Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Inteligencia Artificial con TensorFlow para predicción de comportamientos

Autor: Daniel Conde Ortiz

Tutor: Francisco José Fernández Jiménez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Inteligencia Artificial con TensorFlow para predicción de comportamientos

Autor:
Daniel Conde Ortiz

Tutor:
Francisco José Fernández Jiménez
Profesor colaborador

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Grado: Inteligencia Artificial con TensorFlow para predicción de comportamientos

Autor: Daniel Conde Ortiz

Tutor: Francisco José Fernández Jiménez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A Yaiza

Agradecimientos

En primer lugar, quiero agradecer a mis padres todo su apoyo y confianza durante estos cuatro años, sacando lo mejor de mí, acercándome al mundo de las telecomunicaciones desde pequeño y haciendo posible que hoy pueda escribir estas líneas.

A Virginia, por acompañarme y soportarme en este camino, pese a la distancia. También por entenderme y hacerme feliz en los malos momentos y por aconsejarme siempre lo mejor.

A mi tutor, Francisco José, por haberme brindado la oportunidad de trabajar con él y por la orientación y ayuda proporcionada durante la realización de este proyecto.

Por último, quiero agradecer a mi hermana, Yaiza, su coraje y esfuerzo, con los que me ha enseñado que nunca hay que rendirse pase lo que pase y su inolvidable sonrisa, la que me anima cada día a seguir adelante.

Daniel Conde Ortiz

Sevilla, 2018

Resumen

Debido a la inabarcable cantidad de información que existe en la sociedad actual y la búsqueda continua de atraer y retener al cliente de un servicio, urge investigar y desarrollar nuevos métodos de análisis automático y predicción de gustos y comportamientos.

Este documento recoge el análisis y desarrollo de una aplicación web donde obtendremos datos de los visitantes y actuaremos acorde a sus preferencias. Todo ello procesándolo mediante una red neuronal usando TensorFlow, una librería de Python.

En concreto, se desarrollará una página web, con su respectivo front-end y back-end, que simulará un servicio de streaming de música y en la que se intentará recomendar al usuario los grupos que deberían gustarle mediante sus acciones pasadas en nuestra web.

También se analizarán distintas opciones de redes neuronales, así como ventajas y desventajas de las mismas y se intentará profundizar en el funcionamiento de todo lo que tiene que ver con el aprendizaje máquina o *Machine Learning*.

Abstract

Due to the modern world being overcrowded with information and services trying to make customers spend more time on them, it is urgent to research and develop new automatic analysis methods for this information and methods for predicting likings and behaviour of the customers.

In this document, we analyze and develop a web application which will obtain users' data and change the results given their preferences. All of this will be done using a neural network built with TensorFlow, a Python library.

In fact, we will develop a web page, with its own front-end and back-end, which will emulate a music streaming service and we'll try to recommend groups to the user using their previous actions on this web.

We will also analyze different neural networks as well as their advantages and disadvantages, while also trying to understand and explain Machine Learning deeply.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

API	Interfaz pública de una aplicación
UI	Interfaz de usuario
CPU	Unidad de procesamiento central
GPU	Unidad de procesamiento gráfico
TPU	Unidad de procesamiento tensorial
HTTP	Protocolo de transferencia de hipertexto
URL	Localizador de recursos uniforme
GB	Gigabyte, equivalente a 10^9 bytes
CUDA	Arquitectura Unificada de Dispositivos de Cómputo
cuDNN	Red Neuronal Profunda CUDA

1 INTRODUCCIÓN

Al principio se creó el universo. Eso hizo que se enfadara mucha gente, y la mayoría lo consideró un error.

- Douglas Adams -

Nuestra sociedad se mueve en un océano de información de la que sólo vemos una minúscula parte. Cada minuto se generan medio millón de Tweets, 50 mil posts en Instagram, 3.5 millones de búsquedas en Google, se ven 4 millones de videos en YouTube y Amazon gana 250.000$ en ventas, sólo por citar algunos ejemplos [1].

Todo esto sólo en 5 de las webs más populares del mundo, pero se estima que hay cerca de 2 mil millones de webs en total [2]. Aun teniendo en cuenta que el 75% de esas webs están inactivas y que muchas otras no son tan populares como las mencionadas anteriormente, sigue siendo un gran número y la cantidad de información que generan aún mayor.

Actualmente, en cualquier web medianamente moderna, todo lo que hacemos queda registrado. Cada link que visitamos, cada movimiento de ratón, cuánto nos desplazamos y cuánto estamos viendo qué cosas. Todo ello con el objetivo, al menos teóricamente, de ofrecer al usuario la mejor experiencia, de permitirle encontrar antes lo que desea y de que, al final, eso repercuta positivamente para la compañía, tanto social como económicamente.

Es fácil ver que el manejo de toda esta información es simplemente imposible para cualquier humano y los algoritmos tradicionales tienen numerosos problemas o son ineficientes. Para ello ha surgido un nuevo campo en la computación que emula sistemas biológicos. En concreto, se busca la resolución de problemas complejos a través de sistemas de computación basados en el cerebro humano. A estos sistemas se les llama **Redes Neuronales** [3]**.**

1.1 Objetivos

Este proyecto busca realizar una aproximación a una página web en la que se obtengan datos del usuario que la visita, se procesen mediante redes neuronales y se devuelva información al visitante. En concreto, el entorno será una página de streaming de música que tenga en cuenta los grupos que escucha el usuario y vaya recomendando los más adecuados para conseguir que continúe en nuestra web.

Se intentará que la interacción con la web sea lo más rápida e intuitiva posible, así como que las recomendaciones sean lo suficientemente precisas.

1.2 Alcance y límites

No se intenta construir una web de streaming de música funcional en sí, sino una base que luego pueda ser modificada fácilmente y que cuente con todos los componentes necesarios para funcionar. Por ello, nos centraremos en el funcionamiento conjunto de web + red neuronal. Por ejemplo, un buen diseño de UI quedaría fuera del alcance del proyecto.

Por otro lado, como veremos, para entrenar una red neuronal hace falta una gran cantidad de datos. Como no podemos obtener estos datos por nuestros propios medios, nos valdremos de una API externa que nos los proporcionará.

Finalmente, para que una red neuronal funcione lo suficientemente bien hacen falta, o bien, mucho tiempo entrenando, o mucha capacidad computacional durante el entrenamiento. Como estos dos recursos son limitados, intentaremos aprovecharlos lo mejor posible.

1.3 Decisión adoptada

Unos de los puntos principales de este proyecto era usar TensorFlow para hacer la red neuronal (más adelante se explicará en profundidad), así que esa decisión ya estaba tomada.

TensorFlow puede usarse en varios lenguajes (Java, C, Go, etc.), pero el más usado y completo es Python.

Python es un lenguaje de programación de alto nivel, multiparadigma e interpretado en el que los objetivos son la simpleza y la efectividad. La sintaxis elegante y el tipado dinámico hacen que sea un lenguaje ideal para el desarrollo rápido de cualquier aplicación [4]. Si bien es cierto que puede ser menos eficiente que otros lenguajes, esto se soluciona pudiendo usar módulos escritos en otros lenguajes, por ejemplo, C++.

Para hacer más fácil la unión web - red neuronal, se decidió usar Python también para crear un servidor web. En concreto se usará el módulo Bottle, el cual es ligero y simple, justo lo que necesitamos para nuestro proyecto.

La elección de una página de recomendación de grupos se hizo por dos factores:

- **Set de datos**: Encontrar un set de datos lo suficientemente grande y bueno como para que nuestra red sea entrenada correctamente es complicado. Ante la oportunidad de poder usar los datos de otra página de streaming de música para nuestro proyecto, decidimos seguir ese camino ya que sabíamos que los datos serían fiables.
- **Grupos**: Cuanta más amplia sea la gama de productos a recomendar, más difícil será entrenar a la red, lo que requerirá aún más tiempo y potencia. Así, recomendar canciones específicas sería algo demasiado amplio para nuestro proyecto, por lo que decidimos centrarnos en grupos simplemente.

1.4 Alternativas a la solución

1.4.1 Alternativas de redes neuronales

Actualmente podemos encontrar múltiples herramientas de software para crear redes neuronales [5], por ejemplo:

Logo	Características
Figura 1 - Logo Deeplearging4j [6]	**Deeplearning4j** [6]: Escrito para Java, soporta una amplia gama de algoritmos de "Deep Learning" o aprendizaje profundo. Es de código abierto y tiene su propia librería de computación numérica (ND4J). Funciona tanto con CPUs como con GPUs.

 Figura 2 - Logo PyTorch [7]	**PyTorch** [7]: Desarrollada por Facebook, de código abierto y escrita para Python. Su principal característica es que soporta grafos computacionales dinámicos, es decir, para cada iteración del entrenamiento se define un grafo.
 Figura 3 - Logo Theano [8]	**Theano** [8]: Una de las primeras grandes librerías para redes neuronales. Escrito para Python y es de código abierto. Sirvió de base para desarrollar otros softwares para redes neuronales. Es un híbrido que trata de combinar lo mejor de varios paquetes numéricos para Python. No es muy usado actualmente.
 Figura 4 - Logo Matlab [9]	**Matlab** [9]: Matlab no es un software para desarrollar redes neuronales en sí, lo que nos permite hacerlo es una Toolbox específica: *Neural Network Toolbox*. Esto permite aprovechar al máximo las capacidades de procesamiento numérico de Matlab.
 Figura 5 - Logo Caffe 2 [10]	**Caffe 2** [10]: Basado en *Caffe* y con muchas cosas en común (o parecidas) con *PyTorch* (conceptos, equipos de desarrollo, líneas de investigación, etc.). Es un módulo para Python y C++. Está más centrado en ser ligero de peso, modular y en poder usarse en producción que en la investigación.
 Figura 6 - Logo TensorFlow [11]	**TensorFlow** [11]: Desarrollado y usado por Google en todos sus productos, es una librería de código abierto principalmente enfocada a Python. Soporta multitud de tipos de plataformas y entornos.
 Figura 7 - Logo Sciki-learn [12]	**Scikit-learn** [12]: Es una librería de Machine Learning para Python con algoritmos para clasificación, regresión y agrupamiento (clustering). Empezó como un pequeño proyecto en un evento de Google en 2007 y, actualmente, sigue en desarrollo, con nuevas versiones cada año. La diferencia con las demás librerías es que la mayor parte de su código está escrito en Python.

Tabla 1 - Comparación de redes neuronales

1.4.2 Alternativas de servidores

Aparte de Bottle, existen muchos otros servidores web que pueden ser usados en Python, los más conocidos son: Tornado, CherryPy, Django, Gunicorn, Twisted Web, uWSGI, Waitress, … [13] Todos ofrecen más o menos la misma funcionalidad, cada uno especializándose en distintas materias o siendo más o menos complejo.

Por ejemplo, Django nos permite trabajar con grandes equipos al estar estructurado. Además, tiene una gran comunidad detrás que está constantemente actualizándolo y creando nuevos módulos. También soporta conexiones con bases de datos. Por el contrario, todo esto hace que sea más difícil de aprender.

La elección de Bottle se ha hecho en base a dos elementos:

- **Simpleza**: Tanto la instalación como configuración y utilización de Bottle es realmente fácil e intuitiva, perfecto para un proyecto como el nuestro, en el que no estamos muy familiarizados con el entorno.

- **Componentes**: No tenemos necesidad de usar módulos especiales para nuestra web (funcionalidad que otros servidores implementan más en profundidad). Tan sólo necesitaremos responder a peticiones HTTP simples, como se explicará en los siguientes capítulos.

Como podemos ver en [14], Bottle está entre los más rápidos en respuesta en comparación a los demás. En un futuro, se podría usar cualquier otro servidor de los mencionados anteriormente para mejorar tanto la funcionalidad de la web como su velocidad.

1.5 Conocimientos previos

Este proyecto se comenzó sin conocimientos previos sobre TensorFlow o cualquier otra librería para redes neuronales, y sólo con nociones muy básicas sobre el funcionamiento de las mismas. Para una introducción más en profundidad a este mundo se recomienda lo siguiente:

- En la referencia [15] [16] se puede encontrar una serie de videos en los que se explica detalladamente, pero desde cero y para cualquier público, qué es una red neuronal, cómo funciona, las funciones internas que tienen, etc.
- En el documento [3] se realiza un análisis bastante profundo sobre los distintos tipos de redes neuronales y las diferencias entre ellas.
- En la referencia [17] nos encontramos con los tutoriales oficiales de TensorFlow. Comprenden una amplia variedad de redes neuronales y aplicaciones, como clasificación de texto e imágenes, reconocimiento de audio, representación de datos y más.
- Páginas webs como la de la referencia [18] resultan útiles a la par que curiosas para aprender ya que nos permiten experimentar en directo con distintos tipos de redes, sus parámetros y nos muestran su proceso de aprendizaje.
- Por último, podemos encontrarnos libros, tanto físicos como electrónicos, que profundizan aún más en Machine Learning y Tensorflow, como los de las referencias: [19] [20] [21].

2 REDES NEURONALES

By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.

- Eliezer Yudkowsky -

Una red neuronal se puede definir como un conjunto de algoritmos que intentan simular (o se basan en) un cerebro humano [22]. Estos algoritmos son específicamente diseñados para reconocer patrones. Estos patrones son vectores numéricos, en los que cualquier otro tipo de información debe ser traducida (imágenes, sonidos, palabras, etc.).

La característica más llamativa de las redes neuronales es que "aprenden" a realizar tareas sin necesidad de estar específicamente programadas para ellas. Esto se hace mediante entrenamiento continuo de las mismas, indicándoles qué está bien y qué mal. Así, dependiendo del método de "aprendizaje" que se use, tendremos distintos tipos de redes, según veremos más adelante.

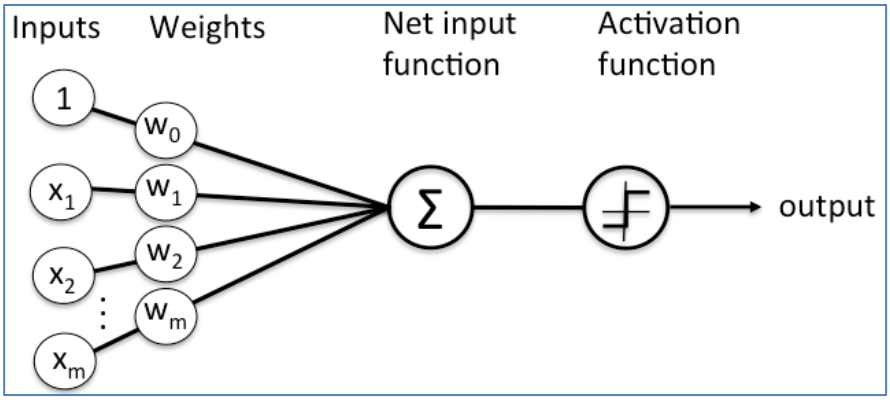

Figura 8 - Neurona [14]

Una red neuronal básica está formada por múltiples capas que a su vez tienen nodos conectados entre las distintas capas; estos nodos son los conocidos como *neuronas*. Cada nodo recibirá información (amplificada y sumada según distintos pesos), la procesará y la transmitirá a los siguientes nodos [23] [22]. Esta transmisión se hará si la información saliente supera cierta función de activación. Todo esto está ilustrado en la Figura 8.

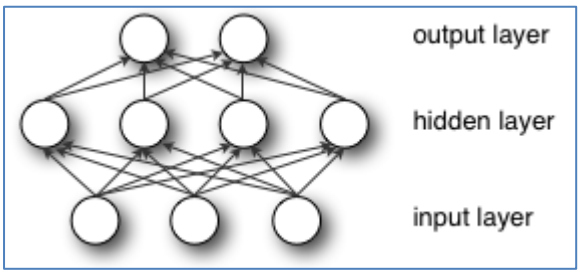

Figura 9 - Capas de una red neuronal [14]

En una red tendremos una capa de entrada, una o varias capas intermedias (capas *ocultas*) y una capa de salida. Dependiendo de cómo se relacionen estas capas y sus neuronas, existen distintos tipos de redes neuronales.

El método de entrenamiento es similar para casi todas las redes neuronales: a la red se le dan unos datos de entrada y esta produce unos datos de salida, se le dice cuál sería la salida correcta según esos datos y la red se encarga de modificar y optimizar los pesos y parámetros de sus funciones internas para conseguir el resultado deseado.

La similitud entre la salida deseada y la obtenida se mide mediante una variable, normalmente llamada "Pérdida", que por sí misma no significa nada, pero podremos usarla para comparar con otras ejecuciones de la red para comprobar si es más o menos acertada.

La red intenta minimizar (o maximizar, según se le indique) el valor de esta variable lo máximo posible, pero aquí podemos encontrarnos el problema del mínimo local. Esto es, un punto en el que todos los puntos cercanos a él tienen un valor mayor, pero existe un mínimo global con aún menor valor. Esto llevaría a la red a creer que está en el punto óptimo, cuando no es así. Afortunadamente, existen métodos para detectar estas situaciones y solventarlas.

Este entrenamiento mediante "fuerza bruta" no está carente de problemas. Por ejemplo, tendremos poco o ningún control (según el tipo de red) sobre los parámetros de la red neuronal. Esto ha llevado a problemas como el del algoritmo de recomendación de YouTube, creado mediante redes neuronales [24], y el cual ha tenido numerosos problemas de recomendaciones inadecuadas a menores, pese a los intentos de los ingenieros de controlarlo.

También nos encontraremos detractores de las redes neuronales ya que consideran que, en conjunto, se comportan como muchas sentencias *if* anidadas, lo cual tiene sentido, aunque también es cierto que, para conseguir el mismo resultado, haría falta mucho más esfuerzo para programar este funcionamiento mediante dichas sentencias del que hace falta para programar la red.

2.1 Aplicaciones de las redes neuronales

Actualmente, las redes neuronales se están abriendo paso en numerosos campos de todas las disciplinas, ya sea como sistemas de recomendación, de clasificación, etc.

Cualquier elemento mínimamente relacionado con inteligencia artificial, muy probablemente implementará una red neuronal, por ejemplo:

- Conducción autónoma en coches [25].
- Robots humanoides que aprenden a caminar [26].
- Bots que aprenden en días lo que a humanos les lleva años, por ejemplo, a competir a alto nivel en videojuegos [27] [28].
- Predicciones (tiempo, transporte, bolsa, …) [29].
- Motores de búsqueda [30].
- Y, como hemos mencionado anteriormente, recomendación y tratamiento de todo otro tipo de información.

También tienen su espacio en las telecomunicaciones. Por ejemplo, pueden servir como ecualizadores, para diseño, control y monitorización de redes y enrutamiento, control de fallos, predicción de tráfico, uso fraudulento de servicios, filtros de SPAM, etc. [31] [32] [33]

En cuanto al futuro de las redes neuronales, es un poco incierto. Hay quien argumenta que no tienen una base teórica sólida, por lo que no se puede construir mucho a partir de ellas, que no hay seguridad de que una red vaya a funcionar bien dados unos parámetros y se preguntan si merece la pena gastar tanto esfuerzo y energía en entrenar redes cuando no sabemos bien qué va a pasar.

Se espera que, con el tiempo, los físicos e ingenieros construyan una buena base sobre la que asentar las redes neuronales; pero aún queda bastante para eso.

2.2 Tipos de redes neuronales

El campo de las redes neuronales e inteligencia artificial actualmente se encuentra en ebullición. Constantemente surgen nuevas líneas de investigación, nuevas formas de conectar las partes de las redes entre sí, nuevas formas de aplicar redes a problemas, etc. Por ello, existen multitud de tipos de redes neuronales, algunos de los cuales se muestran a continuación:

- **Redes prealimentadas:** Se caracteriza por ser el tipo más simple ya que las conexiones entre las neuronas no forman un ciclo. La información sólo se mueve en una dirección.
- **Redes recurrentes:** Son un tipo de red neuronal caracterizadas por tener memoria y sentido temporal. La información se mueve en ambas direcciones de la red. Esto es muy útil en el reconocimiento de secuencias e imágenes.
- **Redes de base radial:** Son redes que calculan la salida de la función en base a la distancia a un punto denominado centro. Se usan para reconocimiento del habla, diagnósticos médicos, series temporales, etc.
- **Redes modulares:** Son redes compuestas de otras subredes que realizan tareas más pequeñas. Por ejemplo, servirían para emular un cerebro en el que cada uno de los sentidos es un módulo.
- **Redes con feedback regulatorio:** Estas redes no usan el feedback para cambiar los parámetros de sus funciones, sino para encontrar la activación óptima de los nodos.
- **Redes físicas:** Son redes en los que se usan componentes eléctricos que son físicos, es decir, que son objetos reales y no sólo software, para emular las neuronas y las sinapsis entre estas.
- **Otros tipos:** Redes dinámicas, redes centradas en la memoria, redes entrenadas instantáneamente [34]**.**

Para nuestro programa usaremos un tipo de red neuronal prealimentada (Feedforward Neural Network): el perceptrón multicapa. El perceptrón multicapa es una red que tiene capacidad para resolver problemas que no son linealmente separables, como asociación de patrones, compresión de datos, etc. En el capítulo 4 se explicará este tipo de red con mayor profundidad.

3 TECNOLOGÍA USADA Y ENTORNO

Hasta la fecha, no se ha diseñado un ordenador que sea consciente de lo que está haciendo; pero, la mayor parte del tiempo, nosotros tampoco lo somos.

- Marvin Minsky -

En este capítulo explicaremos cómo funciona todo el software usado para nuestro proyecto. Como mencionamos en la introducción, nuestro proyecto se basa en TensorFlow para la red neuronal y Bottle para hacer de servidor, ambos usados mediante el lenguaje de programación Python.

3.1 Python

Python es un lenguaje de programación que fue creado por Guido van Rossum en 1991 [35]. Se caracteriza por ser de alto nivel e interpretado, haciendo especial énfasis en la legibilidad del código. Usa tipado dinámico y es un lenguaje multiparadigma, lo que significa que soporta programación orientada a objetos, estructurada, funcional, etc.

Está pensado para ser ampliable, es decir, se compone de un núcleo al que se le pueden ir añadiendo módulos y librerías según se vayan necesitando. También desecha la optimización prematura a favor de la legibilidad ya que, si algo necesita ser optimizado, podrá ser escrito en otro lenguaje más rápido computacionalmente y luego ser importado como módulo.

Una característica llamativa de Python es que no usa llaves como la mayoría de los lenguajes; se vale de la propia indentación para definir su estructura.

Python es ampliamente usado en todas las industrias, por ejemplo: Google, Reddit, en la NASA y en la CIA, Mozilla Firefox, Instagram, Disney, y juegos como Los Sims 4. Por supuesto también para entrenamiento de redes neuronales, como es nuestro caso y cálculos numéricos mediante las librerías *NumPy* y *SciPy* [36].

Como consecuencia, Python se ha convertido en el segundo lenguaje más usado en GitHub [37] y siempre está en el Top 5 en cualquier otro ranking de lenguajes de programación [38].

3.2 TensorFlow

TensorFlow es una librería desarrollada por Google para construir y entrenar redes neuronales. Influenciada por Theano, es la predecesora de DistBelief (otra librería creada por Google) y es usada tanto para la investigación como para la producción de los propios productos de Google [39].

Su arquitectura flexible hace que pueda ser ejecutada en multitud de plataformas (GPUs, CPUs o TPUs) y en ordenadores, clústeres de servidores e incluso dispositivos móviles. Nosotros entrenaremos y usaremos la red en nuestro equipo de trabajo: un ordenador portátil. También nos aprovecharemos de que TensorFlow puede

usar tarjetas gráficas de Nvidia para acelerar el proceso usando la propia del portátil, así aprovecharemos su potencia al máximo.

Como se ha mencionado anteriormente, usaremos TensorFlow mediante Python, pero esto no significa que esté programado en ese lenguaje, lo cual resultaría extremadamente ineficiente. En cambio, esta librería está escrita en una combinación de C++ y CUDA (lenguaje de Nvidia para GPU's). TensorFlow usa Eigen (una librería numérica de alto rendimiento para C++ y CUDA) y cuDNN (una librería de Nvidia para redes neuronales profundas) [40].

El uso de Python nos ahorra problemas como el tipo de las variables que obtenemos como resultado de las ejecuciones, la simpleza en la llamada a las mismas, el manejo de vectores o matrices, etc. TensorFlow también provee interfaces en otros lenguajes como Java, Go, C++, …

TensorFlow usa grafos computacionales estáticos, lo que quiere decir que definimos el modelo de nuestra red neuronal antes de empezar a ejecutarla o entrenarla. Un grafo (Graph) es un conjunto de operaciones que representará a nuestro modelo. Todas las demás interacciones con nuestro modelo, como ejecutar la red, las haremos a través de una sesión (Session).

Los tensores (elementos que dan nombre a esta librería) son objetos geométricos que describen relaciones entre vectores geométricos, escalares y otros tensores. Es decir, son los objetos que maneja la red para producir valores. Así, cada nodo del grafo tomará como entrada 0, 1 o más tensores y producirá un tensor como salida, que será pasado a otros nodos. Por ejemplo, podrían ser entendidos como matrices multidimensionales, aunque son algo más complicados que eso.

Para introducir valores en la red usaremos variables, las cuales serán fijas siempre, o parámetros de sustitución (placeholders), los cuales podremos ir variando su valor cada vez que usemos la red. TensorFlow nos permite utilizar también librerías numéricas avanzadas para nuestras representaciones de valores, como son NumPy y Pandas.

Como vemos, TensorFlow tiene todo lo necesario para crear una red neuronal de calidad y fácilmente, pero también es cierto que tiene detractores. Se dice que la influencia de Theano y el uso de grafos estáticos hacen que sea difícil de depurar o usar en general [41]. En cambio, librerías como PyTorch con su uso de grafos dinámicos y su rapidez en cuanto a prototipado hacen que sea más fácil construir cualquier red sin mucha complicación y proporciona herramientas muy atractivas para los desarrolladores e investigadores.

Aun así, actualmente TensorFlow es una de las librerías para redes neuronales más usadas en el mundo. Por ejemplo, es usada en compañías como AMD, Nvidia, Intel, Airbnb, Twitter, Qualcomm, Ebay, Xiaomi, Coca-Cola, Airbus y, por supuesto, Google [11]. Además, como podemos ver en [37], TensorFlow es el proyecto más bifurcado (*forked*) de GitHub, lo que nos muestra aún más su fama e importancia.

La instalación de TensorFlow para nuestro proyecto se explica en el **Anexo A**.

3.3 Bottle

Bottle es un módulo de Python que nos permite crear un servidor web. Se caracteriza por su simpleza y rapidez y no tiene ninguna dependencia aparte de la librería estándar de Python. Nos servirá tanto para servir archivos (como podrían ser los archivos de CSS y JavaScript) como para procesar las peticiones y devolver una respuesta acorde usando la red [42].

Este módulo es tan fácil de usar como se explica a continuación, ayudado de la Figura 10:

- Se define el método (#1) que va a responder a una petición, normalmente devolviendo una cadena de texto y pudiendo tomar como entrada algún parámetro.
- Se define la ruta a través de la que se va a acceder a ese método (#2).
- Se lanza el servidor en la dirección y puerto que

```
from bottle import route, run

@route('/hola') #2
def index(): #1
    return "¡Hola!"

run(host='localhost', port=8080) #3
```

Figura 10 - Ejemplo de servidor

queramos (#3).

- Y ya estaría todo listo para atender peticiones.

Antes de devolver la respuesta podremos procesar la petición como queramos. Para ello, Bottle nos permite obtener parámetros de esta tales como la URL, las cookies o el tipo de petición (POST, GET, PUT, etc.). También podremos modelar la respuesta, por ejemplo, en el caso de lo que devolvamos sea una imagen, podremos cambiar el 'Content-Type' de la respuesta para que el cliente pueda procesarla bien.

3.4 Página web

Para las páginas web que se van a mostrar al usuario se usarán los tres principales lenguajes en el desarrollo de las mismas:

- HTML, para la base y estructura de la web.
- CSS, para dotar de estilo a la página.
- JavaScript, para todas las peticiones que se quiera hacer al servidor, así como cambiar elementos dinámicamente y modificar las cookies.

Las cookies nos servirán para guardar datos del usuario, como su nombre o los grupos que haya visitado. Después, con cada petición se enviarán estas cookies para que puedan ser procesadas en el servidor mediante la red neuronal y obtener los grupos recomendados.

Para cada grupo generaremos una página web nueva que será servida al usuario con los datos procesados.

3.5 Arquitectura del proyecto

Nuestro proyecto estará estructurado de la forma en la que se puede ver en la Figura 11.

Primero, haremos una recolección de datos para después entrenar la red. Una vez entrenada, podrá ser usada por los usuarios en nuestro servidor web. A la vez, se irá entrenando la red con los datos que nos proporcionen los usuarios para optimizarla.

Todos estos pasos se describirán en profundidad en los siguientes capítulos.

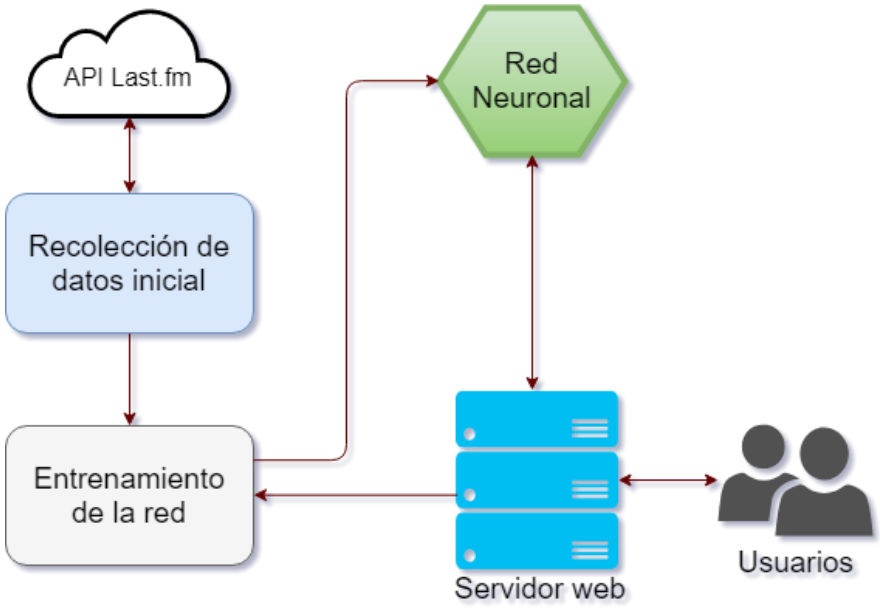

Figura 11 - Arquitectura del proyecto

4 ENTRENAMIENTO DE LA RED NEURONAL

The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.

- Edsger W. Dijkstra -

Para obtener y poder utilizar una red neuronal que funcione bien y sea óptima en sus recomendaciones, necesitaremos entrenarla. Necesitaremos varias cosas: primero, un buen set de datos lo bastante grande y variado como para poder contemplar la mayoría de las opciones y aun así que no resulten muy dispersas; segundo, bastante potencia computacional para poder entrenar a la red en el menor tiempo posible. A falta de potencia, se incrementará el tiempo si fuera necesario.

4.1 Tipo de red neuronal a usar

Como se dijo en el punto 2.1, usaremos un tipo de red neuronal prealimentada: el perceptrón multicapa [43], el cual es una generalización del perceptor simple que, como se explicó anteriormente (punto 2.2), es un tipo de red neuronal prealimentada. La estructura de este tipo de red es la general: una capa de entrada, capas intermedias y una capa de salida, conectadas entre ellas de principio a fin y sin ciclos.

Se considera que es un aproximador universal, es decir, que puede aproximar cualquier función continua en un espacio R^n usando al menos una capa oculta de neuronas. Esto nos permite construir un modelo matemático útil para aproximar o interpolar relaciones no lineales entre elementos de entrada y salida. Esto hace que sea una de las arquitecturas más usadas en la resolución de problemas en la actualidad. Aun así, presenta una serie de limitaciones como su largo proceso de aprendizaje cuando hay muchas variables o la dificultad de poder analizar la red correctamente debido a componentes no lineales y a la alta conectividad.

Un análisis más en profundidad del perceptrón multicapa, incluidas ecuaciones, pueden encontrarse en el Capítulo 3 del Volumen 1 de la referencia [43].

En cuanto a sistemas de recomendación en general, podemos encontrarnos con dos tipos:

- **Basados en contenido**: Para las recomendaciones se tienen en cuenta los elementos en sí y sus características. Por ejemplo, si sabemos que un usuario está interesado en grupos musicales ingleses, le recomendaremos grupos que cumplan eso.
- **Filtrado colaborativo**: Para las recomendaciones no hace falta conocer nada de los elementos ni del usuario, simplemente se recomendarán opciones basándose en acciones pasadas del propio usuario o de otros. Por ejemplo, si dos usuarios escuchan conjuntos de canciones similares, significa que el gusto de esos usuarios es similar; en cambio, si dos canciones son escuchadas por el mismo grupo de

usuarios, significa que esas dos canciones serán parecidas.

Este tipo de filtrado tiene una ventaja: los modelos pueden ser usados para recomendar todo tipo de elementos (libros, películas, canciones, etc.). Pero también tiene una gran desventaja: como basa sus recomendaciones en datos de uso, los ítems nuevos salen perjudicados [44].

Aunque el repositorio del que tomamos los datos (explicado en el punto 4.2) nos ofrece cierta capacidad de saber las características de los elementos, no lo usaremos ya que queremos mantener el conjunto de datos lo más simple posible. Así, el tipo de sistema elegido para nuestro proyecto será el de filtrado colaborativo.

Hay dos categorías de filtrado colaborativo:

- **Basado en usuarios**: Se mide la similitud entre el usuario actual y el resto de los usuarios.
- **Basado en ítems**: Se mide la similitud entre el elemento con el que el usuario está interactuando y el resto de los elementos, por ejemplo, dotándoles de una puntuación (de nuevo, en base a datos obtenidos en pasadas interacciones de otros usuarios).

  La idea general es que usuarios similares comparten intereses similares [45].

En nuestro caso, usaremos el sistema basado en usuarios. Por supuesto, todos estos tipos de sistemas pueden combinarse para formar un motor de recomendación más preciso.

Para nuestra red usaremos un modelo llamado codificador-decodificador [46]. Este modelo lo que hace es transformar la entrada de nuestra red en un set de datos en el que solo quede la información más importante (codificador) y transformarlo de vuelta extrapolando esa información (decodificador). Es un modelo que empezó a usarse en traducción de textos y está dando muy buenos resultados en multitud de campos. Un ejemplo sería: se tiene una imagen a mucha resolución, se baja esa resolución hasta que solo vemos la silueta de los elementos más importantes de la imagen (codificación) y se decide qué elementos son mediante su silueta (decodificación). Esto serviría para detectar el mismo elemento en una imagen, aunque esté desplazado en el espacio.

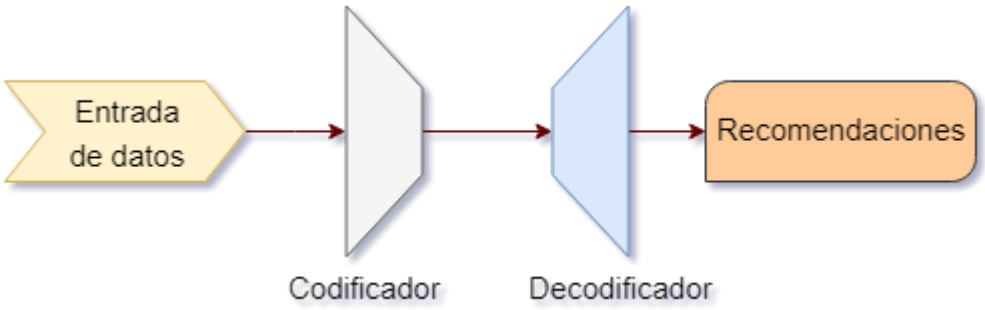

Figura 12 - Estructura de la red

Para minimizar nuestra variable de pérdida, usaremos un optimizador con el método de propagación de la media de los cuadrados (Root Mean Square Propagation o RMSProp) [47]. Este método es una variante del método del descenso estocástico del gradiente (Stochastic Gradient Descent o SGD), el cual es una aproximación estocástica de la optimización del gradiente estocástico [48].

En este método, la proporción (ratio) de aprendizaje es adaptado para cada parámetro y además es capaz de trabajar tanto con el set completo de datos como con partes del mismo, al contrario que otras variaciones similares.

4.2 Repositorios

Como se ha dicho anteriormente, necesitaremos muchos datos para poder entrenar a la red. Si este proyecto se realizara en una empresa que ya tiene experiencia de cara al público, podríamos usar los datos recolectados para el entrenamiento. Como no lo somos, necesitaremos valernos de una base de datos externa.

En internet hay muchas colecciones para entrenamiento de redes neuronales que pueden conseguirse gratuitamente, por ejemplo, a través de las páginas Kaggle [49] y UCI Machine Learning Repository [50]. Sin embargo, estos repositorios se centran más en colecciones de imágenes y texto para fines de investigación.

Si queremos tener datos de interacciones de usuarios con elementos de una web podemos obtener sets de compras en Amazon, por ejemplo. Sin embargo, estos sets se centran más en la puntuación que se le da a cada elemento que en la interacción entre elementos en sí.

Por ello, se decidió usar la API de Last.fm. Last.fm es una red social, además de radio vía Internet, en la que un usuario puede construir su perfil (ya sea añadiendo grupos que escucha en otras páginas mediante un plugin o escuchando el servicio de radio de esta web) e interaccionar con los demás usuarios [51].

Last.fm cuenta con un algoritmo propio de recomendación usado para sugerir grupos para escuchar a los usuarios de su web, concepto muy similar a nuestro proyecto. Además, también ofrece una API abierta al público a través de la cual podemos obtener multitud de datos, como información sobre usuarios, sobre grupos, sobre canciones, etc. Usaremos esta API para obtener los datos para el entrenamiento de nuestra red neuronal.

Para usar esta API tendremos que registrarnos como desarrolladores e indicar para qué la vamos a usar. Este registro puede hacerse en la página de la referencia [52].

Después, nos darán una clave que tendremos que enviar con cada petición. La clave que nos proporcionarán será una cadena de 32 caracteres con números y letras. En el futuro, será sustituida por **[clave]** para hacer más legible el código.

Una petición estándar tendría la siguiente forma:

```
http://ws.audioscrobbler.com/2.0/?method=[conjunto].[método]
&[parámetro_del_método]=[valor]&api_key=[clave]&format=json
```

Por ejemplo, para obtener información sobre el grupo Queen, la petición sería:

```
http://ws.audioscrobbler.com/2.0/?method=artist.getinfo
&artist=Queen&api_key=[clave]&format=json
```

La API podrá devolvernos la información en 2 formatos: XML y JSON. Hemos elegido el formato JSON por ser más intuitivo y legible, además de ser fácil de analizar (*parse*) en Python.

Haremos peticiones para saber la información sobre usuarios y grupos. Sobre cada grupo, cabe la posibilidad de que la API nos envíe información como los géneros con los que está relacionado (mediante etiquetas que han puesto los usuarios) o sobre grupos similares. Lo primero lo desestimaremos como hemos explicado en el punto 4.1 y lo segundo tampoco lo usaremos ya que lo que queremos es crear nuestro propio motor de recomendación desde cero.

4.3 Recolección de datos

Una vez entendido cómo funciona la API, pasaremos a la recolección de datos. Uno de los grandes problemas que se nos presenta es que esta API sólo nos permite hacer una petición cada segundo de media. Es decir, podríamos hacer 10 peticiones en un segundo y en los siguientes 9 segundos no hacer peticiones y no habría problema.

Según la documentación, no está claro dónde está el límite, sólo especifican una petición al segundo de media. Por ello, no es recomendable sobrepasarse demasiado o acabarían bloqueando nuestra clave y no podríamos realizar más peticiones. Esto supone un gran problema ya que nos ralentiza la recogida de datos.

Como datos se usarán los 10 grupos más escuchados de cada usuario. Para ello, primero necesitamos un conjunto de usuarios de los cuales obtener esa información. Lo que haremos es empezar con un usuario que sabemos que existe, el usuario '1' y obtener sus amigos, que serán otros usuarios. De esos nuevos usuarios, de cada uno, obtendremos sus amigos y así iremos repitiendo el proceso.

Todo esto está hecho en el archivo *amigos.py*, que irá haciendo las peticiones (1 por segundo), comprobando que la respuesta es correcta y escribiendo en un archivo de texto todos los usuarios. Si un usuario tiene muchos amigos, los resultados vendrán paginados, por lo que también tendremos que tener en cuenta este factor.

Para no obtener usuarios duplicados tenemos varios métodos:

- Además de escribir todos los usuarios en un fichero, también los guardamos en un vector cada vez

que arrancamos el programa, así, cuando vayamos a escribir un usuario nuevo, comprobamos si ya lo tenemos en el vector.

- Aparte, guardaremos en un fichero de error los usuarios de los que ya hemos hecho petición de amigos, para no volver a hacerla si arrancamos el programa de nuevo.
- Para hacer un filtrado final, hemos creado el programa *duplicado.py* para quitar usuarios duplicados que se hayan podido escapar pese a las acciones anteriores.

Hemos decidido parar cuando teníamos 300.000 usuarios guardados ya que consideramos que es una muestra suficiente para nuestro proyecto, pero podría continuarse este bucle indefinidamente.

Después, de cada uno de estos usuarios, obtendremos sus 10 grupos más escuchados usando el programa *grupos.py*. Este programa hará lo siguiente:

- Por cada usuario, hará una petición a la API de sus 10 grupos más escuchados, aunque puede darse el caso de que recibamos menos de 10 grupos.
- Por cada grupo recibido, guardaremos el nombre en un fichero de texto si no lo hemos hecho antes, y crearemos una cadena con los 10 para escribirla en otro fichero al lado del nombre del usuario en el estilo:

  -Usuario: **[usuario]*[grupo1],[grupo2]**, … ,**[grupo10]**

  Se introducen caracteres especiales como * para luego facilitarnos la lectura del archivo. Este estilo no se ha elegido por ninguna razón en concreto; podría haberse usado otro en su lugar.
- Guardaremos en un archivo de error aquellos usuarios de los cuales haya dado error su petición o haya surgido una excepción al tratar de analizar la respuesta. Por ejemplo, estas excepciones eran muy comunes en grupos con nombres con caracteres chinos.
- Aplicaremos métodos de filtrado para no realizar peticiones duplicadas igual que se hizo con el programa anterior.
- Filtraremos aún más exhaustivamente con el programa *filtrarGrupos.py* para eliminar grupos de la lista que contengan cadenas como "www." o ".com", ya que Last.fm puede enviarnos como grupos algunos nombres de páginas webs.

Al final, nos quedaremos con una lista de 93579 grupos diferentes.

4.4 Procesamiento de los datos

Ya tenemos todos los datos necesarios para el entrenamiento de nuestra red, ahora necesitaremos convertirlos en algo que la red pueda procesar. Para ello usaremos el programa *procesar.py*. Este programa leerá tanto el archivo de grupos como el que contiene las relaciones usuario-grupos e irá escribiendo el resultado en otro fichero.

Para entrenar la red, no necesitamos el nombre de usuario de la relación. Además, cada grupo será convertido a un número según el índice en el que se encuentre en el vector de grupos obtenido tras leer el archivo de grupos. El formato usado para escribir esta vez es el siguiente, siendo NG el número de cada grupo:

-[NG1],[NG2], … , **[NG10]***

De nuevo, se han elegido caracteres especiales para hacer más fácil la lectura en futuros programas. Además, sólo escribiremos aquellas relaciones de las que hayamos podido convertir a números los 10 grupos. Por ejemplo, si una relación tiene menos de 10 grupos o no hemos podido encontrar en la lista de grupos uno de la relación, esa relación no será escrita.

```
-Radiohead
-The Beatles
-Arctic Monkeys
-Lady Gaga
-Lana Del Rey
-Pink Floyd
-Coldplay
-Rihanna
-Muse
-Metallica
```

Figura 13 - Los 10 grupos más escuchados

Así, al final obtenemos un fichero con 276083 relaciones únicas, lo cual consideramos que es un número aceptable para entrenar nuestra red.

También, crearemos otro fichero mediante el programa *ordenaGrupos.py* en el que

escribiremos los grupos según el número de veces que aparezcan en las relaciones, lo que significa que son los más escuchados. Estos datos los usaremos luego en la web para crear un "Top". Por curiosidad, los 10 grupos más escuchados se muestran en la Figura 13.

4.5 Construcción de la red

Es el momento de entrenar a nuestra red. Para nuestro proyecto, nos hemos basado en la red neuronal creada en la referencia [53] y la hemos modificado para nuestra forma de leer datos y el tipo de resultado que queremos.

Actualmente como datos tenemos una matriz de dimensiones 276083 x 93579 o, para entenderlo mejor, una matriz en la que las filas son los usuarios y las columnas son los grupos y cuyos valores es la preferencia de cada usuario por cada grupo. Este tipo de representación de datos se llama factorización matricial.

La red tomará como entrada una matriz de M x 93579 y devolverá como salida una matriz con las mismas dimensiones. Los valores de las matrices estarán normalizados, es decir, variarán de 0 a 1. Para la matriz de entrada, para los 10 grupos de cada usuario se insertará un valor distinto de 0 y para los demás grupos, 0. El hecho de saber qué valor insertar es un poco subjetivo y viene dado por prueba y error. Nosotros hemos elegido los siguientes valores:

- Para el primer grupo de cada relación, se insertará un 1.
- Para los siguientes grupos se irá decrementando el valor en 0.06 hasta llegar al valor que tendrá el último grupo, que será 0.46.

Cuanto más cercano de 1 sea el valor, querrá decir que el usuario tiene más preferencia por ese grupo, lo cual tiene sentido ya que los hemos recibido así ordenados de la API. Que los 10 valores sean distintos de 0 significa que están relacionados entre sí.

En cuanto a la salida de la red, nos devolverá una matriz con las mismas filas, pero con distintos valores en las columnas. Estos valores también variarán de 0 a 1 y cuanto más cercanos sean a 1, significará que son más recomendables para ese usuario.

Si tenemos en cuenta que los elementos de las matrices son de tipo *float* y que ese tipo en Python ocupa 64 bits, la matriz de entrada completa ocuparía 19 GB, algo que nuestro equipo no puede soportar. Para solucionar este problema, observamos que la mayoría de los elementos de la matriz son 0, lo que significa que es un a matriz dispersa. Usaremos la librería *SciPy* de Python que nos permite crear matrices dispersas, en concreto, el método *lil_matrix* (dentro de *scipy.sparse*) y con ello conseguiremos un importante ahorro de memoria. Cuando queramos obtener una matriz completa (incluyendo ceros), tan solo seleccionaremos las filas que queramos de la matriz dispersa y llamaremos al método *todense*.

Tras leer el fichero con las relaciones usuario – grupos convertidos a números y guardados en la matriz dispersa, es el momento de construir la red. Para construir la red tendremos que elegir ciertos parámetros como el número de capas intermedias, el tamaño de esas capas, el peso de las variables en las operaciones, la ratio de aprendizaje, etc. No hay un método exacto para elegir estos parámetros, sino que su elección se basa en prueba y error hasta ir ajustándolos a los óptimos.

Decidimos que dos capas para el codificador y, por lo tanto, otras dos para el decodificador funcionarían bien para nuestra red, de nuevo, sin ningún método en concreto; y, tras múltiples pruebas, comprobamos que cuanto más grandes fueran estas capas, mejor funcionaría la red. El tamaño más grande que les podemos dar sin que nos de error de memoria es de: 235 nodos para la capa 1 y 100 para la capa 2.

```
num_input = numGrupos
num_hidden_1 = 235
num_hidden_2 = 100
```

Figura 14 - Tamaño de las capas de la red

Tras esto, debemos elegir el valor de los pesos y descentramientos (*weights* y *biases*, en inglés). Una forma de poder entender estas variables es mediante la fórmula:

$$y = a * x + b$$

Que representa a la función de activación de cada nodo, donde ***a*** sería el peso y ***b*** el descentramiento. Estas variables serán modificadas durante el entrenamiento de la red, por lo que su valor inicial nos es indiferente.

Por ello, elegimos un valor aleatorio siguiendo una normal y de tipo *float*. Los valores serán guardados en matrices de tamaño definido según el tamaño que indicamos que debían tener las capas ocultas, como podemos ver en la Figura 15.

```
weights = {
    'encoder_h1': tf.Variable(tf.random_normal([num_input, num_hidden_1], dtype=tf.float64)),
    'encoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_hidden_2], dtype=tf.float64)),
    'decoder_h1': tf.Variable(tf.random_normal([num_hidden_2, num_hidden_1], dtype=tf.float64)),
    'decoder_h2': tf.Variable(tf.random_normal([num_hidden_1, num_input], dtype=tf.float64)),
}

biases = {
    'encoder_b1': tf.Variable(tf.random_normal([num_hidden_1], dtype=tf.float64)),
    'encoder_b2': tf.Variable(tf.random_normal([num_hidden_2], dtype=tf.float64)),
    'decoder_b1': tf.Variable(tf.random_normal([num_hidden_1], dtype=tf.float64)),
    'decoder_b2': tf.Variable(tf.random_normal([num_input], dtype=tf.float64)),
}
```

Figura 15 - Valores de pesos y descentramientos

Para nuestra matriz de entrada usaremos un *placeholder* ya que su valor irá cambiando y le daremos el nombre **X**. El parámetro **None** al seleccionar el tamaño de esta matriz significa que esa dimensión podrá tomar cualquier valor.

```
X = tf.placeholder(tf.float64, [None, num_input], name="X1")
```

Figura 16 - Matriz de entrada

Ahora construiremos el codificador y el decodificador. Para la función de activación (la que determina si un nodo pasa su resultado a los siguientes nodos) de los nodos usaremos una sigmoide. Esta función es muy usada en el entrenamiento de redes neuronales ya que cuenta con multitud de ventajas, por ejemplo: su salida siempre estará entre 0 y 1, lo que permite elegir fácilmente si activar o no el nodo, su gradiente es suave, las combinaciones de funciones de este tipo no son lineares, lo que nos permite agrupar capas. Otra opción sería usar la función *tanh* [54].

Como se ha especificado, tanto el codificador como el decodificador estarán compuestos por 2 capas, las cuales serán construidas de forma muy similar:

```
def encoder(x):
    # Capa oculta 1 del codificador con activacion de sigmoide
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['encoder_h1']), biases['encoder_b1']))
    # Capa oculta 2 del codificador con activacion de sigmoide
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['encoder_h2']), biases['encoder_b2']))
    return layer_2
```

Figura 17 - Codificador

```
def decoder(x):
    # Capa oculta 1 del decodificador con activacion de sigmoide
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, weights['decoder_h1']), biases['decoder_b1']))
    # Capa oculta 2 del decodificador con activacion de sigmoide
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, weights['decoder_h2']), biases['decoder_b2']))
    return layer_2
```

Figura 18 - Decodificador

Aunque parezca complicado, lo único que se está haciendo en todas las capas es multiplicar los valores de la matriz recibida por los pesos correspondientes, sumarles los descentramientos y pasar ese valor a la función sigmoide.

En la Figura 20 podemos ver que unimos las partes del modelo como se indicó en la Figura 12:

```
encoder_op = encoder(X)
decoder_op = decoder(encoder_op)

# Prediccion
y_pred = decoder_op

# Los objetivos son
#los datos de entrada.
y_true = X
```

Figura 20 - Unión del modelo

```
loss = tf.losses.mean_squared_error(y_true, y_pred)
optimizer = tf.train.RMSPropOptimizer(0.06).minimize(loss)
```

Figura 19 - Función de pérdida y optimizador

Y en la Figura 19 vemos que la pérdida (*loss*) la calcularemos mediante la media del error al cuadrado de la salida predicha por la red (*y_pred*) y la salida que debería ser (*y_true*). Entrenaremos la red para minimizar esta pérdida mediante un optimizador RMSProp (punto 4.1) con un ritmo de aprendizaje de 0.06. Este valor tampoco hay forma de calcularlo, pero no interesa que sea ni muy alto (la red pasaría detalles por alto al aprender) ni muy bajo (la red aprendería muy lentamente).

Ya estaría todo listo para empezar a entrenar a la red.

4.6 Entrenamiento de la red

Cualquier interacción con la red en TensorFlow se hace con una sesión (*Session*), la cual iniciaremos con: `session = tf.Session()`, siendo *session* la variable que usaremos a partir de ahora. Lo primero que debemos hacer después de esto es inicializar las variables locales y globales, mediante métodos que habremos declarado antes (Figura 21). Con esto conseguiremos que todas las variables de nuestra red tomen valor, por ejemplo, los pesos y descentramientos anteriormente definidos.

```
131  #Antes de la sesión
132  init = tf.global_variables_initializer()
133  local_init = tf.local_variables_initializer()

154  #Dentro de la sesión
155  session.run(init)
156  session.run(local_init)
```

Figura 21 - Inicialización de variables

Como no podemos introducir la matriz entera como entrada en la red debido a limitaciones de memoria, la dividiremos en distintos lotes (*batch*). Aunque no haya limitaciones de memoria, también es recomendable dividir las entradas en lotes. El tamaño de estos lotes es otra variable que decidir: un tamaño grande hará que tardemos menos en entrenar a la red, pero no permitirá entrenar a la red tan profundamente como permitiría un tamaño pequeño. Aun así, un tamaño demasiado pequeño nos perjudicaría ya que la red no sería capaz de tener una visión amplia del set de datos.

Entrenar a la red consiste en realizar varias repeticiones de estas entradas de datos, para que la red vaya modificando sus parámetros. A estas repeticiones se les llama épocas (*epoch*) y también tendremos que elegir cuántas vamos a hacer. Como es obvio, cuantas más hagamos, más durará el entrenamiento total, pero mejor entrenada estará la red. Habrá que encontrar un equilibrio entre ambas partes.

Tras sucesivos entrenamientos y pruebas, el entrenamiento final de nuestra red se ha realizado con un tamaño de lote de 300 (es decir, una matriz de 300 filas y todas las columnas de la matriz original) y 150 épocas.

Una vez tengamos elegido cada batch para entrenar a la matriz, nos bastará con llamar al siguiente método:

```
_, l, summary_str = session.run([optimizer, loss, merged_summary_op], feed_dict={X: batch})
```

Figura 22 - Entrenamiento de la red

En él, vemos que pasamos distintos parámetros a la red. Por un lado, le proporcionamos el optimizador, la función de pérdidas y otro valor que nos permitirá representar la evolución de la red con el tiempo. Por otro lado, le proporcionaremos el batch de entrada.

Este método nos devolverá el optimizador, que no nos interesa y por eso ponemos _; la pérdida de cada vez y otro valor que usaremos más adelante.

Realizar las 150 épocas con lotes de tamaño 300 llevó 3 días, 23 horas, 24 minutos y 38 segundos, es decir, casi 4 días. Tras el entrenamiento debemos guardar la red en un archivo, porque si no, cuando acabe el programa, la sesión desaparecerá y con ella la red y todo el entrenamiento.

Para ello, usaremos la clase *Saver*, la cual iniciaremos antes de empezar con la sesión y, tras acabar, pasaremos la referencia de la sesión para ser guardada en el sitio que especifiquemos. El archivo de esta sesión ocupa más de 1 GB.

```
134  #Antes de iniciar la sesión
135  saver = tf.train.Saver()
```

```
191  #Tras haber acabado el entrenamiento
192  saver.save(session, './red5/')
```

Figura 23 - Guardando la red neuronal

Antes de guardar debemos especificar qué variables queremos usar cuando restauremos la función. Esto se hace al declarar dichas variables, poniéndoles un nombre especial. En nuestro caso será X, la variable de entrada y por eso, en la Figura 16, introducimos el tercer parámetro al crearla.

Por otro lado, TensorFlow también nos proporciona una utilidad para hacer representaciones de nuestra red neuronal. Nosotros la usaremos para representar el valor de la pérdida en función del tiempo. Como podemos ver en la Figura 24, primero tendremos que declarar la variable de pérdida en un espacio con un nombre determinado; luego, se llamaría a dos métodos para decir que vamos a guardar esa y/o más variables y unirlas todas; y, finalmente, cada vez que se ejecute el entrenamiento de la red, guardaremos el valor de pérdida devuelto.

```
115  with tf.name_scope("Loss"):
116      loss = tf.losses.mean_squared_error(y_true, y_pred)
```

```
138  tf.summary.scalar("loss", loss)
139  merged_summary_op = tf.summary.merge_all()
```

```
174  summary_writer.add_summary(summary_str, i*num_batches + num_batch)
```

Figura 24 - Proceso para representar valores

Así, luego podremos llamar al programa TensorBoard (programa incorporado en la instalación de TensorFlow) mediante consola de comandos y este nos proporcionará todas las representaciones que le hayamos indicado mediante una interfaz web:

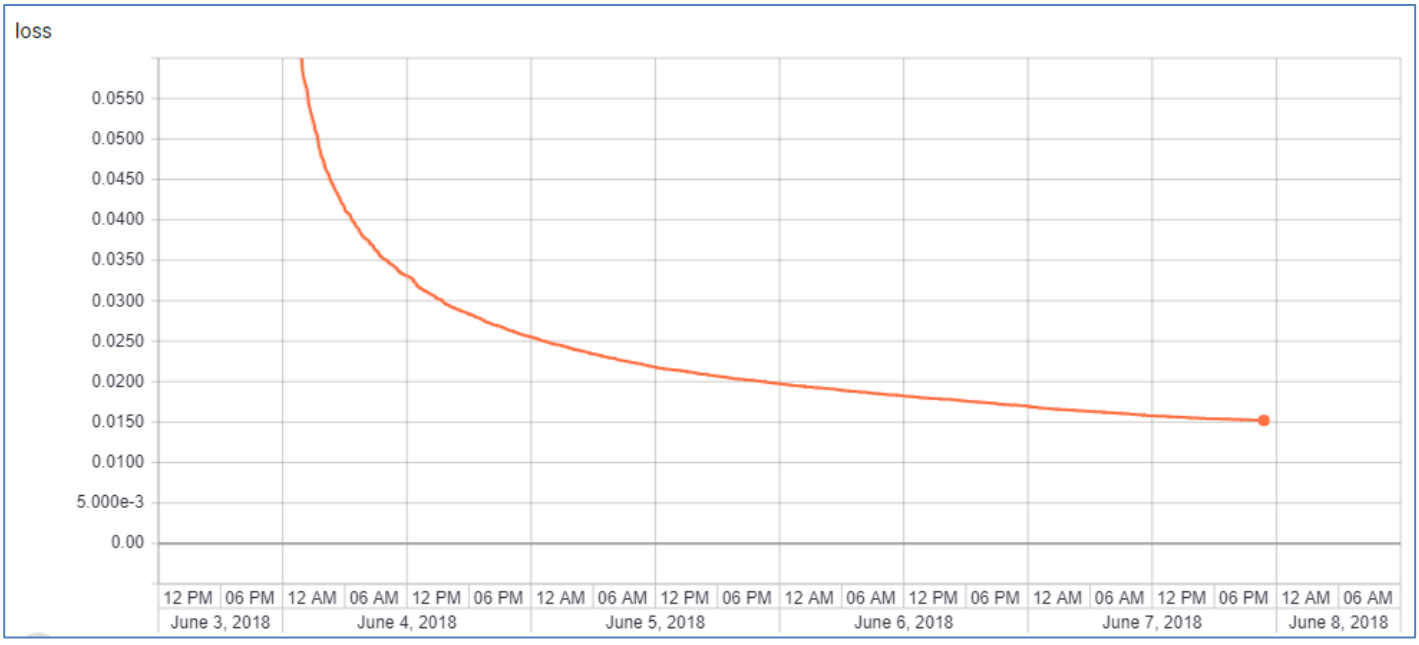

Figura 25 - Variable de pérdida en función del tiempo de entrenamiento

Como se explicó anteriormente, estos valores por sí solos no significan gran cosa, solo nos servirían para compararlos con otros entrenamientos similares y ver si son más o menos eficientes. Aun así, nos permiten ver una cosa: al principio, mejorar la red no es muy difícil, pero en cuanto va avanzando el tiempo y con él las sesiones de entrenamiento, cuesta cada vez más optimizar la red.

TensorBoard también nos ofrece una representación de cómo está construida nuestra red. Si bien no es una representación muy intuitiva, incluyendo que los nombres no son descriptivos, al menos resulta interesante cómo está todo conectado sin que hayamos tenido que hacer malabarismos complejos.

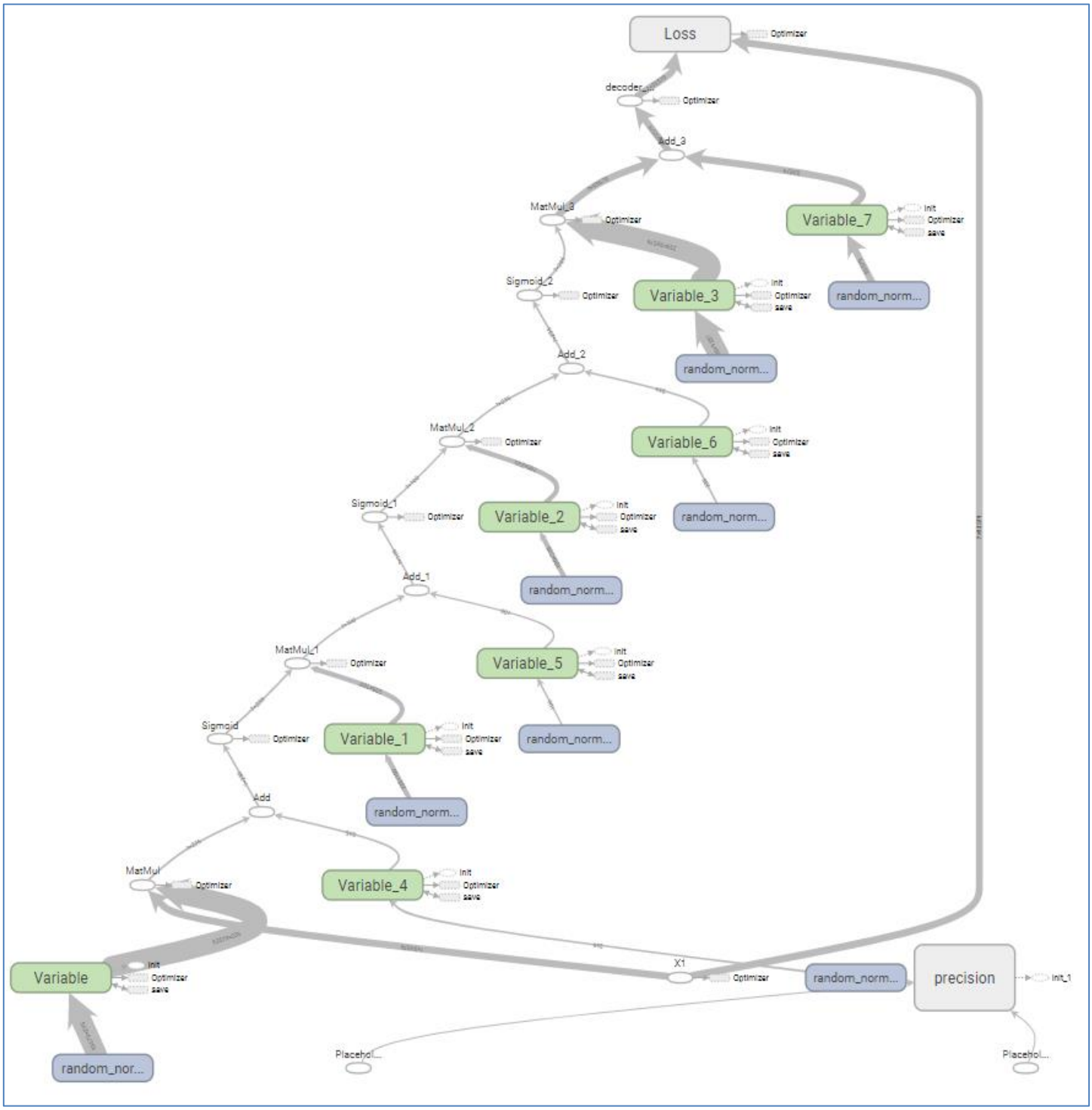

Figura 26 - Representación de la red neuronal

4.7 Pruebas

Probar una red neuronal de recomendación (en concreto, la nuestra) no es fácil ya que, al existir una variedad tan grande de grupos, es imposible conocerlos todos y no podremos predecir qué debería recomendarnos si funcionara bien.

Lo que sí podemos probar es que no funciona mal. Por ejemplo, en los primeros entrenamientos, siempre recomendaba los mismos grupos fuera cual fuera la entrada; por eso se decidió seguir ampliando el tamaño de la red y el tiempo de entrenamiento.

Como se ha visto, los últimos entrenamientos duraban cerca de 4 días, lo que ralentizaba mucho el proceso e impedía detectar fallos rápidamente. Aun así, conseguimos obtener una red aceptable, aunque aún podría ser más precisa con más sesiones de entrenamiento.

Para nuestro proceso de pruebas, lo que hicimos fue generar 27.000 entradas aleatorias, de 10 grupos cada una, para la red y comprobar que no nos devolvía siempre los mismos grupos recomendados. Como sabemos, la red devolverá una matriz igual de grande que la entrada, con cada fila correspondiendo a su respectiva en la entrada, y en la que las columnas cuyos valores que estén más cerca del 1, serán los grupos recomendados.

```
Grupo: 7601 -> veces: 20241
Grupo: 21428 -> veces: 17330
Grupo: 55379 -> veces: 15896
Grupo: 36851 -> veces: 13244
Grupo: 27935 -> veces: 13000
Grupo: 20276 -> veces: 12031
Grupo: 36150 -> veces: 10021
.........
Grupo: 23914 -> veces: 1
Grupo: 10696 -> veces: 1
Grupo: 91643 -> veces: 1
Grupo: 26043 -> veces: 1
Grupo: 5874 -> veces: 1
```

Figura 27 - Fragmento de los resultados de los entrenamientos

Alimentaremos a la red con esta matriz de entrada y analizaremos la salida, quedándonos con el grupo más recomendado de cada set de entrada (conjunto de 10 grupos) y contando las veces que lo obteníamos en total. Una parte de estos datos se encuentra en la Figura 27.

Cada número de grupo luego sería traducido a su nombre real en nuestra aplicación final. Como podemos ver, aún existe cierta preferencia hacia determinados grupos, pero no es tan fuerte como la de los primeros entrenamientos. Esto se reduciría con más sesiones de entrenamiento.

También realizamos este experimento, en vez de escogiendo el grupo más recomendado, escogiendo los 10 y 20 grupos más recomendados y contando de nuevo. Los resultados tienen similar estructura (unos grupos más recomendados que otros), pero la variedad de grupos en ellos es mayor.

4.8 Uso de la red

Cuando queramos volver a usar nuestra red, tan solo tendremos que iniciar la sesión, cargar la red desde el fichero en el que la guardamos anteriormente y restaurarla (Figura 28).

```
session = tf.Session()
saver = tf.train.import_meta_graph('../red/red4/red4.meta')
saver.restore(session,tf.train.latest_checkpoint('../red/red4/'))

graph = tf.get_default_graph()
```

Figura 28 - Restaurando la red

Tras esto, debemos obtener los tensores que usamos tanto para introducir datos en la red (X) como los usados para obtener las predicciones (pesos y descentramientos). Esto se hace llamando en la variable *graph* al método encargado de devolver el tensor según su nombre, como podemos ver en la Figura 29.

Finalmente, construiremos de nuevo el modelo predictivo (codificador-decodificador) con estos valores y llamaremos a la red pasándole el decodificador, es decir, la salida de nuestro modelo (Figura 30).

```
weights = {
    'encoder_h1': graph.get_tensor_by_name("Variable:0"),
    'encoder_h2': graph.get_tensor_by_name("Variable_1:0"),
    'decoder_h1': graph.get_tensor_by_name("Variable_2:0"),
    'decoder_h2': graph.get_tensor_by_name("Variable_3:0"),
}

biases = {
    'encoder_b1': graph.get_tensor_by_name("Variable_4:0"),
    'encoder_b2': graph.get_tensor_by_name("Variable_5:0"),
    'decoder_b1': graph.get_tensor_by_name("Variable_6:0"),
    'decoder_b2': graph.get_tensor_by_name("Variable_7:0"),
}

X = graph.get_tensor_by_name("X1:0")
```

Figura 29 - Obteniendo los tensores

```
preds = session.run(decoder_op, feed_dict={X: nuevaLista.todense()})
```

Figura 30 - Llamando a la red

Si hubiéramos terminado de entrenar y verificar nuestra red, estaría lista para ser usada en producción. Esto significa que no necesitaríamos muchas de las opciones y variables que TensorFlow nos da al hacer el guardado normal. Esto se llama "congelar" la red (*freeze*), y nos proporcionaría un archivo único de extensión .pb y menor tamaño que el anterior en el cual va contenido solo lo indispensable para que la red funcione.

Simplemente nos basta con seleccionar las variables que queremos usar y llamar a la función de guardado, como vemos en la Figura 31:

```
input_graph_def = graph.as_graph_def()
output_graph_def = tf.graph_util.convert_variables_to_constants(
            session, # The session
            input_graph_def,
            ("X1,Variable,Variable_1,Variable_2,Variable_3,
            Variable_4,Variable_5,Variable_6,Variable_7").split(",")
)

output_graph="./freeze/red.pb"
with tf.gfile.GFile(output_graph, "wb") as f:
    f.write(output_graph_def.SerializeToString())
```

Figura 31 - Congelando la red

5 SERVIDOR WEB

People worry that computers will get too smart and take over the world, but the real problem is that they're too stupid and they've already taken over the world.

- Pedro Domingos -

Para poder probar nuestra red, simularemos una página web con la que los usuarios podrán interactuar y que irá recogiendo sus acciones, procesándolas en la red, y generando una serie de recomendaciones. Como se ha indicado anteriormente, nuestro servidor web se ha hecho usando Bottle, un módulo de Python.

Se ha intentado que el diseño gráfico de la web sea atractivo, pero, dado que ese punto no es una parte de este proyecto, el resultado final es mejorable.

5.1 Estructura

La estructura básica de nuestra web puede verse en la Figura 32. En esta imagen podemos ver que contamos con una cabecera donde estaría el título de la página, debajo, los grupos que se han visitado y, a la derecha, 2 elementos para tomar datos mientras se hacen las pruebas: el apodo con el que queremos que se guarden nuestros datos (esto simularía a un usuario que ha iniciado sesión) y un botón para reiniciar todo el proceso y poder probar visitando otros grupos.

Debajo tendríamos 4 cajas:

- La primera y más importante, los 3 grupos recomendados por la red.
- La segunda, un buscador de grupos.
- La tercera, una selección de 3 grupos aleatorios que se genera con cada nueva página y un botón para refrescar esos grupos.
- Una lista de los 3 grupos más escuchados, con opción de acceder a la lista con todos los grupos ordenados.

Excepto en el buscador, con cada grupo se añade una pequeña foto para que resulte más sencillo identificarlos. Cuando accedamos a la página de un grupo, veremos una distribución similar solo que con información y fotos de ese grupo.

Figura 32 - Página inicial de la web

Para el funcionamiento de nuestro servidor, contaremos con distintos ficheros:

- *server.py*: Archivo principal, donde se encuentra el servidor y los métodos para recibir peticiones.
- *grupos.py*: Módulo con métodos encargados de realizar todo lo relacionado con grupos.
- *petición.py:* Módulo encargado de realizar peticiones a la API de Last.fm.
- *recom.py:* Módulo encargado de usar la red neuronal y obtener las recomendaciones.
- *index.html:* Archivo con la estructura de la página web principal.
- *grupos.html:* Archivo con la estructura de la página web de los grupos.
- *script.js:* Archivo con el código JavaScript usado en todas las páginas.
- *style.css:* Archivo con los estilos necesarios para la buena presentación de las páginas.

5.2 Peticiones al servidor

Los métodos del servidor se encuentran en el archivo *server.py*. Todas las peticiones HTTP que recibiremos serán de tipo GET, así que tendremos que distinguirlas mediante la URL. Tendremos:

- **Página principal y grupos:**

  Las principales peticiones que recibiremos en nuestra página web serán la de la página principal y la de los propios grupos, caracterizadas por las URL '/' y ' '/grupos/<*nombre*>' (donde <*nombre*> es el nombre del grupo), respectivamente. En puntos posteriores se explica con detalle el funcionamiento de los métodos encargados de atender estas peticiones.

- **Peticiones de ficheros concretos:**

  Esto sucederá para archivos como el de JavaScript (*.js*), la hoja de estilos (*.css*), imágenes (*.jpg*, *.png*, *.gif* o *.ico*). Para responderlas, simplemente abriremos el archivo pedido, lo leeremos y lo devolveremos al usuario, incluyendo en la respuesta el tipo de archivo (Content-Type) que es, para que el navegador pueda procesarlo bien.

```
@route('/datos/imagenes/<filename>.png')
def png(filename):
    response.content_type = 'image/png'
    return static_file("datos/imagenes/" + filename + ".png",
                       root=".",
                       mimetype='image/png')
```

Figura 33 - Método para las peticiones de archivos PNG

- **Búsquedas:**

  El usuario puede usar la caja de búsqueda para encontrar algún grupo y su contenido se irá actualizando dinámicamente. Para ello compararemos el nombre introducido con la lista de grupos, obteniendo aquellos que tengan esa cadena de texto en alguna parte de su nombre. En caso de que haya más de un resultado, el orden de los mismos será por popularidad, usando la lista ordenada del punto 4.4. La URL para esta petición es '/busca/<*nombre*>'.

- **Aleatorio y top:**

  El contenido de la caja de Aleatorio puede ser refrescado mediante un botón. El servidor atenderá esta petición y devolverá tres nuevos grupos, que serán procesados en el lado del cliente para su correcta visualización. Para el top, si se pulsa el botón de *Top Completo,* devolveremos los primeros 100 grupos ordenados, pudiendo en un futuro paginarse para ver el top completo.

  Las URLs son '/aleatorio' y '/top', respectivamente. Estos métodos se valdrán del módulo *grupos*, que será explicado más adelante, para procesar sus peticiones.

- **Otras peticiones:**

  Cualquier otra petición que no esté dirigida a las URL anteriormente mencionadas, será redirigida a la página principal, como podemos ver en la Figura 34.

```
@route("/<url:re:.+>")
def redir(url):
    redirect('/')
```

Figura 34 - Redirección a página principal

5.3 Páginas principales

Las páginas que podrá ver el usuario serán las de los grupos y la principal. El funcionamiento de los métodos encargados de recibir estas peticiones es muy parecido.

Primero, leeremos el archivo que contiene la estructura de la página a mandar al usuario. La peculiaridad de estos archivos es que están incompletos. En aquellas partes que tengamos que generar contenido dinámicamente, hay cadenas de texto especiales que serán reemplazadas una vez se haya generado el contenido.

Este comportamiento ha sido inspirado por Angular [55], un marco (*framework*) que nos permite desarrollar páginas webs y servicios de front-end muy fácilmente y que se basa en módulos contenidos dentro de otros. Así, una vez se genera la página, se reemplazan cadenas de texto por módulos completos.

En nuestro caso, no pensamos que fuera necesario implementar Angular ya que nuestra web no ofrecía funcionalidades que aprovecharan verdaderamente su potencial y podíamos hacer todo lo que nos proponíamos en JavaScript. Por lo tanto, tendremos estructuras como la de la Figura 35, en la que la cadena *_Recomendados_* sería reemplazada por la lista de recomendados una vez generada.

```
<div id="recomendados">
  <p id="recomendadosP">Recomendados</p>
  _Recomendados_
</div>
```

Figura 35 - Estructura con cadena para ser reemplazada

Este método se usará tanto para los grupos recomendados, como para el top de grupos, los grupos aleatorios y la información sobre el grupo que se está visitando. Una vez se tenga la página completa, se enviará al cliente de vuelta.

Para todo lo relacionado con grupos, se usará el módulo de *grupos*. En él, tendremos guardados en dos vectores la lista de grupos con la que hacer las recomendaciones, y la lista de grupos ordenada, para hacer el top y la búsqueda.

Para hacer el contenido con el que reemplazar el texto de la caja de Aleatorio, simplemente seleccionaremos tres grupos aleatorios de uno de esos vectores y crearemos el contenido. Algo similar haremos para el Top de grupos también y cuando desde el cliente recibamos la petición de generar nuevos aleatorios o mostrar el top completo.

Con cada grupo queremos añadir una imagen del mismo. Para esto nos valdremos de nuevo de la API de Last.fm ya que también nos proporciona URLs con imágenes de los grupos cuando realizamos una petición para obtener información del mismo. Estas peticiones las haremos a través del módulo *petición*, de forma similar a como se hicieron las peticiones en los puntos 4.2 y 4.3.

Guardaremos dinámicamente en un diccionario los grupos ya pedidos para no tener que realizar tantas peticiones y agilizar el proceso. Un diccionario es un elemento de Python que nos permite relacionar fácilmente dos valores: clave y valor. Pudiendo obtener un valor (URL) simplemente introduciendo la clave correspondiente (nombre del grupo).

Por otro lado, también usaremos el módulo de peticiones para añadir, en las páginas de los grupos, una pequeña biografía sobre los mismos. Esta biografía también viene incluida en las peticiones sobre información de cada grupo.

Existe un problema con el nombre de los grupos: es probable que contengan caracteres especiales como barras inclinadas (/), puntos (.), signos de interrogación (?), etc. Esto nos dificulta el crear enlaces para buscar o dirigirnos a estos grupos, ya que serían procesados erróneamente. Por ello, todos los caracteres especiales que den problemas serán reemplazados por la cadena *%TFG____*, incluyendo el nombre del símbolo.

Se ha elegido este formato de cadena para asegurarnos que no nos la encontramos en ningún otro lado y pueda dar falsos positivos. Así, tanto en el lado del back-end (Python) como del front-end (JavaScript), los caracteres serán reemplazados a la hora de enviar una petición o respuesta y el destinatario se encargará de invertir este reemplazamiento. Un ejemplo puede ser visto en la Figura 36.

```
def replaceName(name):
    return name.replace("'","%TFGComa").replace("/","%TFGBarra").replace("?","%TFGPregunta").replace(".","%TFGPunto")
```

Figura 36 - Método para reemplazar caracteres especiales

Para cada grupo del que recibimos petición, comprobamos si existe en nuestra base de datos o simplemente es una petición errónea. Si el grupo no existe, se genera una página indicándolo y no contaría ni para las cookies ni para las recomendaciones.

Desde el cliente, con cada petición recibiremos las cookies, en las que guardaremos distinta información sobre el mismo. Por un lado, tendremos la cookie *apodo,* que nos proporcionará el nombre del usuario para poder guardar información sobre el mismo para futuros entrenamientos de la red neuronal; pero estamos interesados en la cookie *visita*, que es la que nos va a proporcionar la información sobre los grupos que se han visitado anteriormente.

En esta cookie, habrá un máximo de 10 grupos, separados por la cadena **TFG**. De nuevo, esto se hace para no encontrarnos con falsos positivos ya que es probable que, si separamos los grupos, por ejemplo, con comas, encontremos nombres de grupos que contengan ese carácter y la división de nombres se hará más difícil. Por ello, la estructura de la cadena de texto de la cookie tendrá la estructura:

visita=**[grupo1]***TFG***[grupo2]***TFG*…**[grupo10]**

Las cookies estarán separadas entre sí por el carácter *;* (punto y coma). Por lo tanto, la lógica para obtener los grupos será la siguiente:

```
if cookies is not None:
    for i in cookies.split(";"):
        if "visita" in i:
            #Obtenemos los grupos de esta cookie
            gruposCookies = i.split("=")[1].split("*TFG*")
            for j in gruposCookies:
                listaGrupos.append(j)
```

Figura 37 - Obteniendo los grupos desde las cookies

Cabe destacar que el grupo actual (el grupo del que se ha hecho la petición) no estará incluido en las cookies hasta que el cliente no reciba la página final, por lo que tendremos que añadirlo manualmente a la lista de grupos visitados.

Tras tener la lista con los grupos, llamaremos al módulo *grupos* para transformar los nombres en números y luego llamaremos al módulo *recom*, que transformará ese vector de números en una tabla, similar a la que usamos para entrenar y probar la red. Las dimensiones de esta tabla serán: 1 fila (1 usuario) y 93579 columnas (grupos).

Al principio del programa, el módulo de recomendación se encargará de cargar la red desde el archivo donde está guardada y crear todo lo necesario para poder usarla (punto 4.8). Después, cada vez que llamemos al método de recomendar de este módulo, creará la tabla e insertará valores de 1 a 0.46 en saltos de 0.06 para las 10 columnas de los grupos que se han visitado, siendo aquellos con valores más altos los grupos que se han visitado más recientemente.

Pasaremos estos valores a la red y nos devolverá una tabla de las mismas dimensiones donde las columnas con los valores más altos serán los grupos más recomendados. Nos quedaremos con los tres primeros grupos ordenados según estos valores. Estos grupos serán devueltos al módulo *grupos*, que los procesará y creará el contenido de la tabla de recomendación.

Ya tendríamos todo lo necesario para poder devolver la página al usuario. Como se puede observar, hay varios puntos del proceso que requieren algo de tiempo para poderse completar (peticiones a API, uso de la red, etc.). Como queremos que la experiencia del usuario sea lo más fluida posible, no podemos permitir que espere demasiado, por lo que haremos uso de hilos.

Los hilos nos permitirán realizar varias tareas simultáneamente y así agilizar el proceso. Para que los hilos puedan devolver valores debemos pasar un puntero a un vector, en el que insertarán la información. Un ejemplo de uso de hilos puede verse en la siguiente figura, en este caso, para las recomendaciones:

```
61  #Hilo para calcular los grupos para recomendar
62  cadenaResultadoCookies=[]
63  tCookies = threading.Thread(target=leeCookies, args=(request.get_header("Cookie"),cadenaResultadoCookies,nombre,))
64  tCookies.start()
```

```
84  tCookies.join()
85
86  #Insertamos los grupos recomendados
87  pagina = pagina.replace("_Recomendados_", cadenaResultadoCookies[0],1)
```

Figura 38 - Hilo para recomendaciones

Como podemos ver, creamos y empezamos el hilo, luego realizaremos otras operaciones con la página mientras el hilo se ejecuta paralelamente y, al final, esperamos a que el hilo acabe y reemplazamos en la página, la cadena base con el resultado.

Cuando todos los hilos hayan acabado y su contenido sea reemplazado en la página, devolveremos esta al usuario, lugar donde se producirán una serie de operaciones explicadas en el siguiente punto.

Una última observación sobre la red neuronal: la red que usaremos será una red preparada para el ámbito de producción, es decir, una red neuronal congelada (punto 4.8). Aun así, a medida que los usuarios vayan usando nuestra web y recojamos sus datos de uso, querremos entrenar la red para mejorarla. Esto lo haremos con una versión de desarrollo de la web, es decir, la que teníamos antes de congelarla.

El entrenamiento se realizará al margen de la red de la web y, periódicamente, se congelarán las nuevas versiones y se irán sustituyendo las antiguas, con el fin de siempre proporcionar la mayor calidad en cuanto a recomendaciones.

Este entrenamiento continuo se hace ya que nosotros hemos entrenado nuestra red con datos de una página internacional y nuestro público objetivo puede ser más centrado en el ámbito nacional, por lo que los gustos serán diferentes. Una forma de mitigar esto sería filtrar por país (o cualquier otra característica) en la recolección de datos (punto 4.3).

5.4 Front-end

En las páginas que enviemos al cliente incluiremos código JavaScript para ser ejecutado mientras el cliente interactúa con la página. Este código estará compuesto de distintos métodos, cada uno con su finalidad.

Por un lado, tendremos la función de inicio, que se ejecuta nada más cargar la página y es la más importante de todas. Habrá dos variantes: inicio de página principal e inicio de grupos, pero ambas son muy similares.

Esta función se encargará de comprobar si el usuario ha visitado anteriormente la página mediante la cookie *primera* (de primera vez), a la que daremos distintos valores. Si no es así, preguntaremos al usuario por su apodo, para poder usarlo luego en el servidor. Si la primera página que visita el usuario es una de un grupo, será redirigido a la página principal y se realizará la pregunta también. Esto simula un futuro sistema de inicio de sesión.

Una vez se ha comprobado, o guardado si es la primera vez, el nombre del usuario en la cookie *apodo*, añadiremos el grupo que se está visitando actualmente a la cookie *visita*, para luego ser procesada por el servidor como se explicó anteriormente. Como es obvio, si estamos visitando la página principal o un grupo que no existe (obtendremos la página de que no existe el grupo), no añadiremos nada a esta cookie. También, si la cookie ya contiene 10 grupos y añadimos uno más, quitaremos el grupo que se haya visitado hace más tiempo de la misma.

Aparte de la función (o funciones) de inicio, tendremos otras que también nos ayudarán para dotar de funcionalidad a la web, por ejemplo, la función a la que se llama cuando se pulsa el botón de *Empezar de nuevo* (Figura 32), que borrará todas las cookies y redirigirá a la página principal. O a la que se llama cuando queremos visualizar el *Top Completo*, que nos llevará a una nueva página.

Por otro lado, cada vez que tecleemos algo en la caja de *Búsqueda*, se llama a la función *buscar*, que a su vez hará una petición al servidor a la URL de buscar con el contenido de la caja. El servidor nos devolverá una lista con grupos para insertarla debajo de la caja de búsqueda.

Para el botón de aleatorio pasará algo similar: un método que hace una petición al servidor y que obtiene una lista que insertar en una caja. En este caso, hemos adornado la espera con un GIF simbolizando que está cargando.

Para realizar las peticiones hemos creado un método al que le pasamos la URL de las mismas. Como las peticiones al servidor no son instantáneas, este método tiene que ser asíncrono, ya que, si no, bloquearía el funcionamiento de todo el otro código.

Para ello, lo que hace este método es crear una “Promesa” con la petición (Figura 39) y, una vez recibida la respuesta, ejecutaremos el resto del código en la función que ha llamado a esta función (Figura 40).

```
//Función para realizar cualquier petición al backend
function httpGet(nombre)
{
  return new Promise(function (resolve, reject) {
    //Calcula la url de la página
    var loc = window.location.href;
    var index = 0;
    for (var i = 0; i<3; i++){
      index = loc.indexOf("/", index+1);
    }

    //Realiza la petición
    var url = loc.substring(0,index) + nombre;
    var xmlHttp = new XMLHttpRequest();
    xmlHttp.open( "GET", url );
    xmlHttp.onload = resolve;
    xmlHttp.onerror = reject;
    xmlHttp.send();

  });
}
```

Figura 39 - Función para hacer peticiones al servidor

```
//Hace la petición al servidor
httpGet("/buscar/" + texto).then(function (e){
  var respuesta = e.target.response;
  ...
```

Figura 40 - Forma de llamar a la función de realizar peticiones y obtener el resultado de forma asíncrona

Con esto, ya estaría completo el código de JavaScript, en la parte del front-end y, con él, todas las funcionalidades de nuestra web.

6 CONCLUSIONES

I visualize a time when we will be to robots what dogs are to humans, and I'm rooting for the machines

- Claude Shannon -

Como podemos ver, desarrollar una página web de recomendación no supone muchos problemas, aparte del elemento principal: la red neuronal. Si bien entrenar una red neuronal para casos concretos actualmente es muy sencillo, como por ejemplo para detección de imágenes, sonido, realización de tareas específicas, etc.; entrenarla para casos muy amplios como el nuestro no carece de problemas.

El primero es que cuanto más amplio sea el set de datos a relacionar, más difícil será entrenar a la red, en cuanto a temas de tiempo y recursos, así como que también necesitaremos una muestra de datos mayor. En nuestro caso creímos que con cerca de 300 mil muestras era suficiente. Puede que, si hubiéramos tomado una muestra más grande, nuestra red funcionara mejor.

El modelado de la red tampoco es algo casual, dar con la estructura y valores correctos de la misma es más un arte que una ciencia, es decir, no hay ningún método para predecir qué va a funcionar y qué no y la intuición juega un papel fundamental.

Además, nos encontramos con el problema de la caja negra: nuestra red es una "caja" que toma entradas y produce salidas, pero cuyo contenido no podemos modificar manualmente; tan solo podemos indicar cómo queremos que sean las salidas y esperar que funcione bien.

El tiempo también ha sido un aspecto crucial en nuestro proyecto ya que, entre la recolección de datos y entrenamiento de la red, se ha invertido mucho tiempo, sobre todo computacionalmente. Para la recolección de datos, realizamos, más o menos, unas 10 mil peticiones sobre amigos, con las que obtuvimos más de 300 mil usuarios de los que luego realizamos peticiones para información sobre grupos.

A una petición por segundo de media, esto equivale a más de 86 horas de peticiones continuas a las que hay que sumarles cierres repentinos de los programas debidos a fallos de la API o de programación de los mismos debido a caracteres inválidos o errores en general en la recogida de datos. Esto conllevó a reiniciar el proceso completo un par de veces.

Con el entrenamiento de la red pasa algo similar: como no se sabe qué va a funcionar, no queda más remedio que probar. Las primeras pruebas duraban cerca de 13 horas, mientras que las pruebas finales se acercaron a los 4 días de duración, es decir, 4 días de procesamiento y entrenamiento continuo de la red.

Al igual que con la recogida de datos, hubo fallos inesperados que hicieron reiniciar el proceso por completo varias veces. Por ejemplo, aunque nuestro equipo es un ordenador portátil y cuenta con batería propia, mientras se está entrenando la red, no puede ser desconectado de la red eléctrica o fallará todo el proceso y se perderá todo lo entrenado hasta ese momento. Esto incluye cortes de luz y reinicios espontáneos del sistema operativo. Sobra decir que todos estos fallos se descubrieron de la peor forma posible.

Todo esto ha hecho que nuestra red no esté en un estado óptimo, aunque si se acerca poco a poco al mismo. La solución es entrenarla durante más tiempo y con más datos para poder tener recomendaciones más precisas. También podríamos implementar otros sistemas de recomendación como los basados en el contenido de los elementos o incluso probar otras estructuras de red, ya que el perceptrón (el tipo de red neuronal que hemos usado), tiene algunas limitaciones.

Aun así, estamos satisfechos con el resultado ya que, teniendo en cuenta que este proyecto se empezó sin saber nada de TensorFlow, se han conseguido resultados bastante buenos y se ha aprendido mucho sobre redes neuronales y TensorFlow en sí.

Sobre el futuro de las redes neuronales, está claro que hace falta un gran esfuerzo de investigación para poder crear una buena base. Actualmente son una buena solución para problemas secundarios (donde no importa demasiado si el resultado está bien o no), pero para problemas importantes como cuando corre riesgo una vida no son fiables, ya que cuentan con múltiples problemas de seguridad.

Pese a todo, habrá que estar constantemente vigilando las nuevas alternativas que surgen sobre Machine Learning ya que es muy probable que todo esto que hemos visto se use cada vez más en todos los campos posibles y la Inteligencia Artificial haya llegado para quedarse.

6.1 Próximos pasos

Los próximos pasos que seguir en cuanto al proyecto serían, por un lado, mejorar la interfaz web y la comunicación de front-end con back-end, por ejemplo, añadiendo nuevas funcionalidades o cambiando la forma de enviar y recibir datos. Si queremos que nuestra web se convierta en un servicio de streaming de música real, tendremos que implementar todo lo necesario para ello, lo cual ha quedado fuera del alcance de nuestro proyecto.

Por otra parte, perfeccionaríamos la red, tal y como se ha mencionado anteriormente: probando nuevos modelos y valores para la red a la vez que, una vez lanzada la web, continuamos entrenándola con los valores obtenidos de los usuarios que usen nuestra página.

7 GESTIÓN DEL TIEMPO

I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.

- Alan Turing -

El proyecto comenzó a realizarse en cuanto tuve noticia del mismo y hablé con Francisco José sobre cómo iba a desarrollarse. Si bien, esta realización abarca múltiples meses, hay que tener en cuenta que el tiempo dedicado al proyecto ha sido mucho mayor en los últimos meses que en los primeros, debido a diversos factores como exigencias de otras asignaturas en cuanto a exámenes, trabajos y prácticas y también las prácticas como becario en las que he estado involucrado.

En la siguiente página se muestra un diagrama de Gantt en el que se puede ver el tiempo estimado dedicado a cada tarea. El desarrollo ha sido bastante linear, habiendo primero una fase de investigación sobre redes neuronales, su funcionamiento y los distintos tipos, seguida por un periodo largo de realización de tutoriales de TensorFlow.

Este segundo período fue largo ya que el objetivo del proyecto no estaba definido todavía, por lo que el tipo de red neuronal a usar mucho menos; y la forma de trabajar con TensorFlow es bastante diferente según el tipo de red que se quiera usar. También, aprender a usar TensorFlow sin ningún otro tipo de conocimiento sobre software para redes neuronales resulta algo complicado y confuso.

Una vez se hubo comprendido medianamente el funcionamiento y uso de TensorFlow, se pasó a la etapa de investigación y recolección de datos, la cual se alargó más de lo previsto debido a los problemas comentados en el apartado anterior (cortes repentinos, mal formato de los datos, etc.).

Mientras tanto, se investigó sobre el tipo de red a usar y la forma de construirla en TensorFlow. Tras haberse completado la recolección de datos, se comenzó a construir y entrenar la red, probando diferentes técnicas y valores para los parámetros internos (pesos, descentramientos, ritmo de aprendizaje, etc.; explicados en el punto 4.5).

A la vez que se entrenaba la red, y ya que este proceso implicaba bastantes horas inactivas mientras se realizaba, se comenzó a desarrollar la web, avanzando con el front-end y el back-end a partes iguales. El último módulo que se incluyó en la web fue el de recomendación, el que usa a la red neuronal para obtener los grupos recomendados. Esto fue así ya que, durante el entrenamiento de la red, no se nos permitía crear otra sesión (aunque fuera con una red diferente) por problemas de capacidad de la GPU.

Finalmente, durante estas últimas semanas, se ha realizado esta memoria a la vez que se daban los últimos retoques a la web.

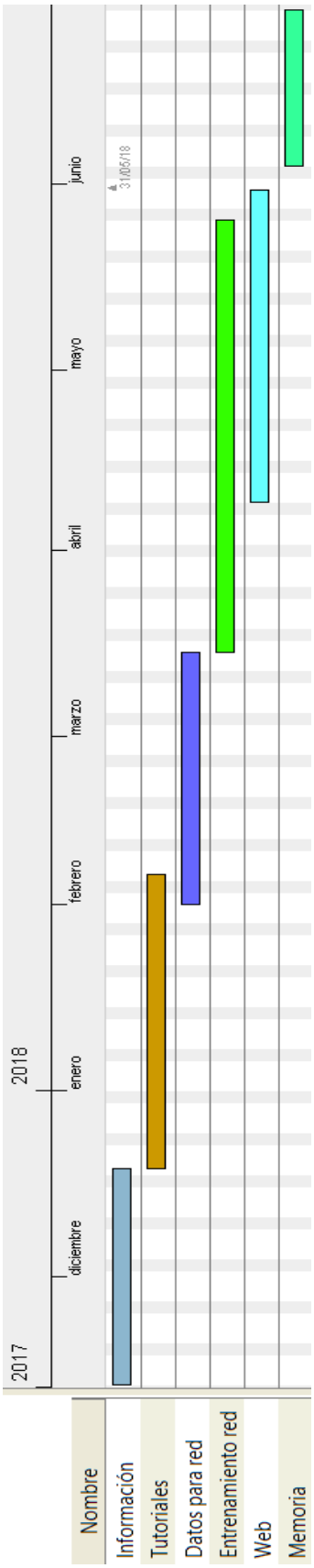

Figura 41 – Diagrama de Gantt para las distintas partes del proyecto

ANEXO A: INSTALACIÓN DE TENSORFLOW

Si bien TensorFlow tiene fama de ser más fácil de instalar que otras librerías para redes neuronales, este proceso no está exento de problemas y puede resultar difícil para desarrolladores novatos.

Como cualquier programa, TensorFlow usará la CPU de nuestro equipo para funcionar, aunque, si contamos con una tarjeta gráfica (GPU) de la marca Nvidia, también existirá la posibilidad de usarla para acelerar la ejecución y los cálculos.

La guía completa de instalación puede encontrarse en las referencias [56] y [57], incluyendo cualquier arquitectura. A continuación, se describirán los pasos que se siguieron para la instalación en nuestro equipo.

Primero, debemos comprobar que nuestra GPU soporta este tipo de computación ya que, pese a que para la mayoría de las GPUs modernas es así, puede darse el caso contrario. Esto se hace comprobándolo en la lista de la referencia [58]. En esta lista también podremos ver la capacidad de computación de la GPU en cuestión, en nuestro caso 5.0.

Tras esto, comprobaremos que tenemos un sistema operativo que soporta CUDA. CUDA es un modelo de computación y programación inventado por Nvidia que nos permite usar la potencia computacional de la GPU para cualquier tarea que se quiera.

Comprobaremos el sistema operativo en la referencia [57], así como que tenemos una versión aceptada de Visual Studio instalada. Si alguno de los pasos anteriores fallara, tendremos o que solucionarlos (instalando versiones que soporten lo que queremos hacer, cambiando el hardware, etc.) o simplemente usar TensorFlow con la CPU.

Tras esto, descargaremos CUDA siguiendo el link que nos proporcionan en [57] y lo instalaremos. Suele tardar bastante, pero no supone ningún problema. Un aspecto a tener en cuenta es que, aunque ponga que soporta Visual Studio 2017 y CUDA 9.0, esto no es así ya que ese soporte se encuentra en desarrollo o, al menos, así era la situación al instalarlo para este proyecto. Por lo tanto, deberemos instalar como máximo las versiones Visual Studio 2015 y CUDA 8.0.

Tras esta instalación verificaremos que los programas se encuentran instalados correctamente, por ejemplo, comprobando que existen los archivos especificados en el punto 2.5.1 de la referencia [57] o ejecutando esos mismos archivos.

Después, debemos instalar cuDNN, una librería para CUDA que nos permite implementar redes neuronales y que será usada por TensorFlow. Para ello, debemos registrarnos como desarrolladores en el Programa para Desarrolladores de Nvidia (*Nvidia Developers Program*), en la referencia [59]. Descargaremos e instalaremos cuDNN y comprobaremos que todo está correctamente instalado.

Finalmente, debemos instalar TensorFlow. TensorFlow puede ser instalado en varios lenguajes: Python, Java, Go o C. Como se explicó anteriormente, usaremos Python, así que debemos instalarlo si no lo hemos hecho con anterioridad. Puede ser descargado desde la referencia [60]. Debemos instalar una de las versiones 3.X de Python, preferiblemente la más actual.

TensorFlow puede instalarse de 2 formas, mediante *pip*, herramienta que proporciona Python para instalar paquetes; o mediante Anaconda, una distribución abierta y libre de Python para ciencia de datos y Machine Learning [61].

Nosotros usaremos *pip*, e instalaremos TensorFlow mediante el siguiente comando:

```
pip3 install --upgrade tensorflow-gpu
```

Tras este paso ya se encuentra todo instalado y sólo nos faltaría realizar un pequeño test en Python para comprobar que, efectivamente, todo funciona correctamente. Para este test podemos utilizar cualquiera de los ejemplos o tutoriales de la referencia [17].

REFERENCIAS

[1] D. N. Sleeps, «https://www.domo.com/learn/data-never-sleeps-5,» 2017.

[2] «Internet Live Stats,» [En línea]. Available: http://www.internetlivestats.com/total-number-of-websites/.

[3] E. C. A. Tepán, *Estudio de los principales tipos de redes neuronales y las herramientas para su aplicación,* Cuenca, 2013.

[4] «Introducción a Python,» [En línea]. Available: http://docs.python.org.ar/tutorial/3/real-index.html.

[5] «Comparación redes neuronales,» [En línea]. Available: https://en.wikipedia.org/wiki/Comparison_of_deep_learning_software.

[6] «Deeplearning4j,» [En línea]. Available: https://deeplearning4j.org/.

[7] «PyTorch,» [En línea]. Available: https://pytorch.org/.

[8] «Theano,» [En línea]. Available: http://deeplearning.net/software/theano/.

[9] «Matlab,» [En línea]. Available: https://es.mathworks.com/products/matlab.html.

[10] «Caffe 2,» [En línea]. Available: https://caffe2.ai/.

[11] «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[12] «Scikit-learn,» [En línea]. Available: http://scikit-learn.org/stable/index.html.

[13] D. Ocean. [En línea]. Available: https://www.digitalocean.com/community/tutorials/a-comparison-of-web-servers-for-python-based-web-applications.

[14] «Klen,» [En línea]. Available: http://klen.github.io/py-frameworks-bench/#results.

[15] «Neural Networks (YouTube),» [En línea]. Available: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi.

[16] «Redes Neuronales (YouTube),» [En línea]. Available: https://www.youtube.com/watch?v=jaEIv_E29sk.

[17] «Tutorial TensorFlow,» [En línea]. Available: https://www.tensorflow.org/tutorials/.

[18] «PlayGround TensorFlow,» [En línea]. Available: http://playground.tensorflow.org.

[19] N. McClure, TensorFlow machine learning cookbook, Birmingham, UK: Packt Publishing, 2017.

[20] M. R. Karim, TensorFlow: powerful predictive analytics with TensorFlow, Birmingham, UK: Packt Publishing, 2018.

[21] N. Shukla, Machine learning with TensorFlow, Shelter Island, NY: Manning Publications, 2018.

[22] DL4J, «Introduction to Deep Neural Networks,» [En línea]. Available: https://deeplearning4j.org/neuralnet-overview.

[23] Wikipedia, «Artificial neural network,» [En línea]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.

[24] J. A. E. S. Paul Covington, «Deep Neural Networks for YouTube Recommendations».

[25] «Tesla Autopilot,» [En línea]. Available: https://www.tesla.com/es_ES/autopilot?redirect=no.

[26] *Atmeh, Ghassan & Ranatunga, Isura & Popa, Dan & Subbarao, Kamesh & Lewis, Frank & Rowe, Patrick. (2014). Implementation of an adaptive, model free, learning controller on the Atlas robot. Proceedings of the American Control Conference. 2887-2892. 10.1109/.*

[27] «Youtube,» [En línea]. Available: https://www.youtube.com/watch?v=jAu1ZsTCA64.

[28] «OpenAI Blog,» [En línea]. Available: https://blog.openai.com/dota-2/.

[29] «TechEmergence,» [En línea]. Available: https://www.techemergence.com/everyday-examples-of-ai/.

[30] «Search Engine Land,» [En línea]. Available: https://searchengineland.com/faq-all-about-the-new-google-rankbrain-algorithm-234440.

[31] S. P. H. a. N. D. R J Frank, «Applications of Neural Networks to Telecommunications Systems,» University of Hertfordshire.

[32] T. Clarkson, «Applications of Neural Networks in Telecommunications,» King's College London.

[33] A. M. Bowen y H. G. Asensio, «Inteligencia artificial. Redes neuronales y aplicaciones,» [En línea]. Available: http://www.it.uc3m.es/jvillena/irc/practicas/10-11/06mem.pdf.

[34] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks.

[35] «Wikipedia Python,» [En línea]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[36] «Quora,» [En línea]. Available: https://www.quora.com/What-is-the-most-famous-software-written-in-Python.

[37] «GitHub,» [En línea]. Available: https://octoverse.github.com/.

[38] «Stackify,» [En línea]. Available: https://stackify.com/popular-programming-languages-2018/.

[39] «Wikipedia (TensorFLow),» [En línea]. Available: https://es.wikipedia.org/wiki/TensorFlow.

[40] «TensorFlow C++,» [En línea]. Available: https://stackoverflow.com/questions/35677724/tensorflow-why-was-python-the-chosen-language?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa.

[41] «Quora,» [En línea]. Available: https://www.quora.com/What-is-the-future-of-TensorFlow.

[42] «BottlePy,» [En línea]. Available: https://bottlepy.org/docs/dev/.

[43] J. P. Valls, Herramientas Matlab para la selección de entradas y predicción neuronal de valores en bolsa, Universidad de Sevilla.

[44] «Spotify Neural Networks,» [En línea]. Available: http://benanne.github.io/2014/08/05/spotify-cnns.html.

[45] «HackerNoon,» [En línea]. Available: https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75.

[46] P. Malhotra, «Quora,» [En línea]. Available: https://www.quora.com/What-are-encoder-decoder-models-in-recurrent-neural-networks.

[47] TensorProp. [En línea]. Available: https://www.tensorflow.org/api_docs/python/tf/train/RMSPropOptimizer.

[48] «Wikipedia Descenso Estocastico,» [En línea]. Available: https://en.wikipedia.org/wiki/Stochastic_gradient_descent#RMSProp.

[49] «Kaggle,» [En línea]. Available: https://www.kaggle.com/datasets?sortBy=hotness&group=public&page=1&pageSize=20&size=all&filetype=all&license=all.

[50] «UCI,» [En línea]. Available: http://archive.ics.uci.edu/ml/datasets.html.

[51] «Last.fm,» [En línea]. Available: https://www.last.fm/es/home.

[52] «Create Accunt Last.fm,» [En línea]. Available: https://www.last.fm/api/account/create.

[53] V. Bellini, «GitHub,» [En línea]. Available: https://vitobellini.github.io/posts/2018/01/03/how-to-build-a-recommender-system-in-tensorflow.html.

[54] «Medium,» [En línea]. Available: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0.

[55] «Angular,» [En línea]. Available: https://angular.io/.

[56] «Installing TensorFlow,» [En línea]. Available: https://www.tensorflow.org/install/install_windows.

[57] «Nvidia Installation TensorFlow,» [En línea]. Available: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/.

[58] «CUDA GPUs,» [En línea]. Available: https://developer.nvidia.com/cuda-gpus.

[59] C. A. Nvidia. [En línea]. Available: https://login.nvgs.nvidia.com/v1/create-account.

[60] «Download Python,» [En línea]. Available: https://www.python.org/downloads/.

[61] «Anaconda,» [En línea]. Available: https://es.wikipedia.org/wiki/Anaconda_(distribuci%C3%B3n_de_Python).

GLOSARIO