por orden de finalizacion
for driver in carrera2021.results['Abbreviation']:
    #Seleccionamos los stints del piloto en cuestion
    stints = driver_stints2021.loc[driver_stints2021['Driver'] == driver]

    #Loopeamos todos los stints del piloto y generamos barra horizontal para
    cada stint
    previous_stint_end = 0
    for _, stint in stints.iterrows():
        plt.barh(
            [driver],
            stint['StintLength'],
            left=previous_stint_end,
            color=compound_colors[stint['Compound']],
            edgecolor = "black"
        )

        previous_stint_end = previous_stint_end + stint['StintLength']

# Titulo
plt.title(f'Estrategia del GP - {circuit} {2021}')

# Etiquetamos eje x
plt.xlabel('Vuelta')

# Invertimos eje y
plt.gca().invert_yaxis()

# Quitamos bordes
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

#Guardamos y mostramos
plt.savefig('strategy_spain_2021.png', dpi=300)

plt.show()

```

```
driver_stints2022 = vueltas2022[['Driver', 'Stint', 'Compound', 'LapNumber']].
    groupby(['Driver', 'Stint', 'Compound']).count().reset_index()
driver_stints2022 = driver_stints2022.rename(columns={'LapNumber': 'StintLength'
    })

driver_stints2022 = driver_stints2022.sort_values(by=['Stint'])
#Iniciamos el dibujo
plt.rcParams["figure.figsize"] = [15, 10]
plt.rcParams["figure.autolayout"] = True

fig, ax = plt.subplots()

#Nos ordena por orden de finalizacion
for driver in carrera2022.results['Abbreviation']:
    #Seleccionamos los stints del piloto en cuestion
    stints = driver_stints2022.loc[driver_stints2022['Driver'] == driver]

    #Loopeamos todos los stints del piloto y generamos barra horizontal para
    cada stint
    previous_stint_end = 0
    for _, stint in stints.iterrows():
        plt.barh(
            [driver],
            stint['StintLength'],
            left=previous_stint_end,
            color=compound_colors[stint['Compound']],
            edgecolor = "black"
        )

        previous_stint_end = previous_stint_end + stint['StintLength']

# Titulo
plt.title(f'Estrategia del GP - {circuit} {2022}')

# Etiquetamos eje x
plt.xlabel('Vuelta')

# Invertimos eje y
plt.gca().invert_yaxis()

# Quitamos bordes
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['left'].set_visible(False)

#Guardamos y mostramos
plt.savefig('strategy_spain_2022.png', dpi=300)

plt.show()

#Aqui creo las particiones de entrenamiento y validacion y hago el procesado
    descrito anteriormente.

vueltas_training=pd.concat([vueltas2018,vueltas2019,vueltas2020,vueltas2021])

vueltas_training.drop(vueltas_training[vueltas_training['LapTime'].isna()].index
    , inplace = True)

vueltas_training.isna().sum()

for i in range(1,len(vueltas_training)):
  if math.isnan(vueltas_training[i:i+1].SpeedI2):
    a=float(vueltas_training['SpeedI2'][i-1:i])
```

```
    b=float(vueltas_training['SpeedI2'][i+1:i+2])
    c=(a+b)/2
    vueltas_training['SpeedI2'][i:i+1]=c

for i in range(1,len(vueltas_training)):
  if math.isnan(vueltas_training[i:i+1].SpeedI1):
    a1=float(vueltas_training['SpeedI1'][i-1:i])
    if i==len(vueltas_training)-1:
      b1=a1
    else:
      b1=float(vueltas_training['SpeedI1'][i+1:i+2])
    c1=(a1+b1)/2
    vueltas_training['SpeedI1'][i:i+1]=c1

for i in range(1,len(vueltas_training)):
  if math.isnan(vueltas_training[i:i+1].SpeedFL):
    aF=float(vueltas_training['SpeedFL'][i-1:i])
    bF=float(vueltas_training['SpeedFL'][i+1:i+2])
    cF=(aF+bF)/2
    vueltas_training['SpeedFL'][i:i+1]=cF

for i in range(1,len(vueltas_training)):
  if math.isnan(vueltas_training[i:i+1].SpeedST):
    aS=float(vueltas_training['SpeedST'][i-1:i])
    bS=float(vueltas_training['SpeedST'][i+1:i+2])
    if math.isnan(vueltas_training[i+1:i+2].SpeedST):
      bS=float(vueltas_training['SpeedST'][i+2:i+3])
    if math.isnan(vueltas_training[i-1:i].SpeedST):
      aS=float(vueltas_training['SpeedST'][i-2:i-1])
    if math.isnan(vueltas_training[i+2:i+3].SpeedST):
      bS=float(vueltas_training['SpeedST'][i+3:i+4])
    if math.isnan(vueltas_training[i-2:i-1].SpeedST):
      aS=float(vueltas_training['SpeedST'][i-3:i-2])
    if math.isnan(vueltas_training[i+3:i+4].SpeedST):
      bS=float(vueltas_training['SpeedST'][i+4:i+5])
    if math.isnan(vueltas_training[i-3:i-2].SpeedST):
      aS=float(vueltas_training['SpeedST'][i-4:i-3])
    if math.isnan(vueltas_training[i+4:i+5].SpeedST):
      bS=float(vueltas_training['SpeedST'][i+5:i+6])
    if math.isnan(vueltas_training[i-4:i-3].SpeedST):
      aS=float(vueltas_training['SpeedST'][i-5:i-4])
    cS=(aS+bS)/2
    vueltas_training['SpeedST'][i:i+1]=cS

vueltas_training['Tiempo_en_segundos']=vueltas_training[:4814]['LapTime'] / np.
    timedelta64(1, 's')
vueltas_training['Tiempo_sector1_en_segundos']=vueltas_training[:4814]['
    Sector1Time'] / np.timedelta64(1, 's')
vueltas_training['Tiempo_sector2_en_segundos']=vueltas_training[:4814]['
    Sector2Time'] / np.timedelta64(1, 's')
vueltas_training['Tiempo_sector3_en_segundos']=vueltas_training[:4814]['
    Sector3Time'] / np.timedelta64(1, 's')

vueltas_training["Valores"]=0

vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 1) & (
    vueltas_training['Compound'] == 'sUPERSOFT'), 1, vueltas_training['Valores'
    ])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 1) & (
    vueltas_training['Compound'] == 'SOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 1) & (
    vueltas_training['Compound'] == 'MEDIUM'), 2, vueltas_training['Valores'])
```

```
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 1) & (vueltas_training['Compound'] == 'HARD'), 3, vueltas_training['Valores'])

vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 2) & (vueltas_training['Compound'] == 'SUPERSOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 2) & (vueltas_training['Compound'] == 'SOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 2) & (vueltas_training['Compound'] == 'MEDIUM'), 2, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 2) & (vueltas_training['Compound'] == 'HARD'), 3, vueltas_training['Valores'])

vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 3) & (vueltas_training['Compound'] == 'SUPERSOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 3) & (vueltas_training['Compound'] == 'SOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 3) & (vueltas_training['Compound'] == 'MEDIUM'), 2, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where((vueltas_training['TyreLife'] == 3) & (vueltas_training['Compound'] == 'HARD'), 3, vueltas_training['Valores'])

vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-1) & (vueltas_training['Compound'] == 'SUPERSOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-1) & (vueltas_training['Compound'] == 'SOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-1) & (vueltas_training['Compound'] == 'MEDIUM'), 2, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-1) & (vueltas_training['Compound'] == 'HARD'), 3, vueltas_training['Valores'])

vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-2) & (vueltas_training['Compound'] == 'SUPERSOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-2) & (vueltas_training['Compound'] == 'SOFT'), 1, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-2) & (vueltas_training['Compound'] == 'MEDIUM'), 2, vueltas_training['Valores'])
vueltas_training['Valores'] = np.where(((vueltas_training['TyreLife'] == 1)-2) & (vueltas_training['Compound'] == 'HARD'), 3, vueltas_training['Valores'])

outputs_train=to_categorical((vueltas_training['Valores'].values), num_classes=4)

training_input=vueltas_training[['LapNumber', 'Stint', 'TyreLife', 'Tiempo_en_segundos', 'SpeedI1', 'SpeedI2', 'SpeedST', 'SpeedFL']]
training_output=outputs_train

training_input.isna().sum()

vueltas_validacion=vueltas2022

vueltas_validacion.drop(vueltas_validacion[vueltas_validacion['LapTime'].isna()].index, inplace = True)

vueltas_validacion['Tiempo_en_segundos']=vueltas_validacion[:1212]['LapTime'] / np.timedelta64(1, 's')
vueltas_validacion['Tiempo_sector1_en_segundos']=vueltas_validacion[:1212]['Sector1Time'] / np.timedelta64(1, 's')
vueltas_validacion['Tiempo_sector2_en_segundos']=vueltas_validacion[:1212]['Sector2Time'] / np.timedelta64(1, 's')
```

```
476 vueltas_validacion['Tiempo_sector3_en_segundos']=vueltas_validacion[:1212]['
    Sector3Time'] / np.timedelta64(1, 's')
477
478 vueltas_validacion["Valores"]=0
479
480 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 1) &
     (vueltas_validacion['Compound'] == 'sUPERSOFT'), 1, vueltas_validacion['
    Valores'])
481 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 1) &
     (vueltas_validacion['Compound'] == 'SOFT'), 1, vueltas_validacion['Valores'
    ])
482 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 1) &
     (vueltas_validacion['Compound'] == 'MEDIUM'), 2, vueltas_validacion['
    Valores'])
483 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 1) &
     (vueltas_validacion['Compound'] == 'HARD'), 3, vueltas_validacion['Valores'
    ])
484
485 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 2) &
     (vueltas_validacion['Compound'] == 'SUPERSOFT'), 1, vueltas_validacion['
    Valores'])
486 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 2) &
     (vueltas_validacion['Compound'] == 'SOFT'), 1, vueltas_validacion['Valores'
    ])
487 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 2) &
     (vueltas_validacion['Compound'] == 'MEDIUM'), 2, vueltas_validacion['
    Valores'])
488 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 2) &
     (vueltas_validacion['Compound'] == 'HARD'), 3, vueltas_validacion['Valores'
    ])
489
490 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 3) &
     (vueltas_validacion['Compound'] == 'SUPERSOFT'), 1, vueltas_validacion['
    Valores'])
491 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 3) &
     (vueltas_validacion['Compound'] == 'SOFT'), 1, vueltas_validacion['Valores'
    ])
492 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 3) &
     (vueltas_validacion['Compound'] == 'MEDIUM'), 2, vueltas_validacion['
    Valores'])
493 vueltas_validacion['Valores'] = np.where((vueltas_validacion['TyreLife'] == 3) &
     (vueltas_validacion['Compound'] == 'HARD'), 3, vueltas_validacion['Valores'
    ])
494
495 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -1) & (vueltas_validacion['Compound'] == 'SUPERSOFT'), 1, vueltas_validacion
    ['Valores'])
496 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -1) & (vueltas_validacion['Compound'] == 'SOFT'), 1, vueltas_validacion['
    Valores'])
497 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -1) & (vueltas_validacion['Compound'] == 'MEDIUM'), 2, vueltas_validacion['
    Valores'])
498 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -1) & (vueltas_validacion['Compound'] == 'HARD'), 3, vueltas_validacion['
    Valores'])
499
500 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -2) & (vueltas_validacion['Compound'] == 'SUPERSOFT'), 1, vueltas_validacion
    ['Valores'])
501 vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1)
    -2) & (vueltas_validacion['Compound'] == 'SOFT'), 1, vueltas_validacion['
    Valores'])
```

```
vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1) -2) & (vueltas_validacion['Compound'] == 'MEDIUM'), 2, vueltas_validacion['Valores'])
vueltas_validacion['Valores'] = np.where(((vueltas_validacion['TyreLife'] == 1) -2) & (vueltas_validacion['Compound'] == 'HARD'), 3, vueltas_validacion['Valores'])

outputs=to_categorical((vueltas_validacion['Valores'].values), num_classes=4)

validacion_input=vueltas_validacion[['LapNumber', 'Stint', 'TyreLife', 'Tiempo_en_segundos','SpeedI1', 'SpeedI2', 'SpeedST', 'SpeedFL']]
validacion_output=outputs

for i in range(1,len(validacion_input)):
  if math.isnan(validacion_input[i:i+1].SpeedI1):
    a1=float(validacion_input['SpeedI1'][i-1:i])
    if math.isnan(validacion_input[i-1:i].SpeedI1):
        a1=float(validacion_input['SpeedI1'][i-2:i-1])
    if i==len(validacion_input)-1:
      b1=a1
    else:
      b1=float(validacion_input['SpeedI1'][i+1:i+2])
      if math.isnan(validacion_input[i+1:i+2].SpeedI1):
        b1=float(validacion_input['SpeedI1'][i+2:i+3])
        if math.isnan(validacion_input[i+2:i+3].SpeedI1):
          b1=float(validacion_input['SpeedI1'][i+3:i+4])
    c1=(a1+b1)/2
    validacion_input['SpeedI1'][i:i+1]=c1

for i in range(1,len(validacion_input)):
  if math.isnan(validacion_input[i:i+1].SpeedST):
    aS=float(validacion_input['SpeedST'][i-1:i])
    bS=float(validacion_input['SpeedST'][i+1:i+2])
    if math.isnan(validacion_input[i+1:i+2].SpeedST):
      bS=float(validacion_input['SpeedST'][i+2:i+3])
      if math.isnan(validacion_input[i+2:i+3].SpeedST):
        bS=float(validacion_input['SpeedST'][i+3:i+4])
    cS=(aS+bS)/2
    validacion_input['SpeedST'][i:i+1]=cS

for i in range(1,len(validacion_input)):
  if math.isnan(validacion_input[i:i+1].SpeedFL):
    aF=float(validacion_input['SpeedFL'][i-1:i])
    if i==len(validacion_input)-1:
      bf=aF
    else:
      bF=float(validacion_input['SpeedFL'][i+1:i+2])
    cF=(aF+bF)/2
    validacion_input['SpeedFL'][i:i+1]=cF

validacion_input.isna().sum()

print(training_input.shape, validacion_input.shape)
print(training_output.shape, validacion_output.shape)

modelo=keras.Sequential()
modelo.add(Dense(units=8, input_dim=8, activation='relu'))
modelo.add(Dense(units=16, activation='relu'))
modelo.add(Dense(units=4, activation='softmax'))

modelo.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

```

```
#Fitting the model
history=modelo.fit(training_input,training_output,batch_size=2,epochs=100,
    validation_split=0.20)

# Capturing learning history per epoch
hist  = pd.DataFrame(history.history)
hist['epoch'] = history.epoch

# Plotting accuracy at different epochs
plt.plot(hist['loss'])
plt.plot(hist['val_loss'])
plt.legend(("train" , "valid") , loc =0)

score=modelo.evaluate(validacion_input, validacion_output)
print(score)

import tensorflow
#Calculate the confusion_matrix
y_pred = modelo.predict(validacion_input)
print("=== Confusion Matrix ===")
matrix = tensorflow.math.confusion_matrix(validacion_output.argmax(axis=1),
    y_pred.argmax(axis=1))
print(matrix)
```

Código B.1: Código empleado en el tratamiento de los datos y en la red neuronal

Apéndice C

Anexo III: Entrenamiento de la red neuronal

A continuación se muestra el proceso de entrenamiento seguido por la red neuronal durante 100 épocas

```
1
2
3  Epoch 1/100
4  1801/1801 [==============================] - 5s 2ms/step - loss: 2.6041 -
       accuracy: 0.4876 - val_loss: 0.9704 - val_accuracy: 0.5183
5  Epoch 2/100
6  1801/1801 [==============================] - 7s 4ms/step - loss: 1.4652 -
       accuracy: 0.5060 - val_loss: 1.0187 - val_accuracy: 0.5516
7  Epoch 3/100
8  1801/1801 [==============================] - 9s 5ms/step - loss: 1.1415 -
       accuracy: 0.5343 - val_loss: 0.8053 - val_accuracy: 0.5949
9  Epoch 4/100
10 1801/1801 [==============================] - 7s 4ms/step - loss: 0.9960 -
       accuracy: 0.5368 - val_loss: 0.7977 - val_accuracy: 0.5782
11 Epoch 5/100
12 1801/1801 [==============================] - 4s 2ms/step - loss: 0.9031 -
       accuracy: 0.5576 - val_loss: 0.8855 - val_accuracy: 0.5383
13 Epoch 6/100
14 1801/1801 [==============================] - 4s 2ms/step - loss: 0.8039 -
       accuracy: 0.6071 - val_loss: 0.7016 - val_accuracy: 0.6626
15 Epoch 7/100
16 1801/1801 [==============================] - 5s 3ms/step - loss: 0.7557 -
       accuracy: 0.6579 - val_loss: 0.8621 - val_accuracy: 0.5516
17 Epoch 8/100
18 1801/1801 [==============================] - 4s 2ms/step - loss: 0.7224 -
       accuracy: 0.6729 - val_loss: 0.5766 - val_accuracy: 0.7980
19 Epoch 9/100
20 1801/1801 [==============================] - 4s 2ms/step - loss: 0.7125 -
       accuracy: 0.6634 - val_loss: 0.6304 - val_accuracy: 0.6792
21 Epoch 10/100
22 1801/1801 [==============================] - 5s 3ms/step - loss: 0.7038 -
       accuracy: 0.6762 - val_loss: 0.6109 - val_accuracy: 0.7802
23 Epoch 11/100
24 1801/1801 [==============================] - 5s 3ms/step - loss: 0.6965 -
       accuracy: 0.6637 - val_loss: 0.5704 - val_accuracy: 0.7048
25 Epoch 12/100
26 1801/1801 [==============================] - 4s 2ms/step - loss: 0.6853 -
       accuracy: 0.6654 - val_loss: 0.5331 - val_accuracy: 0.7336
27 Epoch 13/100
28 1801/1801 [==============================] - 5s 3ms/step - loss: 0.6797 -
       accuracy: 0.6718 - val_loss: 0.6101 - val_accuracy: 0.6471
29 Epoch 14/100
```

```
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6775 -
    accuracy: 0.6765 - val_loss: 0.6870 - val_accuracy: 0.5660
Epoch 15/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.6732 -
    accuracy: 0.6862 - val_loss: 0.5998 - val_accuracy: 0.8102
Epoch 16/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.6749 -
    accuracy: 0.6756 - val_loss: 0.4796 - val_accuracy: 0.7303
Epoch 17/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6645 -
    accuracy: 0.6793 - val_loss: 0.5421 - val_accuracy: 0.7248
Epoch 18/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6623 -
    accuracy: 0.6856 - val_loss: 0.5172 - val_accuracy: 0.8169
Epoch 19/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6534 -
    accuracy: 0.6831 - val_loss: 0.5774 - val_accuracy: 0.7891
Epoch 20/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6465 -
    accuracy: 0.6915 - val_loss: 0.4704 - val_accuracy: 0.8468
Epoch 21/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6517 -
    accuracy: 0.6884 - val_loss: 0.6343 - val_accuracy: 0.6293
Epoch 22/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6385 -
    accuracy: 0.7040 - val_loss: 0.6457 - val_accuracy: 0.6038
Epoch 23/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6311 -
    accuracy: 0.7009 - val_loss: 0.4528 - val_accuracy: 0.8302
Epoch 24/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6324 -
    accuracy: 0.6959 - val_loss: 0.4751 - val_accuracy: 0.7780
Epoch 25/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6176 -
    accuracy: 0.7056 - val_loss: 0.5485 - val_accuracy: 0.8357
Epoch 26/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6180 -
    accuracy: 0.7045 - val_loss: 0.5358 - val_accuracy: 0.7514
Epoch 27/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6162 -
    accuracy: 0.7006 - val_loss: 0.4692 - val_accuracy: 0.7614
Epoch 28/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.6128 -
    accuracy: 0.7073 - val_loss: 0.4967 - val_accuracy: 0.7248
Epoch 29/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6056 -
    accuracy: 0.7095 - val_loss: 0.5517 - val_accuracy: 0.6881
Epoch 30/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6121 -
    accuracy: 0.7095 - val_loss: 0.4796 - val_accuracy: 0.7514
Epoch 31/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.6020 -
    accuracy: 0.7117 - val_loss: 0.4666 - val_accuracy: 0.7392
Epoch 32/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5942 -
    accuracy: 0.7131 - val_loss: 0.5626 - val_accuracy: 0.7769
Epoch 33/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.6114 -
    accuracy: 0.7054 - val_loss: 0.4698 - val_accuracy: 0.7569
Epoch 34/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.6017 -
    accuracy: 0.7117 - val_loss: 0.4677 - val_accuracy: 0.7980
Epoch 35/100
```

```
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5961 -
    accuracy: 0.7129 - val_loss: 0.4802 - val_accuracy: 0.7103
Epoch 36/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5884 -
    accuracy: 0.7176 - val_loss: 0.4686 - val_accuracy: 0.7836
Epoch 37/100
1801/1801 [==============================] - 7s 4ms/step - loss: 0.5890 -
    accuracy: 0.7162 - val_loss: 0.4922 - val_accuracy: 0.7758
Epoch 38/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5884 -
    accuracy: 0.7104 - val_loss: 0.5500 - val_accuracy: 0.6892
Epoch 39/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5863 -
    accuracy: 0.7162 - val_loss: 0.5536 - val_accuracy: 0.6770
Epoch 40/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5957 -
    accuracy: 0.7198 - val_loss: 0.4607 - val_accuracy: 0.7913
Epoch 41/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5929 -
    accuracy: 0.7187 - val_loss: 0.4864 - val_accuracy: 0.7647
Epoch 42/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5888 -
    accuracy: 0.7095 - val_loss: 0.5368 - val_accuracy: 0.7370
Epoch 43/100
1801/1801 [==============================] - 6s 4ms/step - loss: 0.5799 -
    accuracy: 0.7187 - val_loss: 0.4663 - val_accuracy: 0.7625
Epoch 44/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5809 -
    accuracy: 0.7176 - val_loss: 0.4856 - val_accuracy: 0.7536
Epoch 45/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5832 -
    accuracy: 0.7159 - val_loss: 0.4468 - val_accuracy: 0.8024
Epoch 46/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5767 -
    accuracy: 0.7201 - val_loss: 0.4749 - val_accuracy: 0.7802
Epoch 47/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5812 -
    accuracy: 0.7262 - val_loss: 0.4650 - val_accuracy: 0.7214
Epoch 48/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5802 -
    accuracy: 0.7179 - val_loss: 0.5043 - val_accuracy: 0.6915
Epoch 49/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5767 -
    accuracy: 0.7248 - val_loss: 0.4817 - val_accuracy: 0.8013
Epoch 50/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5723 -
    accuracy: 0.7270 - val_loss: 0.4867 - val_accuracy: 0.7436
Epoch 51/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5702 -
    accuracy: 0.7151 - val_loss: 0.5248 - val_accuracy: 0.7303
Epoch 52/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5683 -
    accuracy: 0.7298 - val_loss: 0.4035 - val_accuracy: 0.8546
Epoch 53/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5617 -
    accuracy: 0.7190 - val_loss: 0.4819 - val_accuracy: 0.7381
Epoch 54/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5691 -
    accuracy: 0.7223 - val_loss: 0.4499 - val_accuracy: 0.7614
Epoch 55/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5727 -
    accuracy: 0.7195 - val_loss: 0.4257 - val_accuracy: 0.7913
Epoch 56/100
```

```
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5673 - accuracy: 0.7179 - val_loss: 0.4384 - val_accuracy: 0.7636
Epoch 57/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5704 - accuracy: 0.7190 - val_loss: 0.5371 - val_accuracy: 0.6715
Epoch 58/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5606 - accuracy: 0.7245 - val_loss: 0.4609 - val_accuracy: 0.7902
Epoch 59/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5640 - accuracy: 0.7231 - val_loss: 0.4841 - val_accuracy: 0.8024
Epoch 60/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5657 - accuracy: 0.7156 - val_loss: 0.4426 - val_accuracy: 0.7847
Epoch 61/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5559 - accuracy: 0.7234 - val_loss: 0.4982 - val_accuracy: 0.7536
Epoch 62/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.6242 - accuracy: 0.6995 - val_loss: 0.4309 - val_accuracy: 0.8402
Epoch 63/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5654 - accuracy: 0.7237 - val_loss: 0.6275 - val_accuracy: 0.6437
Epoch 64/100
1801/1801 [==============================] - 7s 4ms/step - loss: 0.5945 - accuracy: 0.7073 - val_loss: 0.4642 - val_accuracy: 0.7669
Epoch 65/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5644 - accuracy: 0.7156 - val_loss: 0.4585 - val_accuracy: 0.7691
Epoch 66/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5550 - accuracy: 0.7301 - val_loss: 0.4805 - val_accuracy: 0.7370
Epoch 67/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5559 - accuracy: 0.7281 - val_loss: 0.5225 - val_accuracy: 0.7159
Epoch 68/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5588 - accuracy: 0.7240 - val_loss: 0.5350 - val_accuracy: 0.7059
Epoch 69/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5522 - accuracy: 0.7240 - val_loss: 0.4464 - val_accuracy: 0.8213
Epoch 70/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5646 - accuracy: 0.7201 - val_loss: 0.6051 - val_accuracy: 0.6670
Epoch 71/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5605 - accuracy: 0.7304 - val_loss: 0.9245 - val_accuracy: 0.6604
Epoch 72/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5682 - accuracy: 0.7220 - val_loss: 0.5474 - val_accuracy: 0.7059
Epoch 73/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5566 - accuracy: 0.7259 - val_loss: 0.4182 - val_accuracy: 0.8024
Epoch 74/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5575 - accuracy: 0.7159 - val_loss: 0.5173 - val_accuracy: 0.6837
Epoch 75/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5542 - accuracy: 0.7226 - val_loss: 0.4847 - val_accuracy: 0.7203
Epoch 76/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5478 - accuracy: 0.7245 - val_loss: 0.4620 - val_accuracy: 0.7625
Epoch 77/100
```

```
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5561 -
    accuracy: 0.7206 - val_loss: 0.4660 - val_accuracy: 0.7547
Epoch 78/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5562 -
    accuracy: 0.7254 - val_loss: 0.4833 - val_accuracy: 0.7070
Epoch 79/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5464 -
    accuracy: 0.7273 - val_loss: 0.4577 - val_accuracy: 0.7181
Epoch 80/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5486 -
    accuracy: 0.7284 - val_loss: 0.4690 - val_accuracy: 0.8313
Epoch 81/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5648 -
    accuracy: 0.7204 - val_loss: 0.5504 - val_accuracy: 0.7037
Epoch 82/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5487 -
    accuracy: 0.7295 - val_loss: 0.4722 - val_accuracy: 0.7381
Epoch 83/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5535 -
    accuracy: 0.7251 - val_loss: 0.4414 - val_accuracy: 0.8069
Epoch 84/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5542 -
    accuracy: 0.7317 - val_loss: 0.4340 - val_accuracy: 0.8391
Epoch 85/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5505 -
    accuracy: 0.7179 - val_loss: 0.4778 - val_accuracy: 0.8491
Epoch 86/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5680 -
    accuracy: 0.7126 - val_loss: 0.5506 - val_accuracy: 0.6659
Epoch 87/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5621 -
    accuracy: 0.7234 - val_loss: 0.4513 - val_accuracy: 0.7403
Epoch 88/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5660 -
    accuracy: 0.7170 - val_loss: 0.4700 - val_accuracy: 0.7714
Epoch 89/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5506 -
    accuracy: 0.7262 - val_loss: 0.4608 - val_accuracy: 0.7547
Epoch 90/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5484 -
    accuracy: 0.7279 - val_loss: 0.5066 - val_accuracy: 0.7203
Epoch 91/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5505 -
    accuracy: 0.7279 - val_loss: 0.4733 - val_accuracy: 0.7125
Epoch 92/100
1801/1801 [==============================] - 6s 3ms/step - loss: 0.5529 -
    accuracy: 0.7256 - val_loss: 0.5055 - val_accuracy: 0.6804
Epoch 93/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5663 -
    accuracy: 0.7209 - val_loss: 0.5716 - val_accuracy: 0.6926
Epoch 94/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5553 -
    accuracy: 0.7140 - val_loss: 0.4178 - val_accuracy: 0.8535
Epoch 95/100
1801/1801 [==============================] - 5s 3ms/step - loss: 0.5476 -
    accuracy: 0.7248 - val_loss: 0.5077 - val_accuracy: 0.7736
Epoch 96/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5443 -
    accuracy: 0.7337 - val_loss: 0.4930 - val_accuracy: 0.7736
Epoch 97/100
1801/1801 [==============================] - 4s 2ms/step - loss: 0.5589 -
    accuracy: 0.7265 - val_loss: 0.4137 - val_accuracy: 0.8946
Epoch 98/100
```

```
198 1801/1801 [==============================] - 5s 3ms/step - loss: 0.5464 -
    accuracy: 0.7331 - val_loss: 0.4274 - val_accuracy: 0.8557
199 Epoch 99/100
200 1801/1801 [==============================] - 5s 3ms/step - loss: 0.5483 -
    accuracy: 0.7204 - val_loss: 0.4454 - val_accuracy: 0.7458
201 Epoch 100/100
202 1801/1801 [==============================] - 4s 2ms/step - loss: 0.5573 -
    accuracy: 0.7231 - val_loss: 0.4762 - val_accuracy: 0.7558
```

Código C.1: Entrenamiento de la red neuronal durante 100 épocas

Bibliografía

[1] MCCARTHY, J., *What is artificial intelligence?*, https://www-formal.stanford.edu/jmc/whatisai.pdf, Stanford University, 2007.

[2] TURING, A. M., *Computing Machinery and intelligence*, https://doi.org/10.1093/mind/LIX.236.433, Mind, Octubre 1950.

[3] MEZIC, A., *Why Unsupervised Machine Learning is the Future of Cybersecurity* , https://technative.io/why-unsupervised-machine-learning-is-the-future-of-cybersecurity, Technative, September 2021.

[4] SILVER, D., SINGH S., PRECUP D., SUTTON R.S., *Reward is enough*, https://doi.org/10.1016/j.artint.2021.103535., DeepMind, 2021.

[5] MORGADO, I., *¿Cómo se forman los recuerdos en el cerebro?*, https://www.lavanguardia.com/ciencia/cuerpo-humano/20181008/452207506868/preguntas-big-vang-recuerdos-cerebro.html, Instituto de Neurociencias de la UAB, Octubre 2018.

[6] HEATON, J., *Introduction to Neural Networks for Java*, https://web.archive.org/web/20140721050413/http://www.heatonresearch.com/node/707, Octubre 2008.

[7] J. A., E. G. D., *El ordenador manda en la Fórmula 1*, https://www.20minutos.es/deportes/noticia/formulauno-informatica-williams-237526/0/, 20 minutos, Mayo 2007.

[8] STUART, G., *Rob Smedley on Car Performance Scores and the data behind the latest F1 Insights TV graphics*, https://www.formula1.com/en/latest/article.rob-smedley-on-car-performance-scores-and-the-data-behind-the-latest-f1.5QoyFi4KL63EULKM4u5mGL.html, Formula1, Julio 2020.

[9] MAPFRE, *El análisis de datos en la Formula 1, la diferencia entre la victoria y la derrota*, https://www.mapfre.com/actualidad/innovacion/formula-1-y-analisis-de-big-data/, Mapfre, Febrero 2020.

[10] CAPARRONI, S., *Aprendizaje supervisado y no supervisado*, http://www.cs.us.es/~fsancho/?e=77, Universidad de Sevilla, Diciembre 2020.

[11] FUMO, J., *Types of Machine Learning Algorithms You Should Know*, https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861 Towards Data Science, Junio 2017.

[12] NASTESKI, V., *An overview of the supervised machine learning methods*, https://www.researchgate.net/profile/Vladimir-Nasteski/publication/328146111_An_overview_of_the_supervised_machine_learning_methods/links/5c1025194585157ac1bba147/An-overview-of-the-supervised-machine-learning-methods.pdf, St. Kliment Ohridski University, Diciembre 2017.

[13] AWASTHI, S., *Five Most Popular Unsupervised Learning Algorithms*, https://dataaspirant.com/unsupervised-learning-algorithms/, DataAspirant, Enero 2021.

[14] MONI, R., *Reinforcement Learning algorithms - an intuitive overview* , https://smartlabai.medium.com/reinforcement-learning-algorithms-an-intuitive-overview-904e2dff5bbc, SmartLab AI, Febrero 2019.

[15] U.S. , *Redes Neuronales Artificiales*, https://rodas5.us.es/file/7352f327-eeab-83c2-d362-8e7f2be2170f/1/redes_neuronales_artificiales_SCORM.zip/page_08.htm, Universidad de Sevilla.

[16] OLIVEIRA, M., *Stochastic Gradient Descent: An intuitive proof*, https://medium.com/oberman-lab/proof-for-stochastic-gradient-descent-335bdc8693d0, Medium, Diciembre 2020.

[17] INTERACTIVECHAOS, *Otros optimizadores*, https://interactivechaos.com/es/manual/tutorial-de-machine-learning/otros-optimizadores, InteractiveChaos.

[18] BROWNLEE, J., *Gradient Descent With Momentum from Scratch*, https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/, Machine Learning Mastery, Febrero 2021.