FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Introducción al Machine Learning con TensorFlow

Iria Álvarez Fidalgo

2021/2022

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRAO DE MATEMÁTICAS

Traballo Fin de Grao

Introducción al Machine Learning con TensorFlow

Iria Álvarez Fidalgo

Febrero, 2022

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Coñecemento: Departamento de matemática aplicada
Título: Introducción al machine learning con TensorFlow
Breve descrición do contido
En esta memoria se estudian los principales conceptos y herramientas relativos al aprendizaje automático supervisado, así como los métodos numéricos de optimización que permiten desarrollar la programación de un algoritmo enfocado a la clasificación de imágenes. Para ello, se emplea Tensorflow, una librería de código abierto diseñada por Google para implementar algoritmos de aprendizaje automático.
Recomendacións
Outras observacións

Índice general

Resumen

En este trabajo se estudian las bases del aprendizaje automático o *Machine Learning* centrándose en el aprendizaje automático supervisado, en el que se le proporcionan a la máquina un conjunto de datos etiquetados. Se detalla el funcionamiento de la clasificación como técnica para estudiar los datos de entrada, que se basa en el uso de redes neuronales artificiales. En concreto, a lo largo del trabajo se analizan las redes neuronales artificiales convolucionales que son de gran utilidad a la hora de procesar datos de tipo cuadrícula, como las imágenes, para ello hacen uso de un optimizador de la función de coste del problema. En este caso estudiaremos las ventajas y el funcionamiento del método de descenso Adam. Finalmente, con el objetivo de poner en práctica todos los conceptos analizados a lo largo del trabajo, se entrena un algoritmo dedicado a la clasificación de imágenes, empleando para ello el conjunto de datos CIFAR-10 y la librería Keras de Tensorflow.

Abstract

This paper examines the basics of machine learning focusing on supervised machine learning, in which a set of tagged data is provided to the machine. It details the functioning of the classification as a technique to study input data, which is based on the use of artificial neural networks. Specifically, throughout the project, artificial convolutional neural networks are analyzed, as they are very useful when processing grid-type data, such as images, for this purpose we employ an optimizer of the cost function of the problem, and in particular we will study the advantages and functioning of the Adam descent method. Finally, with the aim of putting into practice all the concepts analyzed throughout the project, an algorithm dedicated to the classification of images is trained using the CIFAR-10 data set and the Keras library of Tensorflow.

Introducción

El machine learning, traducido al español como aprendizaje automático o aprendizaje de máquinas, es una rama de la inteligencia artificial. Se puede definir como un conjunto de métodos computacionales que emplean la información disponible para hacer predicciones o mejorar su propia actuación. Esta información consiste en grandes volúmenes de datos recolectados, conocidos como Big Data. A partir de ella, el algoritmo puede aprender patrones y dar resultados precisos. Para realizar estos pronósticos, utiliza métodos como: la regresión lineal, la clasificación, que consiste en categorizar los datos, o el agrupamiento, que radica en asignarle a un elemento un grupo.
Aborda una gran cantidad de problemas, entre los que se encuentran la clasificación de texto, empleando el procesamiento del lenguaje natural (NLP). Aplicaciones de visión computarizada, como el reconocimiento de objetos, entre las cuales está la identificación de imágenes. También tiene aplicaciones en el campo de la biología computacional, por ejemplo en el análisis de redes genéticas, o el procesamiento del habla.

TensorFlow es una librería de código abierto para expresar e implementar algoritmos de aprendizaje automático. Desarrollada por Google, inicialmente para uso interno, fue liberada como software de código abierto en 2015. Su nombre se debe a que emplea tensores como herramienta principal para construir los modelos computacionales. Una de sus principales aportaciones radica en el hecho de que permite desarrollar Deep Learning, empleando redes neuronales.

Las redes neuronales artificiales (ANNs) son algoritmos que imitan el comportamiento de las neuronas humanas y el proceso de sinapsis, emulando el hecho de que se encuentran interconectadas por dendritas y axones terminales. Están formadas por cuatro tipos de capas que le proporcionan una estructura similar a las neuronas humanas: una capa de entrada que toma los datos y una capa de salida que puede estar conectada a otras neuronas o actuar como una capa final devolviendo la predicción realizada. Y entre estas hay capas ocultas que realizan cálculos con los datos de entrada, que son las capas de suma y las de activación.

Las redes neuronales convolucionales (CNNs o ConvNets) son una clase de redes neu-

ronales que se especializa en procesar datos topológicos de tipo cuadrícula, imitando así la disposición del ojo humano para identificar distintas características y ser capaz de reconocer objetos. Para poder llevar a cabo esta tarea, se utilizan distintos tipos de capas ocultas que se van especializando, de modo que las primeras capas detectan formas simples, como curvas o líneas, mientras que las capas más profundas son capaces de reconocer formas más complejas, como un rostro o una prenda de ropa. Esto juega un gran papel en la clasificación de imágenes.

Capítulo 1

Machine Learning

1.1. Marco histórico

En el año 1959 Arthur L. Samuels publicó el artículo "Some Studies in Machine Learning Using the Game of Checkers ", convirtiéndose así en la primera persona en desarrollar un programa de aprendizaje automático. En su trabajo (ver [1]) estudió dos procedimientos de machine learning para implementar el juego de las damas, llegando a la conclusión de que un ordenador podía programarse de tal forma que siempre venciese en el juego al programador. Incide en el hecho de que el programa puede aprender a jugar en tan sólo 8 o 10 horas, cuando se le dan: las reglas del juego, una noción de dirección y una lista incompleta y redundante de parámetros, que se piensa que pueden tener algo que ver con el juego, pero cuyos signos y pesos son desconocidos y no se especifican. Además introduce la noción de aprendizaje automático como el campo de estudio que da a los ordenadores la habilidad de aprender sin ser explícitamente programados para ello. También compara la formación de una persona o un animal con el que lleva a cabo un ordenador a lo largo de este proceso y resalta que, programar ordenadores para que aprendan de la experiencia lleva a prescindir de una programación más detallada.

En el año 1958, Frank Rosenblatt introduce el concepto de perceptrón (ver [2]) para responder a cuestiones como: de que forma detecta el sistema biológico la información del mundo físico, cómo se almacena y cómo influencia a los procedimientos de reconocimiento y comportamiento. Rosenblatt define el perceptrón como un sistema nervioso hipotético, diseñado para ilustrar algunas propiedades de estructuras inteligentes, de forma análoga a los sistemas biológicos. Un perceptrón funciona de la siguiente manera: toma varias entradas binarias y genera una única salida binaria. Posteriormente se definió el perceptrón multicapa, como una ampliación del concepto inicial, incorporando las nociones de capas

de entrada, ocultas y de salida.

Estas son las bases que permiten el desarrollo del aprendizaje automático.

1.2. Definición y clases de aprendizaje

***Definición* 1.2.1.** *Machine Learning*

Un programa aprende de la experiencia E con respecto a algunas clases de tareas T y medidas de actuación P, si su actuación y sus tareas en T, medidas por P, mejoran con la experiencia E.

La experiencia E hace referencia a grandes volúmenes de datos recolectados, conocidos como Big Data, para la toma de decisiones T y la forma de medir su desempeño P, para comprobar que los algoritmos mejoran con la adquisición de más experiencia.

Esto es, el programa toma una serie de datos, a partir de los cuales extrae un patrón que utiliza para realizar predicciones. Ahora bien, estas predicciones deben ser precisas. El aprendizaje automático se puede dividir en tres corrientes principales :

- Aprendizaje automático supervisado.
- Aprendizaje automático no supervisado.
- Aprendizaje de refuerzo.

1.2.1. Aprendizaje automático no supervisado

El aprendizaje no supervisado consiste en modelar datos que no están etiquetados. El objetivo es la extracción de información significativa mediante la exploración de la estructura de dichos datos. La labor del algoritmo reside en encontrar patrones de agrupación para elementos que por sus características, en todas las variables introducidas en el modelo, son parecidos.

Dos de las herramientas más útiles a la hora de tratar este tipo de aprendizaje son: el agrupamiento y la reducción dimensional. El agrupamiento consiste en dividir los datos en diferentes conjuntos que contengan elementos similares. Por otra parte, la reducción dimensional consiste en manipular los datos para poder examinarlos desde una perspectiva mucho más simple. Por ejemplo, eliminando características redundantes que no nos aporten información a la hora de agrupar los elementos.

Ejemplo 1.1. *Un ejemplo de un algoritmo no supervisado que emplea el método de agrupamiento es el de recomendación de Netflix. Busca usuarios que tengan visualizaciones similares para asociarlos. En función de esas agrupaciones, recomienda contenidos que hayan sido vistos por sujetos de un mismo grupo.*

Ejemplo 1.2. *Un ejemplo de reducción dimensional es el caso de un modelo que detecta coches en imágenes. Para ello se convertirán primero a blanco y negro, de forma que el algoritmo no tiene que examinar variables que no aportan información en el reconocimiento de coches, como el color.*

1.2.2. Aprendizaje de refuerzo

Este tipo de aprendizaje funciona mediante intervención humana durante el proceso, premiando o penalizando las decisiones que va tomando el algoritmo en cada paso. Esto es, el entrenamiento se realiza con información reunida observando como el modelo interacciona con el medio para aprender que combinación de acciones resulta más favorable, de forma que, en este caso, los datos son las experiencias previas.

Este método se emplea en el campo de la robótica, por ejemplo, en vehículos autónomos o naves espaciales.

1.2.3. Aprendizaje automático supervisado

El aprendizaje automático supervisado consiste en proporcionarle a la máquina datos de entrada etiquetados, para que sea capaz de "aprender de ellos" y realizar ajustes en sus parámetros interiores para adaptarse a los resultados conocidos.

Las etiquetas son valores o categorías asignadas a los conjuntos de datos que se van a utilizar en el proceso de aprendizaje.

***Definición* 1.2.2.** *Etiqueta*

Sea x un conjunto de datos, que toma la forma de un vector de características. La correspondiente etiqueta asociada a x es $f(x)$, también conocida como fundamento de la verdad de x.

En los problemas de clasificación, las etiquetas se corresponden con vectores de categorías, por ejemplo, si queremos clasificar imágenes de animales vertebrados emplearíamos las etiquetas: anfibios, reptiles, aves, peces y mamíferos. Por el contrario, en los problemas de regresión las etiquetas son vectores de valores reales.

Este tipo de aprendizaje automático suele comprender dos fases:

- Fase de entrenamiento. En esta etapa, parte del conjunto de los datos totales, que recibe el nombre de conjunto de entrenamiento, se le proporcionan al algoritmo.

  ***Definición* 1.2.3.** *Conjunto de entrenamiento*

  Partición del total de los datos que se utiliza para entrenar al algoritmo. Está formado por los vectores de características y por los vectores de etiquetas.

  El objetivo de esta fase es que algoritmo aprenda de los datos y establezca una relación entre las características y las etiquetas, es decir, entre los datos de entrada y los de salida. Creando así un patrón a partir del cual se puedan hacer predicciones.

- Fase de prueba o predicción. En esta etapa emplea lo que resta de los datos, que recibe el nombre de conjunto de prueba.

  ***Definición* 1.2.4.** *Conjunto de prueba*

  Partición del total de los datos que se utiliza para evaluar la actuación del algoritmo. A su vez está formado por los vectores de características y etiquetas, no obstante, al modelo solamente se le proporcionarán los primeros.

  Una vez entrenado el algoritmo empleamos parte de los datos para comprobar que funciona correctamente. Solo le proporcionamos los datos de entrada, es decir, el vector de características, de forma que el modelo prediga el vector de etiquetas. A continuación comparamos las predicciones con las etiquetas del conjunto de prueba para medir la efectividad de la actuación del algoritmo.

  Por tanto, una vez el modelo se ha entrenado adecuadamente y los parámetros internos son coherentes con los datos de entrada y los resultados obtenidos para el conjunto de entrenamiento, el modelo podrá realizar predicciones adecuadas ante nuevos datos. A continuación se muestra en la Figura 1.1, un esquema del funcionamiento del aprendizaje automático supervisado.

Este tipo de aprendizaje está enfocado principalmente al uso de la regresión y la clasificación. En el caso de que se quieran predecir valores numéricos, recurrimos a la regresión, sin embargo, si se quiere hacer un pronóstico de la clase o categoría a la que pertenece un elemento entonces hablamos de clasificación.

A lo largo de este trabajo nos centraremos en el aprendizaje automático supervisado.

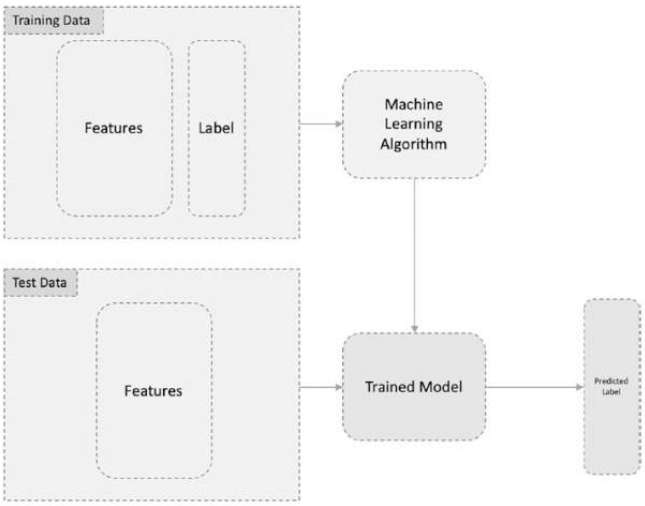


Figura 1.1: Esquema del funcionamiento de los procesos de entrenamiento y prueba (ver [19]).

1.3. Métodos clave en el aprendizaje automático supervisado

La regresión y la clasificación son dos de los principales métodos que se emplean en el aprendizaje automático supervisado. Para datos numéricos se utiliza mayoritariamente la regresión mientras que para aquellos problemas en los que se requiera categorizar elementos es preferible usar la clasificación. Esto es, ambas trabajan con datos de entrada discretos y continuos, no obstante, la regresión genera predicciones continuas mientras que la clasificación da una salida discreta. Además en esta sección también se estudiarán los árboles de decisión, representaciones gráficas que sirven como herramientas a la hora de resolver problemas de aprendizaje automático supervisado y son útiles tanto para la regresión como para la clasificación.

1.3.1. Regresión

Los modelos de regresión sirven para representar la dependencia de una variable Y, llamada variable dependiente o variable respuesta, con respecto a otra variable X, que llamaremos variable independiente o explicativa. Los objetivos de estos modelos son: conocer de qué modo la variable dependiente depende de la explicativa, esto es, describir la forma de dependencia, y realizar predicciones del valor de la variable respuesta, continuas, una vez conocido el valor de la variable explicativa, que puede corresponderse con datos de carácter continuo o discreto, como se puede apreciar en la Figura 1.2.

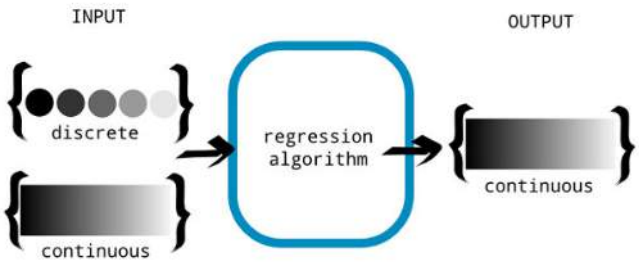


Figura 1.2: Representación del funcionamiento del algoritmo de regresión diferenciando entre datos de entrada continuos y discretos (ver [18]).

1. Regresión lineal

   En Estadística la regresión lineal es una aproximación para modelar un conjunto de datos. Consiste en establecer una aplicación entre una o más variables explicativas, X, que se corresponden con los datos de entrada y una variable dependiente, Y, que se corresponde con los de salida, de forma que nos permita predecir valores numéricos, es decir, se emplea para la predicción de valores continuos. En términos generales, la regresión se suele formalizar como la media condicionada de la variable respuesta en función del valor que tome la variable explicativa, esto es, $m(x) = \mathbb{E}[Y|X = x]$, para cada $x \in X$. Por tanto, la variable respuesta se puede expresar en función de la variable explicativa de la siguiente manera:

   $$Y = m(X) + \epsilon,$$

   donde ϵ es el error y verifica $\mathbb{E}[\epsilon|X = x] = 0$ para todo x.
   Para construir el modelo de regresión específico en cada caso, se tiene en cuenta si hay una o varias variables explicativas, o variables respuesta, si estas son discretas o continuas, la forma de la función de regresión, el tipo de distribución del error, la forma de obtener los datos muestrales, y otros aspectos que permiten configurar el modelo adecuado.

   En el caso de la regresión lineal simple, en una variable, el objetivo es establecer una función:

   $$Y = mX + b,$$

   donde m es la pendiente de la recta de regresión y b es el intercepto.
   En el caso de regresión lineal múltiple la ecuación será:

$$Y = m_1X_1 + m_2X_2 + \ldots + m_nX_n + b,$$

donde $X_1, X_2, \ldots X_n$ son las variables explicativas, $m_1, m_2 \ldots m_n$ son las pendientes y b el intercepto. Tanto en la regresión lineal simple como en la múltiple, las pendientes y el intercepto son parámetros desconocidos que es preciso determinar, esto se consigue empleando el método de mínimos cuadrados.

El algoritmo de aprendizaje automático supervisado obtiene la recta necesaria para realizar predicciones. Para hacerlo mide el error con respecto a los puntos de entrada y el valor Y de salida real; este concepto recibe el nombre de función de coste.

Definición 1.3.1. *Función de coste o de pérdida*

La función de coste mide la diferencia entre una etiqueta predicha y la verdadera etiqueta. Si denotamos el conjunto total de etiquetas por $\mathcal{Y}$ y el conjunto de las etiquetas predichas como $\mathcal{Y}'$, la función de pérdida L es una aplicación $L : \mathcal{Y} \times \mathcal{Y}' \longrightarrow \mathbb{R}^+$.

En la Figura 1.3 se muestra una representación de la función de coste. Cuánto mayor sea el coste peor será la aproximación del parámetro del modelo. Esto nos permite replantear la situación como un problema cuyo objetivo es minimizar el coste.

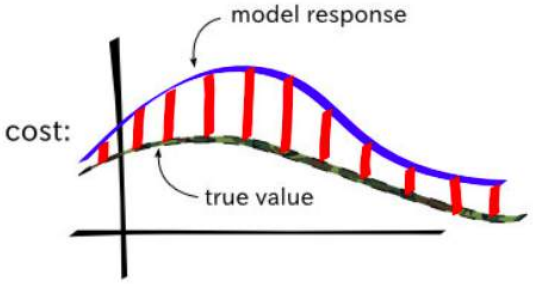


Figura 1.3: Función de coste, calculada como la norma de la diferencia entre los valores predichos y los valores reales (ver [18]).

TensorFlow se encarga de los procedimientos internos del problema de optimización, de forma que solamente es necesario definir la función de coste y emplear un optimizador adecuado. A lo largo de este trabajo emplearemos el algoritmo de optimización de Adam, o algoritmo de estimación de movimiento adaptativo. Es un método de máximo descenso o de gradiente estocástico basado en estimaciones adaptativas de primer y segundo orden. Por tanto, los algoritmos de regresión lineal en TensorFlow

consisten en modelos iterativos que convergen a mejores valores de los parámetros en cada iteración.

***Definición* 1.3.2.** *epoch*

Cada una de las iteraciones del bucle que recorre los datos del problema con el objetivo de mejorar los parámetros recibe el nombre de epoch.

2. Modelo polinomial

   La dependencia que existe entre la variable independiente y las variables explicativas no siempre es lineal. En ocasiones una recta no es suficiente para describir correctamente todos los datos, esto es, los puntos responden mejor a curvas, como por ejemplo, polinomios. La ecuación del modelo de grado n es la siguente:

   $$Y = w_n x^n + \ldots + w_1 x + w_0$$

   A partir de esta ecuación se calcula el error producido entre el comportamiento de los datos y el modelo propuesto, es decir, lo que previamente habíamos definido como función de coste. De esta forma se tiene:

   $$E = Y_i - (w_n x^n + \ldots + w_1 x + w_0)$$

   siendo E el error entre el modelo y los datos experimentales.
   En este caso el problema también se puede reescribir como un algoritmo de optimización donde el objetivo es determinar los coeficientes del polinomio que minimizan ese coste.

1.3.2. Clasificación

La clasificación emplea regresión logística para dar una salida discreta a partir de un conjunto de datos de entrada o características, como se puede ver en la Figura 1.4.

Permite la resolución de dos tipos de problemas:

- Binarios: existen dos tipos de datos de salida o etiquetas. Un ejemplo de esta situación es la clasificación de correo en Spam o No Spam.
- Múltiples: existen un número finito de datos de salida o etiquetas. Por ejemplo, dado un conjunto de imágenes de medios de transporte reconocer de cual se trata. Para resolver este tipo de problemas también se puede hacer uso de la regresión softmax.

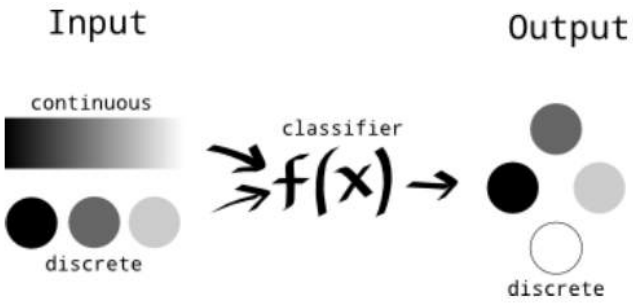


Figura 1.4: Representación del funcionamiento del algoritmo de clasificación diferenciando entre datos de entrada continuos y discretos (ver [18]).

La regresión logística es uno de los principales métodos de clasificación y se emplea para predecir valores discretos o categóricos.
Presenta ventajas con respecto a la regresión lineal, ya que es muy precisa, sus respuestas son medidas de probabilidad y es simple actualizar el modelo con datos nuevos.

El trasfondo del método es parecido al de regresión lineal, toma varias variables explicativas $X_1,\dots X_n$ de modo que:

$$Y = m_1X_1 + m_2X_2 + \dots + m_nX_n + b,$$

donde $m_1, m_2 \dots m_n$ son las pendientes y b el intercepto.
A continuación se le aplica la siguiente función logística a la ecuación lineal:

$$P(Y = 1) = \frac{1}{1 + e^{-(m_1X_1 + m_2X_2 + \dots + m_nX_n + b)}},$$

donde $P(Y = 1)$ es la probabilidad de que $Y = 1$.
Esta función recibe el nombre de función sigmoide y reduce en gran medida los valores de la función de coste, debido a las propiedades logarítmicas. Los términos $m_1, m_2 \dots m_n, \beta$ son parámetros desconocidos que es necesario determinar basándose en los datos obtenidos para una muestra. Encontrar la mejor estimación de estos parámetros pasa por un proceso que genera estimaciones de estos mejorando constantemente las aproximaciones hasta conseguir una solución estable.

En el caso de que las variables sean discretas, es necesario preprocesar los datos, para asegurar que el modelo puede trabajar con ellos. Las variables pueden ser ordinales o nominales. En el primer caso, se pueden ordenar, ya que los valores numéricos son comparables entre ellos. Mientras que en el caso de las variables nominales esto no ocurre, no responden a un orden. Este problema se soluciona añadiendo lo que se conocen como "dummy variables" a cada una de las variables nominales. Esto es, cada una de estas tiene asociado un valor 0 o 1, dependiendo de si la variable pertenece a esa categoría o no (verdadero o falso).

Árboles de decisión

Un árbol de decisión es un método de aprendizaje automático supervisado, puede utilizarse para la regresión y para la clasificación. Para construir árboles de decisión se emplea el algoritmo CART (Classification and Regression Tree algorithm)

***Definición* 1.3.3.** *Árbol de decisión*

Representación gráfica de posibles soluciones a una decisión basadas en ciertas condiciones.

Los árboles de decisión tienen un nodo inicial, conocido como nodo raíz, que representa el conjunto de los datos original al completo. Luego se divide en dos o más grupos homogéneos, llamados nodos de decisión, a través de ramas. En estas ramas se plantea una condición que puede ser verdadera o falsa. Este proceso se repite hasta llegar a las hojas o nodos finales que proporcionan la solución del problema. En la Figura 1.5 se puede ver un ejemplo del funcionamiento de los árboles de decisión.

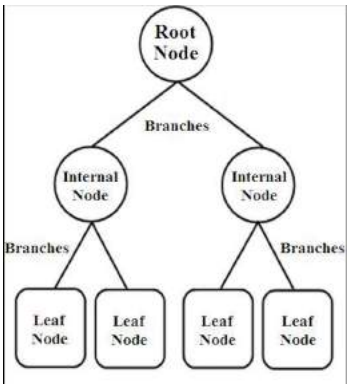


Figura 1.5: Estructura de un árbol de decisión (ver J. Sá, A. Almeida, B. Pereira da Rocha, M. Mota, J.R. De Souza and L. Dentel, Lightning Forecast Using Data Mining Techniques On Hourly Evolution Of The Convective Available Potential Energy).

Para conseguir una predicción óptima es necesario que el algoritmo valore de alguna manera cada una de las soluciones posibles. Con este objetivo se emplean distintas funciones. Para atributos con valores continuos una de las funciones que se utiliza es el índice gini, que mide el grado de impureza de los nodos, esto es, como de desordenados o mezclados quedan los nodos una vez divididos. Es preciso minimizar esta función. Para atributos discretos se emplea la ganancia de información que intenta estimar la información que aporta cada atributo. Se busca maximizar esta función, ya que de esta forma estaremos maximizando la ganancia.

Capítulo 2

Redes Neuronales

Las redes neuronales artificiales (ANNs) son algoritmos que imitan el comportamiento de las neuronas humanas y el proceso de sinapsis que estas llevan a cabo. Están formadas por cuatro tipos de capas interconectadas que le proporcionan una estructura similar a las neuronas biológicas.

En primer lugar la capa de entrada toma los datos, $X_1, X_2, \ldots, X_n$, que llevan asociados unos pesos, $W_1, W_2, \ldots, W_n$. A continuación, las capas ocultas realizan modificaciones en los datos. Formando parte de estas capas se encuentran: la capa de suma, que agrega los valores recibidos, previamente multiplicados por sus pesos, y la capa de activación que permite la interconexión entre neuronas, provocando que una neurona se active y pase una señal hacia la siguiente si el resultado de la capa de suma supera un valor concreto que actúa como umbral. Por último, la capa de salida de datos puede estar conectada a otras neuronas o actuar como una capa final.

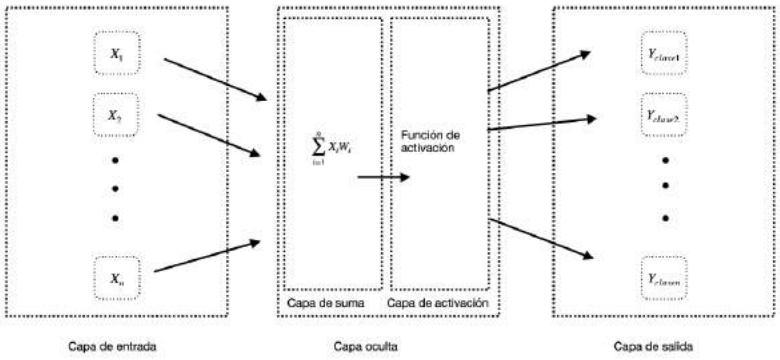


Figura 2.1: Representación de una red neuronal para un problema de clasificación.

En caso de estar resolviendo un problema de regresión, obtendríamos un único valor con-

tinuo en esta última capa, Y_p, mientras que en los problemas de clasificación la capa final generará tantas salidas como clases de elementos considere el modelo, $Y_{clase1}, Y_{clase2}, \ldots, Y_{clasen}$, que representan las probabilidades de que el objeto estudiado pertenezca a cada una de las categorías. Esta última situación se representa en la Figura 2.1. Cabe destacar que todos los datos de entrada deben estar conectados a cada una de las neuronas de la capa oculta y, por tanto, a los datos de salida.
Cuando una red neuronal presenta más de una capa oculta recibe el nombre de *deep neural network* (DNN). Cuantas más capas ocultas presenta la red, mayor es su capacidad para establecer relaciones precisas, no lineales, entre los datos de entrada y los de salida. No obstante esto supone un aumento del coste computacional.

2.1. Propagación en redes neuronales

El objetivo de la propagación en redes neuronales es que las neuronas aprendan por si mismas a ajustar el valor de los pesos para obtener las salidas correctas a través de métodos iterativos. Existen dos modelos: la propagación hacia adelante y la propagación hacia atrás, también conocida como retropropagación o *backpropagation*.

2.1.1. Propagación hacia adelante

Una red neuronal prealimentada o *feedforward*, hace el recorrido de izquierda a derecha hasta obtener los valores de salida.

Siendo $X_1, \ldots, X_n$ los datos de entrada analizaremos como funciona el método. Para simplificar el proceso suponemos que existe una única capa oculta con m neuronas, que denotamos por H_j. Definimos $W_{i,j}$ como el peso que conecta el dato de entrada i-ésimo con la neurona j-ésima. Por tanto, cada una de las m neuronas de la capa oculta se calcula de la siguiente manera:

$$H_j = \sum_{i=1}^{n} W_{i,j} * X_i + b_j \text{ con } 1 \leq j \leq m,$$

donde b_j es el valor del sesgo, esto es el resultado de la capa de suma.
A continuación se les aplica una función de activación, algunas de las más utilizadas son ReLU, softmax, la función sigmoide o la hiperbólica tangente.

2.1.2. Propagación hacia atrás

La propagación hacia atrás, retropropagación o *backpropagation* es un método iterativo que recorre la red de derecha a izquierda. Esto es, además del elemento predicho conocemos

el valor del dato real, ya que se trata de lenguaje supervisado. Esto permite calcular la función de coste, (ver Definición 1.3.1) como la diferencia entre el valor de la predicción y el valor real de la variable.
Con el objetivo de minimizar dicho coste se intenta optimizar el valor de los pesos obteniendo su gradiente. Este gradiente se va actualizando hasta que se alcanza el coste mínimo.

2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN o ConvNets), son una clase de redes neuronales artificiales que emplean el aprendizaje automático supervisado para procesar datos de tipo cuadrícula.
Se trata de un algoritmo muy utilizado en el reconocimiento y clasificación de imágenes, ya que una fotografía se puede entender como una matriz de píxeles, cuyas entradas presentan valores entre 0 y 255 debido a la escala de color.
Este sistema presenta numerosas aplicaciones, como por el ejemplo el diagnóstico de cáncer de mama basado en el estudio de mamografías. En este caso se emplean las CNN para clasificar tumores en benignos o malignos. En [8] se detalla el análisis de esta aplicación en el sector de la medicina.
Estas redes neuronales presentan tres tipos de capas características: las capas convolucionales, las capas de combinación y las totalmente conectadas.

2.2.1. Capas Convolucionales

El objetivo de esta clase de capas es aplicar filtros sobre los datos de entrada. Cada uno de estos filtros recibe el nombre de núcleo convolucional. Además, las capas pueden llevar asociadas funciones de activación.
Los filtros son matrices de números enteros que actúan sobre subconjuntos del total de los datos de entrada, que recordemos se encuentran en forma matricial, de la misma dimensión que el núcleo, y a continuación se suman los resultados para obtener un único valor. Este proceso se repite de forma que los núcleos se "deslizan" hasta recorrer toda la matriz. De este modo el filtro se ha aplicado al conjunto de datos.

***Definición* 2.2.1.** *Stride*

La magnitud del movimiento entre aplicaciones del filtro a los datos de entrada.

En la Figura 2.2 se ejemplifica el funcionamiento de un núcleo de dimensión 2×2 sobre la matriz de datos de entrada, que es de dimensión 4×4. La tabla A muestra los datos de

entrada sin manipular, en la tabla B se puede ver el núcleo convolucional actuando sobre un subconjunto de los datos y en tabla C se puede ver como el núcleo se ha desplazado a lo largo de la tabla con un *stride* de valor (1,1), es decir, se desliza una casilla en horizontal en cada aplicación y al llegar al final se desliza desde el inicio una en vertical. Por último tenemos la matriz de salida, que recibe el nombre de *features map* y contiene los resultados del filtro aplicado.

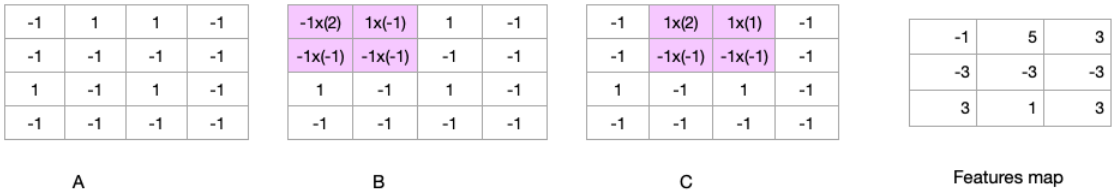


Figura 2.2: Ejemplo de la aplicación de un núcleo convolucional de dimension (2×2) a un conjunto de datos de dimensión (4×4) de stride (1,1).

En ocasiones este tipo de capas lleva asociadas funciones de activación, aunque también es posible que estas funciones estén en las capas totalmente conectadas. A pesar de que existen otras combinaciones eficaces, normalmente se emplea la función ReLU asociada a cada una de las capas internas, mientras que la función softmax se usa en la capa final como función de clasificación.

A continuación detallamos algunas de las funciones de activación:

- ReLU La función ReLU, *Rectified linear units*, le asigna un valor nulo a los elementos negativos de la matriz y da lugar a una función lineal, de pendiente uno, para aquellos elementos que son positivos. Esto es, $R(x) = max\{0, x\}$, asegurando así que no hay valores menores que cero en la matriz, como se puede ver en la Figura 2.3.

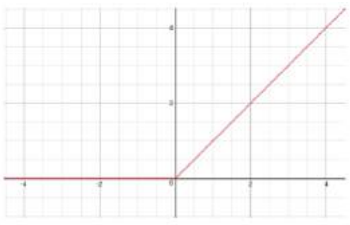


Figura 2.3: Representación gráfica del efecto de la función de activación ReLU. (ver [9]).

- Softmax

  La función softmax es un caso particular de regresión logística que se puede aplicar a datos continuos. Se suele utilizar en la última capa del modelo para llevar a cabo el proceso de clasificación.
  Su funcionamiento es el siguiente, toma como datos de entrada: $Y_1, Y_2, \ldots, Y_n$, teniendo en cuenta que Y_i es de la forma

  $$Y_i = \sum_i^n W_i X_i,$$

  donde X_i son los resultados provenientes de la penúltima capa de la red neuronal y W_i son los pesos correspondientes a la capa con activación softmax asociada.
  A continuación genera las probabilidades de que la muestra pertenezca a la i-ésima clase con $i \in \{1, \ldots, n\}$, donde n es el número de categorías existentes. De esta forma, da probabilidades simples entre cero y uno cuya suma es la unidad. Estas probabilidades se calculan de la siguiente manera:

  $$f(Y_i) = \frac{e^{Y_i}}{\sum_{j=1}^n e^{Y_j}},$$

  de forma que la predicción sobre el elemento en cuestión sería: $y = max_{i \in 1,\ldots,n} f(Y_i)$, es decir, pertenece a la clase que presenta mayor probabilidad. Este proceso se repite para cada uno de los componentes del conjunto.
  Para más información sobre la función softmax se puede consultar [10].

La aplicación de filtros puede provocar una pérdida de información espacial llegando a eliminar algunas características importantes para el desarrollo del modelo, lo cual resulta en una disminución de la precisión del algoritmo.

En [11] se hace un estudio sobre los efectos que tiene el proceso de convolución en lo referente a la pérdida de información.Para evitar este tipo de problemas se usa el zero-padding. Consiste en "rellenar" con ceros las matrices de información hasta que todas tengan la misma dimensión. Su funcionamiento permite a la red neuronal codificar información sobre la posición absoluta a pesar de la presencia de capas de combinación en su estructura. Esto es, permite por ejemplo que la información que se encuentra en los bordes de las fotografías no se pierda.

2.2.2. Capas de combinación

Debido al coste computacional que supone el aumento progresivo del número de neuronas en cada una de las capas ocultas, es necesario reducir la dimensión de la matriz de

datos con la que estamos trabajando. Suponiendo que tenemos imágenes de tamaño $n \times n$[1] píxeles, tendríamos una matriz de datos de entrada de rango n, lo cual implica la necesidad de n^2 neuronas. Luego, en la primera convolución, suponiendo que la capa tiene un filtro de tamaño s, serían necesarias $s \times n^2$ neuronas. Es fácil ver que a medida que aumenta el número de capas se dispararía el número de neuronas necesarias, elevando así el coste computacional del programa.

Una de las técnicas más empleadas para reducir la dimensión de la matriz de datos es el *Max-pooling*, o combinación máxima. Consiste en tomar el máximo de entre un subconjunto de la matriz de datos. Esta técnica toma subconjuntos de la matriz de izquierda a derecha y de arriba a abajo. De esta forma, aplicando Max-pooling de tamaño 2×2 reducimos a la mitad el tamaño de las imágenes, obteniendo así fotografías de tamaño $\frac{n}{2} \times \frac{n}{2}$.

Continuando con el ejemplo de la Figura 2.2, en el cual obteníamos el *features map* como matriz de salida de la capa convolucional, tenemos en la Figura 2.4 la aplicación de la función de activación ReLU para dicha matriz, que convierte en cero aquellas entradas de la matriz que presentaban valores negativos. Por último aplicamos en la tercera tabla Max-pooling de tamaño 2×2 y obtenemos una cuarta tabla con los resultados de está capa de combinación. Nótese que para calcular el primer elemento de la última tabla se hace $max\{0, 5, 0, 0\}$, no obstante, para calcular el elemento en posición $(1, 2)$ se realiza el cálculo $max\{3, 0\}$.

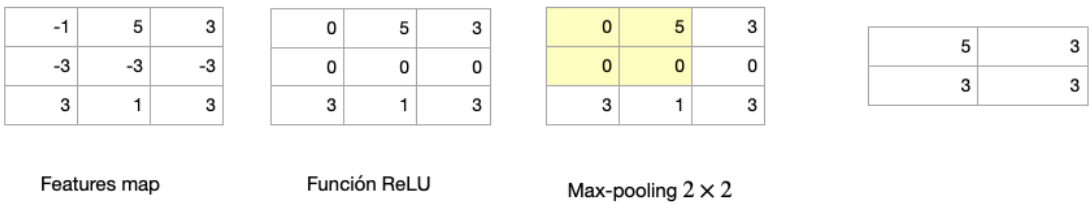


Figura 2.4: Ejemplo de la aplicación de la función ReLU y *Max-pooling* de dimensión 2×2.

2.2.3. Capas totalmente conectadas

Estas capas son del mismo tipo que las que se aplican en cualquier red neuronal para permitir la interconexión entre capas, imitando la estructura de las neuronas humanas.

[1] En este caso estamos suponiendo también que las imágenes a clasificar están en blanco y negro por lo tanto solo es necesario un canal de color, esto es, las imágenes son de dimensión $n \times n \times 1$, en caso de ser a color tendríamos imágenes de dimensión $n \times n \times 3$.

Intervienen en el proceso tras las múltiples capas convolucionales y sus posteriores reducciones en dimensión. En ocasiones son estas capas las que llevan asociadas funciones de activación, incluida la capa final de clasificación, en lugar de ir en las capas convolucionales.

Por tanto, para obtener un modelo preciso y efectivo construiremos un algoritmo que combine capas convolucionales, que pueden llevar asociada una función de activación ReLU en las capas internas, tras las cuales se aplicará una capa de combinación máxima para evitar que el coste computacional aumente. Finalmente aplicaremos una capa que tenga asociada una función softmax que nos generará un vector de probabilidades, lo cual nos permitirá clasificar cada una de las imágenes en la categoría que presente mayor probabilidad. En la Figura 2.5 se puede ver un ejemplo del funcionamiento de una red neuronal convolucional.

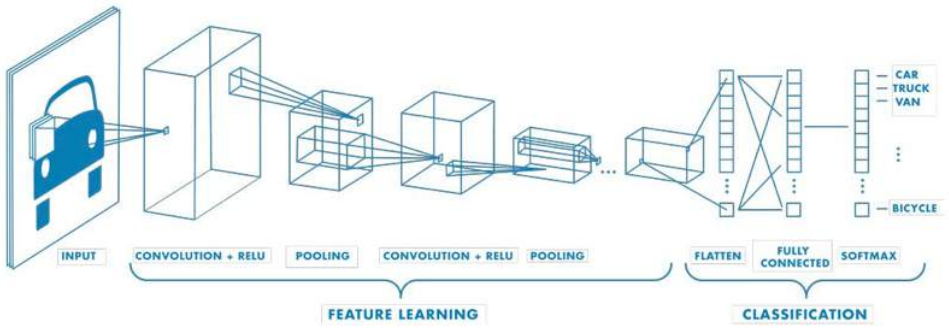


Figura 2.5: Representación gráfica del funcionamiento de una CNN (ver MathWorks, Redes neuronales convolucionales).

Capítulo 3

Métodos numéricos de optimización

El proceso de entrenamiento de una red neuronal pasa por optimizar la función de coste encontrando los pesos $W_1, \ldots, W_n$, adecuados para cada una de las capas. Como ya vimos, el mecanismo de retropropagación o *backpropagation* es el responsable de ir actualizando los valores de los pesos mediante el cálculo de la función de coste, siendo para ello necesario hacer uso de un optimizador.

Existen distintos optimizadores entre los cuales están: *Stochastic Gradient Descent* (SGD) que permite reducir el coste computacional de los modelos tomando muestras aleatorias de entre los datos del conjunto de entrenamiento, AdaGrad o RMSProp. Con el objetivo de mejorar los resultados obtenidos con estos métodos se creó el optimizador Adam, cuyo nombre proviene de *adaptative moment estimation*, y supone un punto intermedio entre las ventajas de los últimos. Todos estos estimadores tienen en común que emplean métodos iterativos para ir mejorando, en cada iteración, el valor de los pesos asociados a las capas neuronales, y en consecuencia la función de coste esto es, utilizan métodos de descenso.

A pesar de la variedad de optimizadores existente, en la actualidad el más empleado es Adam, que se construye teniendo en cuenta métodos que existían con anterioridad, como SGD, AdaGrad o RMSProp con el objetivo de obtener resultados más precisos. En la Figura 3.1 se puede ver que presenta una convergencia más rápida. Se trata de un algoritmo introducido por Kingma y Ba, que emplea método de descenso para minimizar funciones objetivo estocásticas, basándose en estimaciones adaptativas de momentos de menor orden. Es un método eficiente a nivel computacional, que requiere poca memoria y resulta apropiado para resolver problemas con una gran cantidad de parámetros o datos.

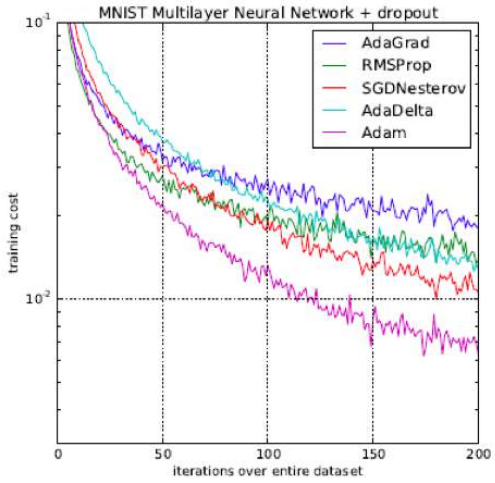


Figura 3.1: Resultado del entrenamiento de una red neuronal multicapa para el conjunto de imágnes MNIST empleando distintos optimizadores (ver [15]).

En este capítulo estudiaremos el funcionamiento del método Adam es necesario analizar, en primer lugar, qué tipo de problema se nos plantea y a continuación como resolverlo. Este método está englobado en una categoría de algoritmos conocidos como métodos de máximo descenso o de gradiente que será preciso analizar para poder entender correctamente el algoritmo que queremos emplear.
Se trata de un problema cuyo objetivo es minimizar la función de coste que es de la forma $J : \mathbb{R}^n \longrightarrow \mathbb{R}$. Por tanto puede ser reformulado de la siguiente manera:

$$\text{mín}_{x\in\mathbb{R}^n}\, J(x),$$

Los métodos de optimización son algoritmos iterativos, que generan una sucesión, x_k, de iterantes que se aproximan al valor en el cual se alcanza el mínimo de la función J, de la siguiente forma:

$$\begin{cases} x_0 \ dado, \\ \\ x_{k+1} = x_k + \alpha_k d_k, \ k \geq 0. \end{cases}$$

donde x_k, d_k y α_k son el k-ésimo iterante, la dirección y el paso respectivamente.

3.1. Método de máximo descenso o de gradiente

En esta sección estudiaremos los métodos de gradiente, que vienen dados porque la dirección de descenso del modelo de optimización es la de máximo descenso o la de menos

el gradiente, $d_k = -\nabla J(x_k)$.

***Definición* 3.1.1.** *Método de máximo descenso o de gradiente*

Sea $J : \Omega \subset \mathbb{R}^n \longrightarrow \mathbb{R}$ la función de coste y $u \in \Omega$ de forma que J se supone derivable en un entorno de u. Se considera el problema minimización sin restricciones:

$$u \in \Omega,\ J(u) = \min_{x \in \Omega} J(x). \tag{3.1}$$

Para ello, se construye una sucesión $\{x_k\}$,

$$J(x_k) \subset J(x_{k-1}).$$

El gradiente de una función $J : \mathbb{R}^n \longrightarrow \mathbb{R}$, es el vector de las derivadas parciales de la función,

$$\nabla J = \left(\frac{\partial J}{\partial x_1}, \ldots, \frac{\partial J}{\partial x_n} \right)^t.$$

Puesto que el gradiente de una función indica el sentido del valor máximo y el objetivo del problema es minimizar la función J, se considera el sentido contrario para llevar a cabo la iteración. Esto es, avanzamos en la dirección de $-\nabla J(x_k)$, que denotamos por d_k. Formalmente se tiene que:

***Definición* 3.1.2.** *Dirección de descenso*

El vector $d_k \in \mathbb{R}^n$ es una dirección de descenso en x_k si verifica:

$$d_k^T \nabla J(x_k) < 0, \quad si\ \nabla J(x_k) \neq 0,$$
$$d_k = 0, \quad si\ \nabla J(x_k) = 0.$$

Por tanto, la estructura del método es la siguiente:

- Paso 1. Se da un vector inicial, x_0, el parámetro de tolerancia $\epsilon > 0$, un número máximo de iteraciones y se emplea el contador de iteraciones $k = 0$.
  Se calcula d_0 la dirección de descenso asociada al gradiente en x_0 de la forma $d_0 = -\nabla J(x_0)$.

- Paso 2. Test de parada: $||d_k|| = ||\nabla J(x_k)|| < \epsilon$.
  Si se verifica el test de parada entonces hemos encontrado el valor que minimiza la función J.

- Paso 3. Seleccionar α_k tal que: $J(x_{k+1}) < J(x_k)$, siendo
  $x_{k+1} = x_k + \alpha_k d_k$.

- Paso 4. Actualizar el valor de d_k de la forma $d_k = -\nabla J(x_k) < 0$.

- Paso 5. $k = k + 1$ y se vuelve al segundo paso.

Una vez conocemos la estructura general del método es necesario estudiar como se actualiza el paso α_k en cada iteración. Existen distintas formas de llevar a cabo la elección del paso α_k, que dan lugar a tres tipos de métodos: métodos con paso fijo, con paso variable y con paso óptimo.

- Métodos de descenso con paso fijo: son tales que $\alpha_k = \alpha > 0, \ \forall k \leq 0$.

- Métodos de descenso con paso variable: emplean distintas reglas para asegurar la condición de descenso y el tamaño del paso, de forma que se mantenga la estabilidad del algoritmo y no dispare el coste computacional.

  Con esta idea, a partir de los datos x_k y d_k construimos $j_k(\alpha) = J(x_k + \alpha d_k)$.
  Para cada α_k existen tres criterios que clasifican el paso: CPG, criterio de paso grande, CPP, criterio de paso pequeño y CPA, criterio de paso aceptable. El hecho de que se verifique este último equivale a que los otros dos no se satisfagan. Por tanto, para asegurar un tamaño de paso aceptable deben verificarse las siguientes hipótesis:

  1. $\{\alpha_k \in \mathbb{R}^+ \text{ tales que } CPG(\alpha_k) = S, CPP(\alpha_k) = S\} = \emptyset$.
  2. $CPP(0) = S$.
  3. $CPG(\infty) = S$.
  4. $\alpha_p, \ \alpha_g \in \mathbb{R}^+$ tales que $\alpha_p < \alpha_g$ y $CPP(\alpha_p) = S, \ CPG(\alpha_g) = S$.
     Entonces existirá $\alpha_a \in (\alpha_p, \alpha_g)$ tal que $CPA(\alpha_a) = S$.

  El primer punto indica que el conjunto de parámetros de tamaño de paso, α, que verifiquen el criterio de paso grande y el criterio de paso pequeño simultáneamente es vacío, es decir, no es posible considerar un tamaño de paso demasiado grande y demasiado pequeño al mismo tiempo. El segundo y tercer punto indican, respectivamente, que cero es un tamaño de paso inadmisible por ser demasiado pequeño e infinito por ser demasiado grande. Finalmente, el cuarto punto establece que, si existen tamaños de paso $\alpha_p, \ \alpha_g \in \mathbb{R}^+$, verificando que α_p es demasiado pequeño y α_g demasiado grande, entonces al intervalo (α_p, α_g) pertenecerá un tamaño de paso admisible, α_a.

  Dos de las reglas más usadas para verificar si la elección del paso es aceptable son las de Goldstein y Wolfe-Powell.
  La regla de Goldstein consiste en elegir α_k de modo que se verifique:

  - $CPG(\alpha_k) : j_k(\alpha) > j_k(0) + j_k'(0)\rho_1\alpha_k, \ \ \rho_1 \in (0, 1)$.

- $CPP(\alpha_k) : j_k(\alpha) < j_k(0) + j_k'(0)\rho_2\alpha_k, \ \ \rho_2 \in (\rho_1, 1)$.
- $CPA(\alpha_k) : j_k(\alpha) \leq j_k(0) + j_k'(0)\rho_1\alpha_k$ y $j_k(\alpha) \geq j_k(0) + j_k'(0)\rho_2\alpha_k$.

En la práctica se toma como α_0 el punto medio del intervalo $[\alpha_p, \alpha_g]$. Si las condiciones de esta regla no se verifican se toma como nuevo intervalo uno de los dos subintervalos $[\alpha_p, \alpha], [\alpha, \alpha_g]$ de $[\alpha_p, \alpha_g]$.

La regla de Wolfe-Powell consiste en elegir α_k de modo que se verifique:

- $CPG(\alpha_k) : j_k(\alpha) > j_k(0) + j_k'(0)m_1\alpha_k, \ \ m_1 \in (0, 1)$.
- $CPP(\alpha_k) : j_k(\alpha) \leq j_k(0) + j_k'(0)m_1\alpha_k$ y $j_k'(\alpha_k) < j_k'(0)m_2, \ \ m_2 \in (m_1, 1)$.
- $CPA(\alpha_k) : j_k(\alpha) \leq j_k(0) + j_k'(0)m_1\alpha_k$ y $j_k(\alpha) \geq j_k'(0)m_2$.

La regla de Wolfe-Powell es más precisa que la de Goldstein, ya que asegura que la $j'(\alpha_k)$ tendrá un valor pequeño. No obstante a nivel computacional Goldstein es menos costoso, ya que solo requiere de la evaluación de J frente a la de Wolfe-Powell que require además de la evaluación de ∇J en cada iteración de la regla.

Por último veremos un resultado para la convergencia de los métodos de gradiente con paso variable.

Teorema 3.1. *Sea $J : \mathbb{R}^n \longrightarrow \mathbb{R}$ diferenciable y tal que existen dos constantes $m, M > 0$ tales que:*

$$||\nabla J(v) - \nabla J(w)|| \leq M||v - w||, \ \ \forall v, w \in \mathbb{R}^n \ \ (\nabla J \text{ es Lipschitziano}).$$
$$(\nabla J(v) - \nabla J(w), v - w) \geq m||v - w||^2, \ \ \forall v, w \in \mathbb{R}^n \ \ (J \text{ es elíptica}).$$

Entonces cualesquiera que sean a, b tales que: $0 < a < b < \frac{2m}{M^2}$, si $\alpha_k \in [a, b], \forall k \geq 0$, el método del gradiente con paso variable es globalmente convergente a la única solución del problema de minimización: encontrar $\bar{x} \in \mathbb{R}^n$ tal que $J(\bar{x}) = \min_{x \in \mathbb{R}^n} J(x)$. Además, la convergencia es al menos lineal:

$$\exists \beta = \beta(m, M, a, b) < 1 \text{ tal que } ||x_{k-1} - \bar{x}|| < \beta||x_k - \bar{x}||.$$

- Métodos de descenso con paso óptimo: se escoge α_k para producir el máximo descenso posible en la dirección d_k. En este caso se tiene que:

$$\alpha_k = \min_{\alpha \in \mathbb{R}^+} j_k(\alpha),$$

siendo $j_k : \mathbb{R} \longrightarrow \mathbb{R}$, con $j_k(\alpha) = J(x_k - \alpha\nabla J(x_k)) = J(x_k + \alpha d_k)$.

Observación 3.2. Como consecuencia de esto es fácil ver que $\nabla J(x_{k+1}) \perp \nabla J(x_k)$. Para demostrarlo, basta notar que α_k es solución de $j_k'(\alpha_k) = 0$. Así pues:

$$0 = j_k'(\alpha_k) = -\nabla J(x_k - \alpha_k \nabla J(x_k))^t \nabla J(x_k).$$

Ahora, teniendo en cuenta que $x_{k+1} = x_k - \alpha_k \nabla J(x_k)$, tenemos que: $0 = \nabla J(x_{k+1}) \nabla J(x_k)$.

El método de máximo descenso se propuso inicialmente para la resolución de sistemas lineales. Como resolver sistemas lineales es equivalente a minimizar funciones cuadráticas definidas positivas, este método fue modificado y desarrollado para problemas de minimización sin restricciones.

***Definición* 3.1.3.** *$J : \mathbb{R}^n \longrightarrow \mathbb{R}$ es un funcional cuadrático si es de la forma:*

$$J(x) = \tfrac{1}{2} x^T A x - b^T x + c,$$

con $A \in \mathscr{M}_{n\times n}$ simétrica definida positiva, $b \in \mathbb{R}^n, c \in \mathbb{R}$.

A continuación obtendremos el paso óptimo para el caso particular de funciones cuadráticas.
En primer lugar calculamos el gradiente de la función cuadrática,

$$J(x_k) = \tfrac{1}{2} x_k^T A x_k - b^T x_k + c.$$

Y obtenemos,

$$\nabla J(x_k) = A x_k - b. \tag{3.2}$$

Luego por la definición de $\alpha_k = \text{mín}_{\alpha \in \mathbb{R}^+} j_k(\alpha)$, siendo $j_k : \mathbb{R} \longrightarrow \mathbb{R}$, con $j_k(\alpha) = J(x_k + \alpha d_k)$ para cada $k \geq 0$, se tiene que α_k es solución de $j_k'(\alpha) = 0$. Por tanto, como ya habíamos visto,

$$0 = j_k'(\alpha_k) = -\nabla J(x_k + \alpha_k d_k)^t \nabla J(x_k).$$

Ahora teniendo en cuenta la expresión de una función cuadrática y (3.2), podemos reescribirlo de la siguiente manera:

$$[A(x_k + \alpha_k d_k) - b]^t d_k = 0,$$

o, equivalentemente,

$$(x_k + \alpha_k d_k)^t A^t d_k - b^t d_k = 0.$$

Desarrollando la expresión anterior,

$$x_k^t A^t d_k - b^t d_k + \alpha_k d_k^t A^t d_k = (x_k^t A^t - b^t) d_k + \alpha_k d_k^t A d_k = 0.$$

Teniendo en cuenta de nuevo (3.2) y la notación, $-d_k = \nabla J(x_k)$ que hemos empleado,

$$-d_k^t d_k - \alpha_k d_k^t A^t d_k = 0.$$

Luego, la elección de paso óptimo se realiza de la siguiente forma, donde se tiene en cuenta que A es simétrica:

$$\alpha_k = \frac{d_k^t d_k}{d_k^t A d_k} = \frac{||\nabla J(x_k)||^2}{d_k^t A d_k}.$$

Teorema 3.3. *Sea $J : \mathbb{R}^n \longrightarrow \mathbb{R}$ una función elíptica, esto es, existe $m > 0$ tal que:*

$$(\nabla J(v) - \nabla J(w), v - w) \geq m||v - w||^2, \ \ \forall v, w \in \mathbb{R}^n.$$

Entonces el método de gradiente con paso óptimo converge al único elemento $u \in \mathbb{R}^n$ solución del problema 3.1.

3.2. Método de Adam

Como ya hemos indicado, el optimizador Adam es uno de lo más utilizados en la actualidad. Se emplea en el proceso de entrenamiento con el objetivo de mejorar el valor de la función de coste hasta minimizarlo. En la sección anterior hemos estudiado en profundidad la estructura de los métodos de descenso, al cual pertenece Adam. Además presenta otra peculiaridad y es que se trata de un modelo estocástico, esto es, toma valores aleatorios con respecto al tiempo para cada una de las variables que estudia. En esta sección analizaremos su funcionamiento (ver [15]). No obstante, los métodos de optimización correspondientes al aprendizaje automático difieren notablemente de la optimización clásica, por lo que para entender correctamente el algoritmo, es preciso estudiar en primer lugar estas diferencias. Veremos como la función objetivo, es decir, la función a minimizar, que usualmente se correspondería con la función de coste, varía ligeramente en este tipo de problemas, según se puede ver en [21]. A continuación veremos las principales diferencias:

1. La función objetivo se descompone en la suma de funciones a lo largo de las muestras del conjunto de entrenamiento. Aquellos métodos que emplean una sola muestra por iteración se denominan estocásticos, como es el caso de Adam.

2. En muchos procesos de aprendizaje automático, nos interesa minimizar, además de la función de coste en si, algún otro parámetro, P, asociado al conjunto de entrenamiento, como por ejemplo la eficiencia, lo cual puede resultar difícil de calcular. Por tanto es necesario hacerlo indirectamente, esto es, minimizar una función de coste distinta $J(\theta)$ con la esperanza de reducir al mismo tiempo P. Entonces se puede reescribir la función de coste de la siguiente forma:

$$J(\theta) = \mathbb{E}[x, y] \sim_{p_{datos}} J^*(f(x;\theta), y), \tag{3.3}$$

donde J^* es la función de coste por muestra del conjunto de datos, $f(x;\theta)$ es la predicción cuando los datos de entrada son x, p_{datos} es la distribución del conjunto de datos e y es la etiqueta correspondiente. El objetivo es reducir el error esperado dado por la función (3.3).

Cabe notar que la distribución del conjunto de datos p_{datos} es desconocida, por tanto es necesario minimizar la pérdida esperada en el conjunto de entrenamiento, esto es, reemplazar la distribución real de los datos, p_{datos}, por una empírica, $\bar{p}_{datos}$ definida por el conjunto de entrenamiento. Así el problema de minimización se modifica de la siguiente forma:

$$J(\theta) = \mathbb{E}[x, y] \sim_{p_{datos}} J^*(f(x;\theta), y) = \frac{1}{m+1} \sum_{i=1}^{m+1} J^*(f(x^{(i)};\theta), y^{(i)}), \tag{3.4}$$

donde m es el número de muestras del conjunto de entrenamiento. Esto es, estamos ante un problema de mínimos cuadrados, en el cual, a partir de un conjunto de pares ordenados, en este caso formados por los datos de entrada, x, y las etiquetas asociadas a estos, y, y una familia de funciones continuas, $\{J_k\}$, donde cada J_k es la función objetivo en el instante k, se busca minimizar la suma de cuadrados de las diferencias, que reciben el nombre de residuos, entre las etiquetas predichas a partir de los datos de entrada, $f(x;\theta)$, y las verdaderas etiquetas asociadas a los datos, y.

Para esta nueva formulación del problema estamos minimizando el riesgo empírico. La desventaja que presenta esta formulación es que puede llevar a *overfitting* [1], esto es, el algoritmo memoriza el conjunto de entrenamiento en lugar de aprender de él.

Una vez hemos visto las peculiaridades propias de los modelos de optimización para algoritmos de aprendizaje automático, estamos en condiciones de conocer el funcionamiento y la estructura del optimizador Adam. En primer lugar definiremos los parámetros, funciones y variables que intervienen en el proceso.

[1] Si el conjunto de datos que utilizamos para entrenar no son representativos de la clase de elementos que estamos estudiando, entonces nos encontramos ante un error de sobreajuste o *overfitting*.

Se trata de un algoritmo en el que las variables dependen del tiempo, por tanto, en primer lugar tendremos el intervalo $[1, T]$ que discretizaremos con un tamaño de paso α.

Por otra parte, la función objetivo $J(\theta)$, se define como una función escalar estocástica diferenciable con respecto a sus parámetros. En cada instante de tiempo podemos denotar la función objetivo por $J_1(\theta), \ldots, J_T(\theta)$, esto es, la función objetivo es la suma de subfunciones evaluadas en distintos subconjuntos del total de los datos. El propósito de este algoritmo es encontrar el valor mínimo de la función, que denotaremos por, $\mathbb{E}(J(\theta))$.
A lo largo de esta sección denotaremos por $g_k = \nabla J_k(\theta)$, al vector gradiente de J_k con respecto al parámetro θ evaluado en el paso de tiempo k. Y por g_k^2, al producto realizado elemento a elemento, $g_k \odot g_k$.
Otros parámetros que juegan un papel importante son: $\beta_1, \beta_2 \in [0, 1)$, que representan la tasa de decaimiento para la estimación de los momentos. Esto es, el algoritmo actualiza en cada iteración el estimador de primer momento, m_k y el estimador de segundo momento, v_k, y β_1, β_2 controlan estas actualizaciones. Normalmente se fijan los valores $\beta_1 = 0{,}9$ y $\beta_2 = 0{,}99$. Nótese que m_k es una estimación del primer momento, es decir representa la media, mientras que v_k, es una estimación del segundo momento del gradiente, que representa la varianza no centrada.

Antes de analizar la estructura del algoritmo es preciso conocer el método de momentum (Polyak, 1964), que fue diseñado para acelerar el aprendizaje. Funciona de la siguiente manera; acumula una media móvil de decaimiento exponencial de los gradientes correspondientes a las iteraciones previas, y continua avanzando en la dirección fijada, como se puede ver en la Figura 3.2. Formalmente, este algoritmo introduce una variable, m, que especifica la dirección y velocidad a la que se desplazan los parámetros a lo largo del espacio de parámetros. Mientras que la velocidad se fija con una media de decaimiento exponencial correspondiente al valor del gradiente con signo negativo, es precisa la existencia de un hiperparámetro, que en este caso concreto hemos denotado por $\beta_1 \in [0, 1)$ que determina como de rápido decaen las contribuciones de los gradientes procedentes de iteraciones previas, de la siguiente manera,

$$m_k = \beta_1 m_{k-1} - \epsilon \nabla_\theta(J(\theta)) = \beta_1 m_{k-1} - \epsilon g_k \ ,$$
$$\theta_k = \theta_{k-1} + m_k.$$

En la Figura 3.2 se puede apreciar una representación gráfica del funcionamiento de este método. Las líneas de contorno representan una función de pérdida cuadrática con una matriz hessiana mal condicionada. El trazado en rojo indica el camino que sigue el método de momentum cuando minimiza la función. Por último, las flechas indican el paso que el método de descenso tomaría en cada punto.

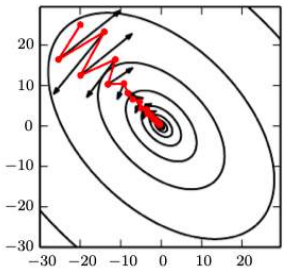


Figura 3.2: Representación gráfica del método de momentum (ver [21]).

El algoritmo Adam incorpora este método directamente como una estimación de primer orden del gradiente. Además incluye correcciones del sesgo para estimar tanto los momentos de primer como de segundo orden. Esto es, emplea el método de momentum para estimar m_k, v_k, con los hiperparámetros β_1, β_2 que representan la tasa de decaimiento exponencial en cada iteración y además mejora dichas estimaciones a través de $\bar{m_k}, \bar{v_k}$.

La estructura del algoritmo es la siguiente:

- Paso 1: dados el tamaño de paso α, los hiperparámetros β_1, β_2, la función objetivo $J(\theta)$ y el valor inicial θ_0 . Además también es necesario inicializar el parámetro $k = 0$, el vector de primer momento $m_0 = 0$ y el vector de segundo momento $v_0 = 0$.
- Paso 2: mientras el parámetro θ_k no converja:

$$\begin{aligned} k &= k + 1, \\ g_k &= \nabla_\theta J_k(\theta_{k-1}), \\ m_k &= \beta_1 m_{k-1} + (1 - \beta_1) g_k, \\ v_k &= \beta_2 v_{k-1} + (1 - \beta_2) g_k^2, \\ \bar{m_k} &= \frac{m_k}{1 - \beta_1^k}, \\ \bar{v_k} &= \frac{v_k}{1 - \beta_2^k}, \\ \theta_k &= \theta_{k-1} - \alpha \frac{\bar{m_k}}{\sqrt{\bar{v_k}} + \epsilon}. \end{aligned}$$

- Paso 3: por último el algoritmo devuelve los parámetros θ_k.

La dirección de descenso del método es $d_k = -\frac{\bar{m_k}}{\sqrt{\bar{v_k}} + \epsilon}$, ya que los parámetros se actualizan de la forma,

$$\theta_k = \theta_{k-1} - \alpha \frac{\bar{m_k}}{\sqrt{\bar{v_k}}+\epsilon}.$$

En este caso la dirección de descenso depende de las estimaciones de primer y segundo momento del gradiente de la función a optimizar y de β_1, β_2 que controlan las actualizaciones de dichas estimaciones. A continuación estudiaremos detalladamente la actualización de tamaño de paso y de la corrección del sesgo, esto es, $\bar{m_k}, \bar{v_k}$.

3.2.1. Regla de actualización

Este algoritmo utiliza momentum para adaptar el tamaño de paso, α, lo cual le proporciona estabilidad al método. Suponiendo que $\epsilon = 0$,

$$\Delta_k = \alpha \frac{\bar{m_k}}{\sqrt{\bar{v_k}}} = \alpha \frac{m_k \sqrt{1-\beta_2^k}}{\sqrt{v_k}(1-\beta_1^k)}$$

donde Δ_k es menos la dirección de descenso multiplicada por el tamaño de paso.
Esto permite que la actualización de este parámetro genere valores que resultan efectivos. Además es posible acotarlo superiormente. Efectivamente, si se verifica la desigualdad $(1-\beta_1) > \sqrt{1-\beta_2}$, entonces se tiene que:

$$|\Delta_k| < \alpha \frac{(1-\beta_1)}{\sqrt{1-\beta_2}}. \tag{3.5}$$

En otro caso:

$$|\Delta_k| < \alpha. \tag{3.6}$$

La primera acotación, (3.5), se utiliza para casos severos de dispersión, como cuando el valor del gradiente es cero en todas las iteraciones excepto en la actual.
Cuando $(1-\beta_1) = \sqrt{1-\beta_2}$ se tiene que $|\frac{\bar{m_k}}{\sqrt{\bar{v_k}}}| < 1$ [2] y por tanto es fácil ver que $|\Delta_k| < \alpha$. Esto indica que el paso efectivo para cada iteración en k, está acotado por el valor del parámetro de tamaño de paso actual α. Esta situación se podría interpretar como si existiera una región de confianza alrededor del valor del parámetro tamaño de paso actual. De modo que a partir de esta región determinada por α, el valor de estimación del gradiente no genera información suficiente.

Por otra parte, como α establece una cota superior de la magnitud de los pasos, deducir su orden nos permitiría que el valor óptimo de la función objetivo se alcanzase a partir del iterante inicial, θ_0, en un número de iteraciones.

Por último vamos a estudiar algunas propiedades de elección del paso. En primer lugar, la relación con respecto al siguiente concepto:

[2] En la mayor parte de los casos se tiene que $\frac{\bar{m_k}}{\sqrt{\bar{v_k}}} \approx \pm 1$, ya que , $|\mathbb{E}[g]/\sqrt{\mathbb{E}[g^2]}| \leq 1$.

***Definición* 3.2.1.** *SNR*

Se define la tasa de señal del ruido, SNR, como $\frac{\bar{m_k}}{\sqrt{\bar{v_k}}}$. *Valores pequeños de SNR provocan que* $\Delta_k = \alpha\frac{\bar{m_k}}{\sqrt{\bar{v_k}}}$ *esté próximo a cero. Ya que SNR toma valores pequeños cuando hay una menor certeza acerca si la dirección de* $\bar{m_k}$ *se corresponde con la dirección del verdadero gradiente.*

Además, para considerar una elección de paso efectiva es necesario que sea invariante al proceso de reescalado del gradiente. Veamos que esto se cumple, reescalar el gradiente con una constante c, supondría reescalar $\bar{m_k}$ con un factor c y $\bar{v_k}$ con un factor c^2 que se cancelarían, de la siguiente manera :

$$cg = \frac{c\bar{m_k}}{\sqrt{c^2\bar{v_k}}} = \frac{c\bar{m_k}}{c\sqrt{\bar{v_k}}} = \frac{\bar{m_k}}{\sqrt{\bar{v_k}}} = g.$$

3.2.2. Actualización de corrección del sesgo

Adam incluye una mejora importante con respecto a otros estimadores, la corrección del sesgo en la estimación de los momentos. En este apartado estudiaremos porqué la actualización de los estimadores de primer y segundo momento del gradiente, m_k, v_k, dan lugar a

$$\bar{m_k} = \frac{m_k}{(1-\beta_1^k)},$$
$$\bar{v_k} = \frac{v_k}{(1-\beta_2^k)}.$$

En primer lugar estudiaremos la actualización del estimador de primer momento del gradiente, m_k, con respecto a la tasa de decaimiento exponencial β_1. Teniendo en cuenta los resultados de las iteraciones previas, podemos reescribirlo de la siguiente manera:

$$\begin{aligned} m_k &= \beta_1 m_{k-1} + (1-\beta_1)g_k = \\ &= \beta_1^2 m_{k-2} + \beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k = \\ &= \beta_1^k m_0 + \beta_1^k(1-\beta_1)g_0 + \beta_1^{k-1}(1-\beta_1)g_1 +, \ldots, +\beta_1(1-\beta_1)g_{k-1} + (1-\beta_1)g_k. \end{aligned}$$

Debido a que el iterante inicial, $m_0 = 0$,

$$m_k = (1-\beta_1)\sum_{i=0}^{k}\beta_1^{k-i}g_i.$$

Queremos conocer el valor de $\mathbb{E}[m_k]$, siendo $g_1, \ldots, g_T$ los gradientes de $J_1, \ldots, J_T$, es decir, los gradientes de cada una de las funciones evaluadas en subconjuntos del total de los datos. Luego, [3]

[3] Teniendo en cuenta que si dos variables x e y son independientes, entonces $\mathbb{E}[xy] = \mathbb{E}[x]\mathbb{E}[y]$.

$$\mathbb{E}[m_k] = \mathbb{E}[g_k](1-\beta_1)\sum_{i=0}^{k}\beta_1^{k-i}.$$

Ahora, teniendo en cuenta que las series geométricas se pueden expresar de la siguiente manera, $\sum_{k=0}^{n-1} x^k = \dfrac{1-x^n}{1-x}$. Se tiene que:

$$\sum_{i=0}^{k}\beta_1^{k-i} = \frac{1-\beta_1^k}{1-\beta_1}.$$

Luego,

$$\mathbb{E}[m_k] = \mathbb{E}[g_k](1-\beta_1^k).$$

Y finalmente,

$$\bar{m_k} = \frac{m_k}{1-\beta_1^k}.$$

De forma análoga estudiaremos la actualización del estimador de segundo momento del gradiente, v_k, con respecto a la tasa de decaimiento exponencial β_2. Teniendo en cuenta los resultados de las iteraciones previas, podemos reescribirlo de la siguiente manera:

$$\begin{aligned} v_k &= \beta_2 v_{k-1} + (1-\beta_2)g_k^2 = \\ &= \beta_2^k v_0 + \beta_2^k(1-\beta_2)g_0^2 + \beta_2^{k-1}(1-\beta_2)g_1^2 + \cdots + \beta_2(1-\beta_2)g_{k-1}^2 + (1-\beta_2)g_k^2. \end{aligned}$$

De nuevo, debido a que el iterante inicial, $v_0 = 0$,

$$v_k = (1-\beta_2)\sum_{i=0}^{k}\beta_2^{k-i}g_i^2.$$

Queremos conocer el valor de $\mathbb{E}[v_k]$, luego

$$\mathbb{E}[v_k] = \mathbb{E}[g_k^2](1-\beta_2)\sum_{i=1}^{k}\beta_2^{k-i}.$$

En este caso, la definición de serie geométrica da lugar a la igualdad:

$$\sum_{i=0}^{k}\beta_2^{k-i} = \frac{1-\beta_2^k}{1-\beta_2}.$$

Luego,

$$\mathbb{E}[v_k] = \mathbb{E}[g_k^2](1-\beta_2^k).$$

Finalmente,

$$\bar{v_k} = \frac{v_k}{1-\beta_2^k}.$$

3.2.3. Convergencia del modelo.

Como ya hemos visto en la sección anterior las condiciones necesarias para la convergencia de cada método pasan por un análisis previo de su estructura y funcionamiento. Para analizar la convergencia de Adam es necesario estudiar en primer lugar el concepto de *regret function* o función de pesar:

***Definición* 3.2.2.** *Regret function*

Dada una sucesión arbitraria de funciones de coste convexas $J_1(\theta), \ldots, J_T(\theta)$. Se define la función de pesar como:

$$R(T) = \sum_{i=1}^{T} [J_k(\theta_k) - J_k(\theta^*)],$$

donde $\theta^* = \min_{\theta \in \chi} \sum_{i=1}^{T} J_k(\theta)$, χ el conjunto de todos los parámetros correspondientes a las iteraciones previas.

Esto es, la función realiza la suma en cada instante de tiempo, de todas las diferencias entre la predicción del algoritmo, $J_k(\theta_k)$ y la mejor aproximación de entre el conjunto χ de resultados previos, $J_k(\theta^*)$.

A continuación se enuncia el teorema para la convergencia del algoritmo dada en [15]:

Teorema 3.4. *Suponiendo que la función J_k tiene gradientes acotados: $||\nabla J_k(\theta)||_2 \leq G$, $||\nabla J_k(\theta)||_\infty \leq G_\infty$, para todo $\theta \in R^d$ y la distancia entre cualquier θ_k generada por Adam está acotada, $||\theta_n - \theta_m||_2 \leq D$, $||\theta_n - \theta_m||_\infty \leq D_\infty$, para $m, n \in \{1, \ldots, T\}$ cualesquiera, y $\beta_1, \beta_2 \in [0, 1)$ satisfaciendo $\frac{\beta_1^2}{\sqrt{\beta_2}} < 1$. Sea $\alpha_t = \frac{\alpha}{\sqrt{t}}$ y $\beta_{1,t} = \beta_1 \lambda^{t-1}$, $\lambda \in (0, 1)$. Entonces para todo $T \geq 1$,*

$$R(T) \geq \frac{D^2}{2\alpha(1-\beta_1)} \sum_{i=1}^{d} \sqrt{T\bar{v}_{T,i}} + \frac{\alpha(1+\beta_1)G_\infty}{(1-\beta_1)\sqrt{1-\beta_2}(1-\gamma)^2} \sum_{i=1}^{d} ||g_{1:T,i}||_2 +$$

$$+ \sum_{i=1}^{d} \frac{D_\infty^2 G_\infty \sqrt{1-\beta_2}}{2\alpha(1-\beta_1)(1-\lambda)^2}$$

No obstante, Josef Goppold y Sebastian Bock descubrieron, ambos a la hora de realizar sus tesis, que hay errores en la prueba que ofrece [15] de este teorema. Una demostración mejorada de este resultado se puede encontrar en [16].

Capítulo 4

Resultados

En los capítulos anteriores hemos visto las bases del aprendizaje automático detallando el aprendizaje supervisado, en el que se le proporcionan a la máquina un conjunto de datos etiquetados, que consta de dos fases: una primera fase de entrenamiento, en la cual el algoritmo aprende de parte de los datos estableciendo una relación entre las características y las etiquetas. Y una segunda fase de prueba que emplea el resto de los datos para comprobar que el mecanismo funciona correctamente y es tan preciso como es posible.
También se estudiaron en el primer capítulo algunos de los métodos principales a la hora de manipular los datos de entrada en el aprendizaje automático supervisado: la regresión lineal y la clasificación. Esta última emplea la regresión logística que genera medidas de probabilidad.

El proceso de clasificación basa su funcionamiento en las redes neuronales artificiales. En concreto, en el capítulo 2 se analizaron las redes neuronales convolucionales que procesan datos de tipo cuadrícula, especialmente empleadas en la clasificación de imágenes. A lo largo del capítulo se estudian los tipos de capas que contiene una CNN.

Por último, en el tercer capítulo se analiza la optimización de la función de coste, esto es, durante la fase de entrenamiento el mecanismo de retropropagación es el responsable de ir actualizando los valores de los pesos asociados a cada capa de la red neuronal, para lo cual hace uso de un optimizador. Este tercer capítulo estudia uno de los métodos que generan mejores resultados y de los más usados en la actualidad, Adam.

Una vez que conocemos el funcionamiento interno de las redes neuronales, dedicaremos este capítulo a ver un caso práctico de aprendizaje automático supervisado que utiliza las CNN en un algoritmo dedicado a la clasificación de imágenes. Emplearemos, en la fase de entrenamiento, el método Adam, que estudiamos en el capítulo anterior, como optimizador.

4.1. Conjunto de datos

Dedicaremos esta sección a conocer las características del conjunto de datos que usaremos para entrenar el programa. Utilizaremos el conjunto CIFAR-10, introducido por Alex Krizhevsky y Geoffrey Hinton (ver[17]). Se trata de 6000 imágenes en las que se pueden distinguir diez tipos de objetos o clases: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión.

Cada una de las fotografías tiene asociado cierto ruido, que aparece en el proceso de etiquetado. El etiquetado se hizo de la siguiente manera: los autores de este conjunto de datos pagaron a grupos de estudiantes para que clasificaran las imágenes. A cada estudiante se le asignó una categoría y debían examinar todas las imágenes pertenecientes a ella, con el objetivo de filtrar aquellas que estuvieran mal etiquetadas. Además se examinaron también las fotografías que se podían encontrar por un hipónimo del término de búsqueda, esto es, una palabra cuyo significado incluye el de otra.
El criterio utilizado para decidir si una imagen pertenece a una clase es el siguiente:

- El nombre de la clase debería estar arriba en la lista de posibles respuestas a la pregunta "¿Qué hay en esta fotografía? "
  Esto es, el foco principal de la fotografía debe ser el objeto perteneciente a una de las clases, por tanto tiene que ser aquello que identificamos en primer lugar, sin importar otros elementos que puedan aparecer en la imagen. Por ejemplo, en las imágenes correspondientes a barcos, es probable que también salga el mar en las fotografías; no obstante, lo que más llame la atención ha de ser el barco.

- La imagen debe ser realista, esto es, se excluyen dibujos.

- La fotografía debe ilustrar únicamente el objeto al que refiere la clase.

- El objeto puede encontrarse parcialmente oculto o ser visto desde un punto de vista poco usual siempre que se pueda identificar fácilmente.

Por último se eliminaron aquellas imágenes duplicadas haciendo uso de la norma L_2 para comparar las fotografías.

Una vez que el conjunto de datos está correctamente etiquetado se puede utilizar para programar empleando lenguaje automático supervisado. Recordemos que el conjunto de imágenes de CIFAR-10 contiene 6000 fotografías y es necesario dividirlo en dos subconjuntos: un set de entrenamiento formado por 5000 imágenes seleccionadas de forma aleatoria, y un set de prueba formado por las 1000 imágenes restantes.

Otra característica de este set de imágenes que resulta importante a la hora de construir el algoritmo, es que se trata de fotografías de tamaño 32×32 píxeles. Como ya vimos en el

Capítulo 2, estas imágenes se pueden expresar como matrices de píxeles. Además se trata de fotografías a color, por lo que necesitaremos tres canales: rojo, verde y azul. En el caso de fotografías en blanco y negro solamente sería necesario un canal, ya que se trata de una escala de grises. Teniendo en cuenta esto, es fácil ver que la estructura de nuestros conjuntos de datos es la siguiente:

$$\text{Conjunto de entrenamiento: } (5000, 32, 32, 3).$$
$$\text{Conjunto de prueba: } (1000, 32, 32, 3).$$

Por otra parte, tenemos los vectores de etiquetas correspondientes a las fases de prueba y entrenamiento. En primer lugar definimos una lista, a la que vamos a llamar *classnames* que contendrá los nombres de cada una de las clases con las que hemos etiquetado las imágenes. A cada clase o etiqueta se le asocia un valor que va del cero al nueve, ya que en total hay diez posibles categorías. Por tanto, los conjuntos de etiquetas se expresarán como vectores fila con 5000 entradas en el caso del conjunto de etiquetas para el entrenamiento y con 1000 entradas para el de prueba.

Finalmente es necesario pre-procesar los datos ya que, para cada uno de los conjuntos de imágenes los valores que toman los píxeles varía entre cero y doscientos cincuenta y cinco en la escala de color. Por tanto es necesario normalizar los datos pasándolos a un rango que vaya de cero a uno. Para ello dividimos nuestros sets de datos de entrenamiento y prueba entre doscientos cincuenta y cinco.

4.2. Construcción del modelo de redes neuronales

Para construir el modelo de redes neuronales haremos uso de Keras, una librería de alto rendimiento de Tensorflow para crear y entrenar modelos de aprendizaje profundo. Se utiliza para la confección rápida de prototipos, la investigación de vanguardia y en producción (ver [22]). Tensorflow es una plataforma de código abierto de Google para el aprendizaje automático. Emplearemos su librería para entrenar CIFAR-10 y posteriormente evaluar su efectividad en la fase de prueba.

En primer lugar, construiremos un modelo de tipo *sequential* en el cual, como vimos en el Capítulo 2 que resultaba más efectivo, alternaremos capas convolucionales con capas de combinación en las que aplicaremos Max-pooling. Finalmente añadiremos capas totalmente conectadas, que en Keras se conocen como capas *Dense*. De entre estas últimas, las capas internas llevan asociadas funciones de activación ReLU, mientras que la última lleva asociada la función softmax.

```
 98
 99  model = keras.Sequential([
100       keras.layers.Conv2D(32,(3,3),input_shape=(32,32,3),padding='same'),
101       keras.layers.MaxPool2D(pool_size=(2,2),strides=(1,1),padding='same'),
102       keras.layers.Conv2D(64,(3,3),padding='same'),
103       keras.layers.MaxPool2D(pool_size=(2,2),strides=(1,1)),
104       keras.layers.Flatten(),
105       keras.layers.Dense(256,activation='relu'),
106       keras.layers.Dense(128,activation='relu'),
107       keras.layers.Dense(10, activation='softmax')
108  ])
```

Figura 4.1: Programación del modelo de red neuronal para el conjunto de datos CIFAR-10.

A continuación describiremos el modelo de red neuronal empleado para entrenar el conjunto de datos CIFAR-10 que se puede ver en la Figura 4.1.

En las líneas de código 100 y 102 añadimos al modelo capas de tipo convolucional, usando el comando *keras.layers.Conv2D*. Como ya vimos, el objetivo de esta clase de capas es aplicar filtros sobre los datos de entrada. Cada uno de estos filtros recibe el nombre de núcleo convolucional. Estos presentan una estructura matricial, al igual que los datos de entrada y se "deslizan", a través de *strides* 2.2.1, hasta recorrer toda la matriz de datos de entrada.

En el caso de la primera capa convolucional aplicada, podemos ver como consta de 32 núcleos de dimensión $(3, 3)$ que se aplican a unos datos de entrada de la forma: $(32, 32, 3)$, esto es debido a que CIFAR-10 está formado por imágenes de tamaño 32×32 a color, por lo que son necesarios 3 canales. Además se hace uso del *padding* para evitar la pérdida de información.

La segunda capa convolucional aplicada presenta una estructura similar; en este caso consta de 64 núcleos de dimensión $(3, 3)$ y también se aplica el padding. No es necesario especificar la forma de los datos de entrada, ya que al ser un modelo de tipo *sequential* toma de forma automática como datos de entrada los datos de salida de la capa anterior.

Combinadas con las capas convolucionales, tenemos en las líneas 101 y 103 del código, capas de combinación cuyo objetivo reside en disminuir la dimensión de la matriz de datos con la que se está trabajando. Consecuentemente tendremos una disminución del coste computacional. La técnica empleada en estas capas es el *Max-pooling* o combinación máxima. Ambas capas modifican subconjuntos de dimensión 2×2 con *strides* de tamaño $(1, 1)$.

A continuación hay una capa *Flatten* en la línea 104 del código. Esta transforma las imágenes de un arreglo bi-dimensional a un arreglo unidimensional de tamaño $n \times n \times 3$ píxeles, donde n es el rango de la matriz de datos procedente de la capa anterior.

Finalmente encontramos tres capas totalmente conectadas. Las dos primeras aplican

activación ReLU, esto es, aseguran que no hay valores menores que cero en el vector. La primera capa cuenta con 256 neuronas, mientras que la segunda consta de 128.
La última capa lleva asociada una función de activación *softmax* de 10 nodos, por tanto generará un arreglo unidimensional de tamaño 10. Cada una de las entradas del vector contiene una calificación que indica la probabilidad de que la imagen que esta siendo estudiada pertenezca a cada una de las diez posibles categorías.

Tras haber construido la red neuronal es necesario crear un algoritmo que entrene el modelo y compruebe que se ha construido correctamente.

- Fase de entrenamiento

  Los comandos *tf.keras.model* agrupan las capas del modelo de red neuronal en un objeto que posee características de entrenamiento e inferencia.

  Una vez que se ha construido el modelo, se puede iniciar la fase de entrenamiento. Para ello haremos uso de algunos de los comandos de la librería keras. En primer lugar, emplearemos *model.compile()* como una etapa previa al inicio del entrenamiento que se corresponde con el análisis que hemos hecho en el Capítulo 3. Esto es, resulta necesario optimizar la función de coste para asegurarnos de que nuestro programa tiene una eficiencia aceptable.
  El comando toma como argumentos: un optimizador (en este caso emplearemos Adam), una función de pérdida (usaremos "sparse categorical crossentropy", que calcula la pérdida cruzada entre las etiquetas y las predicciones. Se trata de una función que tiene la siguiente forma: $f_n(y_{true}, y_{pred})$, donde y_{true} es el fundamento de la verdad de x, es decir, el vector de etiquetas, e y_{pred} es el vector correspondiente a los valores predichos) y una lista de indicadores, para evaluarlos durante las fases de entrenamiento y prueba (en este caso estudiará la precisión de las predicciones, esto es, con que frecuencia las predicciones se corresponden con las etiquetas, o dicho de otra forma el porcentaje de acierto).

  Llegados a este punto, hemos construido una red neuronal que toma como datos de entrada los píxeles de las imágenes de CIFAR-10, que se encuentran en forma matricial, y devuelve un vector con las probabilidades de que cada una de las fotografías pertenezca a cada categoría. Asimismo, hemos empleado el optimizador Adam para mejorar el valor de los pesos asociados a cada capa de la red neuronal, con la consiguiente optimización de la función de coste. Por tanto estamos en condiciones de comenzar con el entrenamiento del modelo, para ello haremos uso del comando *model.fit()*. Este toma como argumentos: el vector de entrenamiento y su respectivo vector de etiquetas y el número de "epochs", esto es, el número de iteraciones para

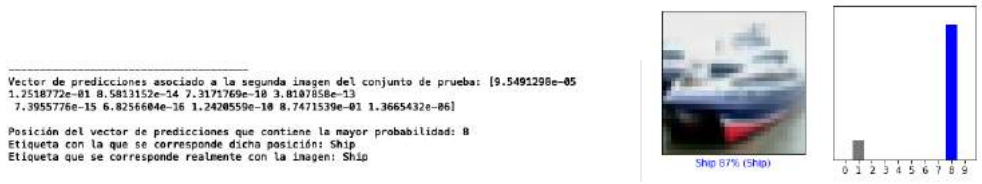
entrenar el modelo, en este caso serán diez.

- Fase de prueba

  Tras la fase de entrenamiento, en la cual el modelo "aprende" del subconjunto de los datos CIFAR-10 dedicado a este propósito, estableciendo una relación entre las etiquetas y el vector de características, se sucede una segunda etapa de prueba o predicción, que emplea lo que resta de los datos para comprobar que el algoritmo funciona correctamente.
  Antes de realizar predicciones es necesario cerciorar que los parámetros internos son coherentes, a través de los valores que toman la función de coste y el indicador que previamente hemos escogido, en este caso la precisión, es decir, el porcentaje de acierto. Esta comprobación la hacemos empleando el comando *model.evaluate()*, que toma como argumentos el conjunto de imágenes y sus respectivas etiquetas, tanto para el set de entrenamiento como para el de prueba. Estos cálculos se realizan por lotes de datos de tamaño treinta y dos por defecto.
  En este caso, obtenemos valores para la función de coste bajos, y para el parámetro de precisión considerablemente altos. Lo cual es un indicativo de que los parámetros internos se están actualizando correctamente.
  Luego estamos en condiciones de realizar predicciones, empleando el comando *model.predict()*, que toma como argumentos las imágenes del conjunto de prueba. De nuevo los cálculos se realizan por lotes de datos de tamaño treinta y dos.

A continuación estudiaremos la validez de dichas predicciones comparándolas con el vector de etiquetas para el conjunto de prueba. Una predicción es un vector de diez elementos, donde cada una de las entradas se corresponde con la probabilidad de que cada etiqueta esté relacionada con la imagen que estamos estudiando. La etiqueta pronosticada se corresponderá con aquella entrada que presente mayor probabilidad. Una vez que tenemos la etiqueta predicha, podemos compararla con aquella que realmente le corresponde. En la Figura 4.2a se puede ver la salida del programa que nos muestra esto mismo para la segunda fotografía del conjunto CIFAR-10. En primer lugar está el vector de probabilidades de que el contenido de la imagen se corresponda con cada etiqueta, a continuación se puede ver que posición ocupa el elemento del vector que presenta una probabilidad más alta y finalmente la etiqueta que se corresponde con dicha posición, *ship*. Podemos ver como el pronóstico que hemos hecho se corresponde con la verdadera etiqueta.
Parece interesante conocer con que probabilidad el algoritmo hará una predicción correcta. En la Figura 4.2b se puede ver que el programa reconoce que la imagen contiene un barco con una eficacia del 87 %. Además, el gráfico de barras nos indica que etiquetas provocan

confusión. Se representa en color azul la barra correspondiente a la probabilidad de que el algoritmo etiquete correctamente la fotografía, y en color gris la probabilidad de que lo haga de forma incorrecta, señalando de esta manera con que otra etiqueta relaciona la imagen. Para el caso de la segunda fotografía podemos ver como la categoría que provoca esta confusión es automóvil.



(a) (b)

Figura 4.2: Comprobación de las predicciones para la segunda imagen de CIFAR-10.

Esta comprobación gráfica resulta especialmente útil debido a la información que proporciona (para las primeras quince imágenes se pueden ver en la Figura 4.3 los resultados obtenidos). Nótese que no todas las imágenes presentan confusión, ya que hay varias fotografías que presentan una eficacia del 100 %; por ejemplo, la primera. Por otro lado, el error a la hora de etiquetar no tiene porque estar ligado únicamente a una sola categoría; por ejemplo, en la tercera fotografía se puede ver como, a pesar de que el algoritmo reconoce el barco con una probabilidad de acierto relativamente alta, del 84 %, hay otras tres clases que asocia de forma errónea a esta imagen, que son aquellas que se corresponden con otros medios de transporte: avión, automóvil y camión. Cabe destacar que, a pesar de que el modelo ha sido capaz de reconocer todas las imágenes y asociarles correctamente una categoría, no presenta una eficacia del 100 % con respecto al conjunto de datos de prueba. Esto se refleja en probabilidades de acierto bajas para algunas fotografías. Este es el caso de la novena imagen, que presenta una probabilidad del 57 % de ser correctamente clasificada. No obstante, podemos afirmar que el algoritmo es suficientemente eficaz a la hora de categorizar imágenes y que puede reconocer con una eficacia media superior al 60 % la etiqueta correspondiente a cada imagen del conjunto de prueba.

Por tanto hemos comprobado como los datos que habíamos obtenido para el modelo son coherentes con los resultados gráficos.

Figura 4.3: Comprobación de las predicciones para las quince primeras fotografías de CIFAR-10.

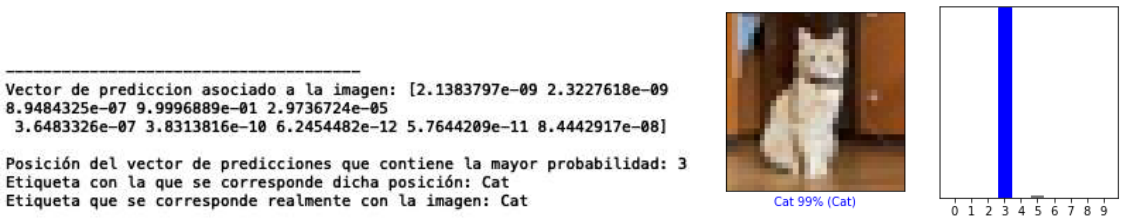
4.3. Comprobación del funcionamiento

En las sección anterior construimos una red neuronal y la entrenamos para la clasificación de imágenes, haciendo uso para ello de un subconjunto de entrenamiento de CIFAR-10, formado por 5000 imágenes. Finalmente empleamos las fotografías restantes para comprobar la eficacia y funcionamiento del algoritmo. A continuación comprobaremos que el programa es capaz de reconocer imágenes, que cumplan las condiciones que se propusieron a la hora de crear CIFAR-10, pero que no pertenezcan a este conjunto de fotografías.

Para comprobar la eficacia del modelo he tomado una fotografía de un gato y de un automóvil que proporcionaremos al algoritmo con la intención de que este las clasifique correctamente. Para que la red sea capaz de reconocer la estructura de cada una de las imágenes y pueda procesarlas deben tener forma $(32, 32, 3)$. Como el resto de las imágenes que hemos empleado, deberán tener un tamaño de 32×32 píxeles. Además se tratará de fotografías a color, ya que la primera capa de la red neuronal toma como argumentos de entrada este tipo de datos. Por último, es preciso normalizar los datos de forma que el rango de valores que toman los píxeles esté entre cero y uno.

Una vez hemos adaptado los datos al algoritmo, podemos predecir la etiqueta que se corresponde con cada imagen, sin más que hacer uso del comando *model.predict()*, como ya hemos visto en la fase de prueba. Esto genera un vector de diez entradas, cada una de las cuales se corresponde con la probabilidad de que la imagen pertenezca a cada catego-

ría. Tomamos la posición en la que se encuentra la mayor probabilidad y vemos con que etiqueta se corresponde. Como se puede observar en la Figura 4.4a la posición con mayor probabilidad es la cuarta[1] que se corresponde con la etiqueta gato, que coincide con la etiqueta real.



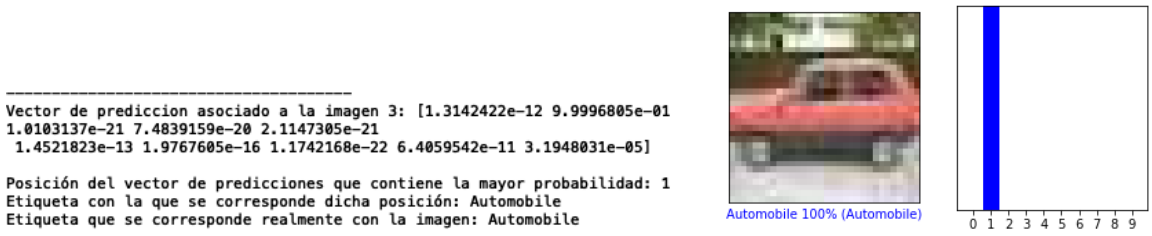
(a) (b)

Figura 4.4: Clasificación de una imagen no perteneciente al conjunto CIFAR-10.

Como habíamos visto en los casos anteriores, en la Figura 4.4b se puede ver, por una parte la fotografía con una indicación de la eficiencia que tiene el algoritmo para detectar la categoría a la que pertenece, en este caso un 99 %. En segundo lugar un gráfico de barras en el cual aparecen: representada en azul la barra correspondiente a la probabilidad de que el algoritmo etiquete correctamente la fotografía, y en gris la probabilidad de que lo haga de forma incorrecta. En este caso con una probabilidad del 1 % clasifica erróneamente la imagen, asignándole la etiqueta perro. Por tanto podemos concluir que el algoritmo creado para la categorización de imágenes es eficiente y funciona correctamente.
De igual modo para la fotografía del automóvil obtenemos los resultados que se pueden ver en la Figura 4.5. En primer lugar, el programa nos devuelve la posición del vector de predicciones con mayor probabilidad, que en este caso es la segunda, que se corresponde con la etiqueta automóvil que coincide con la etiqueta real, como se puede apreciar en la Figura 4.5a. Por otra parte, en la Figura 4.5b podemos ver la imagen del coche con la efectividad que ha tenido el algoritmo a la hora de predecir la etiqueta, en este caso un 100 %, y un gráfico de barras que nos deja ver si existe otra etiqueta que genere confusión a la hora de clasificar la fotografía, en este caso no hay ninguna categoría que algoritmo haga corresponder con la imagen de forma errónea.

Sin embargo, cuando le proporcionamos al algoritmo una imagen que no cumple con los requisitos del criterio que se empleó a la hora de crear el conjunto de datos CIFAR-

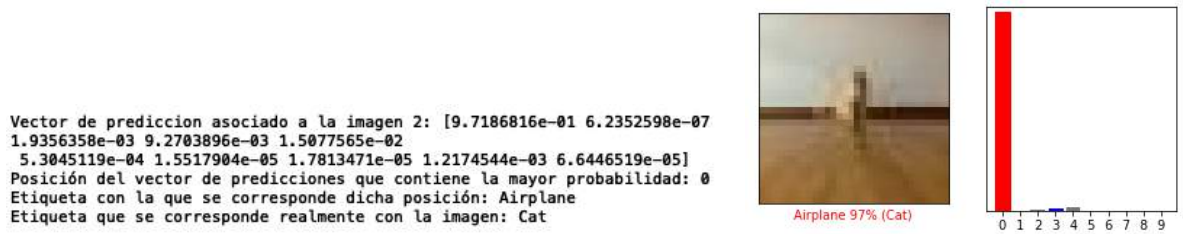
[1] Recordemos que las posiciones del vector van de cero a nueve. Es por ello que en la Figura 4.4a se puede ver como el algoritmo realmente indica la posición 3, que es la cuarta.



(a) (b)

Figura 4.5: Clasificación de una imagen no perteneciente al conjunto CIFAR-10.

10, es probable que no sea capaz de reconocer su contenido ni clasificarla. Para verlo he tomado una fotografía a un gato, esta vez se encuentra más alejado del objetivo, y se la he proporcionado al algoritmo. Teniendo en cuenta que estamos analizando imágenes de tamaño 32×32 píxeles, esto es, nuestro algoritmo no tiene acceso a toda la información que nosotros somos capaces de visualizar con una imagen de alta resolución, sino que analiza una pequeña parte de esa información. Si además el elemento principal de la fotografía se encuentra alejado, la información que el algoritmo percibe, en este caso del gato, es todavía más limitada. Este es el caso de la Figura 4.6, en la cual se puede ver que el número de píxeles que capturan la parte de la imagen en la que se ve al gato es mucho menor que en la Figura 4.4, en la cual el gato se encuentra en primer plano.



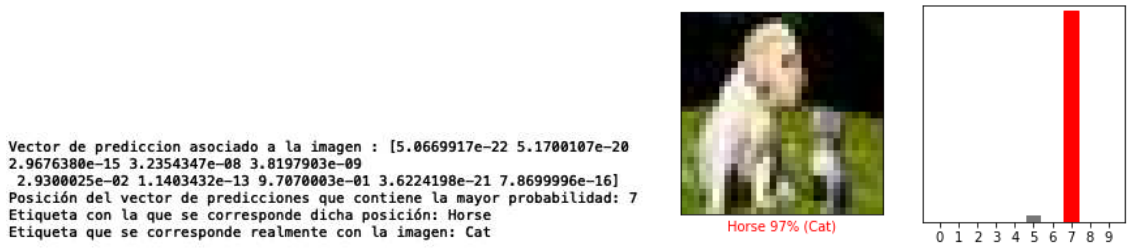
(a) (b)

Figura 4.6: Clasificación de una imagen no perteneciente al conjunto CIFAR-10, donde el elemento principal se encuentra alejado.

En la Figura 4.6a la posición del vector de predicciones con mayor probabilidad es la primera que se corresponde con la etiqueta avión, resulta evidente que este resultado no se corresponde con la etiqueta real. De nuevo, en la Figura 4.6b se puede ver, por una parte la fotografía con una indicación de la eficiencia que tiene el algoritmo para detectar la categoría a la que pertenece, en este caso nos indica que la imagen representa

un avión con probabilidad 97 %. En segundo lugar un gráfico de barras en el cual aparecen: representada en azul la barra correspondiente a que la probabilidad de que el algoritmo etiquete correctamente la fotografía, en este caso hay una probabilidad muy baja de que el algoritmo prediga correctamente la etiqueta gato, en color rojo vemos una barra que acumula mayor probabilidad, esta nos indica que la clasificación es incorrecta, esto es, que la etiqueta predicha no coincide con la real. Finalmente se pueden ver dos barras en color gris, esto es la etiqueta avión no es la única que ha generado confusión a la hora de categorizar la fotografía, existe una probabilidad, más baja, de error en el etiquetado correspondiente a las categorías pájaro y ciervo.

Otro de los requisitos del criterio empleado para crear CIFAR-10 es que la red solo garantiza éxito si en la imagen destaca únicamente un objeto, por tanto, si le paso una fotografía en la que se puede ver un perro y un gato, esto es, la fotografía no aparece únicamente el objeto al que refiere la clase, el algoritmo no la reconoce.



(a) (b)

Figura 4.7: Clasificación de una imagen no perteneciente al conjunto CIFAR-10, en la cual el foco principal de la fotografía no es un único elemento.

Podría darse la posibilidad de que reconociera, por un lado al perro y por otro al gato, es decir, que con probabilidades cercanas al 50 % el algoritmo nos devuelva por un lado la etiqueta gato y por otra la etiqueta perro. No obstante, este resultado no cuadra con el funcionamiento interno del algoritmo, ya que el modelo toma la información que le proporcionamos en forma de píxeles, por tanto en este caso reconocerá el conjunto de píxeles pertenecientes al perro y el conjunto de píxeles que representan al gato como un único conjunto de píxeles representativos de un elemento. En la Figura 4.7b, se puede ver como con una probabilidad del 97 % el algoritmo nos devuelve la etiqueta caballo, y con una probabilidad del 3 % reconoce que en la fotografía hay un perro. Por otro lado en la Figura 4.7a se puede ver el vector de predicciones asociado a la fotografía y que la posición con mayor del vector con mayor probabilidad se corresponde con la etiqueta caballo.

Esto mismo ocurría si no respetásemos cualquiera de los criterios empleados para crear CIFAR-10, como por ejemplo si le pasásemos al algoritmo un dibujo de uno de los elementos.

Capítulo 5

Conclusiones

El programa elaborado en el capítulo anterior es un ejemplo de aprendizaje automático supervisado que emplea la clasificación, uno de los métodos claves para este tipo de aprendizaje. En concreto, a través de la regresión logística hemos sido capaces de resolver un problema de clasificación múltiple, ya que el algoritmo de clasificación de imágenes que hemos implementado distingue entre diez categorías distintas. Hemos obtenido resultados coherentes en el tratamiento de los datos, es decir, hemos visto en la sección donde comprobamos el funcionamiento que, para que el programa nos diera buenos resultados era necesario seguir el criterio empleado a la hora de crear el conjunto de datos CIFAR-10. En este aspecto podríamos decir que cuando construimos un modelo, este se encuentra "atado" en gran medida al conjunto de datos que hemos empleado en las fases de prueba y entrenamiento, sin necesidad de que se de un error por *overfitting*.

Del mismo modo los resultados obtenidos son coherentes a nivel teórico, ya que para elaborar este algoritmo hemos hecho uso de las redes neuronales convolucionales, una clase de redes neuronales artificiales empleadas para procesar datos de tipo cuadrícula, como lo son las fotografías. En la construcción de la red neuronal hemos alternado capas de convolucionales con capas de combinación máxima, finalmente aplicamos capas totalmente conectadas que llevan asocidas una función de activación ReLU y una última capa que tiene asociada una función softmax que genera el vector de probabilidades que nos permite categorizar las fotografías. En el Capítulo 2 veíamos como este tipo estructura era la más ventajosa a la hora de crear una red neuronal, por los resultados que daba y por que el uso de las capas de combinación máxima suponía una reducción del coste computacional.

En el Capítulo 3 estudiamos el método de descenso Adam, descrito por Kingma y Ba en [15], y vimos que en la actualidad es uno de los más empleados debido a que presenta una convergencia más rápida, como veíamos en la Figura 3.1. Empleamos este método para minimizar la función de coste, asegurando de esta manera la coherencia de

los parámetros internos del algoritmo. Finalmente, la librería Keras de Tensorflow nos ha permitido implementar de forma sencilla la construcción de la red neuronal y su posterior entrenamiento y comprobación. Este proceso es una pequeña muestra de las aplicaciones que tiene el machine learning, tanto en la clasificación de imágenes, que se emplea entre otros en el campo de la medicina, como en el procesamiento del habla, clasificación de texto o en robótica a la hora de construir naves espaciales, etc.

Apéndice A

En este apéndice se incluye el código completo para la obtención de los resultados mostrados en el Capítulo 3.

```
#Método de clasificación de imagénes del conjunto de datos CIFAR-10
#emplenado lenguaje automático supervisado y redes neuronales.


-------------------------------------------
#Módulos y librerías necesarios:

import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from skimage.io import imread

-------------------------------------------
Carga de datos y división de estos en los conjuntos de prueba y entrenamiento:

cifar10=tf.keras.datasets.cifar10

(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()



Debido a que se está empleando lenguaje automático supervisado cada una de las
imágenes está etiquetada.
```

```
#A continuación se presenta la lista de etiquetas:

class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
'Dog', 'Frog', 'Horse', 'Ship', 'Truck']


#Cada etiqueta se asocia con un número que va del 0 al 9
#-----------------------------------------
#Breve estudio de los vectores de imágenes y etiquetas correspondientes a los
#datos de entrenamiento y prueba.
print('       ')
print('---------------------------------------')
print('Estructura de los conjuntos de entrenamiento y prueba')
print('El conjunto de entrenamiento es de la forma:',train_images.shape)
print('El conjunto de prueba es de la forma:',test_images.shape)
print('       ')
print('Estructura de los vectores de etiquetas de entrenamiento y prueba')
print('El vector de etiquetas para el entrenamiento es de la forma:',train_labels.shape)
print('El vector de etiquetas para la prueba es de la forma:',test_labels.shape)
print('---------------------------------------')


#Esto es, en el conjunto de entrenamiento hay 50000 imágenes de 32x32,
#mientras que en el conjunto de prueba hay 10000 también de tamaño 32x32.
#También se puede ver como los vectores de etiquetas para el entrenamiento y
#prueba son de la misma longitud que sus respectivos vectores de imágenes.

#-----------------------------------------
#Pre-pocesamiento de los datos.

#Para cada uno de los conjuntos de imágenes, los valores que toman los píxeles
#varía entre 0 y 255 en la escala de color. Por tanto es necesario normalizar
#los datos pasándolos a rango 0-1.
#Los datos se cargan como números enteros por tanto los pasamos a flotante para
#poder realizar las operaciones.
```

```
train_images=train_images.astype('float32')
test_images=test_images.astype('float32')
train_images = train_images / 255.0
test_images = test_images / 255.0

#-----------------------------------------
#Construcción del modelo de redes neuronales


#Descripción de las capas.
#CAPA Flatten :  transforma el formato de las imagenes de un arreglo bi-dimensional
#(de 32 por 32 pixeles) a un arreglo uni dimensional (de 32*32*3 pixeles = 3072 pixeles)

#CAPAS CONV2D: es una capa convolucional, aplica filtros, en caso de la primera 32,
#a los datos de entrada.

#CAPAS MAXPOOL2D: Reducen la dimensión de los datos de entrada.

#CAPAS DENSE: Dense implementa la operación:
#output = activation(dot(input, kernel) + bias) donde la activación es la
#función de activación por elementos pasada como argumento de activación
#(Se aplica en algunos casos la función de activación 'ReLU' y en otros 'softmax'),
# kernel es una matriz de pesos creada por la capa, y bias es un vector de
#sesgos creado por la capa .


#ÚLTIMA CAPA: 10 nodos softmax que devuelve un arreglo de 10 probabilidades que suman 1.
#Cada nodo contiene una calificacion que indica la probabilidad de que la imagen
# evaluada pertenezca a una de las 10 clases.

model = keras.Sequential([
keras.layers.Conv2D(32,(3,3),input_shape=(32,32,3),padding='same'),
keras.layers.MaxPool2D(pool_size=(2,2),strides=(1,1),padding='same'),
keras.layers.Conv2D(64,(3,3),padding='same'),
keras.layers.MaxPool2D(pool_size=(2,2),strides=(1,1)),
keras.layers.Flatten(),
```

```
keras.layers.Dense(256,activation='relu'),
keras.layers.Dense(128,activation='relu'),
keras.layers.Dense(10, activation='softmax')
])


#Utilizamos un optimizador (Adam)


model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])


#------------------------------------------
#Entrenamiento del modelo:


#Usamos el comando model.fit, es llamado asi por que fit (ajusta) el modelo al
# set de datos de entrenamiento:

model.fit(train_images, train_labels, epochs=10)


#------------------------------------------
#Comprobaciones.

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
train_loss, train_acc = model.evaluate(train_images,  train_labels, verbose=2)

print('       ')
print('--------------------------------------')
print('Valores obtenidos para la función de pérdida y la presición en el conjunto de prueba
print('Función de pérdida:', test_loss)
print('Precisión:', test_acc)
```

```
print('       ')
print('Valores obtenidos para la función de pérdida y la presición en el conjunto de entren
print('Función de pérdida:', train_loss)
print('Precisión:', train_acc)




#-----------------------------------------
#Predicciones

predictions = model.predict(test_images)

#Una predicción es un vector de 10 elementos. Donde cada una de las entradas se
#corresponde con la probabilidad de que cada etiqueta esté relacionada con la
#imagen a estudiar.

print('       ')
print('--------------------------------------')
print('Vector de predicciones asociado a la segunda imagen del conjunto de prueba:', predic
print('       ')

#La predicción de la etiqueta para la imagen se corresponderá con aquella
# entrada que presente mayor probabilidad.

print('Posición del vector de predicciones que contiene la mayor probabilidad:', np.argmax(
print('Etiqueta con la que se corresponde dicha posición:', class_names[np.argmax(predictio



#Comprobamos que el modelo asocia la etiqueta correcta, examinando la que
#le corresponde realmente.


print('Etiqueta que se corresponde realmente con la imagen:', class_names[test_labels[1][0]
```

```
#----------------------------------------

#Graficamente también se puede ver la efectividad.
#En primer lugar defino los funciones:
#La primera función toma el vector de predicciones y las etiquetas
#verdaderas y genera un histograma coloreando las barras en las que
#las etiquetas coinciden con las predicciones de azul y en el caso
#contrario de rojo.

def plot_image(i, predictions_array, true_label, img):
predictions_array, true_label, img = predictions_array, true_label[i], img[i]
plt.grid(False)
plt.xticks([])
plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted_label = np.argmax(predictions_array)
if predicted_label == true_label:
color = 'blue'
else:
color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
100*np.max(predictions_array),
class_names[true_label[0]]),
color=color)

def plot_value_array(i, predictions_array, true_label):
predictions_array, true_label = predictions_array, true_label[i]
plt.grid(False)
plt.xticks(range(10))
plt.yticks([])
thisplot = plt.bar(range(10), predictions_array, color="#777777")
plt.ylim([0, 1])
predicted_label = np.argmax(predictions_array)
```

```
thisplot[predicted_label].set_color('red')
thisplot[true_label[0]].set_color('blue')


#Estudiaremos en primer lugar la imagen en posición [1].
#Las etiquetas de prediccion correctas estan en azul y las incorrectas estan en rojo

i = 1
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()


#Vamos a graficar multiples imagenes con sus predicciones.
#Notese que el modelo puede estar equivocado aun cuando tiene mucha confianza.

num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
plt.subplot(num_rows, 2*num_cols, 2*i+1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(num_rows, 2*num_cols, 2*i+2)
plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()


#-----------------------------------------
#RESULTADOS
#Por último usaremos el algoritmo para reconocer  imagenes que no pertenezcan
```

```
#a CIFAR-10.

img_1= imread("imagengato.jpg")
img_2= imread("imagengatolejos.jpg")
img_3= imread("imagencoche.jpg")
img_4= imread("perroygato.jpg")
img_1=img_1.astype('float32')
img_2=img_2.astype('float32')
img_3=img_3.astype('float32')
img_4=img_4.astype('float32')

#Normalizo los datos
img_1 = img_1 / 255.0
img_2 = img_2 / 255.0
img_3 = img_3 / 255.0
img_4 = img_4 / 255.0

#Realizamos la predicción sobre la imagen
for i in range(32):
for j in range(32):
for k in range(3):
test_images[0,i,j,k]=img_1[i,j,k]
test_images[1,i,j,k]=img_2[i,j,k]
test_images[2,i,j,k]=img_3[i,j,k]
test_images[3,i,j,k]=img_4[i,j,k]

test_labels[1]=3
test_labels[2]=1
test_labels[3]=3
ejemplo_pred_1 = model.predict(test_images[0:4])

for i in range(4):
print('       ')
print('---------------------------------------')
print('Vector de prediccion asociado a la imagen :', ejemplo_pred_1[i])
print('Posición del vector de predicciones que contiene la mayor probabilidad:', np.argmax(
```

```
print('Etiqueta con la que se corresponde dicha posición:', class_names[np.argmax(ejemplo_p

#Comprobamos que el modelo asocia la etiqueta correcta, examinando la que
#le corresponde realmente.
print('Etiqueta que se corresponde realmente con la imagen:', class_names[test_labels[i][0]

#Graficamente,
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, ejemplo_pred_1[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, ejemplo_pred_1[i],  test_labels)
plt.show()
```

Bibliografía

[1] A. L. Samuel, Some Studies in Machine Learning Using the Game of Checkers, IBM Journal of Research and Development, 3, 210-229.

[2] F. Rosenblatt, The perceptrón: a probabilistic model for information storage and organization in the brain., Psychological review, 386-408.

[3] M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of Machine Learning, second edition, MIT Press 9780262351362, Adaptive Computation and Machine Learning series, 2018 .

[4] J.J.B. Nuin, E.P. Sanz and E.C. Moreno , Manual Práctico de Inteligencia Artificial En Entornos Sanitarios, Elsevier Health Sciences 9788491138112, 2020.

[5] P. A. Quintana, Métodos numéricos con aplicaciones en Excel, Reverte, 2005.

[6] T. Chen, Introduction to boosted trees, University of Washington Computer Science, 22, 115, 2014.

[7] H. Li, R. Zhao and X. Wang, Highly efficient forward and backward propagation of convolutional neural networks for pixelwise classification, 2014.

[8] R.K. Samala, H.P. Chan, L.M. Hadjiiski, M.A. Helvie, K.H. Cha and C.D. Richter, Multi-task transfer learning deep convolutional neural network: application to computer-aided diagnosis of breast cancer on mammograms, IOP Publishing, 8894, 2017 .

[9] A.F. Agarap, Deep learning using rectified linear units (relu), 2018.

[10] M. Wang, S. Lu, D. Zhu, J. Lin and Z. Wang, A High-Speed and Low-Complexity Architecture for Softmax Function in Deep Learning, 223-226, 2018 .

[11] A. Wiranata, S.A. Wibowo, R. Patmasari, R. Rahmania and R. Mayasari, Investigation of Padding Schemes for Faster R-CNN on Vehicle Detection, 208-212, 2018.

[12] M.A. Islam, M. Kowal, S. Jia, K.G. Derpanis and N.D. Bruce, Position, padding and predictions: A deeper look at position information in cnns, 2021.

[13] J.M. Viaño and M. Burguera, Lecciones de métodos numéricos, Tórculo, 1995.

[14] S.Ruder, An overview of gradient descent optimization algorithms, 2016.

[15] D.P. Kingma and J. Ba, Adam: A method for stochastic optimization, 2014.

[16] S. Bock and J. Goppold and M. Weiß, An improvement of the convergence proof of the ADAM-optimizer, 2018.

[17] A. Krizhevsky, G. Hinton and others, Learning multiple layers of features from tiny images, Citeseer 2009 .

[18] N.Shukla and K. Fricklas, Machine learning with TensorFlow, Manning Greenwich, 2018 .

[19] P. Singh and A. Manure Learn TensorFlow 2.0, Springer.

[20] J.I. Bagnato, Aprende Machine Learning, Leanpub, 2020.

[21] I. Goodfellow, Y. Bengio, A. Courville and Y. Bengio, Deep learning, MIT press Cambridge, 2016 .

[22] Martín Abadi and Ashish Agarwal and Paul Barham and Eugene Brevdo and Zhifeng Chen and Craig Citro and Greg S. Corrado and Andy Davis and Jeffrey Dean and Matthieu Devin and Sanjay Ghemawat and Ian Goodfellow and Andrew Harp and Geoffrey Irving and Michael Isard and Yangqing Jia and Rafal Jozefowicz and Lukasz Kaiser and Manjunath Kudlur and Josh Levenberg and Dandelion Mané and Rajat Monga and Sherry Moore and Derek Murray and Chris Olah and Mike Schuster and Jonathon Shlens and Benoit Steiner and Ilya Sutskever and Kunal Talwar and Paul Tucker and Vincent Vanhoucke and Vijay Vasudevan and Fernanda Viégas and Oriol Vinyals and Pete Warden and Martin Wattenberg and Martin Wicke and Yuan Yu and Xiaoqiang Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.