Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA (SG)
Grado en Ingeniería Informática de Servicios y Aplicaciones

Implementación del modelo de red neuronal RBM

Alumno: Javier Gómez Inés

Tutor(a): Aníbal Bregón Bregón
José Vicente Álvarez Bravo

Agradecimientos

Me gustaría agradecer a mis dos tutores, D. Aníbal Bregón Bregón y D. José Vicente Álvarez Bravo, por permitirme realizar este proyecto y así poder adquirir nuevos conocimientos sobre las redes neuronales y los algoritmos de aprendizaje descritos en este documento. También, agradecerles por el apoyo y guía proporcionados a lo largo de este tiempo de desarrollo.

También agradecer a los profesores del grado por estos años de enseñanza que me han permitido adquirir los conocimientos necesarios para iniciar mi carrera profesional.

Dar las gracias a mi familia, ya que sin su apoyo incondicional no creo que hubiera llegado hasta este momento.

Por último, dar las gracias a mis amigos, tanto aquellos que conozco desde hace más tiempo como los conocidos a lo largo de mis estudios en el grado, por el apoyo proporcionado cuando lo necesitaba, y por el tiempo y discusiones compartidas para entregar las prácticas.

Resumen

La inteligencia artificial ha alcanzado una especial relevancia en nuestra vida cotidiana desempeñando tareas que pueden ir desde proponer nuevos desafíos en tu juego favorito, responder nuestras dudas en relación a un producto concreto o incluso recomendar series similares a nuestra serie favorita. Por esta razón, se ha elegido el contexto de la **Inteligencia Artificial** para el desarrollo de este TFG.

En este proyecto se ha implementada un modelo de **red neuronal** que será capaz de identificar las características presentes en un conjunto de datos de entrenamiento y realizar inferencias sobre los datos de este conjunto.

Para llevar a cabo el proyecto, se ha decidido desarrollar un modelo de red neuronal conocida como RBM (**Restricted Boltzmann Machine**), capaz de identificar estas características por medio de un modelo sencillo de dos únicas capas y sin conexiones entre neuronas de la misma capa, simplificando así la complejidad del sistema, además utiliza un algoritmo de aprendizaje no supervisado, es decir, los datos que emplea no tienen definido un resultado esperado.

Palabras Clave: Inteligencia Artificial, Red Neuronal, Restricted Boltzmann Machine, Python, Tensorflow, Algoritmo no Supervisado

Abstract

Artificial Intelligence has achieved high relevance in our lives, performing daily tasks that may go from offering new challenges in your favourite videogame to answering our doubts about a concrete product, or even recommend new TV series similar to our favourite one. Because of this, **Artificial Intelligence** has been chosen to develop this Degree Final Project.

In this project, we have implemented a **neural network** model that will be able to identify features in a training dataset and make inferences about that dataset.

To carry out this project, we decided to develop a neural network model called RBM (**Restricted Boltzmann Machine**) able to identify those features using a simple model formed by two layers and without any connections between neurons of the same layer, simplifying the complexity of the system, it also uses an **unsupervised learning algorithm,** i.e. the data that it uses doesn't have a defined outcome.

Keywords: Artificial Intelligence, Neural Network, Restricted Boltzmann Machine, Python, Tensorflow, Unsupervised Learning Algorithm.

Índice de Contenido

Índice de Figuras

Índice de Ecuaciones

Índice de Tablas

1 INTRODUCCIÓN

La Inteligencia Artificial (IA) ha conseguido demostrar en los últimos años ser una potente herramienta de apoyo en un gran abanico de campos, desde sistemas de recomendación hasta proporcionando ayuda en algunas tareas u optimizando ciertos procesos dentro de las empresas.

Uno de los sectores en los que la IA ha proporcionado más beneficios es el sector de la salud. Dentro de este ámbito, la Inteligencia Artificial ha contribuido con la mejora de muchos procesos de prevención, diagnóstico y tratamiento de ciertas enfermedades. Uno de los proyectos que se están desarrollando en la actualidad es TrackAI, este proyecto surgido de la colaboración del DIVE-Medical y la empresa tecnológica Huawei consiste en el desarrollo de un software que permite detectar cierta predisposición a padecer futuras enfermedades visuales en pacientes pediátricos, por medio de un algoritmo de Inteligencia Artificial que permite la realización de análisis de los patrones de visión de estos pacientes. Éste solo es uno de los ejemplos en los que la IA ha proporcionado las herramientas necesarias que permiten a los expertos de los distintos campos mejorar ciertos procesos de su ámbito.

Dentro de la IA existen muchas ramas de estudio, de las cuales la más conocida es el Machine Learning (ML) o Aprendizaje Automático, este área de trabajo permite a los sistemas informáticos aprender a resolver problemas a partir de un conjunto de datos seleccionado. El ML presenta un gran abanico de aplicaciones que pueden permitir a las empresas mejorar muchos aspectos de sus actividades, algunos de estos son: Reconocimiento de imágenes, Natural Language Processing o Procesamiento del Lenguaje Natural, Chatbots, sistemas de recomendación, Clasificación y Clustering entre otros muchos.

Una de las principales ramas del ML, se centra en la creación de algoritmos de aprendizaje que simulen el funcionamiento del cerebro humano, este tipo de algoritmos se conocen como Redes neuronales. Al igual que en el cerebro humano, este tipo de redes se componen de neuronas, las cuáles se encargan de realizar una serie de cálculos empleando datos para de esta forma obtener una salida. La estructura de este tipo de modelos se divide en tres tipos de neuronas en función de la capa en la que se encuentren; pueden ser neuronas de la capa de entrada, las cuales representan los datos de entrada que empleará la red para realizar los cálculos, la capa de salida que representa el resultado calculado por la red o la capa oculta, que representa todas aquellas capas de neuronas situadas entre las dos anteriores capas y que realizan los cálculos para obtener la salida.

Cada modelo de red neuronal presenta una estructura distinta, la estructura que estos modelos presentan puede ser tan simple como la del Perceptrón, que está formado por una única capa con una neurona, o tener estructuras tan complejas como las de un modelo de Redes Neuronales Profundas o Deep Neural Network, como por ejemplo el modelo Deep Belief Network o red de Creencia Profunda. El Deep Learning es un área de trabajo de las Redes Neuronales que comprende aquellos modelos que presentan en su estructura más de una capa de neuronas ocultas.

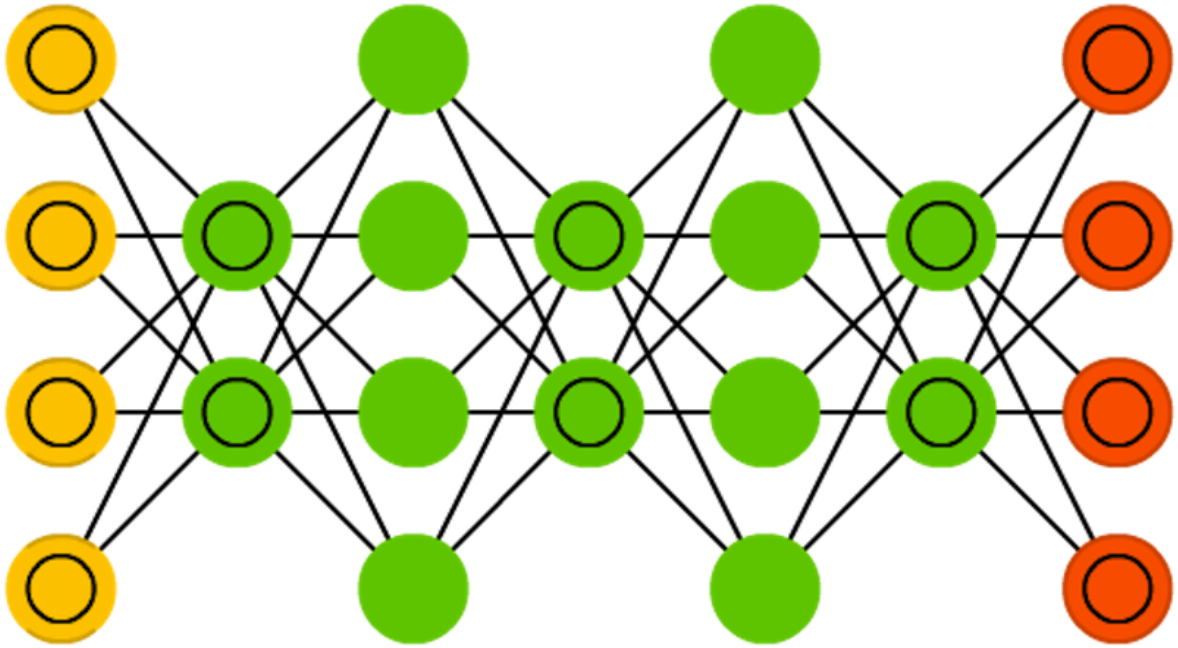


Ilustración 1.1: Estructura de una Deep Belief Network. Fuente: https://medium.com/@icecreamlabs/deep-belief-networks-all-you-need-to-know-68aa9a71cc53

Algunas de las principales aplicaciones tanto de los modelos de Redes Neuronales como de Deep Learning son: Clasificación, Compresión y Descompresión de datos, Reconocimiento de imágenes, Reconocimiento de patrones, Natural Language Processing o Procesamiento del Lenguaje Natural, Reducción de dimensionalidad, Filtrado Colaborativo, entre otros muchos.

Una vez conocidas las definiciones básicas de cada uno de los distintos áreas de trabajo descritos a lo largo de esta introducción, las relaciones entre estos pueden observarse claramente en la Ilustración 1.2.

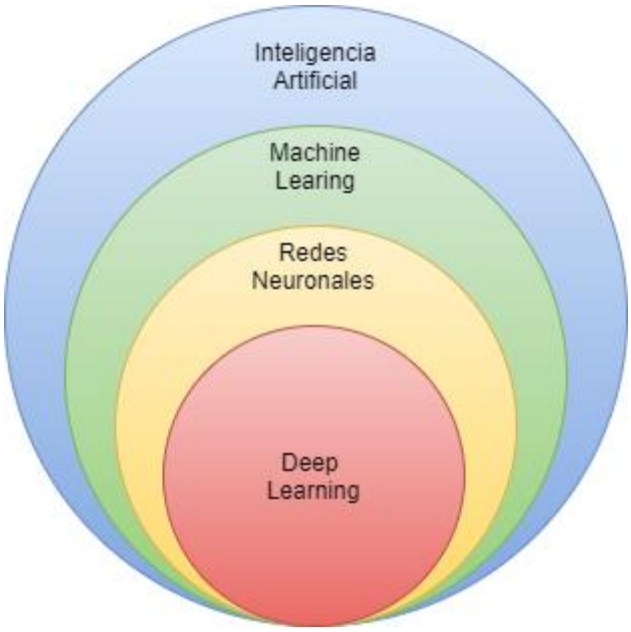


Ilustración 1.2: Relación área de la IA.

1.1 MOTIVACIÓN

Como se ha mencionado en el apartado anterior, la Inteligencia Artificial está tomando una gran relevancia tanto en nuestra vida cotidiana como en un gran abanico de campos de distintas disciplinas. Debido a esta gran transcendencia, han surgido muchas plataformas de desarrollo y herramientas creadas para facilitar el desarrollo de modelos de Machine Learning o Deep Learning.

Con el desarrollo de este proyecto se pretende adquirir los conocimientos básicos sobre el Machine Learning y sobre aquellas tecnologías y herramientas más comúnmente utilizadas dentro de esta área. Para llevarlo a cabo se ha decidido implementar un modelo de red neuronal capaz de identificar las estructuras latentes en un conjunto de datos, con el fin de que éste sea capaz de inferir en base a este conjunto de datos.

Para llevar a cabo la implementación del modelo anteriormente mencionado, se ha decidido desarrollar una implementación del modelo de red neuronal conocida como RBM (Restricted Boltzmann Machine), que permita detectar estas estructuras por medio de un modelo sencillo de dos únicas capas y sin conexiones entre las neuronas de una misma capa, simplificando así la complejidad del sistema. Además, se realizará un estudio y análisis del modelo empleado con el fin de determinar cómo influyen determinados parámetros como el número de registros del conjunto de datos empleado, el número de neuronas por cada capa de la red o la tasa de aprendizaje del modelo, concluyendo cuáles de ellos son más relevantes a la hora de mejorar la precisión del mismo.

1.2 OBJETIVOS

Dentro de este proyecto se han identificado una serie de objetivos, siendo el principal la realización del estudio y análisis del modelo de red neuronal conocido como Restricted Boltzmann Machine; además de éste, también presenta los siguientes objetivos:

1. Implementación de un modelo de la red neuronal conocida como Restricted Boltzmann Machine.
2. Estudio de las tecnologías y herramientas necesarias para el desarrollo y ejecución del modelo.

A su vez, cada uno de estos objetivos se pueden dividir en una serie de sub-objetivos. El primero de los objetivos comprende todas aquellas tareas que se requieran para la realización del estudio y análisis del modelo. Los sub-objetivos que lo componen son los siguientes:

ID Objetivo	Descripción	
OBJ-01	Realización del estudio y análisis del modelo de red neuronal conocido como Restricted Boltzmann Machine.	
	ID Sub-objetivo	Descripción
	OBJ-01.1	Realización de las pruebas pertinentes para la realización del estudio.
	OBJ-01.2	Análisis comparativo de las pruebas realizadas a fin de determinar el comportamiento del modelo en el proceso de aprendizaje.
	OBJ-01.3	Concluir qué factores son más determinantes en el comportamiento del modelo.

Tabla 1.1: Objetivo OBJ-01

El segundo objetivo, está formado por aquellas tareas necesarias para adquirir los conocimientos básicos necesarios para desarrollar el modelo de red neuronal y crear un entorno de ejecución de las pruebas necesarias para realizar el posterior análisis y estudio del modelo. Para ello se requiere la realización de los siguientes sub-objetivos:

ID Objetivo	Descripción	
OBJ-02	Investigación acerca de las tecnologías y herramientas necesarias para el desarrollo y ejecución del modelo.	
	ID Sub-objetivo	Descripción
	OBJ-02.1	Investigación acerca de las tecnologías y herramientas empleadas para el desarrollo del modelo.
	OBJ-02.2	Investigación acerca de las tecnologías y herramientas empleadas para la ejecución de los procesos de entrenamiento y test del modelo.

Tabla 1.2 Objetivo OBJ-02

Por último, el tercer objetivo comprende las tareas necesarias para construir un modelo de red neuronal RBM capaz de capturar la estructura latente de un conjunto de datos, para ello se requiere de la implementación de los siguientes sub-objetivos de los que se compone:

ID Objetivo	Descripción	
OBJ-03	Implementar un modelo de red neuronal RBM.	
	ID Sub-objetivo	Descripción
	OBJ-03.1	Implementación de los métodos necesarios para la realización de la fase de entrenamiento del modelo.
	OBJ-03.2	Implementación de los métodos necesarios para la realización de la fase de test del modelo.
	OBJ-03.3	Implementación de los métodos necesarios para la realización de inferencias sobre los datos del modelo.
	OBJ-03.4	Creación de un fichero con los resultados obtenidos en las fases de entrenamiento y test para la realización del estudio del modelo.

Tabla 1.3 Objetivo OBJ-03

1.3 ORGANIZACIÓN DEL DOCUMENTO

A lo largo de este apartado se detallará la estructura que presenta este documento, explicando brevemente en que se centrará cada uno de los capítulos de los que se componen el mismo.

- Capítulo 1: Introducción. A lo largo de este capítulo del documento se redactará una introducción del contexto sobre el cual se desarrolla el proyecto, incluyendo además la motivación de éste y la especificación de los objetivos que se pretenden alcanzar tras su finalización.
- Capítulo 2: Contexto del proyecto. Contiene la una breve introducción sobre el Machine Learning y el Deep Learning, y la explicación del origen y la evolución del modelo RBM implementado en este proyecto.
- Capítulo 3: Contiene las explicaciones de la metodología de trabajo y el marco de trabajo empleada en este proyecto.
- Capítulo 4: Gestión del Proyecto. Se especificarán la estimación y planificación temporal de este proyecto, detallando el listado de historias de usuario, el listado de tareas en que se dividen estas y su planificación a lo largo del tiempo de desarrollo del proyecto.
- Capítulo 5: Análisis. Contiene la fase de análisis del proyecto, que se compondrá de una breve descripción de los actores, la especificación de las limitaciones y restricciones.
- Capítulo 6: Modelado de los datos. Recoge la descripción de los conjuntos de datos empleados por el proyecto, así como los campos que se utilizarán a la hora de su desarrollo. También, contiene la explicación de todas aquellas transformaciones realizadas sobre los Raw Data con el fin de realizar un correcto procesado de estos datos para que puedan ser utilizados por el proyecto.
- Capítulo 7: Diseño. Describe la fase de diseño del desarrollo del proyecto. En esta fase se mostrarán el diagrama de clases que implementará el proyecto y las arquitecturas lógica y física que seguirá.
- Capítulo 8: Implementación. Recoge la descripción del modelo de red neuronal a desarrollar y cómo funcionan los algoritmos necesarios para el correcto funcionamiento del modelo. Además, se describirá como se ha implementado el proyecto y qué herramientas y tecnologías serán necesarias para su desarrollo.
- Capítulo 9: Pruebas. Contiene la batería de pruebas de caja negra realizadas sobre los principales procesos del modelo para comprobar su correcto funcionamiento.
- Capítulo 10: Estudio y análisis del modelo. Describe el estudio y análisis realizado sobre los resultados obtenidos tras la realización de la batería de pruebas de los procesos de entrenamiento y test del modelo con el fin de determinar que parámetros son más determinantes a la hora de lograr mejores resultados.
- Capítulo 11: Conclusiones y trabajo futuro. Recoge las conclusiones generadas tras la realización de este proyecto y la descripción de las distintas líneas de proyecto futuras que se pueden realizar en base a éste.
- Bibliografía y Webgrafía. Contiene la bibliografía y webgrafía utilizadas como referencia para desarrollar el proyecto y redactar este documento.

- Apéndice A: Manuales de Usuario. Contiene los manuales de usuario necesarios para realizar la correcta instalación y configuración de los entornos de ejecución del modelo.
- Apéndice B: Glosario. Recoge el glosario con un listado de términos empleados a lo largo de este documento.
- Apéndice C: Acrónimos. Recoge todos aquellos acrónimos empleados en este documento.

2 CONTEXTO DEL PROYECTO

A lo largo de este capítulo, se realiza una breve definición sobre el Machine Learning y el Deep Learning, introduciendo también algunos de los algoritmos que estos emplean como métodos de aprendizaje. Además, se detallará el origen del modelo de red neuronal RBM y como ha ido evolucionando con el paso del tiempo.

2.1 MACHINE LEARNING

El Machine Learning es un área de trabajo de la Inteligencia Artificial que emplea algoritmos para dotar a los sistemas de Inteligencia Artificial de la capacidad de aprender automáticamente del entorno y lograr realizar la toma de decisiones en base a ese aprendizaje. Este área de trabajo utiliza una serie de algoritmos, que a su vez emplean un conjunto de técnicas estadísticas, que permiten al sistema aprender iterativamente detectando los distintos patrones que presenta el conjunto de datos empleado para entrenar, estos algoritmos describen y mejoran dichos datos con el fin de obtener mejores resultados con los que realizar acciones en base a los patrones detectados.

Como se ha mencionado antes, el Machine Learning emplea una serie de algoritmos para aprender en base a un conjunto de datos. Dichos algoritmos se pueden clasificar en base a su propósito y las categorías principales que siguen. La clasificación de los algoritmos de machine learning es la siguiente:

1. Aprendizaje supervisado.
2. Aprendizaje no supervisado.
3. Aprendizaje semi-supervisado.
4. Aprendizaje por refuerzo.

2.1.1 Aprendizaje supervisado

Este tipo de algoritmos de aprendizaje se caracterizan por emplear como datos de entrenamiento conjuntos de datos etiquetados, es decir, el conjunto de datos contiene tanto los datos necesarios como el resultado esperado. De esta forma, por medio de los datos etiquetados, el sistema es capaz de encontrar las relaciones y las dependencias entre los datos y su resultado esperado logrando predecir los resultados de datos desconocidos.

Este tipo de algoritmos se emplean para resolver los siguientes problemas:

- Clasificación.
- Regresión.

Algunos de los algoritmos más empleados son el algoritmo de Vecinos Cercanos, Naive Bayes, Árboles de Decisión, Redes Neuronales, Regresión Linear y Máquinas de Vector de Soporte.

2.1.2 Aprendizaje no supervisado

En el aprendizaje no supervisado, el modelo se encarga de identificar los patrones presentes en el conjunto de datos de entrenamiento para posteriormente reaccionar en base a la presencia o ausencia de estos patrones en nuevos datos introducidos en el modelo. Este tipo de algoritmos emplean, a diferencia de los algoritmos supervisados, un conjunto de entrenamiento no etiquetado; es decir, estos datos no presentan un resultado esperado definido, sino que el sistema debe de ser capaz de identificar o determinar este resultado. Para lograr que el modelo pueda aprender a determinar los patrones ocultos en el conjunto de datos, es necesario exponerlo a unos volúmenes de datos de gran tamaño.

Este tipo de algoritmos se emplean en los siguientes problemas:

- Clustering.
- Reducción de dimensionalidad.

Algunos de los algoritmos empleados son Clustering, K-means y Reglas de asociación.

2.1.3 Aprendizaje semi-supervisado

Este tipo de algoritmos surgen como evolución de los algoritmos supervisados empleados en modelos de clasificación. Uno de los principales problemas que surgen al entrenar estos modelos es la necesidad de emplear algoritmos supervisados para la fase de entrenamiento, puesto que este tipo de modelos requieren el uso de conjuntos de datos etiquetados para esta fase. Este tipo de datos son mucho más complicados de obtener que los conjuntos de datos no etiquetados, ya que necesitan de una persona que anote el resultado a cada registro del conjunto. De esta forma, al emplear algoritmos semi-supervisados, los cuales emplean como conjunto de entrenamiento tanto datos etiquetados como no etiquetados, se soluciona el problema de obtención del conjunto de datos, ya que, al combinar el uso de un gran conjunto de datos no etiquetados junto con datos etiquetados, se reduce la cantidad de participación humana necesaria.

Algunos de los algoritmos empleados son:

- Algoritmo esperanza-maximización (EM) con modelos generativos de mezcla (Generative Mixture Models and EM algorithm).
- Algoritmo de entrenamiento por si solo (Self-training algorithm).
- Algoritmo de co-entrenamiento (Co-training algorithm).

2.1.4 Aprendizaje por refuerzo

El aprendizaje por refuerzo consiste en determinar un proceso de aprendizaje estricto, para ello se emplean una serie de acciones, parámetros y valores finales. En este método, el algoritmo (también llamado agente) observa el entorno para realizar acciones con el fin de maximizar la recompensa o bien disminuir el riesgo. Este tipo de aprendizaje permite al agente aprender continuamente del entorno de forma iterativa mientras identifica todos los estados posibles.

Algunos de los algoritmos empleados son:

- Q-Learning.
- Diferencia temporal (Temporal Difference).
- Redes neuronales adversarias profundas.

Este tipo de algoritmos de aprendizaje se emplean en videojuegos basados en juegos de mesa (AlphaGo, AlphaZero), manos robóticas y piloto automático de coches.

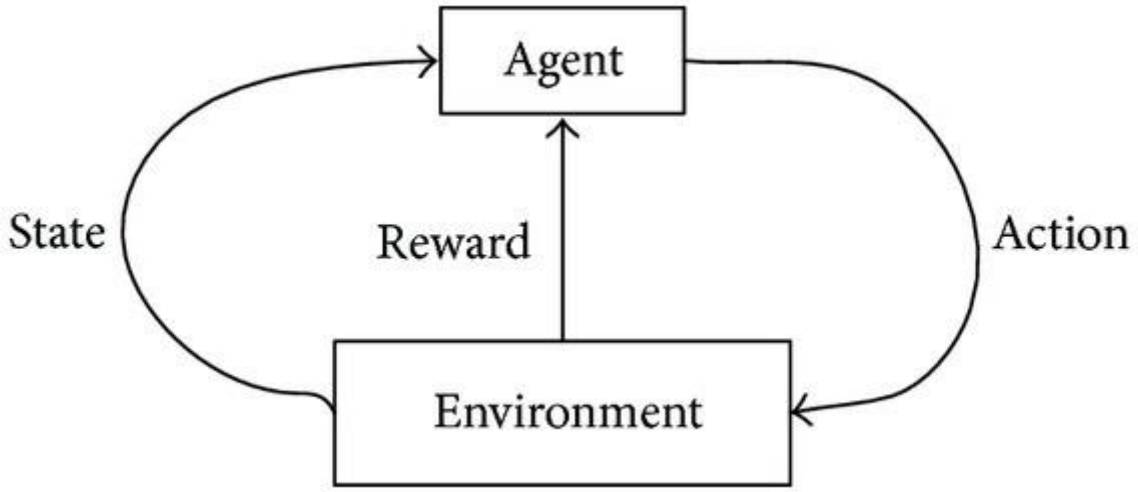


Ilustración 2.1: Aprendizaje por Refuerzo.
Fuente https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

2.1.5 Aplicaciones del Machine Learning

La forma más comúnmente empleada para clasificar los distintos algoritmos de Machine Learning se basa en el conjunto de datos que estos empleen para aprender, el explicado en el apartado anterior. Además de este método también se pueden clasificar estos modelos en función de la aplicación que desempeñen. Algunas de las principales aplicaciones de ML son las siguientes:

2.1.5.1 Clasificación

La Clasificación dentro del ML se encarga de atribuir una serie de clases a un conjunto de datos, ya sea estructurado o no. Este tipo de modelos predictivos tiene como tarea aproximar la función de mapeo de una serie de valores de entrada a unos valores de salida discretas, es decir, se encargan de predecir dentro de que categoría se sitúan los nuevos datos que llegan al modelo.

Este tipo de modelo pertenece a los algoritmos de aprendizaje supervisado. Existen dos métodos de aprendizajes dentro de la clasificación:

- Lazy learners o Estudiantes perezosos: este tipo de métodos simplemente almacenan los datos de entrenamiento hasta que aparezca un dato de la fase de test, de esta forma clasifica utilizando los datos más relacionados de los datos de entrenamiento. Este método presenta un mayor tiempo de predicción, pero un menor tiempo de entrenamiento. Dentro de este método de aprendizaje se sitúan los algoritmos: K-vecinos cercanos y Razonamiento basado en casos.
- Eager learners o Estudiantes ávidos de aprender: por el contrario, por medio de este método, se construye un modelo de clasificación basado en los datos de entrenamiento antes de recibir los datos de la fase de test. Debido a que tiene que obtener una hipótesis que permita clasificar todo el conjunto, presenta un mayor tiempo de entrenamiento,

pero un menor tiempo de predicción. Dentro de este método de aprendizaje se sitúan los algoritmos: Árboles de decisión, Naive Bayes y Redes neuronales.

2.1.5.2 Regresión

Esta aplicación del ML se encarga de modelizar un valor objetivo basado en predictores independientes, es decir, predecir un valor continuo, por ejemplo, predecir el valor de una casa en función de las características de esta.

Existen muchas técnicas de regresiones, entre las que destacan:

- Regresión lineal.
- Regresión polinómica.
- Regresión de vectores de apoyo.

Regresión lineal

Esta técnica de regresión es la más común e interesante, predice un valor de la variable objetivo (Y) basándose en un valor de entrada (X), es decir, estos dos valores presentan una relación linear entre ellos, siguiendo una ecuación como la siguiente:

$$Y = a + bX$$

Ecuación 2.1: Función regresión lineal.

El proceso de aprendizaje de esta técnica se centra en atribuir un valor para los coeficientes a y b de la Ecuación 2.1, obteniendo de esta forma la mejor línea de regresión que se ajusta a estos datos y que además minimice el valor de la función coste empleada para medir el error.

Regresión polinómica

Esta técnica se encarga de transformar las características o valores de entrada (X) en características polinómicas de un grado determinado y para después aplicar regresión lineal sobre ellas. Este tipo de técnicas transforma la Ecuación 2.1 en la siguiente ecuación:

$$Y = a + bX + cX^2$$

Ecuación 2.2: Función regresión pilinómica de grado 2.

El grado de la función de la Ecuación 2.2 varía según el grado de características polinómicas que se desee. Si se aumenta el valor del grado a un valor muy alto, la función se reajusta ya que aprende también del ruido de los datos.

Al igual que en la Regresión lineal el objetivo de la fase de entrenamiento es atribuir un valor a los coeficientes de la Ecuación 2.2, para lograr obtener una línea de regresión que represente los datos y que minimice el valor de la función coste.

Regresión de vectores de apoyo

En esta técnica de regresión, el modelo se encarga de identificar un hiperplano con un margen máximo tal que el máximo número de puntos se encuentre dentro de ese margen. Al contrario que en las regresiones anteriores que trataban de minimizar el error, en esta técnica se pretende ajustar el error dentro de un determinado umbral.

2.1.5.3 Extracción de conocimiento

La Extracción de conocimiento o Knowledge Extraction se centra en obtener las propiedades o estructuras latentes de un conjunto de datos no estructurado. Este tipo de aplicaciones se utiliza por ejemplo para generar comentarios o títulos sobre imágenes.

2.1.5.4 Organización de catálogos

Este tipo de modelos de ML proporcionan a los usuarios una serie de resultados basándose en una cadena de caracteres, siendo el caso de los sistemas basados en recomendaciones. Estos modelos se encargan de proporcionar al usuario una serie de elementos relacionados basándose en el elemento sobre el que mostró interés o bien proporcionando las herramientas necesarias para que realice la búsqueda.

2.1.5.5 Modelos generativos

Los modelos generativos son capaces de generar datos en base a unos datos de entrada introducidos. Estos modelos son idóneos para tareas como la traducción de documentos, donde los resultados son muy variados.

2.2 REDES NEURONALES

Las redes neuronales son una serie de algoritmos de ML que pretenden simular el aprendizaje del cerebro humano. La unidad de procesamiento básica en este tipo de modelos se conoce como neurona, que se compone de los siguientes elementos: los pesos (w) y los datos de entrada (x) que simulan a las dendritas de las neuronas humanas, el sesgo (b), la función sumatoria para calcular la salida generada por los datos de entrada ('$\sum$') y una función de activación que simula la activación de la neurona ($f(\cdot)$).

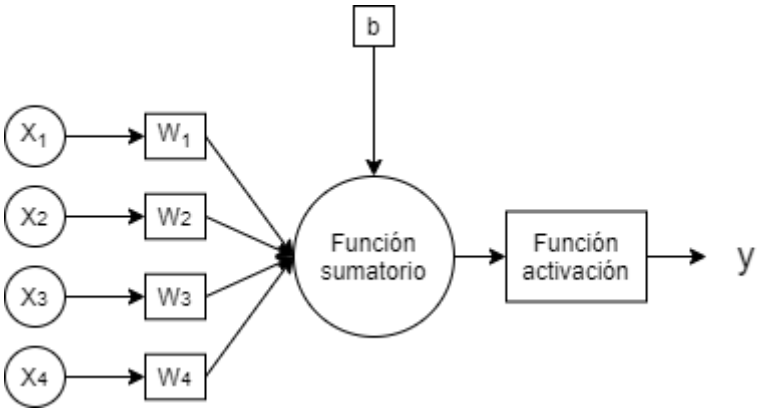


Ilustración 2.2: Estructura de un perceptrón.

Por lo tanto, siguiendo la estructura de la imagen anterior, obtenemos la siguiente ecuación para calcular la salida de la neurona a partir de cuatro datos de entrada:

$$y = f\left(\sum_{i=1}^{n=4} x_i w_i + b\right)$$

Ecuación 2.3: Cálculo salida de un perceptrón.

Las funciones de activación son un factor muy importante dentro de las redes neuronales, ya que se encargan de determinar el valor de salida de cada una de las neuronas. Existen tres tipos principales de funciones de activación:

- Función de activación binaria: esta función está basada en el umbral. De esta forma la función se activará cuando el valor devuelto por la función sumatorio alcance el valor de umbral y mientras tanto esta no se activa.

- Función de activación lineal: esta función sigue una ecuación del tipo $y = Ax$. Calcula el valor de salida en base al valor obtenido de la función sumatorio, siguiendo una función lineal.
- Función de activación no lineal: los modelos que emplean este tipo de funciones de activación siguen funciones no lineales. Este tipo de funciones permiten al modelo crear un complejo mapeo entre los datos de entrada y los datos de salida. Las principales funciones empleadas son las siguientes:
    - Sigmoide o logística.
    - Tangente Hiperbólica o TanH.
    - ReLU o Rectified Linear Unit (Unidad lineal rectificada).
    - Leaky ReLU o ReLU con fugas.
    - Parametric ReLU o ReLU Paramétrica.
    - Softmax.
    - Swish.

El modelo de red neuronal más simple que se puede encontrar es el Perceptrón. Este modelo se compone de una única neurona que utiliza como datos de entrada variables binarias, como se puede observar en la Ilustración 2.2. Debido a su simplicidad, este tipo de modelos no pueden representar problemas complejos (únicamente pueden representar problemas linealmente separables). Por este motivo, se han empleado modelos de redes neuronales multicapa, que combinan varias neuronas para aumentar la capacidad expresiva de la red.

Los modelos de redes neuronales multicapa se caracterizan por presentar los siguientes elementos en su estructura:

- Capa de entrada: contiene los valores de los datos de entrada.
- Capas de neuronas ocultas: se compone por una o más capas de neuronas que se unen a la capa de entrada y a la capa de salida. La estructura de esta capa de neuronas y el número de capas de la red dependen del modelo de red neuronal.
- Capa de salida: contiene los valores de los datos de salida obtenidos a partir de los datos de entrada.

2.3 DEEP LEARNING

El Deep Learning es un área de trabajo del Machine Learning que se centra en la construcción de redes neuronales capaces de imitar la estructura y funcionamiento del cerebro humano. El tipo de redes neuronales construidas por este área de trabajo se caracterizan por disponer de varias capas ocultas, formando cada una un módulo simple no lineal que se encarga de transformar las entradas (comenzando con los datos del conjunto de entrenamiento) en una representación más abstracta de las mismas, logrando aprender funciones muy complejas por medio de la unión de un número suficientes de estos módulos.

Al igual que el Machine Learning, los algoritmos de aprendizaje más comúnmente empleados son los supervisados, siendo concretamente el algoritmo del Gradiente Descendente Estocástico (Stochastic Gradient Descent, SGD) el más empleado en la práctica. Este algoritmo consiste en calcular las salidas y errores correspondientes a una serie de datos de entrada, para luego calcular la media del gradiente de los datos de entrada y posteriormente ajustar los valores de los pesos de cada neurona en función de los resultados obtenidos de cada una de las medias, repitiendo este proceso sobre bloques pequeños de datos de entrada hasta que el promedio de la función objetivo deje de disminuir.

Además del anterior algoritmo de aprendizaje, el Deep Learning utiliza otros muchos algoritmos de aprendizaje tales como el algoritmo Backpropagation o el Learning Rate Decay.

El método de aprendizaje Backpropagation consta de tres procesos. En el primer proceso se propaga el valor de los datos de entrada a través de la red neuronal realizando una propagación hacia adelante con el fin de calcular el resultado de estos valores de entrada. Durante el segundo proceso, el sistema calcula el error entre el valor obtenido y el valor esperado correspondiente esos datos de entrada por medio de una función de coste. Por último, se trasmite el error obtenido a través de la red neuronal, pero esta vez en sentido contrario, actualizando los valores de los parámetros de la red, es decir, pesos y bias.

El método Learning Rate Decay se centra en ajustar la tasa de aprendizaje o Learning Rate en función de unos métodos predefinidos. Existen varios métodos para ajustar esta tasa, algunos de los cuales son reducción basada en el tiempo, reducción escalonada o reducción exponencial. Todos estos métodos de ajuste de la tasa de aprendizaje del modelo suelen complementar al algoritmo de aprendizaje principal de la red neuronal

El Deep Learning está logrando grandes avances en aquellos problemas en que al Machine Learning le ha costado o no ha conseguido resolver. Los mejores resultados que se han obtenido han sido en el descubrimiento de estructuras latentes en conjuntos de datos de alta dimensión, pudiéndose aplicar en un gran conjunto de dominios en ciencia, negocios y gobierno.

2.4 RESTRICTED BOLTZMANN MACHINE

Este tipo de modelo de redes neuronales no supervisadas emplean la noción de energía para modelar aplicaciones como reconstrucción de datos, reducción de dimensionalidad o filtrado colaborativo.

Las redes neuronales feedforward se caracterizan por ser modelos dirigidos en los que la información pasa a través de las distintas capas, desde la capa de entrada pasando por las capas ocultas y terminando en la capa de salida. Por el contrario, los modelos RBM son modelos no dirigidos que emplean los datos de entrada para generar estados probabilísticos con los que lograr identificar las características latentes en el conjunto de datos.

El modelo RBM ha evolucionado a partir de las redes de Hopfield. Este tipo de red neuronal crea un modelo determinista de las relaciones entre los diferentes atributos (inputs) por medio de los pesos entre los diferentes nodos. Posteriormente, las redes de Hopfield evolucionaron a las máquinas de Boltzmann que emplean estados probabilísticos para representar distribuciones de Bernoulli a partir de una serie de atributos binarios. Estas redes, se componen de dos tipos de capas neuronales: una visible y otra oculta. Las neuronas visibles representan los atributos de entrada de la red, mientras que las ocultas representan las variables latentes. A partir de las máquinas de Boltzmann se desarrollaron las máquinas restringidas de Boltzmann.

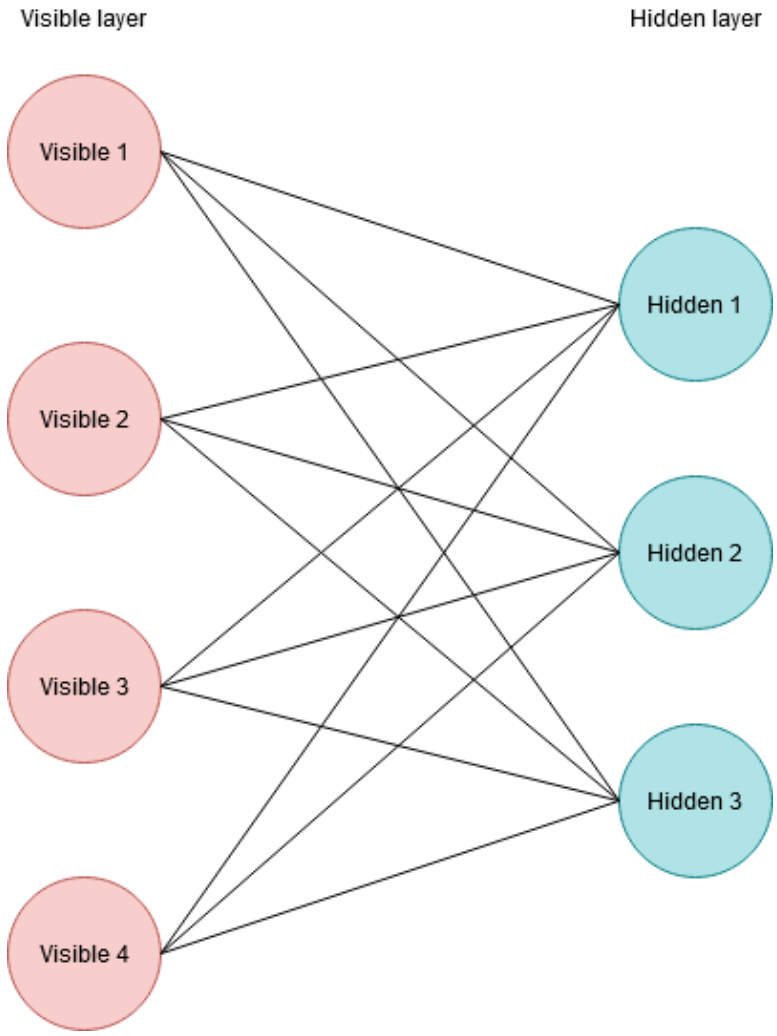


Ilustración 2.3: Estructura RBM

2.4.1 Redes de Hopfield

Este modelo de redes neuronales fueron propuestas en 1982 como un modelo para almacenar información. Este modelo de red neuronal está formado por una única capa de neuronas, que capturan valores binarios, y estos a su vez representan los valores binarios de los datos de entrenamiento. Este tipo de red neuronal permite crear un modelo determinista de las relaciones entre los distintos atributos por medio de la ponderación de pesos entre las distintas neuronas o nodos.

Este modelo presenta una única capa de neuronas donde cada una de ellas está conectada con las otras. Cada conexión tiene asociada un peso, representado como w_{ij} (siendo i y j cada una de las neuronas conectadas y siendo $w_{ij} = w_{ji}$), además, cada una de estas neuronas tiene un

bias asociado (b_i). Cada una de las neuronas se asocia con un estado, que tiene un valor binario {0,1} o en algunos casos {-1,+1}; estos estados se corresponden con la dimensión del conjunto de datos. Por tanto, si queremos almacenar un número '*d*' como dimensión, necesitaremos '*d*' nodos, correspondiendo así el nodo '*i*' con el bit '*i*' de los datos.

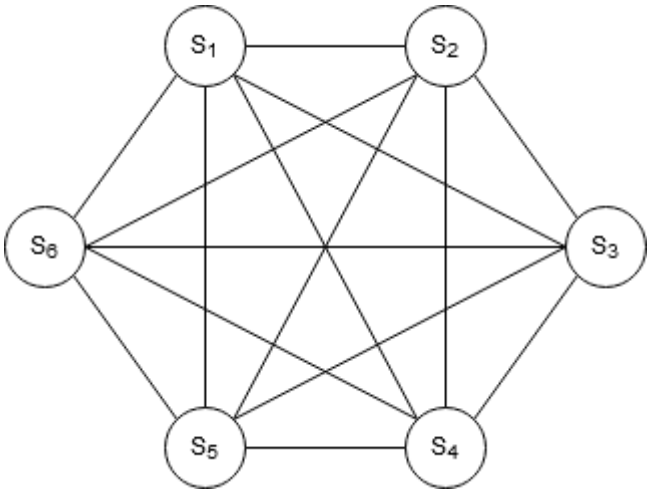


Ilustración 2.4: Estructura Red Hopfield

Este tipo de redes emplean un modelo de optimización para actualizar el valor de los pesos, logrando así que los pesos sean capaces de identificar las relaciones positivas y negativas entre las distintas neuronas de la red. De esta forma, los valores de los pesos positivos muy elevados representan nodos con estados similares y aquellos valores negativos muy elevados representan estados diferentes. La función objetiva de este modelo de red neuronal se denomina función de energía y es análoga a la función de coste de otros modelos de red neuronal. En la fase de entrenamiento del modelo, se pretende ajustar el valor de los pesos de forma que se minimice el valor de la función de energía, siempre que el valor de los atributos de entrenamiento sea binario.

La función de energía de las redes de Hopfield, se definen por medio de la siguiente ecuación:

$$E = -\sum_{i} b_i s_i - \sum_{i,j:i<j} w_{ij} s_i s_j$$

Ecuación 2.4: Energía Hopfield.

2.4.1.1 ¿Cómo se logra alcanzar la configuración de estados óptima en una red Hopfield entrenada?

Como se ha mencionado en el apartado anterior, la fase de entrenamiento de las redes de Hopfield pretende minimizar la función de energía de la red, es decir, alcanzar el equilibrio energético del modelo. Cada uno de los valores mínimos de esta función se corresponde con cada uno de los valores memorizados del conjunto de datos de entrenamiento o con un valor que representa un conjunto de estos datos. Para llevar a cabo esta fase de entrenamiento se emplea un algoritmo que permita encontrar un mínimo local, a través de la definición de una combinación de estados que representan cada una de las neuronas, tal que el cambio de uno de los valores de los estados no reduce el valor de la función de energía.

La búsqueda de la configuración óptima de los estados de la red, ayuda al modelo a capturar las memorias asociativas que inherentemente tiende a aprender, porque dado un conjunto de

datos de entrada, el modelo intercambiará el valor de los bits de entrada con el fin de mejorar la función de energía hasta encontrar una configuración que no mejore dicha función. El mínimo local obtenido (combinación final de estados) a menudo difiere en pocos bits del conjunto de estados inicial, por lo tanto, memoriza un patrón estrechamente relacionado; es decir, el mínimo local pertenece a menudo al conjunto de datos de entrenamiento.

Para logra alcanzar el próximo mínimo local o incluso el mínimo global se puede emplear la regla de actualización de umbral, de esta forma comparando la energía de los casos de los estados con valor 1 y los de valor 0 en la Ecuación 2.4, obtenemos la siguiente ecuación:

$$\Delta E_i = E_{s_i=0} - E_{s_i=1} = b_i + \sum_{j:j\neq i} w_{ij}s_j$$

Ecuación 2.5: Incremento energético.

La diferencia energética (ΔE_i) debe tener un valor mayor de 0 para realizar el cambio del estado s_i de 0 a 1. Obteniendo así la siguiente regla de actualización:

$$s_i = \begin{cases} 1. \quad Si \quad b_i + \sum_{j:j\neq i} w_{ij}s_j \geq 0 \\ \\ 0 \end{cases}$$

Ecuación 2.6: Regla de actualización energía Hopfield

De esta forma empleando la regla de actualización anterior de forma iterativa con cada uno de los estados s_i, se puede alcanzar un mínimo local, aunque se establezcan los valores de los pesos y bias de la red previamente.

2.4.1.2 Entrenamiento de las redes de Hopfield

La fase de entrenamiento de las redes de Hopfield emplea como método más común la "*regla de aprendizaje de Hebbian".* De acuerdo con esta regla, la sinapsis entre dos neuronas está estrechamente relacionada con el valor de la salida, es decir, la sinapsis entre ellas se refuerza cuando la salida de cada una está estrechamente correlacionada con la otra.

Siguiendo la regla de Hebbian, se obtiene que el valor de los pesos de la red viene dado por la siguiente ecuación:

$$w_{ij} = 4\frac{\sum_{k=1}^{n} (x_{ki} - 0{,}5) \cdot \left(x_{kj} - 0{,}5\right)}{n}$$

Ecuación 2.7: Cálculo de los pesos según la Regla de Hebbian.

donde $x_{ki}, x_{kj} \in \{0,1\}$ y representan los valores de los estados de los bit i y j en la iteración k del proceso de entrenamiento, n representa el número de iteraciones del proceso de entrenamiento. Por lo tanto siguiendo la regla de Hebbian, si los bit i y j están correlacionados el valor de $(x_{ki} - 0{,}5) \cdot \left(x_{kj} - 0{,}5\right)$ debe ser positivo y por tanto el valor de los pesos entre ambos bit también debe ser positivo, y negativo si no presentan esta correlación.

Por otro lado, también se puede aplicar esta regla sin normalizar el denominador, puesto que, en la práctica se puede implementar un algoritmo de aprendizaje incremental para actualizaciones de registros específicos, pudiendo realizar estas actualizaciones únicamente sobre el registro de entrenamiento *k*. De esta forma, se obtiene la siguiente ecuación:

$$w_{ij} \Leftarrow w_{ij} + 4\,(x_{ki} - 0{,}5) \cdot \left(x_{kj} - 0{,}5\right) \forall i, j$$

Ecuación 2.8: Actualización de los pesos en las redes de Hopfield (I).

El parámetro del bias puede actualizarse considerando que únicamente el estado ficticio está activado y el bias representa el peso entre el estado ficticio y el estado *i*:

$$b_i \Leftarrow b_i + 2(x_{ki} - 0{,}5)\ \forall i$$

Ecuación 2.9: Actualización de los bias en las redes de Hopfield (I).

Por último, en el caso de que los valores de los registros del conjunto de datos pertenezcan al conjunto $\{-1, +1\}$, las ecuaciones anteriores deben ser modificadas de la siguiente forma:

$$w_{ij} \Leftarrow w_{ij} + x_{ki} x_{kj}\ \forall i, j$$

Ecuación 2.10: Actualización de los pesos en las redes de Hopfield (II).

$$b_i \Leftarrow b_i + x_{ki}\ \forall i$$

Ecuación 2.11: Actualización de los bias en las redes de Hopfield (II).

Además de este método de aprendizaje, se pueden utilizar otros métodos como el Storkey learning rule o regla de aprendizaje de Storkey. Esta regla de aprendizaje de Storkey, también conocida como regla de aprendizaje pseudo-inversa, proporciona al modelo una mayor capacidad frente a la proporcionada por la regla de Hebbian, aunque se pierde el factor de aprendizaje incremental, inmediato y local de esta última. Por este motivo, la regla de Hebbian es la más comúnmente empleada, aunque en algunos casos puede ser conveniente emplear otras reglas que permitan obtener una mayor capacidad al modelo.

2.4.1.3 Mejorando las redes de Hopfield

Aunque no es una práctica estándar, se puede añadir una capa de neuronas ocultas a las redes de Hopfield, con el fin de mejorar su capacidad para capturar la estructura latente dentro de un conjunto de datos. De esta forma, los pesos entre dichas capas representan la relación entre los datos de entrenamiento y la estructura latente. En algunos casos, es posible representar los datos con un número reducido de neuronas ocultas.

Las neuronas de la capa oculta proporcionan una representación de los datos de entrenamiento en términos de '*m*' bits, siendo '*m*' el número de neuronas que componen la capa oculta; siendo esta una versión comprimida de tamaño '*m*' de estos datos. Por lo tanto, por medio de la representación de las neuronas ocultas se pueden obtener las aproximaciones a los patrones que representan al conjunto de datos. De esta forma, si añadimos una capa de neuronas ocultas y permitimos que los estados puedan adquirir valores probabilísticos, obtenemos un nuevo modelo de red neuronal conocido como Máquina de Boltzmann.

2.4.2 Máquinas de Boltzmann

Como se ha mencionado antes, este tipo de redes neuronales evolucionaron a partir de las redes de Hopfield; siendo así una generalización probabilística de dichas redes.

Las redes de Hopfield establecen el valor de cada uno de sus estados como 0 o 1, en función del valor de la energía de dicho estado, si tiene valor negativo es 0 y si es positivo 1.

Por otro lado, las máquinas de Boltzmann asignan una distribución probabilística al estado en función del valor de la energía, por medio de la aplicación de una función sigmoide a dicho valor de la energía:

$$P(s_i = 1 | s_1, \dots, s_{i-1}, s_{i+1}, s_q) = \frac{1}{1 + \exp(-\Delta E_i)}$$

Ecuación 2.12: Función sigmoide de la energía de Hopfield

Esta distribución es definida sobre varias configuraciones de estados, en función de una configuración de pesos y bias. La energía de una configuración en particular $\bar{s} = (\bar{v}, \bar{h})$ se denota como $E(\bar{s}) = E([\bar{v}, \bar{h}])$ y de forma similar a la de una red de Hopfield:

$$E(\bar{s}) = -\sum_i b_i s_i - \sum_{i,j:i<j} w_{ij} s_i s_j$$

Ecuación 2.13: Enegía de una configuración (s).

Sin embargo, estas configuraciones solo se conocen de forma probabilística en el caso de las máquinas de Boltzmann. La distribución condicional de la Ecuación 2.12 se deriva de una definición más fundamental de la probabilidad incondicional $P(\bar{s})$ de un configuración de $\bar{s}$:

$$P(\bar{s}) \propto exp(-E(\bar{s})) = \frac{1}{Z} exp(-E(\bar{s}))$$

Ecuación 2.14: Probabilidad incondicional de una configuración (s) .

La normalización a través del factor Z garantiza que todas las probabilidades de todas las configuraciones sumen 1:

$$Z = \sum_{\bar{s}} exp(-E(\bar{s}))$$

Ecuación 2.15: Normalización factor Z

El factor Z, también conocida como función de partición, es una función muy difícil de calcular explícitamente porque se corresponde con un número exponencial de términos que corresponden a todas las configuraciones de estados posibles. Por lo tanto, el cálculo de $P(\bar{s}) = P(\bar{v}, \bar{h})$ no es posible. A pesar de esto, si se puede calcular muchos tipos de probabilidades condicionales como $P(\bar{v}|\bar{h})$ ya que son proporciones y el factor de normalización se anula en el cálculo, obteniendo de esta forma la Ecuación 2.12.

Uno de los beneficios de establecer los estados de forma probabilística consiste en la posibilidad de crear nuevos valores para los datos a partir de estos estados, aunque debido a la dependencia entre estados visibles y estados ocultos dificulta en gran medida la generación de los nuevos datos. En cambio, otros modelos permiten esa generación al no presentar dicha dependencia entre los dos tipos de estados pudiéndose realizar primero un proceso secuencial de muestreo de los estados ocultos y después generar datos visibles de éstos.

2.4.2.1 Estructura del modelo

Como se ha mencionado en la sección anterior, este modelo de red neuronal es una generalización probabilística de las redes de Hopfield, surgidas al añadir a esta red una segunda capa compuesta por neuronas ocultas.

Al igual que en el modelo de Hopfield, todas las neuronas que componen la red estarán interconectadas entre ellas, indiferentemente de la capa en la que se encuentren, generando así la siguiente estructura del modelo:

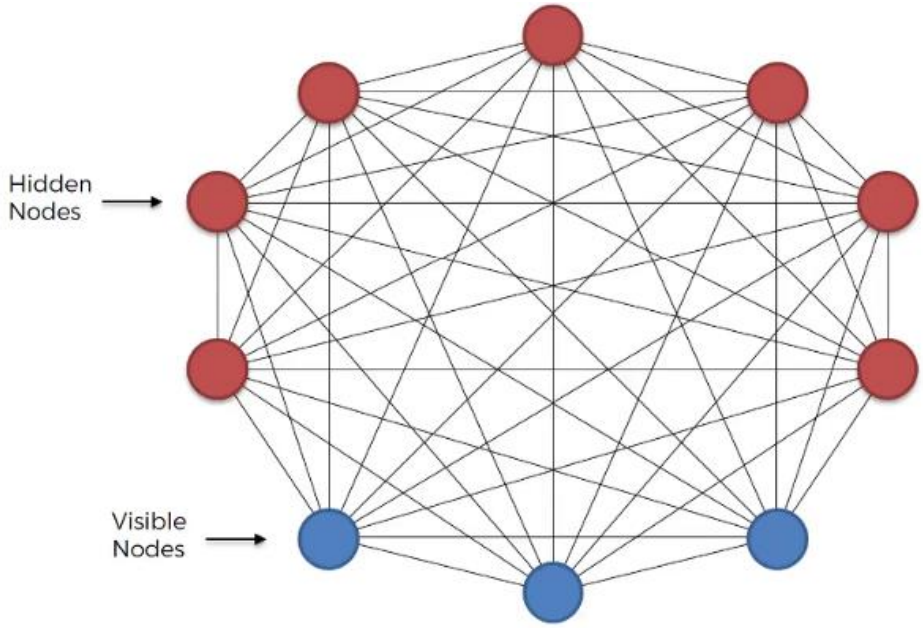


Ilustración 2.5: Estructura Máquina de Boltzmann.
Fuente: https://medium.com/datadriveninvestor/an-intuitive-introduction-of-boltzmann-machine-8ec54980d789

2.4.2.2 Generación de datos

Debido a la alta complejidad que presentan las máquinas de Boltzmann para generar datos, provocada por la dependencia circular entre los estados visibles y ocultos, es necesario, definir un proceso iterativo que permita generar datos de muestra y solucione el problema de estas dependencias. La máquina de Boltzmann recorre de forma iterativa los distintos estados por medio de una distribución condicional generada a partir de los estados de la iteración anterior; comenzando en la primera iteración con un conjunto de estados aleatorio, obteniendo las probabilidades condicionales de estos datos iniciales y generando a partir de estas probabilidades los nuevos valores de los estados que se usaran como valores en la siguiente iteración. La ejecución de este proceso durante varias iteraciones permite obtener muestras de los datos generadas aleatoriamente a partir de los valores de los estados visibles. Este proceso, que se realiza el número de veces necesario hasta alcanzar el equilibrio térmico de la red, se conoce como Gibb Sampling.

Una vez alcanzado el equilibrio térmico de la red, los estados ocultos representan la estructura latente de los datos que ha sido capturada por el modelo, donde las dimensiones en los datos generados estarán correlacionadas entre ellas por medio de los pesos de los estados. De esta forma, aquellos pesos que presenten valores grandes representarán nodos altamente correlacionados. Por lo tanto, si el modelo ha sido entrenado correctamente generará datos que contengan estas correlaciones, aunque los estados se inicialicen aleatoriamente.

2.4.2.3 Proceso de aprendizaje

El proceso de entrenamiento de este modelo de red neuronal consiste en obtener una configuración de los valores de los pesos y bias que permita maximizar la probabilidad de registro del conjunto de datos de entrenamiento empleado. La posibilidad de registro de los estados individuales se calcula empleando la siguiente ecuación:

$$\log[P(\bar{s})] = -E(\bar{s}) - \log(Z)$$

Ecuación 2.16: Logaritmo de la Probabilidad incondicional de una configuración (s).

Por lo tanto, el cálculo de $\frac{\partial \log \log[\mathrm{P}(\bar{\mathrm{s}})]}{\partial \mathrm{w_{ij}}}$ requiere del cálculo de la derivada negativa de la energía, aunque presenta un término adicional que implica a la función de partición. La función de partición es lineal en el peso w_{ij} con coeficiente $-s_i s_j$. Por lo tanto, la derivada parcial de la energía con respecto al peso w_{ij} es $-s_i s_j$. Obteniendo de esta forma la siguiente ecuación:

$$\frac{\partial \log \log[\mathrm{P}(\bar{\mathrm{s}})]}{\partial \mathrm{w_{ij}}} = \langle s_{i,} s_j \rangle_{data} - \langle s_{i,} s_j \rangle_{model}$$

Ecuación 2.17: Ecuación derivada parcia de la energía.

En la ecuación anterior, $\langle s_{i,} s_j \rangle_{data}$ representa el valor promedio obtenido al fijar los estados visibles como valores de un dato de entrenamiento. Por otro lado, $\langle s_{i,} s_j \rangle_{model}$ representa el valor promedio de $s_i s_j$ en equilibrio térmico sin fijar los estados visibles como datos de entrenamiento. Para obtener este promedio se ejecuta el proceso de Gibb Sampling sobre múltiples instancias. Por lo tanto, emplearemos la diferencia centrada en los datos de entrada y los datos obtenidos por el modelo con el proceso del Gibb Sampling para realizar la actualización de los valores de los parámetros pesos y bias.

Este proceso de cálculo de la derivada parcial se repetirá con todos los valores que componen cada mini-batch, en los que se divide el conjunto de entrenamiento. Una vez calculadas todas las derivadas parciales de cada uno de los datos que forman el mini-batch actual, se procederá a calcular el valor promedio de dichos valores. Posteriormente, se emplea el valor promedio para actualizar los valores de los pesos y bias, siguiendo las siguientes ecuaciones:

$$w_{ij} \Leftarrow w_{ij} + \alpha \left(\langle s_{i,} s_j \rangle_{data} - \langle s_{i,} s_j \rangle_{model} \right)$$

Ecuación 2.18: Valor de actualización de los pesos.

$$b_i \Leftarrow b_i + \alpha \left(\langle s_{i,} 1 \rangle_{data} - \langle s_{i,} 1 \rangle_{model} \right)$$

Ecuación 2.19: Valor de actualización de los bias.

El principal problema del procedimiento del Gibb Sampling, se basa en el alto número de iteraciones que se necesitan para alcanzar el equilibrio térmico.

2.4.3 Logros obtenidos por el modelo RBM

El concurso de Netflix fue una competición abierta organizada por la plataforma de visionado de contenido multimedia, cuyo objetivo perseguía encontrar el mejor algoritmo de Filtrado Colaborativo capaz de predecir la calificación de los usuarios sobre un conjunto de películas en base a otras valoraciones previas, sin mas información que los identificadores de estos.

Una vez registrados en el concurso, los participantes podían acceder a los conjuntos de entrenamiento y test proporcionados. El conjunto de entrenamiento se componía por más de 100 millones de valoraciones de más de 480000 clientes, elegidos al azar en casi 18000 títulos de películas. Estos datos fueron recolectados entre octubre de 1998 y diciembre de 2005, reflejando las calificaciones recibidas por Netflix durante ese periodo. La valoración de las películas se ponderaba de 1 a 5, siendo estos valores enteros. Con el fin de mantener el anonimato de los usuarios, se eliminaron todos los datos referentes a éstos del conjunto de datos, a excepción de la fecha de la valoración y, por otro lado, los identificadores de cada uno de los usuarios se sustituyeron por otros asignados de forma aleatoria.

El conjunto de test se componía de 2.8 millones de pares identificador de usuario/película con las fechas de valoración, pero estos registros carecían de valoración. Los datos de este conjunto se componían de las calificaciones más recientes de un subconjunto de los mismos usuarios sobre un subconjunto de las mismas películas.

Con el fin de valorar el algoritmo de predicción seleccionado por cada equipo, estos debían lograr que este algoritmo valorará todos aquellos pares de identificador de usuario/película que formaba el conjunto de entrenamiento, y enviar los resultados obtenidos para valorarlo en el mismo formato que los conjuntos de entrenamiento y test.

El concurso lo ganó el equipo “Bellkor’s Pragmatic Chaos” el 26 de julio de 2009, este equipo empleo para ganar el concurso un modelo de red neuronal Restricted Boltzmann Machine.

2.4.4 Deep Boltzmann Machine

Este tipo de modelo es la evolución de las Restricted Boltzmann Machine que difiere de las RBM en la estructura de la red. Al contrario que las RBM, las Deep Boltzmann Machine forman una estructura de red neuronal multicapa, en donde cada capa captura las complicadas correlaciones entre ella y la capa anterior. Este tipo de redes, al igual que las Deep Belief, tienen el potencial computacional de capturar representaciones internas en los conjuntos de datos cuya complejidad cada vez aumenta más, consiguiendo una alta capacidad resolutiva en problemas de reconocimiento de objetos y transcripción de conversaciones o discursos a texto.

3 METODOLOGÍA

En este capítulo se realizar una descripción de la metodología y el marco de trabajo empleados para desarrollar este proyecto, para ello se ha decido emplear una metodología ágil, ya que permite su desarrollo de forma iterativa e incremental. Son metodologías que se encuentran en auge debido a la rapidez con la que evoluciona la tecnología, ya que proporcionan una gran flexibilidad frente a los posibles cambios y una eficacia a la hora de trabajar, abaratando costes y reduciendo el tiempo del proyecto.

Este tipo de metodologías proporcionan una serie de ventajas frente a las metodologías tradicionales, como son:

- Mejoran la satisfacción del cliente dado que este está involucrado y comprometido con el proyecto durante todo su desarrollo. Tras la finalización de cada una de las etapas de desarrollo, se informa al cliente sobre los progresos realizados y los logros alcanzados para aumentar su motivación con el fin de que se involucre y lograr añadir sus conocimientos al proyecto.
- Mejoran la motivación e implicación del equipo de desarrollo, puesto que todos sus miembros conocen el estado del proyecto.
- Permite el ahorro de tiempo y costes de desarrollo, al utilizar una forma de trabajo es más eficiente y rápida.
- Se logra mejorar la eficacia y rapidez de trabajo por medio de entregas parciales, logrando evitar también el desarrollo de características innecesarias.
- La continua interacción entre cliente y desarrolladores permite el aumento de la calidad del producto.
- También permite alertar rápidamente de posibles errores en el proyecto.

En concreto, para este proyecto se emplea Scrum como marco de trabajo, por ser el más conocido por este equipo de desarrollo y uno de los marcos de trabajo más empleados en la actualidad.

Scrum, como se ha mencionado antes, es un marco de trabajo empleado en proyectos de desarrollo basados en metodologías ágiles que permite al equipo presentar en pequeños periodos de tiempo entregables al cliente. Este marco de trabajo presenta tres características:

- Transparencia: todos los miembros del equipo tienen conocimiento de lo que ocurre en el proyecto.
- Inspección: los miembros del equipo periódicamente inspeccionan el progreso del proyecto
- Adaptabilidad: el equipo se ajusta para lograr los distintos objetivos de cada una de las etapas de desarrollo o sprint.

3.1.1 Roles de Scrum

Con el fin de lograr entregar valor y ofrecer resultados de calidad para así cumplir los objetivos del cliente, Scrum emplea equipos de desarrollo auto-organizados y multifuncionales, haciendo así a cada uno responsable de un conjunto de tareas determinadas y de su finalización en los plazos acordados. De esta forma, se garantiza la entrega de valor del equipo completo, sin que sea necesaria la ayuda o supervisión minuciosa de otros miembros.

En Scrum existen tres roles importantes, los cuales son los siguientes:

- Scrum Master: es el líder del equipo de trabajo ágil. Su misión principal consiste en el cumplimiento de todos los objetivos del proyecto, para ello debe de asegurarse que todas y cada una de las técnicas Scrum empleadas son comprendidas y aplicadas en la organización. Además, es el encargado de solucionar cualquier posible problema que pueda aparecer en los distintos sprint que componen el proyecto.
- Product Owner: es el enlace del equipo de desarrollo con los distintos stakeholders, se encarga de mantener una comunicación constante con dichos stakeholder. Además, se encarga de maximizar el valor de trabajo del equipo de desarrollo, por medio del Product Backlog, el cual es el encargado de organizar y gestionar.
- Equipo de desarrollo: se componen por todas aquellas personas encargadas de realizar las tareas, para ello se encargan de satisfacer los criterios de aceptación de las historias, determinar la complejidad de las tareas y negociar el alcance de los distintos sprint.

3.1.2 Artefactos de Scrum

Son una serie de herramientas que emplean los equipos Scrum para maximizar la transparencia dentro del equipo y permitir que todo el equipo tenga la misma visión de proyecto. Los artefactos que se emplea en Scrum son los siguientes:

- Product Backlog: es un inventario, el cual contiene un listado de las distintas tareas o funcionalidades necesarias para el desarrollo del proyecto. Es el resultado del trabajo realizado por el Product Owner con los distintos stakeholders del proyecto. Esta lista está ordenada en función de la prioridad de las distintas tareas o funcionalidades, no es necesario que la lista esté completa al principio del proyecto, pero sí deben estar presentes los requisitos más importantes. Esta lista presenta los siguientes elementos:
    - Funcionalidades.
    - Bugs.
    - Historias de usuario.
    - Tareas técnicas.
    - Trabajo de investigación.
- Sprint Backlog: consiste en una lista específica de aquellas funcionalidades o tareas del Product Backlog (Product Backlog Item) que se van a abordar dentro de cada sprint, este listado debe de mantenerse actualizado por medio de las Daily Meeting. Los distintos elementos del Sprint Backlog se dividen en tareas más pequeñas, con el objetivo de facilitar su implementación en un incremento.

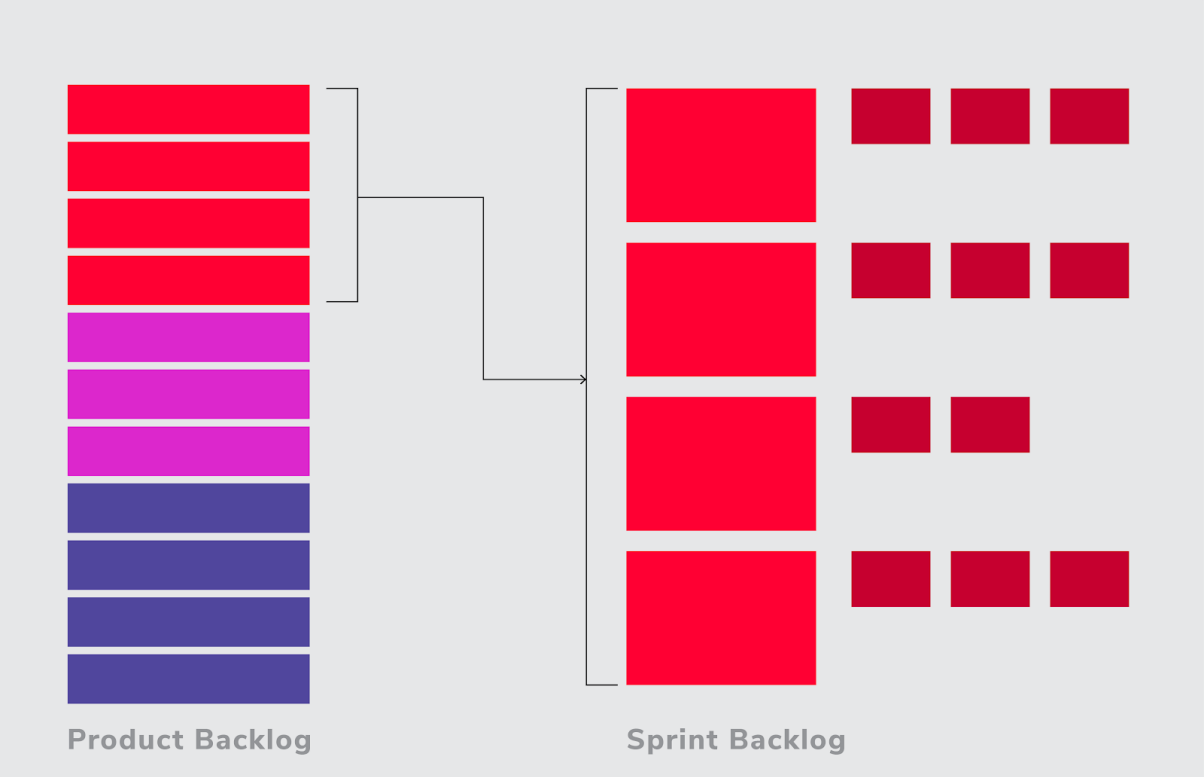


Ilustración 3.1: Product Backlog y Sprint Backlog. Fuente: https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html

- Incremento: consiste en un entregable software que se obtiene al final de cada sprint tras la realización de todas las tareas abordadas en el sprint actual. Los incrementos se desarrollarán de forma iterativa e incremental con respecto a los anteriores.

3.1.3 Eventos de Scrum

Scrum se caracteriza por la retroalimentación que proporciona la incorporación del cliente en el desarrollo, permitiendo así poder ofrecer un entregable tras finalizar cada sprint del proyecto; el cual será refinado y se le añadirán las nuevas funcionalidades con el próximo sprint.

Con el fin de obtener esa retroalimentación, Scrum dispone de los siguientes eventos:

- Sprint: es la base de Scrum, consiste en un periodo de tiempo de entre una o cuatro semanas en el cual se realizan un conjunto de tareas y contiene al resto de eventos pertenecientes a Scrum. Su duración máxima no debe de superar las cuatro semanas de duración debido a que sino puede provocar la pérdida de feedback importante para el proyecto y comprometer dicho proyecto.
- Sprint Planning: es una reunión que se realiza al comienzo de cada sprint donde participa el equipo de Scrum al completo para así inspeccionar el Product Backlog, con el fin de decidir qué tareas o Product Backlog Items se van a abordar durante el sprint actual. En esta reunión, el Product Owner presenta el Product Backlog actualizado y el Equipo de desarrollo se encarga de determinar cuántos Product Backlog Item serán capaces de abordar en el sprint actual teniendo en cuenta el objetivo de este sprint. Después de decidir los elementos que serán capaces de abordar, el Equipo de desarrollo divide estos elementos en tareas más pequeñas. Por medio de esta reunión, se obtiene una actualización del Product Backlog, el Sprint Backlog y el Sprint Goal (objetivo del

sprint). La duración máxima de esta reunión debe ser de ocho horas para sprint de cuatro semanas.

Ilustración 3.2: Sprint Planning. Fuente: https://jeronimopalacios.com/scrum/

- Daily Meeting: es una reunión diaria de planificación de quince minutos de duración del Equipo de desarrollo, en la cual cada miembro expone las tareas que está abordando y qué impedimentos se le han presentado; de esta forma, se puede determinar si el Sprint Goal puede estar en riesgo de no ser alcanzado, de ser así se debería proceder a mitigar esos riesgos.
- Sprint Review: consiste en una reunión al final del sprint, en la cual el Product Owner se reúne con los distintos stakeholder para presentar el incremento actual, con el fin de inspeccionar dicho incremento y adaptarlo. Esta reunión permite medir la situación actual y actualizar el Product Backlog con nuevas condiciones que puedan afectar al negocio.
- Sprint Retrospective: es una reunión realizada al final de cada sprint, justo después de la Sprint Review. En esta reunión, el equipo Scrum reflexiona sobre el último sprint, presentando qué ha ido bien, qué ha ido mal y qué se podría mejorar para poder identificar posibles mejoras para el próximo sprint.

4 GESTIÓN DEL PROYECTO

A lo largo de este capítulo se especificará el Product Backlog con el listado de las historias de usuario y las tareas en las cuales se dividirá cada una de estas. Además, se procederá a detallar la planificación temporal que se ha seguido a lo largo del desarrollo de este proyecto.

4.1 PRODUCT BACKLOG

El objetivo de este capítulo consiste en especificar y describir el Product Backlog con el listado de las historias de usuario que deberá cumplir el proyecto. Estas historias de usuario consisten en descripciones cortas y simples de las características que ha de cumplir un proyecto, éstas han de ser contadas desde la perspectiva de un usuario del sistema. Las historias han de presentar la siguiente plantilla o formato:

- Como <Usuario>.
- Quiero <Cierto objetivo>.
- Por cierto <Motivo>.

A continuación, se detalla el Product Backlog con las distintas historias de usuario que presenta el proyecto.

ID	Descripción
US-01	El administrador del sistema podrá realizar la ejecución de las fases de entrenamiento y test del modelo de red neuronal.
US-02	El administrador del sistema podrá realizar inferencias sobre los casos de estudio.
US-03	El administrador del sistema podrá visualizar los resultados obtenidos en los procesos de entrenamiento y test.
US-04	Investigación acerca de las tecnologías y herramientas necesarias para el desarrollo del modelo.
US-05	Realización del estudio y análisis del modelo de red neuronal conocido como Restricted Boltzmann Machine.
US-06	Desarrollo de la documentación del proyecto.
US-07	Configuración de los entornos y plataformas necesarias para el desarrollo del modelo.

Tabla 4.1: Historias de usuario del Product Backlog.

4.1.1 Historias de usuario

En este apartado se va a especificar de forma detallada las diversas historias de usuario descritas en el Product Backlog. Antes de realizar la especificación se debe definir la jerarquía de las historias de usuario, y cómo se agrupan los elementos que las componen.

- La historia de usuario es la unidad básica que se puede identificar en el marco de trabajo de las metodologías ágiles.
- La historia de usuario se puede dividir a su vez en tareas.
- Una épica es la agrupación en bloque de historias de usuario.
- Una iniciativa es la agrupación de varias épicas.

A continuación, se detalla la especificación de las historias de usuario relacionadas con la implementación de las distintas funcionalidades de las que consta el modelo de red neuronal:

ID Historia	Descripción historia	Estatus	Prioridad	Estimaciones	Propietario
US-01	Como administrador del sistema deseo realizar los procesos de entrenamiento y test, para poder inferir sobre los casos de estudio.	Finalizada	Alta		
Criterios de aceptación					
1	Realizar procesos de entrenamiento y test.				
Dado que	Ejecuto el fichero				
Cuando	Se construye el modelo de red neuronal.				
Entonces	Se inician los procesos de entrenamiento y test del modelo.				

Tabla 4.2: Historia de usuario US-01.

ID Historia	Descripción historia	Estatus	Prioridad	Estimaciones	Propietario
US-02	Como administrador del sistema deseo realizar inferencias sobre los casos de estudio.	Finalizada	Alta		
Criterios de aceptación					
1	Inferencia casos de estudio.				
Dado que	Han finalizado los procesos de entrenamiento y test.				
Cuando					
Entonces	Se permite realizar inferencias sobre los datos.				

Tabla 4.3: Historia de usuario US-02.

ID Historia	Descripción historia	Estatus	Prioridad	Estimaciones	Propietario
US-03	Como administrador del sistema deseo poder visualizar un fichero con los resultados de los procesos de entrenamiento y test.	Finalizada	Alta		
Criterios de aceptación					
1	Creación fichero ‘TXT’ con resultados entrenamiento y test.				
Dado que	Han finalizado los procesos de entrenamiento y test.				
Cuando	Se haya creado el fichero ‘TXT’ con los resultados.				
Entonces	Se podrá visualizar dicho fichero.				

Tabla 4.4: Historia de usuario US-03.

A continuación, se procede a especificar las historias de usuario que componen las fases de estudio y análisis del modelo, investigación sobre las tecnologías y herramientas necesarias para el desarrollo del modelo y la redacción de la documentación complementaria al proyecto:

- US-04 Investigación acerca de las tecnologías y herramientas necesarias para el desarrollo del modelo.

  Esta historia de usuario comprende las tareas necesarias para la adquisición de los conocimientos básicos sobre las herramientas y tecnologías empleadas para la construcción del modelo de red neuronal (Tensorflow, Numpy, CSV...) y de un entorno de desarrollo para la ejecución de las distintas pruebas necesarias para la realización del estudio y análisis del modelo (Anaconda, Spyder, Google Cloud Platform…).

- US-05 Realización del estudio y análisis del modelo de red neuronal conocido como Restricted Boltzmann Machine.

  Tras finalizar las correspondientes pruebas, se realizará el estudio y análisis del modelo con el fin de determinar qué factores son los más determinantes a la hora de conseguir que el modelo logre ser más preciso.

- US-06 Desarrollo de la documentación del proyecto.

  Esta historia de usuario comprende las distintas tareas para la redacción de la documentación del proyecto. Además, se incluirá toda aquella investigación necesaria para su redacción.

- US-07 Configuración de los entornos y plataformas necesarias para el desarrollo del modelo.

  Esta historia de usuario comprende las tareas necesarias para la instalación y/o configuración de los distintos entornos de desarrollo, plataformas, librerías y APIs empleadas para el desarrollo del proyecto.

4.1.1.1 Especificación tareas

A continuación, una vez especificadas las distintas historias de usuario que presenta el proyecto, se procederá a realizar el desglose de dichas historias de usuario:

ID Historia	US-01	
ID Tarea	Descripción	Dificultad
T-01	Implementación modelo RBM.	3 Puntos de Historia.
	Esta tarea consiste en la implementación de la estructura del modelo de red neuronal RBM, es decir, definición de las variables correspondientes a los pesos, bias, tasa de aprendizaje, número de neuronas visible y ocultas,...	
T-02	Implementación algoritmo Gibb Sampling.	5 Puntos de Historia.
	Esta tarea comprende la implementación de un método que comprenda la ejecución del algoritmo de Gibb Sampling, necesario para la fase de entrenamiento.	
T-03	Implementación del método Compute Gradient.	3 Puntos de Historia.
	Esta tarea comprende la implementación de un método que permita el cálculo de los gradientes por medio de los valores obtenidos a través del algoritmo Gibb Sampling, necesarios para realizar la actualización de los valores de bias y pesos del modelo.	
T-04	Implementación método actualización bias y pesos.	2 Punto de Historia.
	Esta tarea comprende el desarrollo del método necesario para obtener el valor de actualización de los parámetros de los bias y pesos.	
T-05	Implementación procesos de entrenamiento y test.	8 Puntos de Historia.
	Esta tarea comprende el desarrollo de una clase capaz realizar las fases de entrenamiento y test del modelo.	
T-06	Implementación pre-procesado de los datos del CSV.	5 Puntos de Historia.
	Esta tarea comprende el desarrollo de las funcionalidades necesarias para realizar la lectura y pre-procesado de los datos correspondientes a los casos de estudio.	

Tabla 4.5: Especificación tareas US-01.

ID Historia	US-02	
ID Tarea	Descripción	Dificultad
T-07	Implementación método inferencia.	1 Punto de Historia.
	Esta tarea consiste en la implementación del método necesario para la realización de inferencias sobre los casos de estudio.	

Tabla 4.6: Especificación tareas US-02.

ID Historia	US-03	
ID Tarea	Descripción	Dificultad
T-08	Creación fichero con los resultados de las pruebas.	5 Puntos de Historia.
	Esta tarea consiste en el desarrollo de la funcionalidad que permita la creación de un fichero con los resultados de las fases de entrenamiento y test.	

Tabla 4.7: Especificación tareas US-03.

ID Historia	US-04	
ID Tarea	Descripción	Dificultad
T-09	Investigación sobre las opciones de desarrollo.	3 Puntos de Historia.
	Investigación sobre las opciones para realizar el desarrollo del modelo; es decir, entorno de desarrollo, lenguaje de programación, APIs, CVS...	
T-10	Investigación herramientas y tecnologías de desarrollo.	3 Puntos de Historia.
	Investigación realizada para adquirir los conocimientos básicos necesarios sobre las tecnologías y herramientas empleadas para realizar el desarrollo del modelo.	

Tabla 4.8: Especificación tareas US-04.

ID Historia	US-05	
ID Tarea	Descripción	Dificultad
T-11	Análisis comparativo de las pruebas realizadas.	5 Puntos de Historia.
	Esta tarea comprende la comparación de las distintas pruebas realizadas con el fin de determinar qué configuración de los parámetros del modelo ha obtenido mejores resultados y cómo afecta el valor de cada parámetro al modelo en la fase de entrenamiento.	
T-12	Conclusión factores determinantes.	2 Puntos de Historia.
	Esta tarea abarca la determinación en base a los resultados obtenidos en la tarea T-11 para determinar qué parámetros influyen más en el aprendizaje del modelo	
T-30	Ejecución batería pruebas Estudio y Análisis.	13 Puntos de Historia
	Esta tarea comprende la ejecución de las batería de pruebas realizadas para llevar a cabo el Estudio y análisis del modelo.	

Tabla 4.9: Especificación tareas US-05.

ID Historia	US-06	
ID Tarea	Descripción	Dificultad
T-13	Investigación modelo RBM	10 Puntos de historia.
	Esta tarea comprende la adquisición de los conocimientos pertinentes a cerca del modelo de red neuronal que va a desarrollar el equipo de proyecto.	
T-14	Investigación metodología Scrum e historias de usuario	2 Puntos de Historia.
	Investigación realizada para adquirir los conocimientos básicos necesarios referentes a la metodología Scrum y el método de especificación de requisitos basado en historias de usuario a emplear.	
T-15	Documentación Introducción y Contexto de negocio.	2 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Introducción.	

T-16	Documentación Contexto del Proyecto.	5 Puntos de Historia.
	Esta tarea se corresponde a la redacción de los apartado Contexto del Proyecto.	
T-17	Documentación Análisis.	2 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Análisis.	
T-18	Documentación Modelado de datos	2 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Modelado de datos.	
T-19	Documentación Plan de proyecto y Estimación.	2 Puntos de Historia.
	Esta tarea se corresponde a la redacción de los apartados Gestión del Proyecto.	
T-20	Documentación Diseño.	3 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Diseño	
T-21	Documentación Implementación del modelo.	5 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Implementación.	
T-22	Documentación Pruebas modelo.	2 Punto de Historia.
	Esta tarea se corresponde a la redacción del apartado Pruebas.	
T-23	Documentación Análisis y estudio del modelo.	3 Puntos de Historia.
	Esta tarea se corresponde a la redacción del apartado Estudio y análisis del modelo.	
T-24	Documentación Conclusiones y Trabajo futuras.	1 Punto de Historia.
	Esta tarea se corresponde a la redacción del aparatado Conclusiones y trabajo futuro.	
T-25	Documentación Anexo.	

	Esta tarea se corresponde a la redacción del aparatado **¡Error! No se encuentra el origen de la referencia.**.	1 Punto de Historia.
T-31	Revisión de la documentación	1 Punto de Historia.
	Esta tarea comprende la revisión de los distintos apartados descritos en este documento.	

Tabla 4.10: Especificación tareas US-06.

ID Historia	US-07	
ID Tarea	Descripción	Dificultad
T-26	Instalación y configuración del entorno de desarrollo Anaconda en un sistema Windows.	1 Puntos de Historia.
	Esta tarea comprende la instalación y configuración del entorno de desarrollo de Anaconda para la implementación del modelo.	
T-27	Instalación API y librerías Python.	1 Puntos de Historia.
	Esta tarea abarca la instalación e integración de las APIs y librerías necesarias para el desarrollo del modelo de red neuronal.	
T-28	Creación cuenta GCP y configuración de entorno de desarrollo en cloud.	1 Puntos de Historia.
	Esta tarea comprende la creación de una cuenta en GCP y la configuración de una máquina virtual para la ejecución de las pruebas necesarias para la realización del análisis y estudio del modelo.	
T-29	Instalación y configuración del entorno de desarrollo Anaconda en una máquina virtual Debian.	1 Puntos de Historia.
	Esta tarea comprende la instalación y configuración del entorno de desarrollo de Anaconda en una máquina virtual Debian en una cuenta de GCP para la realización de las pruebas necesarias para la realización del análisis y estudio del modelo.	

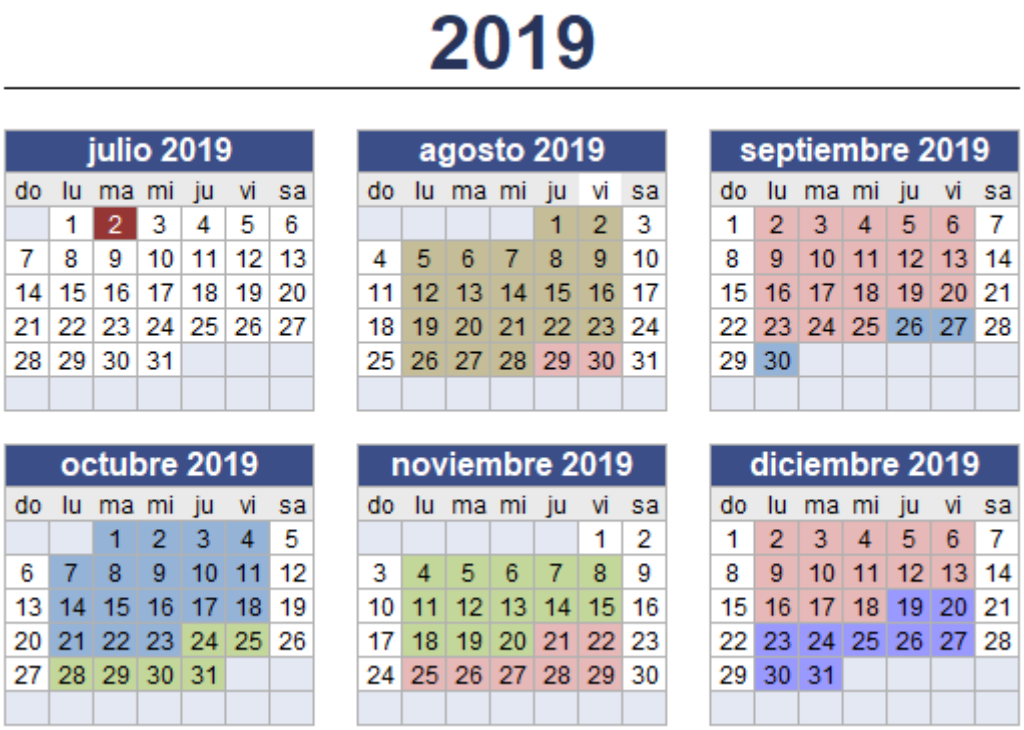
Tabla 4.11: Especificación tareas US-07.

4.2 SPRINT BACKLOG

En este apartado, se procederá a detallar el Sprint Backlog, que como se ha mencionado, contiene un listado de las tareas, especificadas en el apartado Product Backlog, detalladas según el sprint en el que se van a desarrollar. Con dicha lista se persigue mostrar las distintas tareas que se requiere realizar hasta lograr alcanzar el Sprint Goal.

El proyecto se ha dividido en 19 sprints de una duración de 4 semanas cada uno, desarrollando el proyecto únicamente de lunes a viernes y dedicándolo 5 horas al día.

Durante los cuatro primeros sprints de desarrollo, se concluyó que el tiempo dedicado al proyecto no sería posible ajustarlo a la planificación anteriormente especificada, por tener que compaginarlo con la realización de las prácticas extracurriculares durante éstos. Eso significó que el desarrollo del proyecto se viera ralentizado desde el día 1 de agosto de 2019 hasta el 19 de noviembre de 2019.



2019

julio 2019

do	lu	ma	mi	ju	vi	sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

agosto 2019

do	lu	ma	mi	ju	vi	sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

septiembre 2019

do	lu	ma	mi	ju	vi	sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

octubre 2019

do	lu	ma	mi	ju	vi	sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

noviembre 2019

do	lu	ma	mi	ju	vi	sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

diciembre 2019

do	lu	ma	mi	ju	vi	sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Ilustración 4.1: Calendario Planificación 2019.

2020

enero 2020

do	lu	ma	mi	ju	vi	sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

febrero 2020

do	lu	ma	mi	ju	vi	sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

marzo 2020

do	lu	ma	mi	ju	vi	sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

abril 2020

do	lu	ma	mi	ju	vi	sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

mayo 2020

do	lu	ma	mi	ju	vi	sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

junio 2020

do	lu	ma	mi	ju	vi	sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

julio 2020

do	lu	ma	mi	ju	vi	sa
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

agosto 2020

do	lu	ma	mi	ju	vi	sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

septiembre 2020

do	lu	ma	mi	ju	vi	sa
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30			

Ilustración 4.2: Calendario Planificación 2020 (I).

octubre 2020

do	lu	ma	mi	ju	vi	sa
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

noviembre 2020

do	lu	ma	mi	ju	vi	sa
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Ilustración 4.3: Calendario Planificación 2020 (II).

Ilustración 4.4: Calendario Planificación 2021(I).

Ilustración 4.5: Calendario Planificación 2021(II).

Además, la primera reunión con los tutores se realizó el día 2 de julio de 2019, pero no fue hasta un mes después cuando se comenzó a desarrollar el proyecto, siendo la fecha de inicio del desarrollo del proyecto el día 1 de agosto de 2019.

Una vez fijadas las fechas de inicio y fin de cada uno de los sprint previstos anteriormente, se procederá a distribuir cada una de las tareas del Product Backlog en los sprint determinados y estimar la duración prevista de cada tarea. Para realizar la estimación en horas de desarrollo de las distintas tareas, Scrum emplea la experiencia obtenida de otros proyectos para realizar este proceso; debido a que este proyecto es de investigación y, por tanto, se dispone de ninguna experiencia previa respecto a proyectos de este tipo, es necesario realizar una estimación intuitiva de las horas de cada tarea. Por este motivo, el tiempo de desarrollo de cada tarea puede verse desviado de la estimación prevista, así que cuando ocurra alguna desviación, será necesario realizar ajustes del Sprint Backlog.

Sprint 1	**Duración**	**01/08/2019**	**28/08/2019**	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-09	3 Puntos de historia	01/08/2019	14/08/2019	10
T-10	3 Puntos de historia	15/08/2019	28/08/2019	10
Total Esfuerzo	6 Puntos de historia		Total Horas	20

Tabla 4.12: Sprint 1.

Durante el primer sprint, el objetivo principal se centró en la investigación de las tecnologías y herramientas que proporcionarán el entorno y las herramientas de desarrollo necesarias para realizar el proyecto. El incremento producido por este sprint es la selección del entorno y las herramientas de desarrollo necesarias para implementar el modelo.

Sprint 2	Duración	29/08/2019	25/09/2019	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-26	1 Punto de historia	29/08/2019	29/08/2019	2
T-27	1 Punto de historia	29/08/2019	29/08/2019	2
T-14	2 Puntos de historia	30/08/2019	06/09/2019	5
T-13	10 Puntos de historia	09/09/2019	25/09/2019	11
Total Esfuerzo	14 Puntos de historia		Total Horas	20

Tabla 4.13: Sprint 2.

La realización del segundo sprint se centró en la configuración del entorno de desarrollo con el que implementar el modelo y en la adquisición de los conocimientos necesarios sobre la metodología Scrum y el modelo de red neuronal RBM. El incremento obtenido tras la realización de este sprint es los conocimientos necesarios sobre la metodología y parte de los necesarios sobre el modelo de red.

Sprint 3	Duración	26/09/2019	23/10/2019	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	10 Puntos de historia	26/09/2019	02/10/2019	5
T-01	3 Puntos de historia	03/10/2019	23/10/2019	10
Total Esfuerzo	13 Puntos de historia		Total Horas	15

Tabla 4.14: Sprint 3.

Durante el desarrollo del tercer sprint se realizaron las tareas de adquisición de los conocimientos sobre el modelo y se comenzó la implementación del código del modelo, en concreto la definición e inicialización de los parámetros del modelo. El incremento obtenido tras la finalización de este sprint está formado por la mejora de los conocimientos e construcción del modelo de red neuronal.

Sprint 4	Duración	24/10/2019	20/11/2019	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	10 Puntos de historia	24/10/2019	05/10/2019	10
T-09	3 Puntos de historia	06/10/2019	08/10/2019	5
T-01	3 Puntos de historia	11/11/2019	20/11/2019	10
Total Esfuerzo	16 Puntos de historia		Total Horas	25

Tabla 4.15: Sprint 4.

A finales del sprint anterior, se observó que se había implementado de forma incorrecta el modelo de red neuronal, por lo tanto, durante el desarrollo del Sprint 4 fue necesario realizar la corrección pertinente. Por este motivo, se procedió a revisar cuales fueron los errores cometidos y mejorar los conocimientos sobre el modelo y las herramientas de desarrollo. El incremento obtenido está formado por la mejora de los conocimientos referentes al modelo y la implementación del modelo.

Sprint 5	Duración	21/11/2019	18/12/2019	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	13 Puntos de historia	21/11/2019	29/11/2019	35
T-02	5 Puntos de historia	02/12/2019	10/12/2019	35
T-03	3 Puntos de historia	11/12/2019	16/12/2019	20
T-04	2 Puntos de historia	17/12/2019	18/12/2019	10
Total Esfuerzo	23 Puntos de historia		Total Horas	100

Tabla 4.16: Sprint 5.

Tras la corrección en el sprint anterior del modelo RBM, se procedió a implementar los métodos correspondientes al proceso de entrenamiento del modelo. De esta forma, se implementaron los métodos correspondientes a los procesos Gibb Sampling, cálculo de los gradientes y la actualización de los parámetros del modelo.

Sprint 6	Duración	19/12/2019	15/01/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	13 Puntos de historia	19/12/2019	20/12/2019	15
T-04	2 Puntos de historia	23/12/2019	24/12/2019	10
T-05	8 Puntos de historia	26/12/2019	15/01/2020	60
Total Esfuerzo	23 Puntos de historia		Total Horas	85

Tabla 4.17: Sprint 6.

Durante el sprint sexto, se inició la implementación del proceso de entrenamiento, para el cual fue necesario realizar una breve investigación con el fin de profundizar en el funcionamiento de este proceso.

Sprint 7	Duración	16/01/2020	12/02/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-05	8 Puntos de historia	16/01/2020	21/01/2020	20
T-06	8 Puntos de historia	22/01/2020	31/01/2020	40
T-07	1 Punto de historia	03/02/2020	03/02/2020	5
T-08	5 Puntos de historia	04/02/2020	12/02/2020	35
Total Esfuerzo	21 Puntos de historia		Total Horas	100

Tabla 4.18: Sprint 7.

Durante este sprint se finalizó el proceso de entrenamiento, comprendiendo además la implementación de los procesos referentes al pre-procesado de los datos y la generación del fichero con resultados, también, se implementó el proceso de inferencia del modelo. El incremento obtenido en este sprint está formado por el proceso de entrenamiento y el proceso de inferencia.

Sprint 8	Duración	13/02/2020	11/03/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-05	8 Puntos de historia	13/02/2020	21/02/2020	35
T-22	2 Puntos de historia	24/02/2020	29/02/2020	25
T-05	8 Puntos de historia	02/03/2020	11/03/2020	40
Total Esfuerzo	18 Puntos de historia		Total Horas	100

Tabla 4.19: Sprint 8.

Tras la finalización del sprint anterior, se procedió a verificar el correcto funcionamiento de los procesos implementados hasta la fecha, para ello, se realizaron una serie de baterías de pruebas con el fin de comprobar su correcto funcionamiento y posteriormente se corrigieron los errores detectados.

Sprint 9	Duración	12/03/2020	08/04/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	13 Puntos de historia	12/03/2020	20/03/2020	35
T-16	5 Puntos de historia	25/03/2020	27/03/2020	15
T-17	2 Puntos de historia	01/04/2020	08/04/2020	24
Total Esfuerzo	20 Puntos de historia		Total Horas	74

Tabla 4.20: Sprint 9.

Durante el desarrollo de este sprint, se ha iniciado la redacción de los apartados referentes al Contexto del Proyecto y Análisis de este documento.

Sprint 10	Duración	09/04/2020	06/05/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-13	13 Puntos de historia	09/04/2020	15/04/2020	20
T-21	5 Puntos de historia	20/04/2020	23/04/2020	16
T-21	5 Puntos de historia	29/04/2020	01/05/2020	12
T-17	2 Puntos de historia	04/05/2020	06/05/2020	12
Total Esfuerzo	20 Puntos de historia		Total Horas	60

Tabla 4.21: Sprint 10.

Durante este sprint, se realizó la redacción del apartado Implementación y se continuó con la del apartado Análisis.

Sprint 11	Duración	07/05/2020	03/06/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-05	8 Puntos de historia	07/05/2020	13/05/2020	20
T-08	5 Puntos de historia	14/05/2020	15/05/2020	8
T-30	13 Puntos de historia	18/05/2020	03/06/2020	52
Total Esfuerzo	21 Puntos de historia		Total Horas	80

Tabla 4.22: Sprint 11.

Durante el desarrollo de este sprint, se inició la implementación del proceso de test del modelo y se modificó el fichero con los resultados para que mostrará también los resultados de este proceso, también se inició la ejecución de las pruebas necesarias para realizar el apartado Estudio y análisis del modelo.

Sprint 12	Duración	04/06/2020	01/07/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-22	2 Puntos de historia	04/06/2020	11/06/2020	30
T-05	8 Puntos de historia	12/06/2020	16/06/2020	15
T-30	13 Puntos de historia	17/06/2020	01/07/2020	55
Total Esfuerzo	23 Puntos de historia		Total Horas	100

Tabla 4.23: Sprint 12.

En el desarrollo de este sprint, se realizaron las baterías de pruebas correspondientes al proceso de entrenamiento y test, se corrigieron los fallos detectados y se continuaron las pruebas pertenecientes al estudio del modelo.

Sprint 13	Duración	02/07/2020	29/07/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-10	3 Puntos de historia	02/07/2020	03/07/2020	12
T-28	1 Punto de historia	06/07/2020	06/07/2020	2,5
T-29	1 Punto de historia	06/07/2020	06/07/2020	2,5
T-06	8 Puntos de historia	07/07/2020	07/07/2020	5
T-30	13 Puntos de historia	08/07/2020	29/07/2020	80
T-16	5 Puntos de historia	08/07/2020	17/07/2020	40
T-17	2 Puntos de historia	20/07/2020	29/07/2020	40
Total Esfuerzo	33 Puntos de historia		Total Horas	182

Tabla 4.24: Sprint 13.

Tras la finalización del sprint anterior, se detectó que la ejecución de las pruebas necesarias para el desarrollo del estudio del modelo requería una gran cantidad de tiempo, por lo tanto, se decidió ejecutar esta serie de pruebas en una plataforma cloud, con el fin de poder reducir el tiempo de ejecución de éstas y poder redactar de forma paralela otros apartados de este documento. Por otra parte, se decidió también realizar dos bloques de pruebas para el apartado Estudio y análisis del modelo, cada uno con dos conjuntos de datos sobre el Titanic de distinto tamaño.

Sprint 14	Duración	30/07/2020	26/08/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-15	2 Puntos de historia	30/07/2020	07/08/2020	35
T-18	2 Puntos de historia	10/08/2020	14/08/2020	25
T-20	3 Puntos de historia	17/08/2020	19/08/2020	15
T-21	5 Puntos de historia	20/08/2020	26/08/2020	25
Total Esfuerzo	12 Puntos de historia		Total Horas	100

Tabla 4.25: Sprint 14.

Durante el desarrollo de este sprint, se redactaron los apartados Introducción, Modelado de datos y Diseño, además se continuó desarrollando el apartado Implementación.

Sprint 15	Duración	27/08/2020	25/09/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-21	5 Puntos de historia	27/08/2020	04/09/2020	35
T-11	5 Puntos de historia	04/09/2020	09/09/2020	20
T-12	2 Puntos de historia	10/09/2020	14/09/2020	15
T-23	3 Puntos de historia	15/09/2020	16/09/2020	10
T-24	1 Punto de historia	17/09/2020	18/09/2020	10
T-25	1 Punto de historia	18/09/2020	18/09/2020	5
T-19	2 Puntos de historia	22/09/2020	23/09/2020	10
Total Esfuerzo	19 Puntos de historia		Total Horas	105

Tabla 4.26: Sprint 15.

En este sprint, se continuó con el desarrollo del apartado correspondiente a la implementación del modelo y además se comenzó a desarrollar los apartados Estudio y análisis del modelo, Conclusiones y trabajo futuro, Manuales de Usuario, Glosario y Acrónimos, Bibliografía y Webgrafía y Gestión del Proyecto.

Sprint 16	Duración	24/09/2020	21/10/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-15	2 Puntos de historia	28/09/2020	30/09/2020	12
T-21	5 Puntos de historia	01/10/2020	09/10/2020	28
T-19	2 Puntos de historia	19/10/2020	21/10/2020	12
Total Esfuerzo	9 Puntos de historia		Total Horas	52

Tabla 4.27: Sprint 16.

Durante este sprint 16, se realizaron las correcciones de los errores encontrados en los apartados Introducción e Implementación. Por otro lado, se continuó redactando el apartado Gestión del Proyecto.

Sprint 17	Duración	22/10/2020	18/11/2020	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-19	2 Puntos de historia	22/10/2020	23/10/2020	8
T-19	2 Puntos de historia	27/10/2020	28/10/2020	8
T-13	10 Puntos de historia	02/11/2020	06/11/2020	20
T-21	5 Puntos de historia	09/11/2020	12/11/2020	16
T-31	1 Punto de Historia.	13/11/2020	13/11/2020	3
T-31	1 Punto de Historia.	16/11/2020	18/11/2020	9
Total Esfuerzo	9 Puntos de historia		Total Horas	64

Tabla 4.28: Sprint 17.

Durante el sprint 17, se han finalizado todos aquellos apartados que no se habían podido finalizar en el sprint anterior, siendo estos los apartados Gestión del Proyecto, Implementación, Manuales de Usuario, Glosario, Acrónimos y Bibliografía y Webgrafía; además, se comenzó a revisar los distintos apartados de este documento.

Sprint 18	Duración	05/02/2021	08/02/2021	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-25	1 Puntos de historia	05/02/2021	08/02/2021	10
Total Esfuerzo	1 Puntos de historia		Total Horas	10

Tabla 4.29: Sprint 18.

En el mes de febrero de 2021 se observó que faltaban por redactar los apartados referentes a los manuales de usuario para la instalación del entorno de ejecución de los scripts desarrollados y fue necesario añadir un sprint adicional entre las fechas 05/02/2021 y 08/02/2021, para redactar estos manuales.

Sprint 19	Duración	02/03/2021	07/03/2021	
ID Tarea	Esfuerzo previsto	Fecha inicio	Fecha fin	Horas empleadas
T-15	2 Puntos de Historia.	02/02/2021	02/02/2021	3
T-16	5 Puntos de Historia.	03/03/2021	03/03/2021	2
T-17	2 Puntos de Historia.	04/03/2021	04/03/2021	0.5
T-19	2 Puntos de Historia.	04/03/2021	04/03/2021	0.5
T-20	3 Puntos de Historia.	05/03/2021	05/03/2021	4
T-25	1 Punto de Historia.	07/03/2021	07/03/2021	1
T-31	1 Punto de Historia.	07/03/2021	07/03/2021	1
Total Esfuerzo	16 Puntos de historia		Total Horas	12

Tabla 4.30:Sprint 19.

Tras la finalización de una de las revisiones realizadas sobre este documento, se observaron una serie de errores en los capítulos de Introducción, Clasificación, Regresión, Redes neuronales, Metodología, Gestión del Proyecto, Análisis, Arquitectura física y Bibliografía y Webgrafía, por lo tanto, fue necesario crear un nuevo sprint para corregir los errores pertinentes.

5 ANÁLISIS

En este apartado se lleva a cabo el análisis del proyecto, especificando de forma detallada los actores del sistema, las limitaciones y las restricciones que presenta dicho proyecto.

5.1 DESCRIPCIÓN DE ACTORES

En esta sección del documento se identificará aquellos actores que interactúan con nuestro sistema, considerando actor del sistema a toda entidad externa que interactúa directa o indirectamente con ese sistema.

En el caso de este proyecto, el sistema solo constará de un único actor que podrá realizar tanto tareas de ejecución de los procesos de entrenamiento y test como de inferencia sobre los datos.

ID	ACT-01
Nombre	Administrador del sistema
Versión	1.0.1
Descripción	Representa aquel usuario que se encargará de realizar las fases de entrenamiento y de test del modelo.

Tabla 5.1: Descripción de actores.

5.2 LIMITACIONES Y RESTRICCIONES

A la hora de desarrollar el proyecto se han encontrado una serie de limitaciones, siendo la principal de ellas el limitado número de registros del que dispone el conjunto de datos utilizado. Al emplear como conjunto de datos de entrenamiento el referente al hundimiento del Titanic, únicamente se dispone de un máximo de 2200 registros. Otra limitación, consiste en la restricción de uso de variables únicamente binarias, que viene dada por el propio modelo de red neuronal; el modelo RBM solo emplea como entrada de datos variables binarias.

Por otra parte, también se dispone de una limitación económica. Con el fin de optimizar la ejecución de las pruebas necesarias para la realización del apartado Estudio y análisis del modelo, se ha decidido emplear Google Cloud Platform para realizar esta parte del proyecto, accediendo a la prueba gratuita de un año de esta plataforma y al presupuesto que Google proporciona de 300$ para invertir en su plataforma minimizando en parte los costes del proyecto.

6 MODELADO DE DATOS

En esta sección se definirán el conjunto de datos empleado para realizar los procesos de entrenamiento y test del modelo a estudiar, antes y después de realizar el pre-procesado de estos mismos.

6.1 RAW DATA

A lo largo de esta subsección, se describe el conjunto de datos empleado para la realización de este proyecto. Se ha elegido un conjunto los datos sobre el hundimiento del Titanic, que recoge la información sobre las pasajeros y tripulación que se embarcaron en este transatlántico.

Para la realización del estudio y análisis del modelo de red neuronal se ha decidido emplear dos dataset diferentes sobre las personas a bordo del transatlántico.

El primer conjunto de datos consta de 1313 registros, con 10 columnas cada uno de los registros y un peso de 89 KB. El fichero que contiene los datos es un archivo CSV, cuyo separador es la ‘,’.

Los campos de este fichero son los siguientes:

Nombre	Tipo de dato	Descripción
PassengerId	Int	Es un identificador numérico de pasajero.
Survived	Boolean	Determina si la persona sobrevivió o no al hundimiento.
Pclass	String	Contiene la clase a la que pertenecía la persona. ● '1st': Primera clase. ● '2nd': Segunda clase. ● '3rd': Tercera clase.
Sex	String	Contiene el sexo de la persona. ● 'Male'. ● 'Female'.
Age	Int	Contiene la edad de la persona.
Name	String	Contiene el nombre de la persona.
SibSp	Int	Número de hermanos/as o cónyuges a bordo del transatlántico.
Parch	Int	Número de padres/madres o hijos/as a bordo del transatlántico.
Ticket	String	Identificador del billete .
Fare	Float	Tarifa del billete.
Cabin	String	Identificador del camarote.
Embarked	String	Puerto en el que se embarcó.

Tabla 6.1: Campos fichero Titanic con 1316.

El segundo fichero que contiene los datos del Titanic consta de 2201 registros. A diferencia del anterior fichero además de los datos de los pasajeros, también contiene la información de la tripulación del transatlántico. Por otro lado, este fichero solo contiene los siguientes campos:

Nombre	Tipo de dato	Descripción
Survived	Boolean	Determina si la persona sobrevivió o no al hundimiento.
Pclass	String	Contiene la clase a la que pertenecía la persona. ● '1st': Primera clase. ● '2nd': Segunda clase. ● '3rd': Tercera clase. ● 'crew': Tripulación.
Sex	String	Contiene el sexo de la persona. ● 'Male'. ● 'Female'.
Age	Int	Contiene la edad de la persona.

Tabla 6.2: Campos fichero Titanic con 2201 registros.

6.2 PROCESAMIENTO DE LOS DATOS

La realización de un correcto procesado de los datos es un proceso vital a la hora de lograr la correcta realización de los procesos de entrenamiento y test del modelo. Para ello estos datos se deben adaptar para poder ser usados por el modelo.

Una de las principales desventajas del modelo de red neuronal que se está estudiando en este proyecto es que el tipo de datos que la red emplea se limita a datos con valores binarios, por lo tanto, se debe convertir todos aquellos datos que no presenten este tipo de valor.

Los campos que se han decidido emplear de los distintos conjuntos de datos son: 'Survived', 'Pclass', 'Sex' y 'Age'. Debido a que sólo los campos 'Survived' y 'Sex' se ajustan a este tipo de valor se va a requerir realizar una adaptación de los datos a valores binarios.

A continuación, se especificará como se ha realizado el procesado de los distintos campos empleados del conjunto de datos con el fin de transformar los valores no binarios en valores binarios.

Con respecto al campo 'Sex', al tener valor binario ('Male' o 'Female'), se ha sustituido el valor 'Female' por 1 y el valor 'Male' por 0. Igual ocurre con el campo 'Survived', como el campo tiene un valor binario únicamente se ha sustituido el valor 'Yes' por 1 y 'No' por 0.

Por el contrario, el campo 'Age' es un tipo de dato entero, por lo que se necesita sustituir el valor por uno binario. Para ello, se ha considerado a aquellas personas mayores de edad (mayor de 18 años) como valor 1 y aquellas menores de edad como 0.

Por último, el campo ‘PClass’ comprende las distintas clases de las que dispone el transatlántico (‘1st’, ‘2nd’, ‘3rd’ o ‘Crew’). Para ajustar el valor de este campo se ha realizado un sesgo en función de la proporción de supervivientes y fallecidos que presenta cada una de las clases. Realizando un análisis sobre la distribución de supervivientes y fallecidos de cada clase del campo, hemos obtenido los siguientes resultados:

- Clase ‘1st’: esta clase presenta un total de 325 pasajeros, de los cuales el 62.46% logró sobrevivir al hundimiento, mientras que el 38.54% restante no logró sobrevivir. Como este valor para el campo presenta un mayor porcentaje de supervivientes que de fallecidos se ha considerado que el valor de esta clase se sustituirá por valor 1.
- Clase ‘2nd’: esta clase presenta un total de 285 pasajeros, de los cuales solo el 41.40% logró sobrevivir y el otro 58.60% no lo logró. Por lo tanto, al presentar un mayor porcentaje de fallecidos que de supervivientes se ha considerado sustituir el valor de esta clase por un valor de 0.
- Clase ‘3rd’: esta clase presenta un total de 706 pasajeros, de los cuales solo el 25.21% logró sobrevivir y el otro 74.78% no lo logró. Por lo tanto, al presentar una gran diferencia entre el porcentaje de fallecidos y de supervivientes se ha considerado sustituir el valor de esta clase por un valor de 0.
- Clase ‘Crew’: esta clase presenta un total de 885 miembros del personal de la tripulación, al igual que en el caso anterior, existe una gran diferencia entre fallecidos y supervivientes, siendo estos de 76.05% y 23.95% respectivamente. Por lo tanto, se ha considerado que este valor del campo se sustituirá por un valor de 0.

```
In [22]: prueba._classInput()
Clase 1st, total personas: 325 número muertes: 122 número supervivientes: 203
Porce muertos: 37.53846153846154 / Porcen supervivientes: 62.46153846153846


Clase 2nd, total personas: 285 número muertes: 167 número supervivientes: 118
Porce muertos: 58.59649122807018 / Porcen supervivientes: 41.40350877192983


Clase 3rd, total personas: 706 número muertes: 528 número supervivientes: 178
Porce muertos: 74.78753541076487 / Porcen supervivientes: 25.21246458923513


Clase crew, total personas: 885 número muertes: 673 número supervivientes: 212
Porce muertos: 76.045197740113 / Porcen supervivientes: 23.954802259887007
```

Ilustración 6.1: Sesgo campo PClass

Por otro lado, todos aquellos valores de campo vacíos o nulos se han considerado como valor 0 para todos los campos anteriormente citados.

7 DISEÑO

A lo largo de este capítulo se pretende desarrollar el trabajo de diseño previo a la implementación del modelo. En este capítulo se detallará el diagrama de clases que implementará el proyecto de desarrollo del modelo y la arquitectura lógica del mismo.

7.1 DIAGRAMA DE CLASES

Este apartado mostrará el diagrama de clases que detallará la estructura de las clases que componen el sistema y como se relacionan entre ellas.

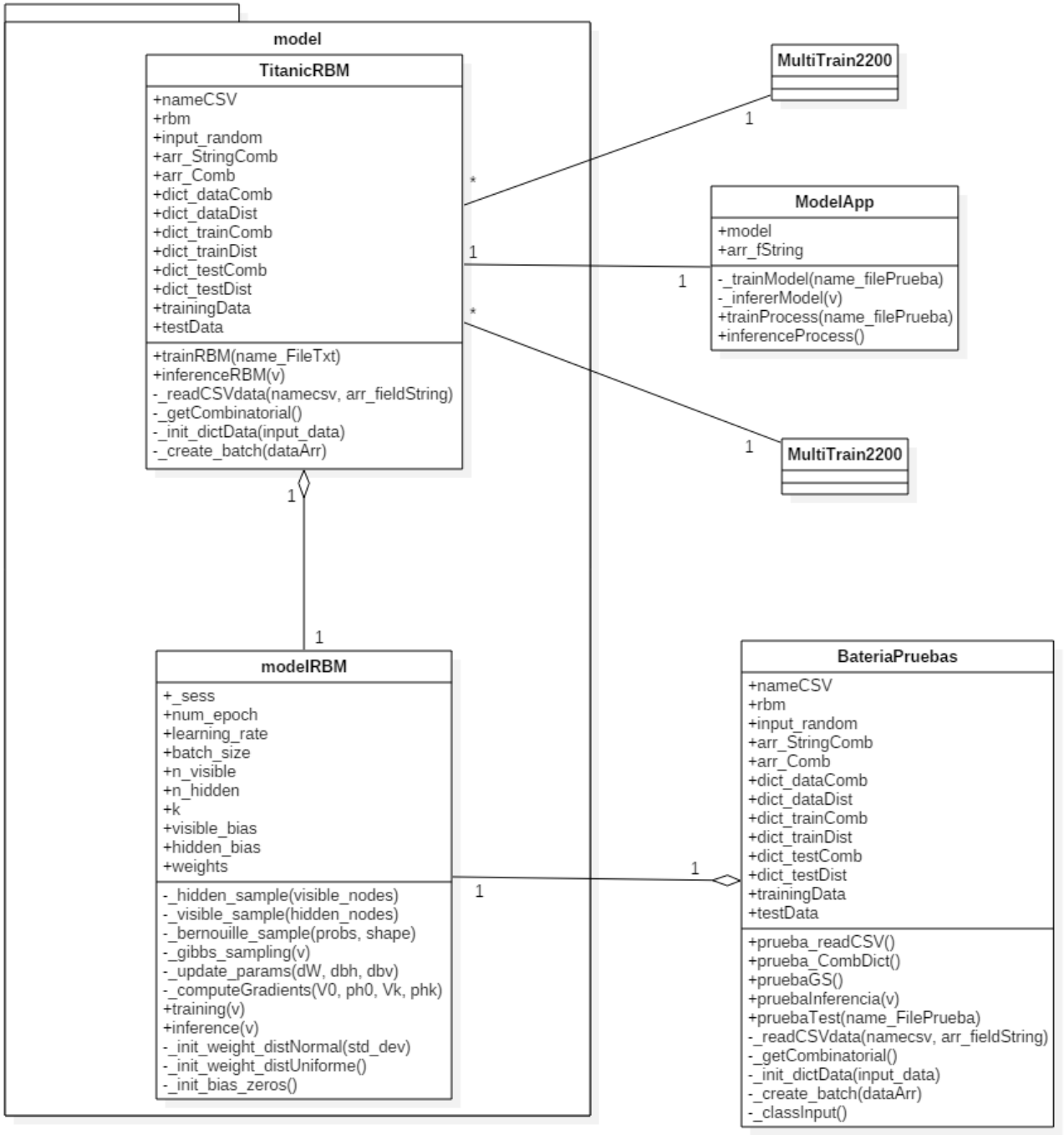


Ilustración 7.1: Diagrama de clases.

7.2 ARQUITECTURA FÍSICA

En este apartado se realizará la especificación de los componentes físicos que presenta el sistema y como se relacionan entre ellos. La arquitectura de este sistema mostrará cómo se comunica el cliente o usuario del sistema con el servidor cloud de Google GCP para ejecutar la batería de pruebas realizadas para llevar a cabo el estudio y análisis del modelo.

Esta plataforma proporciona una serie de recursos físicos y virtuales, ubicados en centros de datos de Google a lo largo del mundo, que permiten al usuario el desarrollo de aplicaciones cloud. Como se ha mencionado, los centros de datos de Google se dividen en las siguientes zonas: América del Norte, América del Sur, Asia, Australia y Europa, y que a su vez se dividen en sub-zonas; esta división por zonas proporciona varios beneficios como la redundancia en caso de fallos y una menor latencia. De esta forma, los recursos se dividen en tres niveles:

- Globales: Imágenes de discos pre-configuradas, instantáneas de disco y redes.
- Regionales: Direcciones IP estáticas externas.
- Zonales: Instancias de VM, los tipos de VM y los discos de VM.

En el siguiente diagrama se presenta la relación entre el alcance de los recursos.

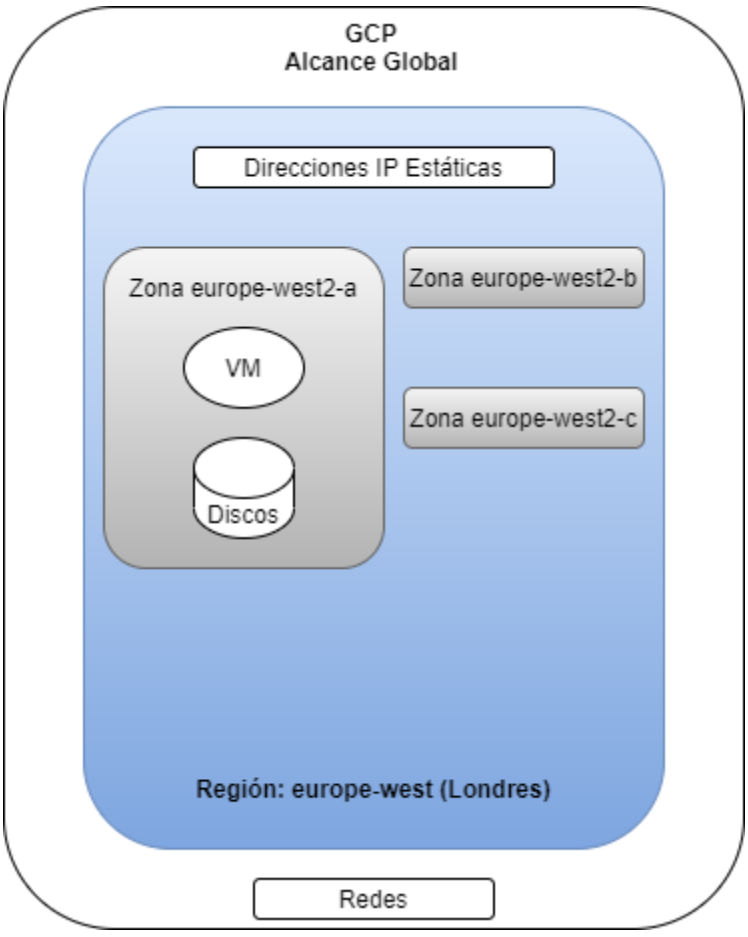


Ilustración 7.2: Estructura de los recursos de GCP.

Una vez explicado cómo se estructuran los recursos en GCP, se puede proceder a describir cómo se estructurará el entorno cloud empleado para el desarrollo del estudio y análisis del modelo, siguiendo la estructura mostrada en la siguiente imagen.

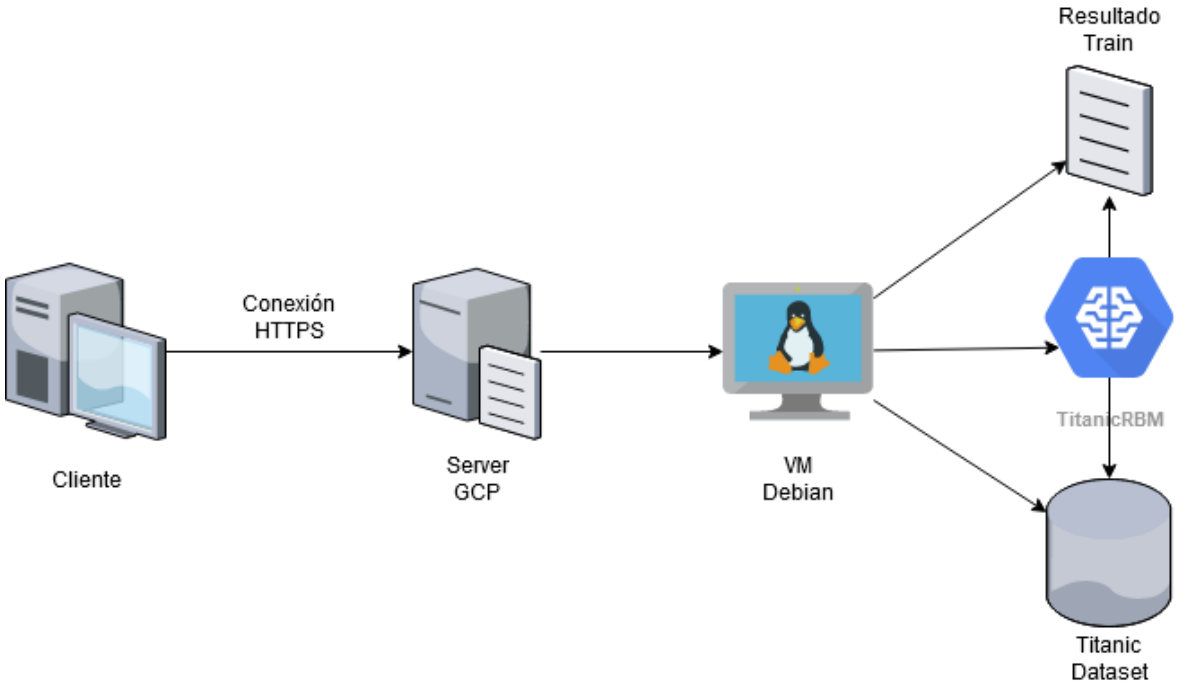


Ilustración 7.3: Arquitectura física.

Como se ha mencionado a lo largo de este documento, se ha configurado un entorno en esta plataforma para desarrollar la batería de pruebas necesarias para realizar el estudio del modelo desarrollado; la configuración de dicho entorno se explicará más adelante en el capítulo Instalación y configuración en Debian v.10. El entorno de desarrollo empleado ha sido una máquina virtual Debian v.10. situada en la zona 'europe-west2-a', por lo tanto, cuando el cliente solicite el acceso a dicha máquina virtual, el servidor de esta zona desplegará esta VM y mostrará al cliente una ventana emergente que le proporcionará un entorno de ejecución CLI (Command-Line Interface) que permite el acceso a los recursos almacenados en esta máquina virtual. Por último, a través de este entorno CLI el cliente podrá acceder tanto a los scripts Python y a los conjuntos de datos de entrenamiento como a los resultados obtenidos después de la ejecución de los scripts.

8 IMPLEMENTACIÓN

A lo largo de este apartado, se presentará una explicación del modelo investigado en este proyecto, las Máquinas Restringidas de Boltzmann incluyendo también una explicación de todos aquellos algoritmos y procesos que emplea en sus fases de entrenamiento y test. Posteriormente, se mostrarán aquellas posibles aplicaciones que presenta y como deben modificarse los algoritmos que emplea para poder implementarlas. Por último, se realizará una explicación de cómo se ha implementado el modelo de este proyecto, detallando así todas las herramientas y tecnologías utilizadas, además de la estructura de la aplicación.

8.1 RESTRICTED BOLTZMANN MACHINE

Este tipo de redes neuronales son un caso especial de máquinas de Boltzmann, que carecen de la complejidad de las conexiones entre neuronas de la misma capa generando así una complejidad mucho menor, manteniendo únicamente las conexiones entre las neuronas de las capas visible y oculta.

Por lo tanto, se asume que las neuronas ocultas serán de h_1 hasta h_m y las neuronas visibles serán de v_1 hasta v_n, siendo m y n los números totales de neuronas ocultas y visibles respectivamente. Cada una de las neuronas tendrá asociado un bias, denotado como $b_i^{(v)}$ o $b_j^{(h)}$, en función de la capa en la que se encuentre la neurona (v o h) y siendo i y j los números que representan su posición en su respectiva capa. Los pesos de las relaciones entre las neuronas de la capa visible y la oculta se representarán como w_{ij}, siendo i y j los números de las neuronas que representan en cada capa, además el valor de w_{ij} será igual al de w_{ji}.

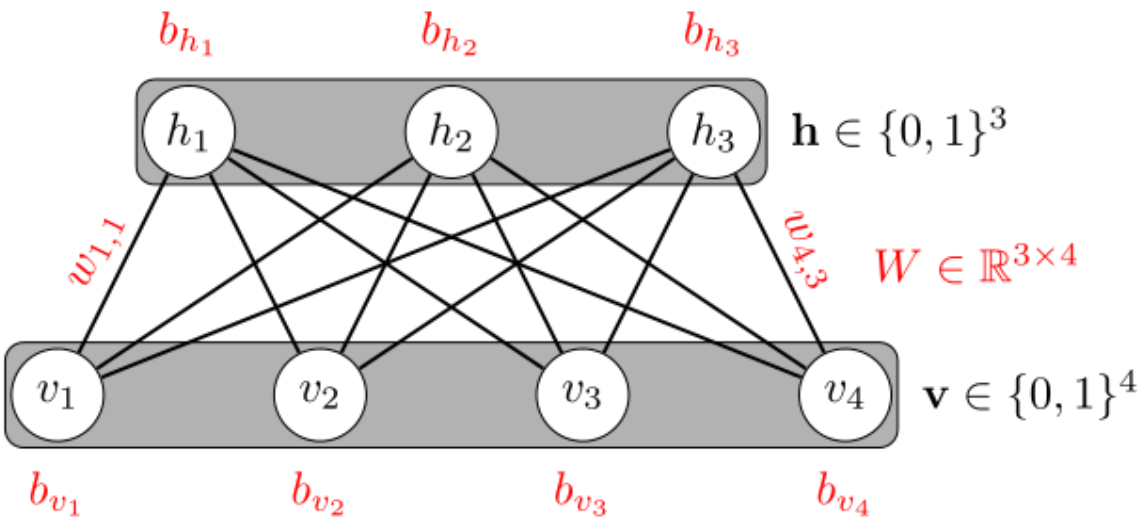


Ilustración 8.1: Estructura y parámetros RBM. Fuente: https://commons.wikimedia.org/wiki/File:Restricted-boltzmann-machine.svg

8.1.1 Proceso de aprendizaje

El proceso de aprendizaje de las RBMs es similar al de las Boltzmann Machine, para ello se emplea generalmente el algoritmo de Contrastive Divergence (CD). Además, es posible crear un algoritmo de aprendizaje eficiente mediante el procesamiento en lotes o batch de los datos del conjunto de entrenamiento. Como se ha mencionado anteriormente, el proceso de entrenamiento de este tipo de modelo se asemeja en gran medida al explicado en el apartado Máquinas de Boltzmann, por lo tanto, este proceso también presentará tres fases:

- El proceso inicia con el cálculo de las probabilidades de los estados de la capa oculta a partir de los datos introducidos en la capa visible obtenidos del conjunto de entrenamiento, repitiendo este proceso tantas veces como registros contenga el mini-batch. De esta forma, se obtiene el valor $\langle s_{i,} s_j \rangle_{data}$ de la Ecuación 2.17.
- Una vez calculadas estas probabilidades de los estados ocultos, se obtendrán las probabilidades visibles correspondientes a partir de estas probabilidades, repitiendo este proceso un número de 'k' iteraciones, para obtener así el valor de $\langle s_{i,} s_j \rangle_{model}$ de la Ecuación 2.17. Cada iteración del proceso anterior se conoce como Gibb Sampling.
- Por último, una vez obtenidos los dos valores antes mencionados se aplicará las siguientes ecuaciones para actualizar los valores de los pesos y bias del modelo. Los valores de $\langle s_{i,} s_j \rangle_{data}$ y $\langle s_{i,} s_j \rangle_{model}$ se corresponden con los valores $\langle v_{i,} h_j \rangle_{data}$ y $\langle v_{i,} h_j \rangle_{model}$ respectivamente en las siguientes ecuaciones:

$$w_{ij} \Leftarrow w_{ij} + \alpha \left(\langle v_{i,} h_j \rangle_{data} - \langle v_{i,} h_j \rangle_{model} \right)$$

Ecuación 8.1: Actualización pesos RBM.

$$b_i^{(v)} \Leftarrow b_i^{(v)} + \alpha \left(\langle v_{i,} 1 \rangle_{data} - \langle v_{i,} 1 \rangle_{model} \right)$$

Ecuación 8.2: Actualización bias de los estados visible RBM.

$$b_j^{(h)} \Leftarrow b_j^{(h)} + \alpha \left(\langle 1, h_j \rangle_{data} - \langle 1, h_j \rangle_{model} \right)$$

Ecuación 8.3: Actualización bias de los estados ocultos RBM.

Este proceso de entrenamiento requiere de una cierta cantidad de experiencia práctica para realizar una correcta toma de decisiones con respecto a ciertos parámetros del modelo, por ejemplo: el valor inicial de los pesos, el valor de la tasa de aprendizaje, el número de neuronas que componen la capa oculta, etc…

8.1.1.1 Contrastive Divergence

El algoritmo Contrastive Divergence es un algoritmo de aprendizaje que emplea una variante del método de estimación Maximum Likelihood junto con métodos de Markov Chain Monte Carlo (MCMC) para converger estimaciones no sesgadas. Hinton (2002) demostró que al juntar estos dos métodos en uno y realizar el proceso del MCMC durante unos pocos pasos, se soluciona el problema del método Maximum Likelihood para realizar estimaciones de promedios con tendencia exponencial en campos aleatorios de Markov y el problema del tiempo de convergencia en estimaciones no sesgadas del MCMC.

Como se mencionó en el apartado anterior, este algoritmo se compone de 3 fases.

- Fase 1.
  En la primera fase del proceso, se utiliza cada uno de los registros que comprenden el batch procesado para establecer el valor de los estados visibles de la red. Estos estados se representarán como v_i. A partir de estos valores de los estados visibles, la red obtiene los valores de las probabilidades ocultas que se expresarán como $p(h_j = 1 \mid v)$, estas probabilidades vienen dadas por la siguiente ecuación:

$$p(h_j = 1 \mid v) = \sigma\left(b_j + \sum_h v_i w_{ij}\right)$$

Ecuación 8.4: Cálculo de la probabilidad del estado oculto 'j'.

  Además, aplicando el método MCMC a estas probabilidades, se obtendrá un valor para los estados ocultos del modelo h_j.

- Fase 2.
  En la segunda fase del modelo, una vez obtenido el valor de h_j, a partir del valor de estos estados se generará por medio del algoritmo Gibb Sampling un valor para las probabilidades visibles ($p(v_i = 1 \mid h)$), siguiendo la siguiente ecuación:

$$p(v_j = 1 \mid h) = \sigma\left(b_i + \sum_h h_j {w^T}_{ij}\right)$$

Ecuación 8.5: Cálculo de la probabilidad del estado visible 'i'.

  Al igual que en la fase anterior, con estas probabilidades se obtendrá un nuevo valor de v_i aplicando nuevamente MCMC. Este proceso se repetirá tantas iteraciones como se especifique en el valor de k; por lo general, emplear una única iteración k obtiene buenos resultados.

- Fase 3.
  Una vez finalizado el proceso de CD, el modelo ha obtenido el par de valores v_i, h_j a partir de los datos entrenamiento y el par v_i, h_j reconstruidos por el modelo, expresándose como $\langle v_i, h_j \rangle_{data}$ y $\langle v_i, h_j \rangle_{model}$ respectivamente. Este proceso se repite con cada uno de los registros del batch procesado para así calcular los gradientes con los que se actualizarán los parámetros de los pesos y bias, como se explicará posteriormente.

Gibbs Sampling

Este algoritmo es una de las técnicas de la cadena de Markov Monte Carlo o MCMC (Markov Chain Monte Carlo) con las que se realiza una inferencia bayesiana o una integración numérica a partir de una distribución probabilística dada por un modelo de datos. Por medio de este algoritmo, se podrá obtener los valores de $\{v_i,h_j\}_{model}$ a partir del valor de los estados visibles que se inicializaron con los registros del conjunto de entrenamiento.

Algoritmo
Como se ha mencionado anteriormente, este algoritmo es empleado en la segunda fase del proceso de CD para así obtener el valor de la reconstrucción de v_i, h_j.

Tras la obtención de las probabilidades $p(h_j = 1 \mid v)$, se les aplica el método MCMC para así generar un muestreo aleatorio de h_j. Con este muestreo, por medio de la Ecuación 8.5, se generarán las probabilidades $p(v_j = 1 \mid h)$ a las que se aplicará el MCMC, para obtener así los valores de los estados visibles (v_i) del modelo.

Este proceso se ejecuta tantas veces como valor entero tenga el parámetro k. Tras finalizar la iteración k del algoritmo, el modelo habrá sido capaz de generar la reconstrucción $\langle v_i, h_j \rangle_{model}$.

Cálculo de los Gradientes
El cálculo de los gradientes representa la tercera y última fase del algoritmo CD que proporciona al modelo el método necesario para realizar la actualización de los parámetros correspondientes a los pesos y bias de este mismo.

Este método emplea los valores obtenidos $\langle v_i, h_j \rangle_{data}$ y $\langle v_i, h_j \rangle_{model}$ para llevar a cabo, por medio de la Ecuación 8.1, la Ecuación 8.2 y la Ecuación 8.3.

8.1.1.2 Tasa de aprendizaje
El parámetro de la tasa de aprendizaje es muy importante dentro del proceso de entrenamiento del modelo. Si el valor de esta tasa es muy grande, el error de reconstrucción de los datos aumentará drasticamente y provocará que los valores de los pesos y bias se disparen.

Por otra parte, si su valor es reducido mientras el proceso de entrenamiento está en ejecución, el valor del error de reconstrucción suele caer de forma significativa. Esto no necesariamente puede significar algo bueno, debido a que el nivel de ruido en las actualizaciones de los pesos estocásticos es menor. Además, suele implicar una reducción en la velocidad de aprendizaje a largo plazo. Sin embargo, llegado el final del aprendizaje, suele ser conveniente reducir la tasa de aprendizaje puesto que realizar varias actualizaciones es una alternativa para eliminar parte del ruido del valor final de los pesos.

Una buena práctica para establecer el valor de la tasa de aprendizaje consiste en realizar un histograma de las actualizaciones de los pesos y otro para los propios pesos, de esta forma se puede comprobar que las actualizaciones sean del orden de magnitud de 10^{-3} el de sus respectivos pesos. Así, cuando una unidad presente un gran abanico de valores, las actualizaciones deben ser muy pequeñas, ya que muchas pequeñas actualizaciones pueden invertir el signo del gradiente. Por el contrario, en el caso de los bias pueden ser mayores.

8.1.1.3 Inicialización del valor de los pesos y bias

El valor inicial del parámetro correspondiente a los pesos habitualmente se obtiene mediante la generación de pequeños valores aleatorios obtenidos por medio de una función Gaussiana de media cero con una desviación típica de 0.01. La inicialización de este parámetro a grandes valores provoca que el modelo obtenga al inicio una mayor velocidad de aprendizaje, pero puede generar un modelo final ligeramente peor. Por otro lado, se debe controlar que el valor inicial de los pesos del modelo no genere valores de las probabilidades ocultas muy cercanas a 0 o 1, ya que esto puede provocar una ralentización significativa del proceso de aprendizaje. Además, si las estadísticas empleadas para el aprendizaje son estocásticas, los valores de los pesos se pueden inicializar a 0, puesto que, esto provocará que el ruido en las estadísticas desvié las unidades ocultas unas de otras, incluso si tienen conectividades idénticas.

Con el fin de mejorar el proceso de aprendizaje, se puede inicializar el valor de los bias de las unidades visibles siguiendo la siguiente ecuación:

$$\log\left[{p_i}/{(1 - p_i)}\right]$$

Ecuación 8.6: Valor de inicialización bias de los estados visibles.

siendo p_i la proporción de vectores de entrenamiento en la que se encuentra la unidad 'i'. Mediante la inicialización de estos parámetros, el modelo evitará utilizar las unidades ocultas para activar la unidad 'i' con la probabilidad aproximada de p_i durante la primera etapa del proceso.

Debido a que algunas unidades ocultas se activan en raras ocasiones (su probabilidad de activación es mucho menor que 1), siendo más fáciles de interpretar que aquellas que se activan con más frecuencia, es necesario especificar un 'objetivo de dispersión' con el fin de lograr generarlas. Por esto, se emplea un factor de penalización que permita mejorar la probabilidad de activación de estas unidades ocultas, expresándose este como 'q'. Este factor 'q' viene determinado por la siguiente ecuación:

$$q_{new} = \lambda q_{old} + (1 - \lambda) q_{current}$$

Ecuación 8.7: Probabilidad real de activación de 'q'

donde $q_{current}$ es la probabilidad de activación de la unidad oculta en el batch actual. Además, el factor de penalización natural viene dado por:

$$Sparsity\ penalty \propto -p \log p - (1 - p)\log(1 - q)$$

Ecuación 8.8: Factor de penalización.

Al utilizar la probabilidad de objetivo de dispersión de 't', permite poder inicializar los valores del parámetro de los bias ocultos siguiendo la siguiente ecuación:

$$\log\left[{t}/{(1 - t)}\right]$$

Ecuación 8.9: Valor de inicialización bias de los estados ocultos.

De lo contrario, la inicialización de los bias ocultos puede establecerse a 0 o también a valores negativos grandes del orden de -4.

8.1.1.4 División de los registros de prueba en batch

El proceso de actualización de los valores de los pesos y de los bias de cada una de las neuronas se realiza después de estimar cada uno de los gradientes pertenecientes a los registros de entrenamiento. La realización repetida de este proceso aumenta la complejidad computacional del proceso, para evitar este aumento de la complejidad se puede dividir los datos de entrenamiento en bloques de N registros. De esta forma, se pueden almacenar los distintos gradientes de los registros de cada bloque para calcular la media de éstos y así solo actualizar los valores de los pesos y bias una única vez por bloque.

Un error grave a la hora de dividir el conjunto de entrenamiento en batchs cuando se utiliza un algoritmo de aprendizaje basado en el Stochastic Gradient Descent es dividir este conjunto de datos en bloques de gran tamaño. Una práctica recomendable a la hora de dividir un conjunto de entrenamiento que contiene un pequeño número de clases equiprobables en batchs, es establecer el valor del tamaño del batch al mismo número de clases equiprobables que presenta el dataset a dividir y que cada uno de estos batch contenga un ejemplo de cada clase. Esto reduce el error de muestreo al estimar el valor de los gradientes de todo el conjunto de entrenamiento.

8.1.1.5 Número de unidades ocultas

Para establecer un número sensato de unidades ocultas es mejor evitar las intuiciones que puedan derivarse del proceso de aprendizaje. La principal preocupación a evitar al establecer el número de unidades ocultas es el sobre-entrenamiento, por lo tanto, para establecer correctamente el número de unidades ocultas, se debe estimar el número de bits que se necesita para describir cada uno de los vectores de datos, multiplicar esta estimación por el número de casos de entrenamiento del conjunto de datos y utilizar un orden de magnitud menor del valor obtenido como número de unidades ocultas. Además, si se utiliza un objetivo de dispersión muy pequeño, se puede emplear un número mayor; por otro lado, si los casos de entrenamiento son muy redundantes, es necesario emplear un número menor.

8.1.1.6 Tipos de unidades

Las RBM emplean valores binarios tanto en las neuronas visibles como en las ocultas, a pesar de esto se pueden emplear datos no binarios como datos de entrenamiento. Para ello, se requiere adaptar el modelo para permitir manejar estos tipos de datos como valores para los estados visibles:

Softmax y unidades multinomiales

La función de regresión Softmax consiste en una forma de regresión logística que normaliza un conjunto de datos de entrada a un conjunto de valores que sigue una función probabilística cuya suma total de los valores tiene valor 1. Los valores de las salidas obtenidas por este tipo de entradas comprenden su valor dentro del intervalo [0,1]. Esto permite evitar la clasificación binaria y adaptar estas clases al modelo. Por esto se denomina a veces al Softmax como una regresión logística multinomial.

La probabilidad de activación de una unidad binaria viene dada por la siguiente función sigmoide del total de la entrada, x:

$$p = \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{e^{x} + e^{0}}$$

Ecuación 8.10: Probabilidad activación unidad multinomial.

La ecuación anterior puede generalizarse para los K estados alternativos presentes en este tipo de regresión:

$$p_j = \frac{e^{x_j}}{\sum_{i=1}^{K} e^{x_i}}$$

Ecuación 8.11: Generalización probabilidad de activación de unidades multinomiales.

Por medio de esta función, el modelo puede tratar apropiadamente aquellos datos de entrada que presentan K valores alternativamente que no estén ordenados de ninguna forma. Las unidades Softmax pueden ser interpretadas como un conjunto de unidades binarias cuyos estados están restringidos, de forma que sólo uno de los estados K tiene valor 1 y el resto valor 0. Siguiendo este procedimiento se puede interpretar que la regla de las unidades Softmax es idéntica a la regla de las unidades binarias, siendo la única diferencia el método por el cual calculan las probabilidades de los estados y se interpretan las muestras del conjunto de datos.

Unidades Gaussianas visibles y ocultas

El empleo de unidades logísticas a menudo genera una pobre representación cuando se emplean como datos de entrenamiento imágenes o coeficientes de Mel-Cepstrum para representar el habla, por ello, se puede emplear como unidades visibles lineales con ruido Gaussiano independiente. Al emplear este tipo de unidades visibles, es necesario modificar la ecuación de la energía de Hopfield:

$$E(v,h) = \sum_{i \in visible} \frac{\left(v_i - {b_i}^{(v)}\right)^2}{2\sigma_i^2} - \sum_{j \in ocultas} {b_j}^{(h)} h_j - \sum_{i,j} \frac{v_i}{\sigma_i} h_j w_{ij}$$

Ecuación 8.12: Modificación Energía Hopfield para unidades visibles Gaussianas.

donde σ_i es la desviación estándar del ruido Gaussiano para la unidad v_i.

Es posible calcular la varianza del ruido de cada unidad visible, pero el uso del algoritmo convencional de CD lo dificulta en gran medida. En muchas aplicaciones, es más sencillo normalizar cada componente de los datos para obtener una media cero y una varianza unitaria, para luego utilizar reconstrucciones sin ruido en la ecuación anterior.

La tasa de aprendizaje debe de ser del orden de una o dos magnitudes más pequeña que la tasa empleada cuando se utilizan unidades binarias. Se requiere que este valor sea pequeño porque no existe un límite superior al tamaño de un componente al realizar la reconstrucción, por lo tanto, si el valor de este componente aumenta mucho, provocará que los pesos adquieran un valor de aprendizaje muy grande también.

Además del caso anterior, también se puede aplicar el concepto anterior a las unidades ocultas, lo que empeora el problema en gran medida. Por lo tanto, se debe adaptar la Ecuación 8.12:

$$E(v,h) = \sum_{i \in visible} \frac{\left(v_i - {b_i}^{(v)}\right)^2}{2\sigma_i^2} - \sum_{j \in ocultas} \frac{\left(h_j - {b_j}^{(h)}\right)^2}{2\sigma_j^2} - \sum_{i,j} \frac{v_i}{\sigma_i} \frac{h_j}{\sigma_j} w_{ij}$$

Ecuación 8.13: Modificación Energía Hopfield para unidades Gaussianas.

En este caso también se requiere que el valor de la tasa de aprendizaje sea lo suficientemente pequeña como para evitar que los pesos adquieran valores muy grandes, al no existir tampoco un límite inferior al tamaño de un componente al realizar la reconstrucción.

Unidades binomiales

Una manera simple de emplear unidades que presenten ruido de tipo entero con un rango de valores de 0 a N consiste en crear N copias separadas de una unidad binaria y asignarlas a cada copia los mismos valores de los parámetros de pesos y bias. Al recibir todas las copias el valor total de entrada, todas ellas generarán la misma probabilidad de activarse y sólo se calcula una vez. El número esperado de copias activas se representa como N_p y la variante de este número será $N_p(1-p)$. Para valores de 'p' pequeños, ésta actúa como unidad de Poisson, pero conforme el valor de 'p' se aproxima a 1, la variancia tiende a reducirse de nuevo, lo que puede no ser deseable. Además, para valores pequeños de p, el crecimiento de ésta es exponencial en el total de entrada.

Una ventaja al emplear valores de pesos compartidos para sintetizar un nuevo tipo de unidad a partir de las unidades binarias es que las matemáticas que subyacen al modelo de RBM que emplea unidades binarias se mantiene sin cambios.

8.1.2 Aplicaciones de las RBM

En este apartado se estudiarán algunas de las posibles aplicaciones del modelo de red neuronal Restricted Boltzmann Machine. Este tipo de red neuronal puede ser empleada en una gran variedad de aplicaciones tanto en aplicaciones supervisadas como no supervisadas, además este modelo permite ser empleado para complementar otro modelo de red neuronal, por ejemplo, obtener una configuración de pesos y bias para un modelo Feedforward por medio de la realización de un pre-entrenamiento de los datos con un modelo de RBM o también para crear un modelo de Autoencoders, en los cuales se necesitan dos modelos de RBM para su implementación (para los procesos de compresión y descompresión de los datos).

A continuación, se detallarán algunas de las aplicaciones más comunes para las redes neuronales RBM.

8.1.2.1 Reducción de dimensionalidad y reconstrucción de datos

Una de las aplicaciones básicas de este tipo de modelo de red neuronal es la reducción de dimensionalidad y el aprendizaje no supervisado. Las neuronas ocultas de la red se encargan de capturar la estructura latente de los datos, con el fin de obtener una representación reducida de los datos de entrenamiento. Para comprender el proceso de reconstrucción de los datos, primero se debe comprender la equivalencia entre un modelo no dirigido de RBM y un modelo dirigido de un modelo tradicional de red neuronal.

A pesar de que las RBM son un modelo no dirigido, éste se puede "desdoblar" como un modelo dirigido en donde la inferencia sobre los datos ocurre en una dirección en particular. El modelo no dirigido de RBM se puede considerar como un modelo dirigido donde existen infinitas capas. Este desdoblamiento es particularmente útil cuando el número de neuronas visibles están establecidas a valores específicos, ya que al desdoblar la red, el número de capas se reduce hasta exactamente el doble del número de capas presentes en la red original. Por otro lado, sustituir el muestreo probabilístico discreto por unidades sigmoides continuas, permite al modelo operar como un autoencoder virtual que presenta tanto parte codificadora como descodificadora.

El modelo RBM emplea los mismos valores de pesos tanto para aprender $\bar{h}$ a partir de $\bar{v}$ como viceversa, aunque en el caso de $\bar{v}$ a partir de $\bar{h}$ se emplea la traspuesta de estos pesos, la única diferencia se encuentra en los bias, ya que se emplea diferentes valores para aprender $\bar{h}$ a partir de $\bar{v}$ que $\bar{v}$ a partir de $\bar{h}$. Por lo tanto, las ecuaciones se reescribirían de la siguiente manera:

$$\bar{h} \sim f\left(\bar{v}, \bar{b}^{(h)}, W\right)$$

Ecuación 8.14: Modificación probabilidad activación unidad oculta.

$$\bar{v} \sim f\left(\bar{h}, \bar{b}^{(v)}, W^T\right)$$

Ecuación 8.15: Modificación probabilidad activación unidad visible.

Siendo la función $f(\cdot)$ una función sigmoide al igual que en un modelo de RBM binario convencional, ésta constituye una gran variación con respecto a este tipo de modelos tradicionales. Si se sustituyen los estados visibles con valores del conjunto de entrenamiento, se pueden realizar únicamente dos iteraciones de estas operaciones de reconstrucción para reconstruir los valores de los estados visibles con aproximaciones de valores reales.

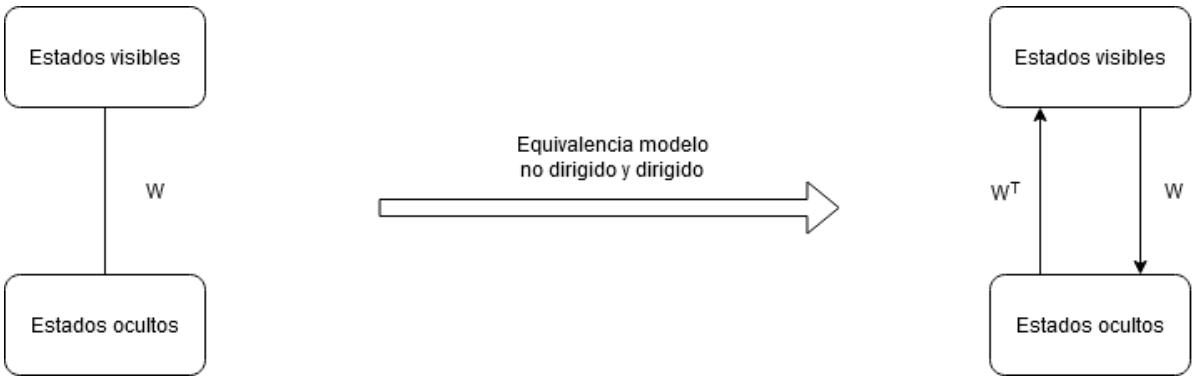


Ilustración 8.2: Equivalencia modelo dirigido y no dirigido.

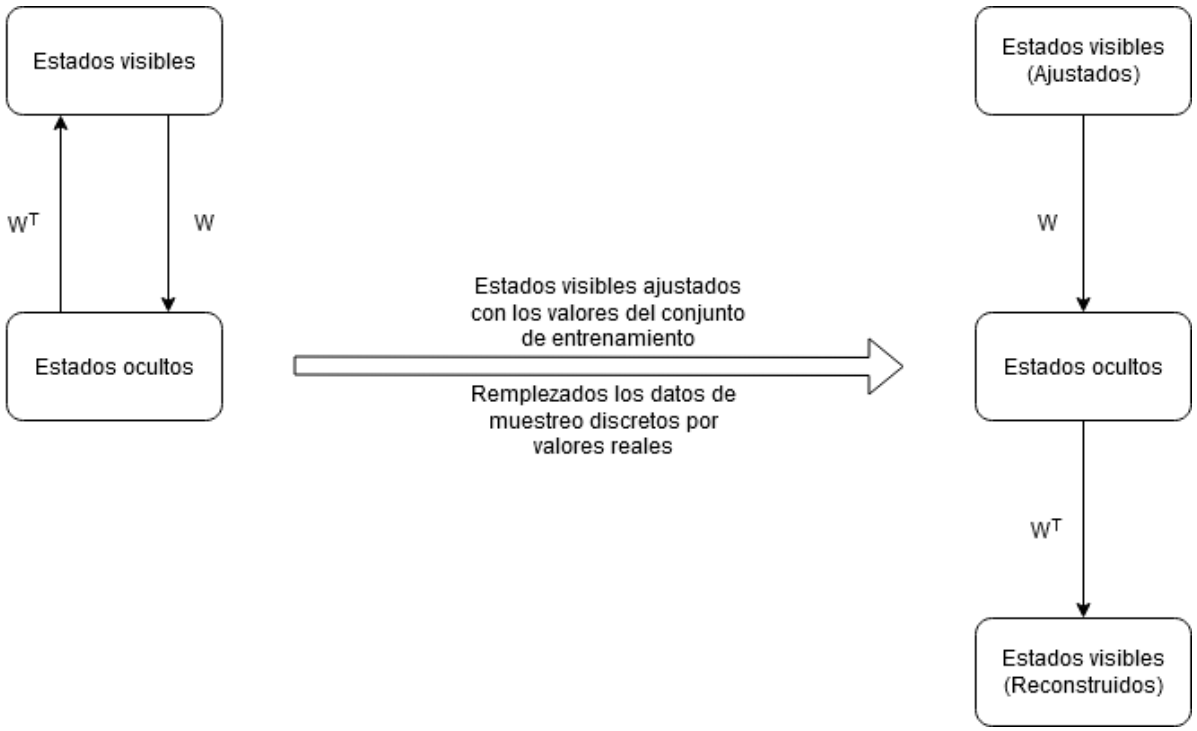


Ilustración 8.3: Reconstrucción modelo RBM como modelo convencional.

De esta forma, aproximando la red neuronal entrenada RBM a un modelo tradicional reemplazando los datos del muestreo discreto por activaciones sigmoides de valor continuo, obtenemos las siguientes modificaciones en las ecuaciones anteriores.

$$\bar{h} = f(\bar{v}, \bar{b}^{(h)}, W)$$

Ecuación 8.16: Probabilidad de activación unidad oculta para modelo RBM como modelo convencional.

$$\bar{v}' = f(\bar{h}, \bar{b}^{(v)}, W^T)$$

Ecuación 8.17: Probabilidad de activación unidad visible para modelo RBM como modelo convencional.

Nótese que de esta forma, el valor obtenido de $\bar{v}'$ es un valor real y representa la versión reconstruida de $\bar{v}$ siendo éste un dato del conjunto de entrenamiento. Por lo tanto, al no utilizar muestreos y que todos los cálculos se realizan en función a las previsiones, sólo es necesario realizar una iteración de los procesos de reducción y de reconstrucción. Además, estos procesos de reducción y reconstrucción se corresponden con las fases de compresión y descompresión de los modelos Autoencoders.

A primera vista puede resultar mucho más sencillo implementar un modelo de Autoencoder de la forma tradicional para alcanzar los mismos objetivos, pero emplear modelos entrenados de RBM apilados asemejando un modelo tradicional de red neuronal puede conllevar ciertas ventajas. El entrenamiento de los modelos de RBM apilados no suponen los mismos retos que los que pueden aparecer al entrenar un modelo de Deep Learning, como la desaparición y explosión de los gradientes. Además, proporciona unos excelentes puntos de inicialización para modelos Autoencoder de poca profundidad y de partida para modelos de Deep Learning de Autoencoders. Este principio condujo al desarrollo de modelos RBM empleados como métodos de pre-entrenamiento para otros modelos.

8.1.2.2 Filtrado colaborativo

Como se ha mencionado en el apartado anterior, las RBM se pueden emplear como modelado no supervisado de los Autoencoders y para reducción de dimensionalidad. Otra de las aplicaciones de la reducción de dimensionalidad, consiste en el filtrado colaborativo, el cual es una técnica de recomendación. Esta técnica fue uno de los componentes del conjunto de entrada ganadora del concurso de Netflix.

Uno de los principales retos de esta técnica, se basa en el uso de matrices de valoraciones con alguno de los valores sin especificar, lo que dificulta el diseño de la arquitectura de la red en mayor medida que con la arquitectura tradicional del modelo de reducción de dimensionalidad. En este apartado, se explicará cómo se puede crear una diferente instancia de entrenamiento y una red neuronal distinta para cada usuario dependiendo de calificaciones observadas por el usuario, además todas estas instancias distintas de RBM comparten entre ellas los valores de los pesos. En este tipo de aplicación, el principal problema es que los valores de las calificaciones de las matrices de datos están ponderados con valores de 1 a 5 y las RBM únicamente pueden emplean valores binarios como datos de entrada. Para solucionar este problema se emplean 5 unidades Softmax en las neuronas visibles, correspondientes a cada valor de la calificación. Cada una de las entradas de los datos se corresponde con una de las películas calificadas, que se componen a su vez de 5 unidades binarias, representando cada una de las valoraciones de las 5 posibles. Por lo tanto, el conjunto de entrenamiento se compondrá por ‘n’ número de registros y que a su vez cada uno se compondrán de una distribución probabilística Softmax, de notada por $v_i^{(1)}, \dots, v_i^{(5)}$.

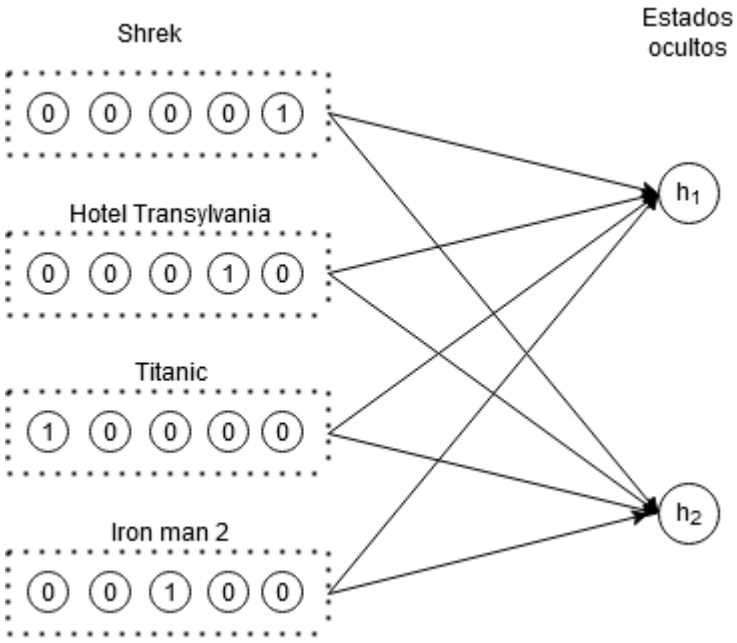


Ilustración 8.4: Ejemplo Collaborative Filtering.

Únicamente uno de los valores de ${v_i}^{(k)}$ puede ser 1, mientras el resto deben de ser 0. Además, el valor de los pesos que unen cada estado visible con cada uno de los 'm' estados ocultos va asociado al valor de la unidad Softmax, es decir, existen 5 pesos asociados a cada una de las películas, uno por valoración. Por lo tanto, los pesos se denotan de la siguiente forma ${w_{ij}}^{(k)}$. Por otro lado, las unidades visibles, al igual que los pesos, presentan 5 valores de bias, denotándose como ${b_i}^{(k)}$; por el contrario, los estados ocultos sólo tendrán un único bias, denotado como b_j.

La fase de entrenamiento de esta aplicación se asemeja al método de Gibb Sampling del modelo Restricted Boltzmann Machine, la única diferencia es la generación de los estados visibles a partir del modelo de las unidades Softmax. Con respecto al cálculo de los gradientes para la actualización de los parámetros, se corresponderán con la siguiente ecuación:

$$ {w_{ij}}^{(k)} \Leftarrow {w_{ij}}^{(k)} + \alpha \left(\langle {v_i}^{(k)}, h_j \rangle_{data} - \langle {v_i}^{(k)}, h_j \rangle_{model} \right) \ \forall k $$

Ecuación 8.18: Actualización pesos para Collaborative Filtering.

Por lo tanto, como se ha podido observar, cada usuario utiliza una instancia distinta de RBM en los datos, pero todas ellas comparten los mismos parámetros de pesos en aquellas unidades visibles que ambos hayan valorado.

8.1.2.3 Clasificación

Los modelos de RBM desarrollados para clasificación son empleados en la mayoría de los casos como módulo de pre-entrenamiento para otros modelos de redes neuronales. En estos casos, el modelo se desarrolla según se explicó en el apartado Reducción de dimensionalidad y reconstrucción de datos, consiguiendo de esta forma que el modelo se comporte como una red neuronal tradicional con unidades sigmoides, y cuyos pesos se derivan del modelo no supervisado del mismo en lugar del método backpropagation o propagación hacía atrás. La parte codificadora de la red RBM se une a una capa de salida que realiza la fase de clasificación, por otro lado, los pesos de este modelo se entrenan mediante el método de propagación hacía

atrás. Por medio de este ajuste del modelo de RBM se obtiene un modelo de Deep Learning para clasificación y, además, fue uno de los primeros enfoques para realizar una fase de pre-entrenamiento de un modelo de Deep Neural Network convencional.

Sin embargo, existe otro método alternativo que integra en mayor medida el entrenamiento y la inferencia del modelo RBM para desarrollar un modelo para clasificación. Este tipo de enfoque se asemeja al Collaborative Filtering explicado en el aparado Filtrado colaborativo y además proporciona una ayuda para desarrollar el modelo para clasificación.

La clasificación se puede considerar como una versión simplificada del problema de los registros sin especificar del filtrado colaborativo, donde los valores sin especificar pertenecen a una misma columna y que representa la clase, al contrario que en el filtrado colaborativo.

Para solucionar el problema de la clasificación empleando redes RBM, es necesario modificar la estructura de las mismas de la siguiente forma:

- La capa visible estará formada por dos tipos de unidades binarias, una parte de ellas representará las características de los registros y otra parte representa las distintas clases de los registros. Por lo tanto, existirán '*d*' nodos para las unidades de las características y '*k*' unidades para las clases. En caso de las últimas, sólo podrá tener valor '1' la unidad que represente el tipo de clase del registro. De esta forma, las unidades que representan las características se denotarán como $v_1^{(f)} \dots v_d^{(f)}$ y aquellas unidades que representan las unidades de las clases como $v_1^{(c)} \dots v_k^{(c)}$.
- Por otro lado, la capa oculta estará formada por 'm' unidades binarias, denotadas como $h_1 \dots h_m$

Según la estructura anterior, el modelo presentará dos matrices de pesos que conectará con la capa oculta, según a qué parte de las unidades visibles pertenezca la unidad. De esta forma, los pesos que conectan las unidades $v^{(f)}$ se denotarán como w_{ij} y las unidades $v^{(c)}$ como u_{ij}. En el caso de los bias visibles, también hay que distinguir en función de si pertenece a las unidades correspondientes a las características o a las de las clases, denotándose como $b_i^{(f)}$ y $b_i^{(c)}$ respectivamente; por otro lado, los bias ocultos se denotarán al igual que en los anteriores casos como b_j.

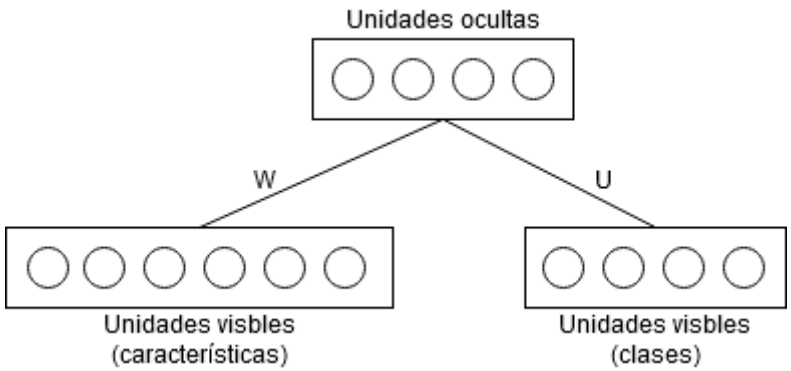


Ilustración 8.5: Estructura RBM para clasificación.

Con esta nueva estructura del modelo es necesario modificar las ecuaciones que permiten calcular los estados visibles y ocultos.

Para el cálculo de los estados ocultos, la ecuación resultante es la siguiente:

$$P\left(h_j = 1 \middle| \bar{v}^{(f)}, \bar{v}^{(c)}\right) = \frac{1}{1 + \exp\left(-b_j - \sum_{i=1}^{d} v_i^{(f)} w_{ij} - \sum_{i=1}^{k} v_i^{(c)} u_{ij}\right)}$$

Ecuación 8.19: Probabilidad estados ocultos.

Para calcular los estados visibles en este modelo hace falta distinguir entre los dos tipos de unidades visibles. En el caso de los estados visibles correspondientes a las características su definición se asemeja a la del modelo convencional de RBM, obteniendo la siguiente ecuación:

$$P\left(v_i^{(f)} = 1 \mid \bar{h}\right) = \frac{1}{1 + exp\left(-b_i^{(f)} - \sum_{j=1}^{m} h_j w_{ij}\right)}$$

Ecuación 8.20: Probabilidad estados visibles correspondientes a las características.

Por el contrario, el cálculo de los estados visibles asociados a las clases debe modificarse aplicando una función softmax a la sigmoide (Ecuación 8.11), obteniendo la siguiente ecuación:

$$P\left(v_i^{(c)} = 1 \mid \bar{h}\right) = \frac{exp\left(b_i^{(c)} + \sum_j h_j u_{ij}\right)}{\sum_{l=1}^{k} exp\left(b_l^{(c)} + \sum_j h_j u_{ij}\right)}$$

Ecuación 8.21: Probabilidad estados visibles correspondientes a las clases.

Las ecuaciones de actualización de los valores de los pesos también deben ser modificadas en función del tipo de unidad visible con la que se corresponda.

$$w_{ij} \Leftarrow w_{ij} + \alpha\left(\langle v_i^{(f)}, h_j \rangle_{data} + \langle v_i^{(f)}, h_j \rangle_{model}\right)$$

Ecuación 8.22: Actualización pesos unidades visibles (características).

$$u_{ij} \Leftarrow u_{ij} + \alpha\left(\langle v_i^{(c)}, h_j \rangle_{data} + \langle v_i^{(c)}, h_j \rangle_{model}\right)$$

Ecuación 8.23: Actualización pesos unidades visibles (clases).

A pesar de que este enfoque es una extensión directa del filtrado colaborativo, no optimiza totalmente la precisión de la clasificación, por lo que no necesariamente se va a obtener una mayor precisión solamente incluyendo las distintas clases a los valores de entrada. Por lo tanto, el proceso de entrenamiento debería centrarse en la optimización de la precisión, para ello, se suele utilizar un enfoque discriminatorio para la formación de la RBM en el que los pesos se entrenan para maximizar la probabilidad condicional de clase del verdadero valor de la clase. En un modelo convencional RBM, el proceso de entrenamiento se centraría en maximizar la probabilidad de ambas unidades visibles (características y clases), mientras, que en este tipo de enfoque la función objetivo pretende maximizar la probabilidad condicional de clase, $y \in \{1 \dots k\}\, P\left(v_y^{(c)} = 1 \mid \bar{v}^{(f)}\right)$, es decir, maximizar la precisión de la clasificación. Aunque es posible emplear el Contrastive Divergence en este enfoque, el problema se simplifica puesto que se puede estimar $P\left(v_y^{(c)} = 1 \mid \bar{v}^{(f)}\right)$ sin tener que emplear el enfoque iterativo del modelo convencional. El cálculo de la probabilidad condicional de clase se puede expresar de la siguiente forma:

$$P\left(v_y^{(c)} = 1 \mid \bar{v}^{(f)}\right) = \frac{exp\left(b_y^{(c)} \prod_{j=1}^{m}\left[1 + exp\left(b_j^{(h)} + u_{yj} + \sum_i w_{ij} v_i^{(f)}\right)\right]\right)}{\sum_{l=1}^{k} \exp\left(b_l^{(c)}\right) \prod_{j=1}^{m}\left[1 + \exp\left(b_j^{(h)} + u_{lj} + \sum_i w_{ij} v_i^{(f)}\right)\right]}$$

Ecuación 8.24: Probabilidad condicional de clase.

8.2 HERRAMIENTAS EMPLEADAS

En este apartado se describirán las distintas herramientas que han sido utilizadas para la realización del proyecto.

- Google Cloud Platform: es una plataforma propiedad de Google que proporciona los recursos físicos (Computadoras, máquinas virtuales o discos duros) necesarios para el desarrollo de proyectos cloud sin la necesidad de preocuparse por el mantenimiento de los recursos físicos. Algunos de los servicios que proporciona son los siguientes:
    - Procesamiento y Hosting.
    - Almacenamiento.
    - Base de datos.
    - Machine learning.
- Debian v.10: es una distribución del sistema operativo libre GNU/LINUX.
- GitHub: es una plataforma online basada en el Sistema de Control de Versiones (VCS) que proporciona un entorno en el que registrar los cambios realizados en un proyecto, con el fin de regresar a versiones anteriores en caso de ser necesario.
- Tensorflow: es un API Open Source para desarrollar modelos de machine learning en lenguajes como Python, JavaScript o Java. Esta herramienta proporciona una serie de módulos que permiten el desarrollo de dichos modelos de forma sencilla tanto para desarrolladores expertos como aquellos que se estén iniciando en esta tecnología. La versión empleada es la v.2.0.
- Numpy: es un API que proporciona al desarrollador las herramientas necesarias para la creación y gestión eficiente y sencilla de arrays multi-dimensionales, entre otras funciones. La versión empleada es la v.1.18.1.
- CSV: es un módulo de Python que proporciona al desarrollador las clases necesarias para la lectura y gestión de ficheros CSV de manera sencilla.
- OS: es un API de Python destinando a acceder desde código a funcionalidades del sistema operativo. Este API comprende algunas funcionalidades como lectura y escritura de ficheros, creación de directorios del sistema, manipulación del path del sistema o incluso creación de ficheros temporales.

8.3 TECNOLOGÍAS UTILIZADAS

En este apartado se describirán las distintas tecnologías y lenguajes de programación que han sido utilizados para la realización del proyecto.

- Python: es un lenguaje de programación de alto nivel de uso muy extendido. Es "interpretado", es decir no requiere de compilación previa para ser ejecutado, y que se centra en la legibilidad y simplicidad del código. Este lenguaje proporciona al desarrollador una flexibilidad y sencillez de uso, además de una gran cantidad de librerías de uso específico que permiten al desarrollador resolver una gran variedad de problemas. En este proyecto se emplea Python debido a su uso extendido en el campo de la A.I. y a la gran cantidad de librerías que proporciona para desarrollar dicho proyecto. La versión empleada es la v.3.7.6.
- Anaconda: es una plataforma que proporciona al desarrollador una herramienta que funciona como gestor de paquetes y de entornos, también proporciona distribuciones libres y open-source de los lenguajes Python y R. Esta plataforma contiene un gran número de paquetes y colecciones de los lenguajes anteriores. Además, incluye distribuciones para los sistemas Windows, Linux y MacOS. Para este proyecto emplearemos tanto para el sistema Windows como Debian la distribución Anaconda Individual Edition, empleando la versión v.4.8.1.
- Spyder: es un Entorno de Desarrollo Integrado o IDE que proporciona al desarrollador un entorno gratuito para el desarrollo de proyectos en el lenguaje Python. La versión empleada para este proyecto es la v.4.0.1.

8.4 IMPLEMENTACIÓN DEL MODELO

En esta sección, se presentarán las distintas clases y directorios que compondrán la aplicación desarrollada en este proyecto. La aplicación presenta la siguiente estructura de directorios:

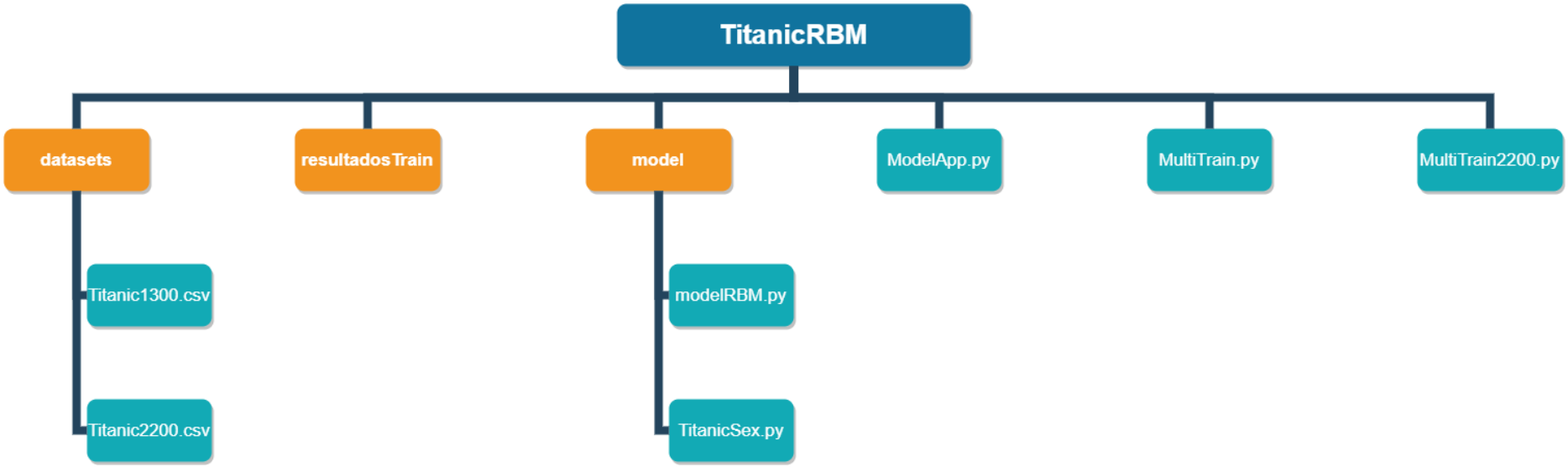


Ilustración 8.6: Estructura de directorios de la aplicación del proyecto.

- Datasets: contiene los dos ficheros CSV empleados como dataset con los cuales ejecutar los procesos de entrenamiento y test del modelo, obteniendo así los ficheros de pruebas necesarios para realizar el estudio y análisis del modelo.
- ResultadosTrain: en este directorio se almacenan todas las pruebas obtenidas tras la finalización de los procesos de entrenamiento y test realizados mediante la ejecución del fichero Main 'ModelApp.py'. Los ficheros que genera la aplicación se nombran de la siguiente forma:

Variable	Letra
Número neuronas visibles	V
Número neuronas ocultas	H
Número de épocas	E
Tasa de aprendizaje	LR
Tamaño del batch	BS

Tabla 8.1: Nomenclatura ficheros con resultados.

- Model: este directorio contiene las clases que implementan las principales funcionalidades del modelo de red neuronal, las clases que almacena son las siguientes:
    - ModelRBM.py: esta clase implementa todas las variables y funcionalidades necesarias para la ejecución de los procesos de entrenamiento, test e inferencia del modelo. Implementa los métodos de Gibb Sampling, el cálculo de los gradientes para actualizar los parámetros del modelo (pesos y bias), la generación del muestreo de Bernouille y el método de inferencia sobre los datos.
    - TitanicRBM.py: este fichero Python contiene los métodos necesarios para ejecutar correctamente los procesos de entrenamiento, test e inferencia del modelo de red neuronal, además de generar el fichero con los resultados de estos procesos y realizar el preprocesado del conjunto de datos empleado.
- MultiTrain.py: esta clase Python se encarga de ejecutar todos entrenamientos y test necesarios para llevar a cabo el estudio y análisis del modelo realizado, generando 60 ficheros de resultados para el dataset Titanic1300.csv.
- MultiTrain2200.py: este fichero Python se encarga de ejecutar todos entrenamientos y test necesarios para llevar a cabo el estudio y análisis del modelo realizado, generando 60 ficheros de resultados para el dataset Titanic2200.csv.
- ModelApp.py: este fichero Python contiene un programa Main que realiza un entrenamiento inicial, guardando los resultados en el directorio ResultadosTrain, y posteriormente muestra un pequeño menú que permite al usuario volver a entrenar el modelo o realizar inferencias sobre los datos del Titanic.

9 PRUEBAS

En este capítulo se presentarán las distintas baterías de pruebas realizadas para comprobar si existen posibles fallos en las distintas funcionalidades del proyecto. Este proceso se puede llevar a cabo por medio de dos tipos de técnicas:

- Técnicas de caja blanca o estructurales.
- Técnicas de caja negra o funcionales.

Para este proyecto, sólo emplearemos las pruebas de caja negra ya que únicamente queremos estudiar el comportamiento del sistema frente a una serie de entradas.

9.1 PRUEBAS DE CAJA NEGRA

A lo largo de este apartado, se van detallar la serie de pruebas de caja negra para estudiar el comportamiento de las distintas funcionalidades del sistema frente a una serie de entradas. Previamente se va a definir dichas pruebas.

Las técnicas de pruebas de caja negra son unas técnicas de testing en las cuales se prueban las distintas funcionalidades de una aplicación sin tener en cuenta la parte interna de dicha aplicación, es decir, se obvia la estructura del código, la arquitectura, los detalles de la implementación de los módulos, paquetes o rutas. Este tipo de pruebas se basan en los requisitos de la aplicación y de sus especificaciones técnicas. Por lo tanto, este tipo de pruebas se centran únicamente en las entradas, las salidas y si se realiza la acción.

Para realizar estas pruebas se deben seguir los siguientes pasos:

1. Realizar un análisis de requisitos y especificaciones técnicas.
2. El tester se encargará de diseñar una batería de entradas válidas y no válidas para el desarrollo de la prueba.
3. Basándose en las distintas entradas, el tester deberá asignar las salidas esperadas.
4. Diseñar los casos de prueba.
5. Ejecutar los casos de prueba.
6. Comparar la salida obtenida frente a la esperada.
7. En caso de que no coincidan las salidas, se procederá a la reparación del código.

Existen distintos tipos de pruebas de caja negra, a continuación, se presentarán las más conocidas:

- Particiones equivalentes: este tipo de técnica de caja negra consiste en diseñar y clasificar entradas para una funcionalidad en clases de equivalencias, las cuales son un conjunto de datos de entrada donde el comportamiento del software es igual para todos los datos, esperando que se procesen de la misma forma. Se pueden diseñar tanto para entradas válidas como para las no válidas. El criterio de clasificación se determina en función de las salidas, funcionalidades, valores internos, etc.
- Análisis de valores límite: esta técnica se basa en la evidencia experimental de que los errores suelen presentar mayor probabilidad de aparición en los casos extremos de los campos de entrada. Para realizar esta técnica es necesario determinar previamente las

clases de equivalencia y elegir qué casos ejercitan estos valores límite, por lo tanto, se considera que esta técnica complementa a la de particiones equivalente.

- Pruebas de casos de uso: este tipo de pruebas consisten en la realización de las pruebas en función de los distintos casos de usuarios del sistema. A menudo, estas pruebas presentan precondiciones que deben de haber sido cumplidas previamente. Tras finalizar la ejecución de las funcionalidades se analizarán los resultados obtenidos, los cuales se corresponden con las post-condiciones del caso de uso.
- Pruebas de historias de usuario: las pruebas de historias de usuario consisten, al igual que la anterior técnica, en la realización de las pruebas en función de las historias de usuario que presenta el sistema. Los casos de prueba de este tipo de técnicas emplean los criterios de aceptación de las historias de usuario para determinar la cobertura mínima de dichos casos.

Para este proyecto, se ha decidido emplear las historias de usuario como método para la realización de las pruebas puesto que se han empleado para realizar la especificación de las distintas funcionalidades de este proyecto.

9.1.1 Pruebas realizadas

En este apartado se especifican las distintas pruebas realizadas por medio del método de historias de usuario. Como se ha mencionado en el apartado anterior, se realizará las baterías de pruebas sobre las historias de usuario que presenta el proyecto y se comprobará el correcto funcionamiento del código en base a los criterios de aceptación de la historia sobre la historia que se realiza el caso de prueba.

Para realizar correctamente la batería de pruebas se ha empleado la siguiente configuración de la red neuronal:

Variable	Valor
Número de épocas	1
Número neuronas visibles	4
Número neuronas ocultas	2
Tamaño del batch	8
Tasa de aprendizaje	0.01
Variable k Gibb Sampling	1
Fichero CSV	Titanic2200.csv
Variables a emplear del fichero CSV	Sex, Age, Survived y PClass
Directorio de las pruebas	pruebasPrecision2200

Tabla 9.1: Configuración red neuronal batería prueba.

A continuación, se especificarán las distintas pruebas realizadas a cada una de las historias de usuario involucradas en la implementación del código del modelo de red neuronal RBM para probar si este código cumple los criterios de aceptación de las mismas.

El primer bloque de pruebas se corresponde a aquellas pruebas referentes a la historia de usuario US-01. Este bloque de pruebas comprende tanto la prueba de los procesos de entrenamiento y test, como las pruebas de todos aquellos métodos que intervengan en el proceso.

BBT-01	Prueba pre-procesado de los datos
Propósito	El propósito de esta prueba consiste en la comprobación del proceso de lectura del fichero CSV que contiene el conjunto de datos, realización del pre-procesado de esos datos y la división en batch de los conjuntos de datos de entrenamiento y test.
Pre-requisitos	Ejecución del fichero main.
Datos entrada	Nombre del fichero CSV. Listado con los nombre de las variables del fichero CSV a utilizar.
Resultado esperado	Listado con los datos a emplear del fichero CSV.
Resultado obtenido	Array con los datos necesarios del CSV.
Resultado de la prueba	Correcto.

Tabla 9.2: Prueba pre-procesado de los datos.

```
Valor dato de entrada: [0. 1. 1. 1.]
Valor dato de entrada: [1. 1. 0. 0.]
Valor dato de entrada: [0. 1. 1. 1.]
Valor dato de entrada: [1. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 0. 0. 1.]
Batch número: 80

Valor dato de entrada: [0. 1. 0. 1.]
Valor dato de entrada: [1. 1. 0. 1.]
Valor dato de entrada: [1. 1. 0. 1.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 1. 0.]
Valor dato de entrada: [1. 0. 0. 1.]
Valor dato de entrada: [0. 1. 0. 0.]
Batch número: 81

Valor dato de entrada: [0. 1. 0. 1.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 1. 1.]
Valor dato de entrada: [1. 1. 0. 1.]
Valor dato de entrada: [1. 1. 0. 1.]
Valor dato de entrada: [0. 1. 1. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [1. 1. 1. 1.]
Batch número: 82

Valor dato de entrada: [0. 1. 0. 1.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [0. 1. 0. 0.]
Valor dato de entrada: [1. 1. 1. 1.]
Valor dato de entrada: [1. 1. 1. 1.]
Batch número: 83

Valor dato de entrada: [1. 1. 0. 1.]
Valor dato de entrada: [0. 1. 0. 0.]
```

Ilustración 9.1: Resultados prueba pre-procesado de los datos.

BBT-02	Prueba inicialización de diccionarios.
Propósito	El propósito de esta prueba consiste en la comprobación de los procesos de inicialización de los diccionarios que capturan las distribuciones de los valores de los conjuntos de datos empleados y de las inferencias que generan, además de los procesos internos que requieren.
Pre-requisitos	Ejecución del fichero main.
Datos entrada	Nombre del fichero CSV. Listado con los nombre de las variables del fichero CSV a utilizar.
Resultado esperado	Listado con los valores de los diccionarios y array con las combinaciones.
Resultado obtenido	Listado con los valores de los diccionarios y array con las combinaciones.
Resultado de la prueba	Correcto.

Tabla 9.3: Prueba inicialización de diccionarios.

```
In [20]: prueba.prueba_CombDict()

Array string valores combinatoria
Valor: 1111
Valor: 0101
Valor: 0100
Valor: 1000
Valor: 1101
Valor: 1100
Valor: 0110
Valor: 1001
Valor: 0111
Valor: 0000
Valor: 1110
Valor: 0001
Valor: 0011
Valor: 1011
Array valores combinatoria
Valor: [1. 1. 1. 1.]
Valor: [0. 1. 0. 1.]
Valor: [0. 1. 0. 0.]
Valor: [1. 0. 0. 0.]
Valor: [1. 1. 0. 1.]
Valor: [1. 1. 0. 0.]
Valor: [0. 1. 1. 0.]
Valor: [1. 0. 0. 1.]
Valor: [0. 1. 1. 1.]
Valor: [0. 0. 0. 0.]
Valor: [1. 1. 1. 0.]
Valor: [0. 0. 0. 1.]
Valor: [0. 0. 1. 1.]
Valor: [1. 0. 1. 1.]
```

Ilustración 9.2: Resultados prueba inicialización de diccionarios (I).

```
Distribución data set completo
Valor key: 1111 distribución: 140
Valor key: 0101 distribución: 281
Valor key: 0100 distribución: 1211
Valor key: 1000 distribución: 17
Valor key: 1101 distribución: 176
Valor key: 1100 distribución: 105
Valor key: 0110 distribución: 118
Valor key: 1001 distribución: 27
Valor key: 0111 distribución: 57
Valor key: 0000 distribución: 35
Valor key: 1110 distribución: 4
Valor key: 0001 distribución: 24
Valor key: 0011 distribución: 5
Valor key: 1011 distribución: 1

Data set completo
Valor key: 1111 data set: [array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32),
array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1.,
1., 1., 1.], dtype=float32), array([1., 1., 1., 1.], dtype=float32), array([1., 1., 1., 1.],
```

Ilustración 9.3: Resultados prueba inicialización de diccionarios (II).

BBT-03	Prueba Gibb Sampling y Compute Gradients
Propósito	El propósito de esta prueba es la comprobación del correcto funcionamiento de los métodos del proceso de entrenamiento referentes a los algoritmos de Gibb Sampling y al cálculo de los gradientes.
Pre-requisitos	Realizar pre-procesado de los datos. Iniciar el proceso de entrenamiento.
Datos entrada	Ninguno.
Resultado esperado	Valores de v0, ph0, vK, phK, dw, dbh y dbv con formato correcto.
Resultado obtenido	Valores de v0, ph0, vK, phK, dw, dbh y dbv con formato correcto.
Resultado de la prueba	Correcto.

Tabla 9.4: Prueba Gibb Samplig y CD.

```
Prueba GS:
V0:
tf.Tensor(
[[ 0.32455558  0.7136673 ]
 [ 0.          0.        ]
 [ 0.          0.        ]
 [-1.0632198  -1.038819  ]], shape=(4, 2), dtype=float32)
PH0: [[-0.4141087   0.38851562]]
Prueba GS 2:
VK: [[ 1.]
 [ 0.]
 [ 0.]
 [-2.]]
PHK: [1.0, 0.0, 0.0, -1.0]
Prueba diff:
dW: [[0.32455558 0.7136673 ]]
dbh: [[0. 0. 0. 1.]]
dbv: [[0.73866427 0.32515165]]
```

Ilustración 9.4: Resultado prueba Gibb Sampling y CD.

BBT-04	Prueba procesos entrenamiento y test
Propósito	El propósito de esta prueba es la comprobación del correcto funcionamiento de los métodos de los procesos de entrenamiento y test.
Pre-requisitos	Creación de la red neuronal. Pre-procesado de los datos.
Datos entrada	Nombre del fichero TXT con los resultados.
Resultado esperado	Creación del fichero TXT. Visualización resultados de la precisión por pantalla.
Resultado obtenido	Creación del fichero TXT. Visualización resultados de la precisión por pantalla.
Resultado de la prueba	Correcto.

Tabla 9.5: Prueba procesos train y test .

```
La época actual es: 1
Precisión training: 0.49144473671913147
Fin proceso training
Precisión test: 0.20370370149612427
Precisión test: 0.32098764181137085
Precisión test: 0.37037035822868347
Precisión test: 0.4000000059604645
Precisión test: 0.4197530746459961
Precisión test: 0.4338624179363251
Precisión test: 0.43981480598449707
Precisión test: 0.45679008960723877
Precisión test: 0.4555555284023285
Precisión test: 0.44781142473220825
Precisión test: 0.4537036716938019
Precisión test: 0.4472934305667877
Precisión test: 0.4417989253997803
Precisión test: 0.4444444477558136
Precisión test: 0.4444444179534912
Precisión test: 0.4444444179534912
Precisión test: 0.44238677620887756
Precisión test: 0.4327484965324402
Precisión test: 0.43703699111938477
Precisión test: 0.4356260895729065
Precisión test: 0.4427609145641327
Precisión test: 0.43800315260887146
Precisión test: 0.4398147761821747
Precisión test: 0.4444443881511688
Precisión test: 0.45014238357543945
Precisión test: 0.4540465772151947
Precisión test: 0.45502638816833496
Precisión test: 0.4533843696117401
Precisión test: 0.45185181498527527
Precisión test: 0.4551970958709717
Precisión test: 0.4548610746860504
Precisión test: 0.4601570963859558
Precisión test: 0.4596949815750122
Precisión test: 0.4634920358657837
Precisión test: 0.4619341492652893
Precisión test: 0.45945945382118225
Precisión test: 0.4561403691768646
Precisión test: 0.45868948101997375
Precisión test: 0.4592592716217041
Precisión test: 0.4598012864589691
Precisión test: 0.46296295523643494
Precisión test: 0.4633936285972595
Precisión test: 0.4638047218322754
Precisión test: 0.4658436179161072
Precisión test: 0.46537843346595764
```

Ilustración 9.5: Resultados prueba procesos train y test (I) .

```
Precisión test: 0.466509073972702
Precisión test: 0.4691357910633087
Precisión test: 0.46863189339637756
Precisión test: 0.4703703820705414
Precisión test: 0.4698620140552521
Precisión test: 0.470085471868515
Precisión test: 0.46960169076919556
Precisión test: 0.4691358208656311
Precisión test: 0.4727272689342499
Precisión test: 0.46957674622535706
Precisión test: 0.46978560090065
Precisión test: 0.47126439213752747
Precisión test: 0.4682988226413727
Precisión test: 0.46666669845581055
Precisión test: 0.4675167202949524
Precisión test: 0.4689367115497589
Precisión test: 0.47031161189079285
Precisión test: 0.4699074327945709
Precisión test: 0.46951571106910706
Precisión test: 0.4691358208656311
Precisión test: 0.46876731514930725
Precisión test: 0.4678649604320526
Precisión test: 0.4685990810394287
Precisión test: 0.46931222081184387
Precisión test: 0.4715702533721924
Precisión test: 0.47170788049697876
Precisión test: 0.4718417227268219
Precisión test: 0.47497501969337463
Precisión test: 0.4745679199695587
Precisión test: 0.47417154908180237
Precisión test: 0.4761905074119568
Precisión test: 0.4743589758872986
Precisión test: 0.47304263710975647
Precisión test: 0.47268515825271606
Precisión test: 0.47325095534324646
Precisión test: 0.4719963073730469
Precisión test: 0.4730030596256256
Precisión test: 0.4753085970878601
Fin proceso test
```

Ilustración 9.6: Resultados prueba procesos train y test (II) .

La siguiente prueba se corresponde a la historia de usuario US-02, al igual que con el bloque de pruebas anterior, se han realizado tanto las pruebas de los métodos involucrados como la prueba de la funcionalidad de la historia de usuario.

BBT-05	Prueba Inferencia
Propósito	El propósito de esta prueba es la comprobación del correcto funcionamiento del método de inferencia que permite al usuario inferir sobre los datos de estudio después de realizar los procesos de entrenamiento y test.
Prerequisitos	Realizar el entrenamiento y test del modelo.
Datos entrada	Array con los valores referentes a un registro del conjunto de datos.
Resultado esperado	Visualización resultados obtenidos en la inferencia por pantalla (v inicia, ph0, v bernouille, phk).
Resultado obtenido	Visualización resultados obtenidos en la inferencia por pantalla (v inicia, ph0, v bernouille, phk).
Resultado de la prueba	Correcto.

Tabla 9.6: Prueba Inferencia.

```
In [14]: prueba.pruebaInferencia([1.0,0.0,1.0,-1.0])
Valor inicial v: [1.0, 0.0, 1.0, -1.0]
Valor phv: tf.Tensor([[0.5984902  0.34180015]], shape=(1, 2), dtype=float32)
Valor v bernouille: tf.Tensor([[0. 1. 1. 0.]], shape=(1, 4), dtype=float32)
Valor pvh: tf.Tensor([[0.3980682  0.6411466  0.22243929 0.34381396]], shape=(1, 4), dtype=float32)
```

Ilustración 9.7: Resultados prueba Inferencia.

10 ESTUDIO Y ANÁLISIS DEL MODELO

En este apartado, se ha realizado un estudio y análisis de los parámetros de la red con el fin de identificar cuáles de estos parámetros influyen de forma significativa en el aprendizaje del modelo.

Para realizar este estudio, se han realizado dos bloques de pruebas de los procesos de entrenamiento y test, cada uno de estos bloques con 60 pruebas. Cada bloque de entrenamiento, se corresponde con un conjunto de datos de entrenamiento, uno de ellos de 2201 registros y el otro de 1313 registros.

En cada una de las pruebas realizadas, se alternan los valores de las variables 'Número de neuronas ocultas', 'Tamaño del batch' y 'Tasa de aprendizaje', además del número de registros del conjunto de datos.

10.1 ESTRUCTURA FICHERO DE PRUEBAS

En este apartado, se definirá la estructura de los ficheros con los resultados generados por los procesos de entrenamiento y test del modelo.

Los ficheros han sido nombrados en función de los valores que tienen los parámetros de la red neuronal, por ejemplo, 'pruebaMultiV4H2E2000LR0.001BS16.txt'. En el ejemplo anterior, los valores se corresponden con los siguientes parámetros:

Nombre parámetro	Valores parámetro	Letra
Número neuronas visibles.	4 neuronas.	V
Número neuronas ocultas.	2 neuronas.	H
Número de épocas.	2000 épocas.	E
Tasa de aprendizaje.	0,001	LR
Tamaño del batch.	16 registros/batch	BS

Tabla 10.1: Valor de los parámetros de la prueba de ejemplo.

La estructura del fichero se divide en los siguientes apartados:

1. Valores iniciales de los parámetros de la red: en este apartado se muestran los valores de los parámetros expuestos en la tabla anterior y sobre los que se realizará el estudio con el fin de identificar cuáles de ellos son los más determinantes a la hora de lograr un mejor aprendizaje de la red.
2. Precisión obtenida después de la realización de la época actual: en esta sección del fichero se muestra la evolución de la red según se realiza el entrenamiento, pudiendo así observar la tendencia que presenta la red durante el entrenamiento con una configuración específica de parámetros.

3. Valores iniciales y posteriores al entrenamiento de los pesos y bias: en este apartado del fichero se puede observar y comparar los valores iniciales y finales de los pesos y bias.
4. Precisión proceso test: en esta sección se muestra la precisión obtenida del modelo con el conjunto de test tras finalizar el entrenamiento.
5. Distribución de la combinación de los valores de los datos de entrada en los distintos conjuntos de datos: este apartado permite comprobar si la red es capaz de reconstruir la misma distribución de valores en los datos de entrada después del entrenamiento, con el fin de comprobar cuán preciso es el modelo. Así se muestran las siguientes distribuciones:
    a. Distribución de valores en el conjunto de datos completo.
    b. Distribución de valores en el conjunto de datos de entrenamiento.
    c. Distribución de valores en la reconstrucción del conjunto de datos de entrenamiento.
    d. Distribución de valores en el conjunto de datos de test.
    e. Distribución de valores en la reconstrucción del conjunto de datos de test.

10.2 COMPARATIVA FICHEROS

Con el fin de realizar el estudio del modelo, se han realizado una serie de pruebas combinando valores de los distintos parámetros que influyen en el proceso de aprendizaje, Los parámetros a estudiar son los siguientes y presentan los siguientes valores:

Nombre del parámetro	Valores del parámetro
Número de registros	2201 registros o 1313 registros.
Número de épocas	2000 épocas.
Número de neuronas ocultas	2 neuronas, 4 neuronas u 8 neuronas.
Tamaño del batch	8 registros, 16 registros, 32 registros, 40 registros o 64 registros.
Tasa de aprendizaje	1,0, 0,1, 0,01 o 0,001.

Tabla 10.2: Valores de los parámetros a estudiar.

Otro parámetro que influye en el aprendizaje del modelo es el número de neuronas visibles de las que se compone la red neuronal. Este parámetro viene determinado por el número de variables de entrada que se introducen a la red. En nuestro caso al emplear el conjunto de datos del hundimiento del Titanic se han empleado únicamente 4 entradas, como se ha especificado en el apartado Modelado de datos.

La combinación de estos parámetros ha generado 120 pruebas, que se han dividido en dos bloques almacenados en dos directorios, en función del conjunto de datos que se ha empleado para la prueba.

Con el objetivo de realizar el estudio y análisis del modelo, se seleccionarán los 5 ficheros con los mejores resultados de cada bloque de entrenamiento y se analizará cuáles han sido los valores de los parámetros que mejores resultados han obtenido en las diferentes pruebas.

Para realizar la comparativa de los ficheros se han recolectado los siguientes datos de cada una de las pruebas:

- Precisión primera época de entrenamiento del modelo: representa la precisión del modelo tras la realización de la primera época del proceso de entrenamiento.
- Precisión máxima de modelo: este dato representa el valor de precisión máximo obtenido por el modelo durante la realización de este proceso.
- Época precisión máxima: este valor representa la época del proceso de entrenamiento en la que se obtuvo el valor máximo de precisión del modelo.
- Precisión modelo proceso test: representa el valor de precisión del modelo obtenido durante la realización del proceso de test.
- Precisión final modelo: este dato representa el valor de precisión del modelo obtenido tras la finalización del proceso de entrenamiento. Este valor ha sido extraído en aquellas pruebas que alcancen el valor máximo de entrenamiento en un número reducido de épocas, para comprobar la tendencia de aprendizaje de la prueba.

A continuación, se especificarán los distintos resultados obtenidos divididos por los bloques de entrenamiento mencionados anteriormente:

- Bloque del conjunto de entrenamiento de 1316 registros:

Número neuronas ocultas	Tasa de aprendizaje	Tamaño batch	Datos obtenidos				
2	0,001	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,455953	0,570752	1991	0,597285	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,443148	0,594685	1812	0,57251	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,468574	0,57762	1587	0,562761	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,480894	0,584312	1972	0,535096	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,425971	0,557437	1380	0,582243	
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,493661	0,608246	815	0,547636	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,473484	0,626613	1616	0,529581	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,499094	0,62197	1636	0,549683	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,467997	0,604776	1657	0,498717	

		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,492987	0,593048	1631	0,53395	
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,54426	0,581647	50	0,55876	0,554799
		32	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,549571	0,620761	421	0,59485	0,490981
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,531849	0,573594	1868	0,563829	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,512709	0,568457	798	0,572168	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,521038	0,54495	238	0,527596	
	0,1	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,506943	0,571108	869	0,606209	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,514624	0,579238	75	0,600649	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,503902	0,585476	814	0,592017	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,502926	0,569816	1300	0,556356	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,503352	0,550253	1283	0,563725	

Tabla 10.3: Resultados bloque de entrenamiento 1316 registro (I).

Número neuronas ocultas	Tasa de aprendizaje	Tamaño batch	Datos obtenidos			
4	0,001	16	Prec init	Precisión max.	Época max prec	Prec test
			0,486896	0,597742	1533	0,599421
		32	Prec init	Precisión max.	Época max prec	Prec test
			0,518159	0,618236	1675	0,560966
		40	Prec init	Precisión max.	Época max prec	Prec test
			0,46772	0,587404	361	0,593085
		64	Prec init	Precisión max.	Época max prec	Prec test
			0,477957	0,610785	1910	0,587072
		8	Prec init	Precisión max.	Época max prec	Prec test
			0,470888	0,560071	1380	0,582243
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test
			0,507833	0,627545	466	0,539027
		32	Prec init	Precisión max.	Época max prec	Prec test
			0,516776	0,640206	1816	0,552849
		40	Prec init	Precisión max.	Época max prec	Prec test
			0,488058	0,636647	1457	0,540896
		64	Prec init	Precisión max.	Época max prec	Prec test
			0,508361	0,579076	63	0,589529
		8	Prec init	Precisión max.	Época max prec	Prec test
			0,498186	0,576252	26	0,588961
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test
			0,556081	0,565731	1522	0,598856
		32	Prec init	Precisión max.	Época max prec	Prec test
			0,577678	0,625713	84	0,577741
		40	Prec init	Precisión max.	Época max prec	Prec test
			0,576577	0,608583	1246	0,603432
		64	Prec init	Precisión max.	Época max prec	Prec test
			0,566001	0,575083	651	0,552403
		8	Prec init	Precisión max.	Época max prec	Prec test
			0,507354	0,531051	1567	0,566448
	0,1	16	Prec init	Precisión max.	Época max prec	Prec test
			0,536177	0,581362	1262	0,587481
		32	Prec init	Precisión max.	Época max prec	Prec test
			0,539152	0,606324	219	0,569083
		40	Prec init	Precisión max.	Época max prec	Prec test
			0,55304	0,622876	158	0,590354

		64	Prec init	Precisión max.	Época max prec	Prec test
			0,502832	0,572073	696	0,557638
		8	Prec init	Precisión max.	Época max prec	Prec test
			0,523878	0,5482	1329	0,577341

Tabla 10.4: Resultados bloque de entrenamiento 1316 (II).

Número neuronas ocultas	**Tasa de aprendizaje**	**Tamaño batch**	**Datos obtenidos**				
8	0,001	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,494445	0,609813	225	0,595839	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,455456	0,598001	77	0,602092	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,490143	0,607997	464	0,586761	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,503354	0,600334	900	0,562553	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,474856	0,5612	68	0,582062	
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,547108	0,596353	22	0,611865	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,530643	0,636824	1488	0,555375	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,552063	0,606498	45	0,589718	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,514527	0,628166	923	0,561004	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,523227	0,611076	1513	0,58424	
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,534966	0,570645	713	0,596342	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,566534	0,594213	1519	0,588023	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,559685	0,581268	1523	0,58709	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,583559	0,605905	1396	0,607104	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,505573	0,534174	1099	0,57952	

	0,1	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,577517	0,613587	876	0,603758	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,581192	0,64732	1300	0,589285	
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,560864	0,584903	2	0,583148	0,560408
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,546059	0,595004	1940	0,565811	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,53479	0,561097	772	0,556644	

Tabla 10.5: Resultados bloque de entrenamiento 1316 (III).

- Bloque del conjunto de entrenamiento de 2200 registros

Número neuronas ocultas	Tasa de aprendizaje	Tamaño batch	Datos obtenidos				
2	0,001	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,47826	0,505833	173	0,45858	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,516155	0,543106	1740	0,469986	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,528156	0,564805	1169	0,531498	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,479974	0,518417	1108	0,450202	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,448435	0,481421	900	0,435626	
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,457833	0,50105	788	0,46998	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,481912	0,508352	505	0,459354	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,492943	0,52277	1606	0,455688	
		64	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,469489	0,510946	27	0,459424	0,484171
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,447182	0,477251	699	0,440476	
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,52022	0,566194	576	0,452804	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,496346	0,521874	1657	0,476294	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,497133	0,528272	786	0,461763	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,467997	0,604776	1657	0,498717	
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,470521	0,476947	81	0,434303	0,463469
	0,1	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,53939	0,576042	728	0,542331	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,479204	0,539034	1471	0,457614	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,522148	0,55308	1688	0,50889	

		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,488997	0,513923	153	0,463405	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,463022	0,515195	636	0,444003	

Tabla 10.6: Resultados bloque de entrenamiento 2201 (I).

Número neuronas ocultas	**Tasa de aprendizaje**	**Tamaño batch**	**Datos obtenidos**				
4	0,001	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,520258	0,555165	1433	0,599421	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,469374	0,52034	287	0,490043	
		40	Prec init	Precisión max.	Época max prec	Prec test	
			0,499364	0,533212	161	0,484001	
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,486205	0,523335	1048	0,481444	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,489475	0,538301	1557	0,496031	
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0.6136457	0,728566	1778	0,752576	0.718462
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,512651	0,554071	1370	0,469938	
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,489805	0,52822	21	0,483478	0,50146
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,461963	0,509652	1388	0,452766	
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,475658	0,499499	363	0,470017	0,472204
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,495498	0,499981	1888	0,446116	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,499318	0,511931	1686	0,467932	
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,504198	0,511058	44	0,470688	0,508098
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,488979	0,510244	1428	0,469388	
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,451961	0,475909	108	0,454144	0,455648
	0,1	16	Prec init	Precisión max.	Época max prec	Prec test	

			0,496342	0,507315	693	0,587481	
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,521117	0,536136	1424	0,469068	
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,507914	0,513262	4	0,45992	0,480538
		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,572004	0,650758	1815	0,59152	
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,593195	0,657574	1399	0,62522	0,648446

Tabla 10.7: Resultados bloque de entrenamiento 2201 (II).

Número neuronas ocultas	Tasa de aprendizaje	Tamaño batch	Datos obtenidos				
8	0,001	16	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,454569	0,502701	234	0,442772	0,489026
		32	Prec init	Precisión max.	Época max prec	Prec test	
			0,581571	0,633446	1899	0,629694	
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,60193	0,652165	321	0,605928	0,6266
		64	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,5	0,554622	252	0,484345	0,526904
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,506121	0,53227	316	0,465167	0,5109
	0,01	16	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,477397	0,503882	8	0,463292	0,485781
		32	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,490037	0,512121	17	0,469126	0,486874
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,485105	0,523488	14	0,453287	0,495179
		64	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,599223	0,671135	715	0,596041	0,645007
		8	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,466047	0,476213	2	0,443562	0,45606
	1,0	16	Prec init	Precisión max.	Época max prec	Prec test	
			0,492816	0,502157	671	0,470892	
		32	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,520075	0,520075	1	0,454835	0,495075
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,510518	0,512093	950	0,46156	0,486159
		64	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,493491	0,508025	663	0,471704	0,485925
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,455415	0,473009	1447	0,454585	
	0,1	16	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,549913	0,554777	287	0,513147	0,531062
		32	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,555189	0,589548	321	0,505316	0,571818
		40	Prec init	Precisión max.	Época max prec	Prec test	Prec fin
			0,507444	0,517295	2	0,476465	0,481542

		64	Prec init	Precisión max.	Época max prec	Prec test	
			0,513794	0,546967	722	0,508389	
		8	Prec init	Precisión max.	Época max prec	Prec test	
			0,471577	0,474047	1417	0,438712	

Tabla 10.8: Resultados bloque de entrenamiento 2201 (III).

A continuación, se procederá a realizar la selección de las 10 pruebas, 5 pruebas de cada bloque, en función de los valores de los resultados obtenidos. Para realizar esta selección previa, se compararán los valores obtenidos del dato 'Precisión máxima' por cada una de las pruebas y se seleccionarán aquellas pruebas que hayan logrado un valor mayor en este dato. Por otro lado, en caso de ser necesario, se comprobará el valor del dato 'Precisión Test' con el fin de comprobar la evolución de los parámetros de la red durante el aprendizaje.

Con respecto al bloque de entrenamiento de 1313 registros, se ha obtenido la siguiente distribución en los valores de los datos anteriormente mencionados:

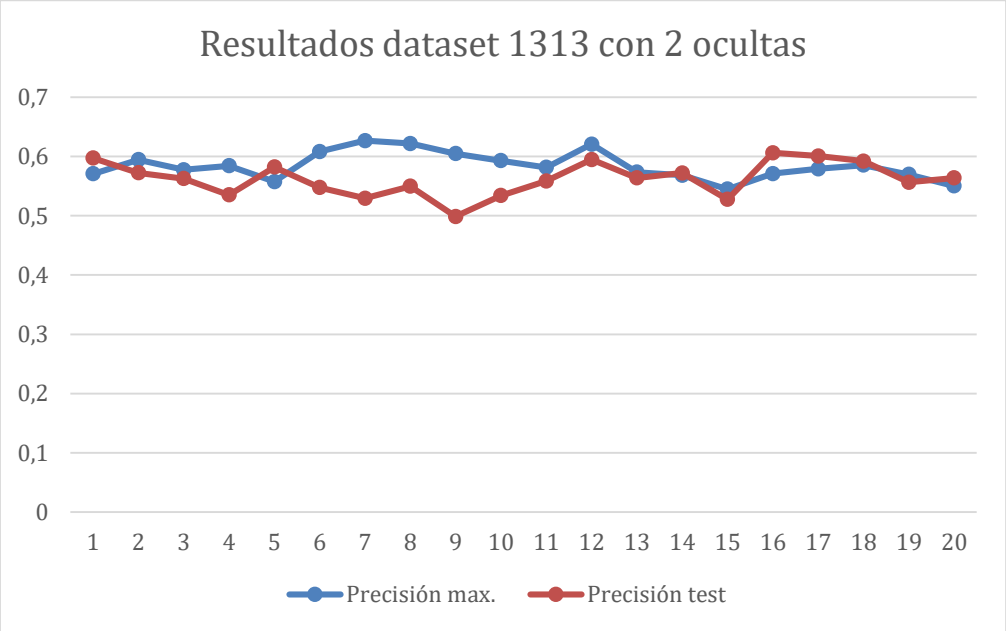


Ilustración 10.1: Gráfico resultados pruebas dataset 1313 registros con 2 neuronas ocultas.

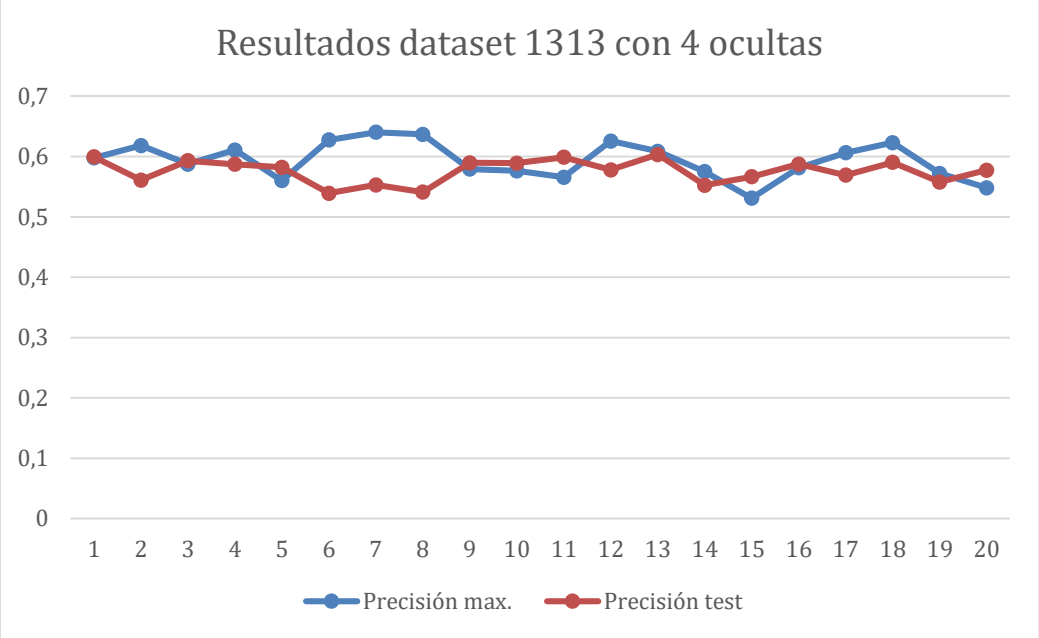


Ilustración 10.2: Gráfico resultados pruebas dataset 1313 registros con 4 neuronas ocultas.

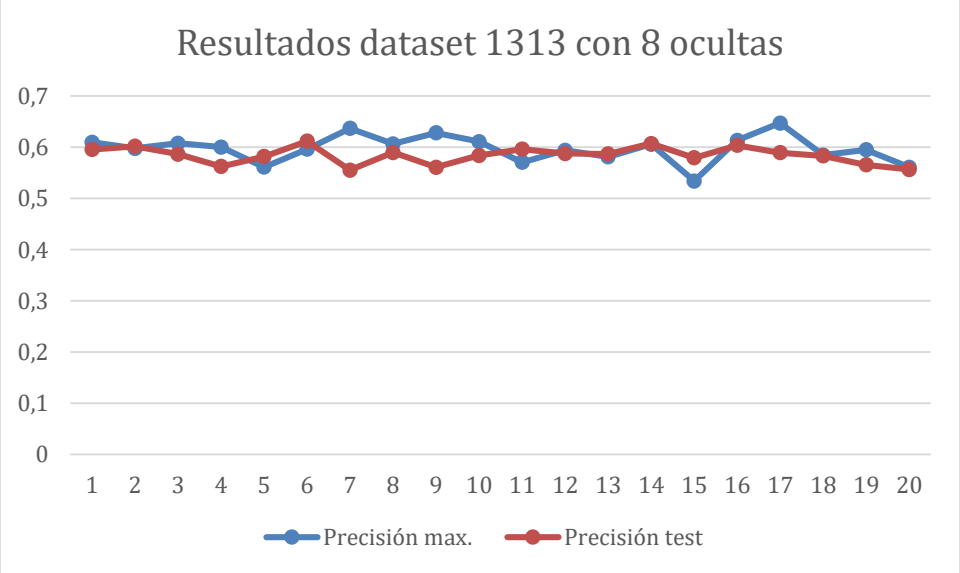


Ilustración 10.3: Gráfico resultados pruebas dataset 1313 registros con 8 neuronas ocultas.

Como se puede observar, los datos de estos resultados oscilan entre los valores “0,64732” y “0,531051”, para el dato ‘Precisión máxima’, y ‘’0,611865” y “0,498717” para ‘Precisión test’.

Las pruebas seleccionadas de este bloque de entrenamiento son las siguientes:

Número Ocultas	LR	Tamaño batch	Precisión init	Precisión max.	Época max prec	Precisión test
8	0,1	32	0,581192	0,64732	1300	0,589285
4	0,01	32	0,516776	0,640206	1816	0,552849
8	0,01	32	0,530643	0,636824	1488	0,555375
4	0,01	40	0,488058	0,636647	1457	0,540896
8	0,01	64	0,514527	0,628166	923	0,561004

Tabla 10.9: Mejores resultados bloque Titanic1300.csv.

Por otro lado, en el bloque de entrenamiento de 2201 registros se han obtenido la siguiente distribución en los valores de los datos:

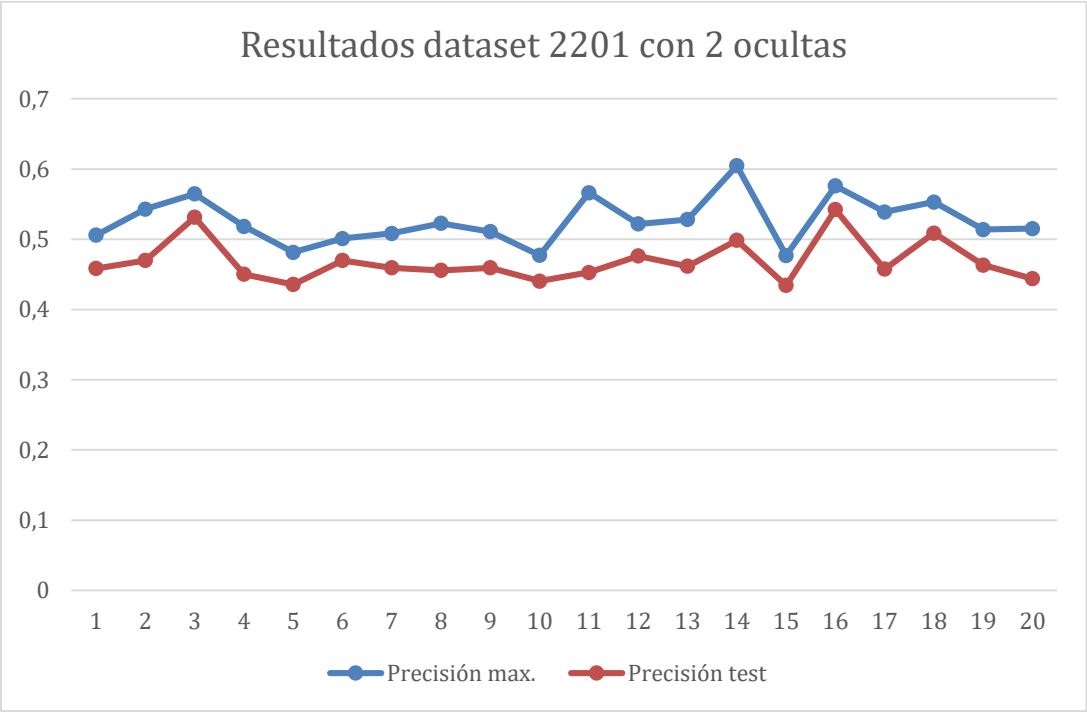


Ilustración 10.4: Gráfico resultados pruebas dataset 2201 registros con 2 neuronas ocultas.

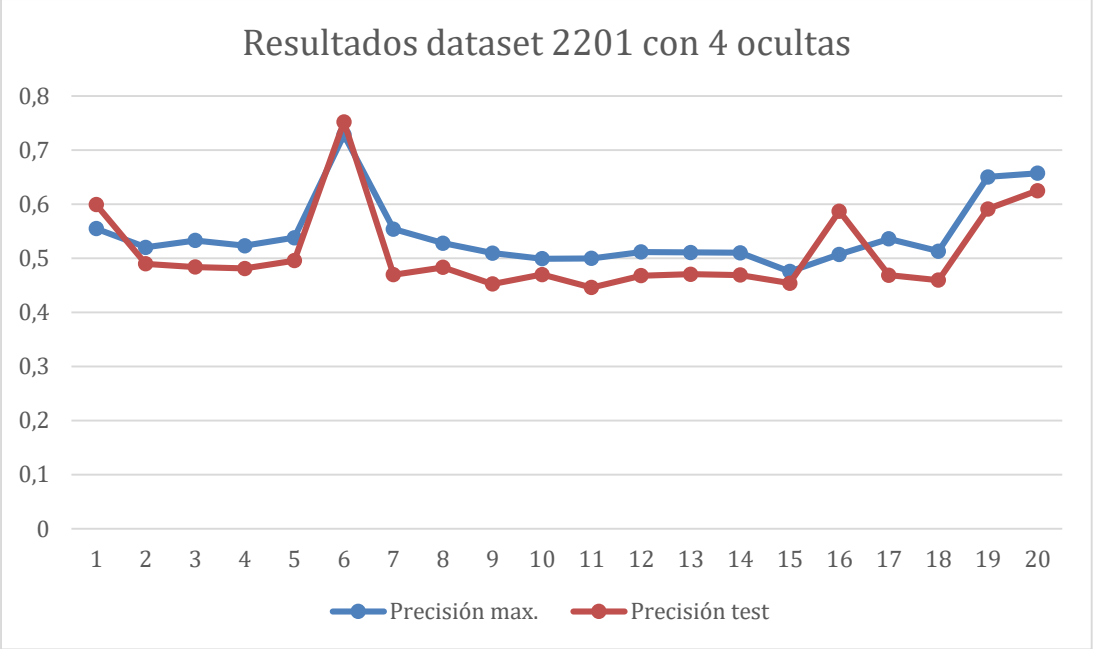


Ilustración 10.5: Gráfico resultados pruebas dataset 2201 registros con 4 neuronas ocultas.

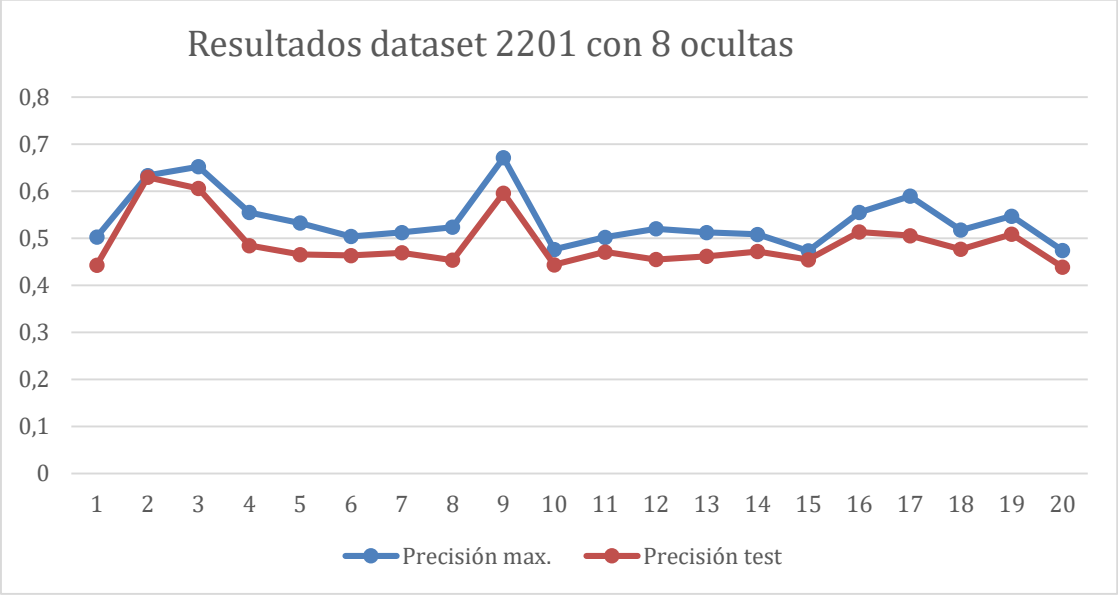


Ilustración 10.6: Gráfico resultados pruebas dataset 2201 registros con 8 neuronas ocultas.

Como se puede observar, al igual que en el bloque anterior, los datos oscilan entre los mismos valores, oscilando estos entre "0,728566" y "0,473009" para 'Precisión máxima' y para 'Precisión Test' entre "0,752576" y "0,434303". A excepción de la prueba que ha obtenido los mejores resultados del segundo bloque, el resto de pruebas oscilan entre los mismos valores que el bloque de pruebas anterior, siendo estos "0,671135" y "0,628166".

Las pruebas seleccionadas de este bloque de entrenamiento son las siguientes:

Número Ocultas	LR	Tamaño batch	Precisión init	Precisión max.	Época max prec	Precisión test
4	0,01	16	0.6136457	0,728566	1778	0,752576
8	0,01	64	0,599223	0,671135	715	0,596041
4	0,1	8	0,593195	0,657574	1399	0,62522
8	0,001	40	0,60193	0,652165	321	0,605928
4	0,1	64	0,572004	0,650758	1815	0,59152

Tabla 10.10: Mejores resultados bloque Titanic2200.csv.

Como se puede apreciar en los resultados seleccionados, la prueba que ha obtenido mejores resultados pertenece al bloque de entrenamiento de 2201 registros y presenta la siguiente configuración de parámetros:

Nombre del parámetro	Valores del parámetro
Número de neuronas ocultas	4 neuronas.
Tamaño del batch	16 registros.
Tasa de aprendizaje	0,01.

Tabla 10.11: Valores parámetros batería con mejor resultado.

Por lo tanto, se utilizará la configuración de parámetros de esta prueba como configuración base para la implementación de la aplicación del proyecto.

10.3 PARÁMETROS RELEVANTES

En este apartado, se estudiarán los resultados obtenidos con el fin de determinar que parámetros del modelo influyen en mayor medida para obtener mejores resultados a la hora de entrenar el modelo.

Este tipo de modelo de red neuronal, emplea un algoritmo de aprendizaje no supervisado para realizar el proceso de entrenamiento. Por este motivo es necesario exponer al modelo a grandes volúmenes de datos para realizar este proceso con el fin de que este modelo sea capaz de capturar las estructuras latentes de los datos y lograr una mejor precisión, como se mencionó en el apartado Aprendizaje no supervisado. Por lo tanto, se puede determinar que el número de registros disponibles en el conjunto de datos de entrenamiento influyen en gran medida en el proceso de aprendizaje del modelo. Además, en las pruebas realizadas se puede observar cómo el modelo es incapaz de obtener una precisión mayor por muchas iteraciones o épocas que se realicen en este proceso; a partir de cierta iteración, como consecuencia del tamaño reducido del conjunto de datos, el modelo no logra mejorar la precisión, provocando que el valor obtenido de esta precisión por el modelo se estanque dentro de un intervalo, alternando su valor dentro de este.

Como se puede observar en los resultados de las pruebas, la diferencia entre los resultados de un bloque y otro son mínimas, salvo en el caso de la prueba del bloque de 2201 registros mencionada anteriormente. Además, los mejores resultados obtenidos presentan valores

parecidos en los parámetros ‘Número de neuronas ocultas’ y ‘Tasa de aprendizaje’, siendo estos valores 4 y 8 neuronas ocultas, y 0.01 para la tasa de aprendizaje.

El número de unidades ocultas que presenta la red influye también en el proceso de aprendizaje puesto que estas unidades representan las características que presenta el conjunto de datos y que el modelo puede identificar. La errónea selección del valor de este parámetro puede provocar un sobre-entrenamiento del modelo, como se detalló en el apartado Número de unidades ocultas.

La tasa de aprendizaje es un parámetro muy importante en el proceso de aprendizaje del modelo ya que su valor puede provocar grandes fluctuaciones en los valores de los pesos y bias. Por lo general, como se mencionó en el apartado Tasa de aprendizaje, suele ser conveniente establecer el valor de esta tasa a uno reducido ya que la realización de muchas pequeñas actualizaciones pueden fácilmente invertir el valor del gradiente con el que se actualizan los valores de los pesos y bias.

En cuanto al tamaño del batch, éste tiene una menor relevancia a la hora de obtener mejores resultados en el proceso de aprendizaje puesto que no presenta un valor concreto que suponga una mejora relevante en los resultados de las pruebas. A pesar de que este parámetro no haya tenido una relevancia destacable en la batería de pruebas de entrenamiento realizada, sí cabe mencionar que influye en el proceso de aprendizaje del modelo ya que al dividir correctamente el conjunto de entrenamiento en pequeños lotes o batchs se reduce la complejidad computacional del cálculo de los gradientes, como ya se mencionó en el apartado División de los registros de prueba en batch.

La correcta inicialización del valor de los parámetros de los pesos y bias también influye en gran medida en el proceso de aprendizaje del modelo. La correcta forma de inicialización de estos parámetros se detalla en el apartado Inicialización del valor de los pesos y bias.

Todos y cada uno de los parámetros expuestos a lo largo de este apartado tienen una gran importancia en el proceso de aprendizaje del modelo, por lo que, la errónea configuración de éstos puede generar problemas a la hora de realizar este proceso.

Como muestran los resultados expuestos en el apartado anterior, salvo en el entrenamiento que obtuvo de valor máximo de precisión 72,8566% y una precisión final de 71,8462%, el resto de los entrenamientos presentan unos valores de ‘Precisión máxima’ comprendido entre [46-68]% y de ‘Precisión final’ entre [45-64]%. Esta escasez de mejora y la fluctuación en el valor de la precisión durante el proceso puede verse fuertemente influenciado por el número muy reducido de registros que componen el conjunto de entrenamiento, puesto que a pesar de aumentar considerablemente el número de épocas del proceso de entrenamiento (se incrementó a un valor 10000 iteraciones), el modelo fue incapaz de obtener un mejor valor de precisión. Por lo tanto, se puede inferir que el número de registro del conjunto de entrenamiento es el parámetro que influye más significativamente en el aprendizaje del modelo. Además, durante la investigación realizada sobre el modelo, se consultaron otras implementaciones de este tipo de modelo, en las cuales se empleaban conjuntos de datos cuyo tamaño es del orden del 10^6 registros; como es el caso de Artem Oppermann 2018, cuyos resultados tras la realización del proceso de entrenamiento, con un número muy reducido de épocas, obtuvo una precisión del modelo del 78%.

Por otro lado, la tasa de aprendizaje sí presenta un valor predominante en las pruebas con mejores resultados, este valor es ‘0.01’. De esta forma, se puede deducir que este valor de la tasa de aprendizaje influye en el proceso de aprendizaje.

El número de unidades ocultas también ha mostrado a través de las pruebas realizadas que aquellos procesos de entrenamiento que presentaba 2 unidades ocultas en su configuración han obtenido peores resultados que aquellas que tenían 4 u 8 unidades en la capa oculta.

Número Ocultas	LR	Tamaño batch	Precisión init	Precisión max.	Época max prec	Precisión test
8	0,1	32	0,581192	0,64732	1300	0,589285
4	0,01	32	0,516776	0,640206	1816	0,552849
8	0,01	32	0,530643	0,636824	1488	0,555375
4	0,01	40	0,488058	0,636647	1457	0,540896
8	0,01	64	0,514527	0,628166	923	0,561004

Tabla 10.12: Tabla mejores resultados dataset 1316.

Número Ocultas	LR	Tamaño batch	Precisión init	Precisión max.	Época max prec	Precisión test
4	0,01	16	0.6136457	0,728566	1778	0,752576
8	0,01	64	0,599223	0,671135	715	0,596041
4	0,1	8	0,593195	0,657574	1399	0,62522
8	0,001	40	0,60193	0,652165	321	0,605928
4	0,1	64	0,572004	0,650758	1815	0,59152

Tabla 10.13: Tabla mejores resultados dataset 2201.

11 CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentarán las conclusiones generadas tras la finalización del proyecto y los objetivos cumplidos. También se detallarán las posibles líneas de trabajo futuras que podrían realizarse empleando como base el proyecto desarrollado.

Una de las principales dificultades surgidas a lo largo del desarrollo del proyecto, ha sido el completo desconocimiento tanto del modelo y los algoritmos desarrollados y estudiados, como la falta de conocimientos sobre las tecnologías y herramientas necesarias para su desarrollo. Esto, junto con la escasez de tiempo disponible durante los primeros meses de desarrollo, propiciaron que el desarrollo del proyecto avanzará a un ritmo muy lento, como se puede observar en el apartado Sprint Backlog.

A lo largo de este documento, se ha detallado cómo este modelo requiere emplear valores binarios como datos de entrada, limitando en gran medida el número de conjuntos de datos disponibles. En el caso de emplear alguno de los tipos de valores descritos en el apartado Tipos de unidades, es necesario adaptar los algoritmos de los procesos de entrenamiento y test al tipo de dato empleado, aumentando considerablemente la dificultad y complejidad de estos procesos. Una posible solución para aumentar el número de conjuntos de datos disponibles, hubiera sido emplear unidades Softmax para adaptar aquellos datos a valores binarios; por este motivo, inicialmente se decidió emplear el conjunto de datos sobre el hundimiento del Titanic, ya que estos datos son fácilmente convertibles a datos binarios por medio de la realización de un sesgo de los valores.

Como consecuencia de esta elección, se generó un problema referente al tamaño de este conjunto. Puesto que este tipo de modelo emplea un algoritmo de aprendizaje no supervisado, requiere de volúmenes de datos del orden del millón o superior.

Por último, se presentó otra dificultad provocada por la complejidad computacional de los algoritmos empleados en el proceso de aprendizaje. Inicialmente se realizaron una serie de baterías de pruebas en el ordenador que se ha desarrollado el proyecto. Esta complejidad computacional provocaba que la realización de una prueba del proceso de aprendizaje con 2000 épocas concluyera tras aproximadamente 1 día de ejecución. Por este motivo, se decidió realizar la ejecución de la batería de pruebas necesarias para el apartado Estudio y análisis del modelo en la plataforma cloud GCP, con el fin de agilizar el proceso de entrenamiento. Se eligió esta plataforma porque Google proporcionaba una prueba gratuita de un año y un presupuesto de 300$ para el desarrollo de aplicaciones cloud.

11.1 CONCLUSIONES

La primera conclusión extraída al finalizar el TFG ha sido el gran abanico de áreas de trabajo que se pueden abarcar con el Machine Learning. Actualmente se generan cantidades abismales de datos de cualquier ámbito y tipo, que pueden ser empleados por un modelo de Machine Learning para aprender en base a ellos y facilitar muchos procesos de trabajo o incluso llegar a optimizar estos procesos.

Otra conclusión extraída tras la realización de las pruebas del proceso de aprendizaje es como la computación de procesos en cloud puede agilizar la realización de estos procesos. Las plataformas cloud proporciona el hardware necesario para realizar procesos que supongan una gran carga computacional en un menor tiempo, además de evitar al usuario el trabajo de mantener estos dispositivos en perfectas condiciones.

Por otro lado, al concluir la realización de las pruebas, se ha podido observar como la correcta selección de los datos de entrenamiento y su correcta gestión suponen un factor muy importante al desarrollar aplicaciones de este tipo. Además de esta conclusión, también se ha observado como la correcta configuración de los parámetros presentes en el proceso de aprendizaje suponen un factor muy influyente en el aprendizaje del modelo RBM, ya que su mala configuración puede suponer problemas como el sobre-entrenamiento de la red o una ralentización de este proceso.

11.2 LÍNEAS DE TRABAJOS FUTURAS

Aunque se ha implementado un modelo de red neuronal RBM completamente operativo y funcional, este proyecto presenta varias líneas de trabajo futuras empleando este modelo como base:

- **Dashboard para la aplicación principal**: como línea de trabajo a corto plazo se podría desarrollar una herramienta de visualización o Dashboard para la aplicación principal que permita visualizar los distintos resultados obtenidos por las pruebas de una forma cómoda y realizar nuevos procesos de entrenamiento o de inferencia.
- **Implementación unidades Softmax**: otra línea de trabajo futura a corto plazo sería la implementación de un método para el pre-procesado de los datos que permita convertir aquellos valores del conjunto de datos no binarios en unidades Softmax para así aumentar el número de conjuntos de datos que puede emplear el modelo.
- **Mejora del proceso Contrasctive Divergence**: como se ha explicado en el apartado Contrastive Divergence, este proceso se realiza en batch calculando cada uno de los errores de las reconstrucciones individualmente para cada uno de los registros del batch. Una de las principales ventajas que proporciona Tensorflow es la capacidad de realizar cálculos con matrices de grandes dimensiones, por este motivo, se podría modificar este método para realizar el cálculo de los errores de las reconstrucciones de todos los registros que componen el batch en una única iteración.

Con respecto a las posibles líneas de trabajo futuras a largo plazo se podrían implementar las siguientes aplicaciones:

- **Implementación Autoencoder**: como se explicó en el apartado Reducción de dimensionalidad y reconstrucción de datos, los modelos RBM pueden emplearse para realizar la implementación de un Autoencoder. Implementando la variante de RBM presentada en el apartado mencionado se puede implementar un Autoencoder capaz de comprimir y descomprimir los datos de entrenamiento.
- **Filtrado colaborativo:** una de las principales aplicaciones del modelo RBM y que ganó el premio del concurso de Netflix, es el Filtrado Colaborativo. La implementación de este modelo destinado a esta aplicación proporciona un buen punto de vista para entender el método de weight-sharing o compartición de los valores de los pesos.
- **Proceso de pre-entrenamiento para otro modelo de red neuronal:** las RBM presentan otra aplicación muy interesante como su implementación realizando un proceso de pre-entrenamiento de otra red neuronal, como por ejemplo una red Feedforward.

11.3 APRENDIZAJE PERSONAL

En esta sección se presentarán todos aquellos conocimientos obtenidos tras la finalización de este proyecto de desarrollo y que me serán de gran ayuda en mi futuro laboral.

En primer lugar, he mejorado y ampliado mis conocimientos sobre Machine Learning y algunos de los principales algoritmos de aprendizaje empleados en la actualidad en este área de trabajo, sobretodo destacando todos aquellos conocimientos referentes al modelo de red neuronal desarrollado obteniendo una evolución en este área muy positiva.

Además, se han obtenido unos conocimientos sobre las herramientas y tecnologías de desarrollo más utilizadas actualmente en Machine Learning, destacando Tensorflow y Numpy, permitiéndome embarcar en la realización de otros proyectos en este área de trabajo.

También se han visto reforzados mis conocimientos sobre el lenguaje de programación Python y más en concreto destinado al paradigma de Programación Orientada a Objetos, incluyendo también sus APIs CSV, OS y los anteriormente mencionados Numpy y Tensorflow.

De cara a la planificación, he mejorado mis conocimientos sobre las metodologías Ágiles y las ventajas que proporcionan a este tipo de proyectos de desarrollo, mencionando en especial, el marco de trabajo Scrum, bajo el que se ha desarrollado este proyecto.

Además de los conocimientos anteriores, la realización de este proyecto de investigación ha mejorado mis capacidades para identificar aquellas fuentes de datos fiables de las que no lo son del todo.

Por último, ha sido necesario aplicar algunos de los conocimientos aprendidos a lo largo de estos años de desarrollo en algunas de las asignaturas del grado para realizar ciertas partes del proyecto:

- Conocimientos de Machine Learning: los conocimientos proporcionados en la asignatura de **Sistemas Inteligentes** me han facilitado la comprensión de algunos de los algoritmos empleados a la hora de desarrollar este proyecto.
- Conocimientos de programación orientada a objetos: los conocimientos adquiridos en las asignaturas de **Programación Orientada a Objetos**, **Protocolos y Comunicaciones Seguras** y **Programación** y **Estructura de Datos** han facilitado el desarrollo de la aplicación principal del proyecto.
- Tratamiento de grandes volúmenes de datos: la realización de un correcto tratamiento y limpieza de los datos ha sido facilitada por los conocimientos impartidos en las asignaturas de **Programación Software Empresarial** y **Administración de Bases de Datos**, además de los conocimientos necesarios para manejar ficheros CSV y el tratamiento de estos en batch.
- Configuración del entorno de desarrollo cloud: los conocimientos adquiridos en las asignaturas de **Administración de Sistemas Operativos** y **Utilización de Sistemas Operativos** sobre la instalación de herramientas en sistemas GNU/Linux han sido de gran ayuda a la hora de instalar y configurar el entorno para el desarrollo de las pruebas del modelo.
- Documentación: para el desarrollo de los distintos apartados que componen este documento, han resultado de gran utilidad los conocimientos adquiridos en las asignaturas de **Programación Software Empresarial**, **Modelado del Software** y **Gestión de Proyectos basados en las Tecnologías de la Información**.

BIBLIOGRAFÍA Y WEBGRAFÍA

BIBLIOGRAFÍA

- Charu C. Aggarwal. Neural Network and Deep Learning. ISBN 978-3-319-94463-0.2018.
- A. Storkey. Increasing the capacity of a Hopfield network without sacrificing functionality. Artificial Neural Networks, pp. 451–456, 1997.
- Matt Harrison. Machine Learning Pocket Reference Working with structured data in Python. ISBN 978-1-492-04754-4.2019.
- Geoffrey E. Hinton. Training Products of Experts by Minimizing Constrastive Divergence. 2002.
- Geoffrey E. Hinton. Boltzmann Machine. 2007.
- Geoffrey E. Hinton. A Practical Guide to Training Restricted Boltzmann Machine. 2010.
- Miguel Á. Carreira-Perpiñán, Geoffrey E. Hinton. On Contrastive Divergence Learning.
- Yann LeCun, Yoshua Bengio, Geoffrey E. Hinton. Deep Learning. 2015.
- Kenneth Scerri. MCMC Methods for data modeling.
- Xiaojin Zhu. Semi-Supervised Learning with Graphs. May 2005.

WEBGRAFÍA

- 10 Powerful Examples of AI used in Healthcare Today. (Visitado 30/09/2020). https://getreferralmd.com/2019/04/10-powerful-examples-of-ai-used-in-healthcare-today/
- For the First Time, a Robot Passed a Medical Licensing Exam. (Visitado 30/09/2020). https://futurism.com/first-time-robot-passed-medical-licensing-exam
- TrackAI. (Visitado 29/09/2020). https://dive-medical.com/es/TrackAI.html
- Softmax Regression. (Visitado 20/09/2020). https://towardsdatascience.com/softmax-function-simplified-714068bf8156
- Binomial Distribution. (Visitado 20/09/2020). https://en.wikipedia.org/wiki/Binomial_distribution
- Poisson Distribution. (Visitado 20/09/2020). https://en.wikipedia.org/wiki/Poisson_distribution
- Netflix Prize. (Visitado 11/09/2020). https://www.netflixprize.com/community/topic_1537.html 11/09/2020
- Types of Machine Learning Algorithms You Should Know. (Visitado 15/07/2020). https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861
- AI, Machine Learning, Deep Learning Explained Simply. (Visitado 12/07/2020). https://towardsdatascience.com/ai-machine-learning-deep-learning-explained-simply-7b553da5b960
- A guide to machine learning algorithms and their applications. (Visitado 12/07/2020). https://www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html
- Unsupervised Machine Learning. (Visitado 13/07/2020). https://towardsdatascience.com/unsupervised-machine-learning-9329c97d6d9f
- Supervised Learning vs. Unsupervised Learning – A Quick Guide for Beginners. (Visitado 13/07/2020). https://www.analyticsvidhya.com/blog/2020/04/supervised-learning-unsupervised-learning/
- Probability concepts explained: Maximum likelihood estimation. (Visitado 16/11/2020). https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1
- Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning. (Visitado 08/09/2020). https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1
- Feed Forward Neural Network. (Visitado 13/11/2020). https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network
- Deep Belief Network. (Visitado 13/11/2020). https://www.sciencedirect.com/topics/engineering/deep-belief-network
- Estimación de la desviación estándar no sesgada. (Visitado 16/11/2020). https://es.wikipedia.org/wiki/Estimaci%C3%B3n_de_la_desviaci%C3%B3n_est%C3%A1ndar_no_sesgada
- Bayes' Theorem and Conditional Probability. (Visitado 17/11/2020). https://brilliant.org/wiki/bayes-theorem/
- El Ruido Gaussiano. (Visitado 17/11/2020). https://media4.obspm.fr/public/VAU/instrumentacion/observar/analizar/ruido-gaussiano/APPRENDRE.html
- Deep Learning meets Physics: Restricted Boltzmann Machines Part I. (Visitado 29/04/2020). https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15

- Deep Learning meets Physics: Restricted Boltzmann Machines Part II. (Visitado 30/04/2020), https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-ii-4b159dce1ffb
- TensorFlow Module tf. (Visitado 18/06/2020). https://www.tensorflow.org/api_docs/python/tf
- An Intuitive Introduction Of Boltzmann Machine. (Visitado 14/07/2020). https://medium.com/datadriveninvestor/an-intuitive-introduction-of-boltzmann-machine-8ec54980d789
- PathMind A.I. Wiki. (Visitado 13/11/2020). https://pathmind.com/wiki/index
- Las metodologías ágiles más utilizadas y sus ventajas dentro de la empresa. (Visitado 22/09/2020). https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/
- Metodología Scrum: qué es y cómo funciona. (Visitado 22/09/2020). https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html
- Escribiendo Historias de Usuario. (Visitado 23/09/2020). https://www.scrum.mx/informate/historias-de-usuario
- Gestión ágil de proyectos. Cómo pueden ayudar las metodologías ágiles a tu equipo de software. (23/09/2020). https://www.atlassian.com/es/agile/project-management
- Scrum, la guía definitiva. (Visitado 23/09/2020). https://jeronimopalacios.com/scrum/
- https://commons.wikimedia.org/wiki/File:Restricted-boltzmann-machine.svg
- Anaconda Individual Edition. (Visitado 09/11/2020). https://docs.anaconda.com/anaconda/
- Getting started with Anaconda. (Visitado 09/11/2020). https://docs.anaconda.com/anaconda/user-guide/getting-started/
- Welcome to Spyder's Documentation. (Visitado 20/08/2020). https://docs.spyder-ide.org/
- TensorFlow. (Visitado 20/08/2020). https://www.tensorflow.org/overview
- Numpy. (Visitado 21/08/2020). https://numpy.org/
- CSV - CSV File Reading and Writing. (Visitado 21/08/2020). https://docs.python.org/3/library/csv.html
- OS - Miscellaneous operating system interfaces. (Visitado 21/09/2020). https://docs.python.org/3/library/os.html
- UML Class Diagram Tutorial. (Visitado 20/09/2020). https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
- Descripción general de Google Cloud. (Visitado 06/07/2020). https://cloud.google.com/docs/overview
- Pruebas de Caja Negra. (Visitado 10/06/2020). https://howtotesting.com/testing-funcional/pruebas-de-caja-negra/
- TensorFlow. (Visitado 07/02/2021). https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/
- How to set up Anaconda under Google Cloud VM and transfer files on Windows. (Visitado 07/02/2021). https://medium.com/google-cloud/set-up-anaconda-under-google-cloud-vm-on-windows-f71fc1064bd7
- Descripción general de Google Cloud. (Visitado 06/03/2021). https://cloud.google.com/docs/overview?hl=es-419
- Main Types of Neural Networks and its Applications — Tutorial. (Visitado 06/03/2021). https://pub.towardsai.net/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

- 7 Types of Neural Network Activation Functions: How to Choose? (Visitado 07/03/2021). https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/

APÉNDICES

A lo largo de esta última sección, se redactarán los manuales de instalación y configuración tanto de los entornos de ejecución como paquetes u herramientas necesarios para la ejecución de los scripts de Python desarrollados en este proyecto. Por otro lado, se añadirá un apartado con un breve glosario, que contendrá aquellas definiciones de los términos utilizados a lo largo de este documento y que no han sido definidos en los apartados anteriores. Por último, se detallarán aquellos acrónimos utilizados en este documento, especificando los significados en inglés y castellano, y si existe su acrónimo en castellano.

A. MANUALES DE USUARIO

A lo largo de este apéndice se detallarán los pasos necesarios para preparar el entorno de ejecución necesario para los scripts desarrollados. De esta forma, se detallarán dos procesos diferentes de instalación y configuración, en función del sistema operativo en el cuál se ejecuten los scripts; los cuales serán para un sistema Debian y para un Windows 10, puesto que han sido los empleados para el desarrollo de este proyecto.

Instalación y configuración en Windows 10

En este manual se detallarán los procesos de instalación en un sistema Windows 10 de las herramientas que se han empleado para desarrollar este proceso.

Instalación y configuración Anaconda y Python 3

Como paso inicial, se debe instalar una herramienta que permita desarrollar y ejecutar scripts o aplicaciones Python, en el caso de este proyecto se ha decidido emplear Anaconda. Anaconda nos permitirá acceder y utilizar una gran cantidad de APIs y herramientas con las que desarrollar scripts de Python.

Se instalará la distribución Anaconda Individual Edition (https://www.anaconda.com/products/individual), se ha elegido esta distribución porque es OpenSource y nos proporciona con ella los ficheros correspondientes a Python 3.

Una vez descargado el instalador correspondiente a la versión de Windows que se necesite (32 o 64 bits), se procederá a realizar la instalación de Anaconda. Una vez finalizada la instalación de Anaconda, se habrán instalados los paquetes que viene incluidos con la distribución de Anaconda. Se puede confirmar que paquetes se han descargado junto con la instalación iniciando la herramienta Anaconda Navigator, selecionando la opción del menú lateral 'Environments', eligiendo el entorno de ejecución (por defecto al instalar Anaconda, solo se dispone de un entorno de ejecución llamado base) que se desea comprobar y elegimos en el desplegable visualizar los paquetes instalados, con la opción 'Installed'.

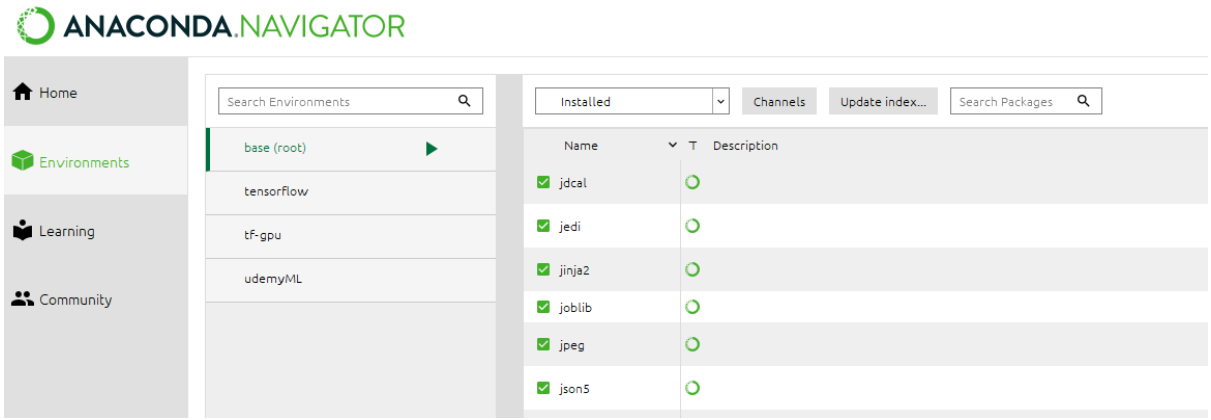


Ilustración A.0.1: Listado de paquetes Anaconda.

Instalación y creación de un entorno para TensorFlow

Para realizar la instalación de TensorFlow en nuestra distribución de Anaconda y la creación de un entorno de ejecución emplearemos la terminal Anaconda Prompt.

La creación de un entorno que permita la utilización del API de TensorFlow se puede realizar de dos formas distintas, en función del componente que emplea el sistema para realizar la ejecución, esta ejecución se puede realizar por medio de la CPU o con la GPU. En el caso de este proyecto se ha empleado una ejecución en GPU, puesto que se obtiene una mayor velocidad de computo a la hora de realizar cálculos con matrices de grandes dimensiones gracias a que los componentes GPU disponen de más núcleos para ejecutar estos scripts. Se recomienda utilizar la ejecución en GPU para este proyecto, ya que los cálculos y algoritmos que se emplean son complejos.

A continuación, detallarán los comandos necesarios para crear un entorno de ejecución tanto en CPU como GPU para la última versión de TensorFlow (2.0):

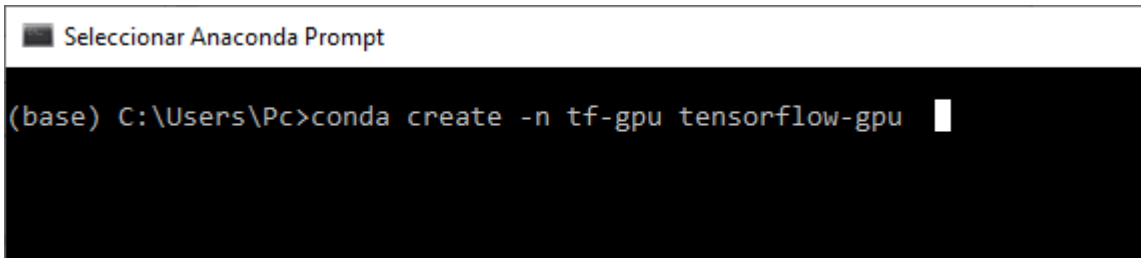


Ilustración A.2: Crear entorno TensorFlow en GPU.

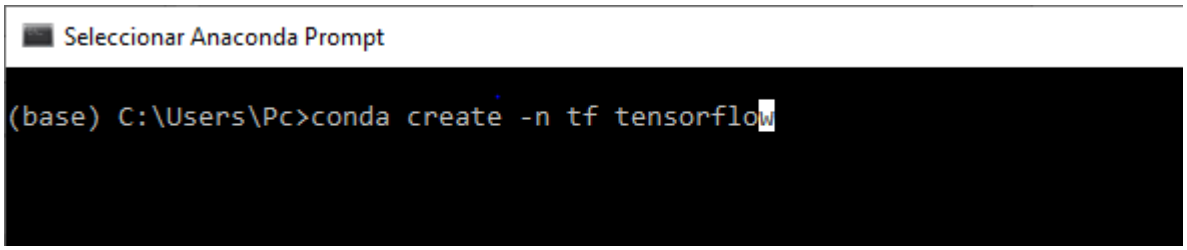


Ilustración A.3: Crear entorno TensorFlow en CPU.

Con estos comandos se procederá a instalar los paquetes correspondientes a TensorFlow y Keras, este último no se ha utilizado para este proyecto, y además se creará el entorno de ejecución elegido.

Instalación paquetes de Python

En este apartado se detallarán el comando de la terminal de Anaconda empleado para instalar en el nuevo entorno de ejecución los paquetes necesarios para ejecutar los scripts desarrollados en este proyecto, la mayoría de los paquetes empleados pueden estar previamente descargados en el entorno, por si alguno de ellos no se hubiera descargado correctamente; se detallarán los comandos necesarios para instalar correctamente esos paquetes.

Lo primero de todo, se debe comprobar que entorno de ejecución está activo, si el entorno activo no fuera el que ha sido creado en el apartado Instalación y creación de un entorno para TensorFlow, se deberá ejecutar el siguiente comando por la terminal de Anaconda:

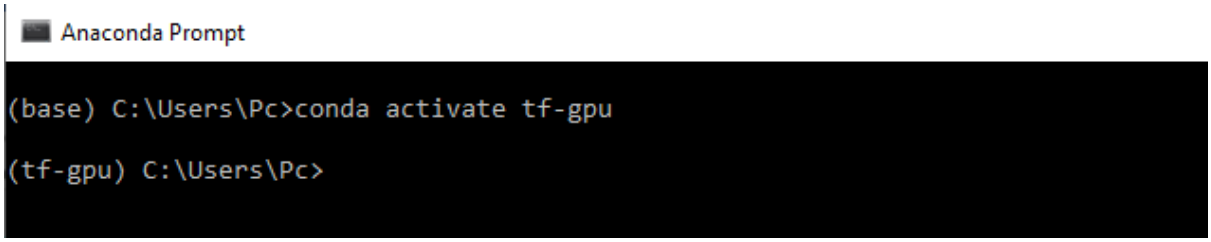


Ilustración A.4: Activar entorno de TensorFlow.

Como se puede observar en la imagen anterior, inicialmente el entorno activo era el entorno 'base', por lo tanto, había que activar el creado en el apartado anterior. Una vez ejecutado el script de la imagen anterior, Anaconda desactiva el entorno base y activa el entorno que usaremos para ejecutar los scripts.

Una vez activado el entorno a usar, podemos ejecutar el siguiente comando para instalar cualquier paquete de Python en este entorno. Para ello usaremos el siguiente comando:

pip install [nombre del paquete].

En el caso de que el paquete ya estuviera instalado en el entorno, después de ejecutar el comando se mostrará el siguiente mensaje:

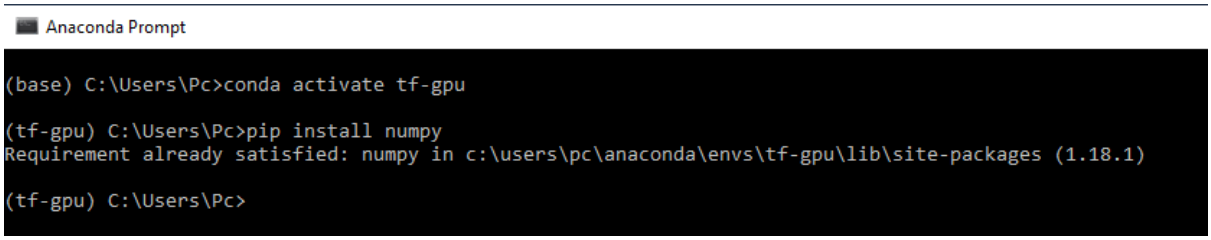


Ilustración A.5: Instalación paquete Numpy.

En el caso de este proyecto, se intentó realizar la instalación del paquete Numpy, el cual ya estaba instalado.

Una vez instalados todos los paquetes en el entorno de ejecución de TensorFlow, a continuación, se procederá a descargar los scripts para su ejecución.

Instalación y configuración de GitHub

Los scripts desarrollados se han almacenado en un repositorio de GitHub, por lo tanto, es necesario descargar dicho contenido de este repositorio. Para poder acceder a ellos, en este proyecto se ha utilizado la aplicación de escritorio de GitHub, puesto que permite acceder fácilmente a este contenido y trabajar con ello cómodamente.

Inicialmente, se descargará el ejecutable correspondiente al sistema operativo del que se disponga, en el caso del utilizado en este proyecto es Windows 10, desde la siguiente página web: https://desktop.github.com/, una vez descargado este ejecutable; se procederá a instalar la aplicación.

Una vez instalada en el sistema, se procederá a descargar el contenido del repositorio a éste. Para ello se utilizará el siguiente enlace del repositorio: https://github.com/RinkakuGomez/TitanicRBM.git. Para descargar este contenido en el sistema, seleccionamos la opción 'New repository' del menú File de la aplicación desktop de GitHub o usamos el atajo de teclado Ctrl+N, desplegándose la siguiente ventana:

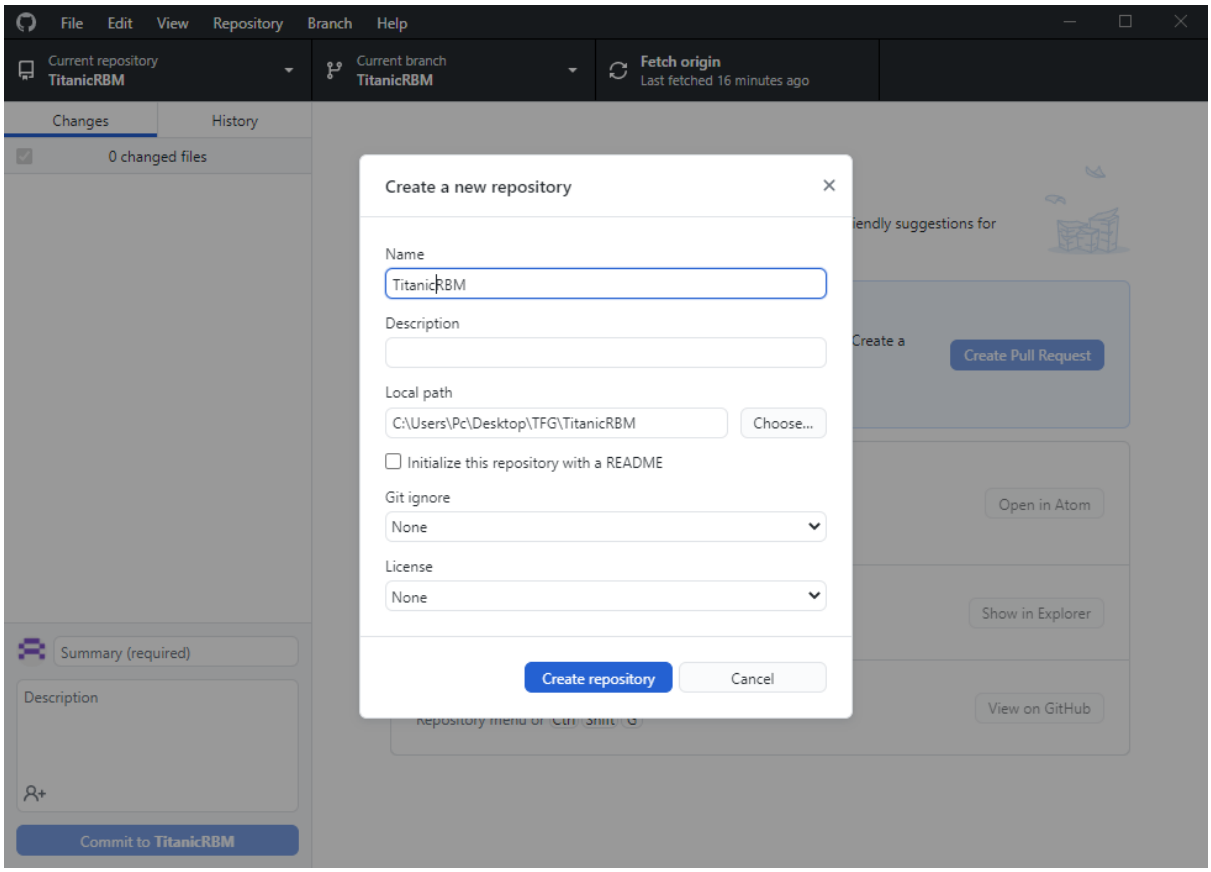


Ilustración A.6: Creación repositorio GitHub.

A continuación, introduciremos el path donde se desea almacenar el repositorio y se le da un nombre. Una vez realizado esto se elige la opción 'Create repository' y GitHub creará este repositorio.

Instalación y configuración en Debian v.10

En este apartado se detallarán los pasos realizados para configurar la máquina virtual Debian v.10 empleada en GCP para llevar a cabo las baterías de pruebas para el apartado Estudio y análisis del modelo.

Configuración Máquina Virtual Debian v.10

Inicialmente se debe crear una cuenta en GCP con una cuenta mail de Google haciendo click en 'Empezar gratis' en el siguiente enlace: https://cloud.google.com/. Después se mostrará un formulario en el que habrá que elegir la cuenta mail de Google que deseamos utilizar para iniciar la prueba gratuita de 300$ y/o 90 días de uso, aceptamos los términos del servicio y aparecerá un formulario en el que se deberá introducir algunos datos personales, incluyendo la tarjeta de crédito sobre la que se cargarán los costes una vez finalice la prueba gratuita, los cargos no se comenzarán a realizar automáticamente sobre la tarjeta una vez finalice la prueba gratuita; una vez introducidos los datos se hace click en el botón 'Iniciar versión de prueba gratuita' para poder crear la cuenta.

Una vez finalizado el proceso de creación de la cuenta, deberemos acceder a la página principal de nuestra cuenta de GCP.

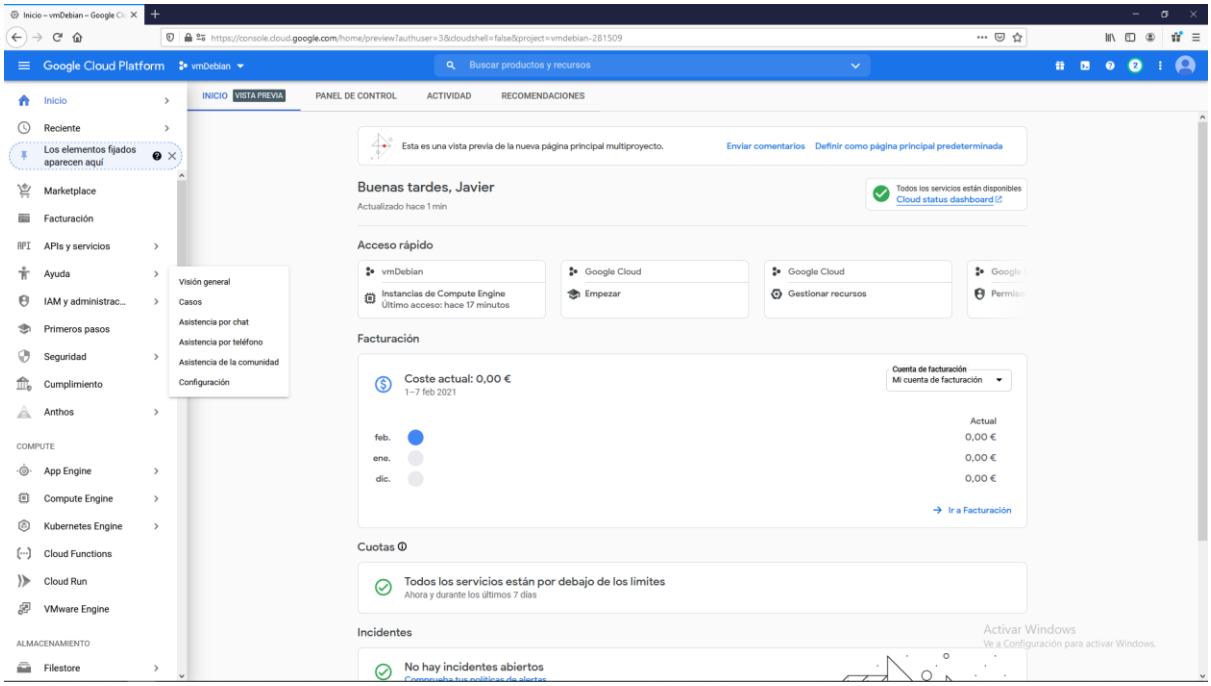


Ilustración A.7: Página principal de la cuenta de GCP.

Una vez dentro de la plataforma y en la ventana de inicio, se accederá a la opción del menú lateral 'Instancias de VM' de la subsección 'Compute Engine' dentro de la sección 'Compute' para crear una máquina virtual en la plataforma con la que trabajar.

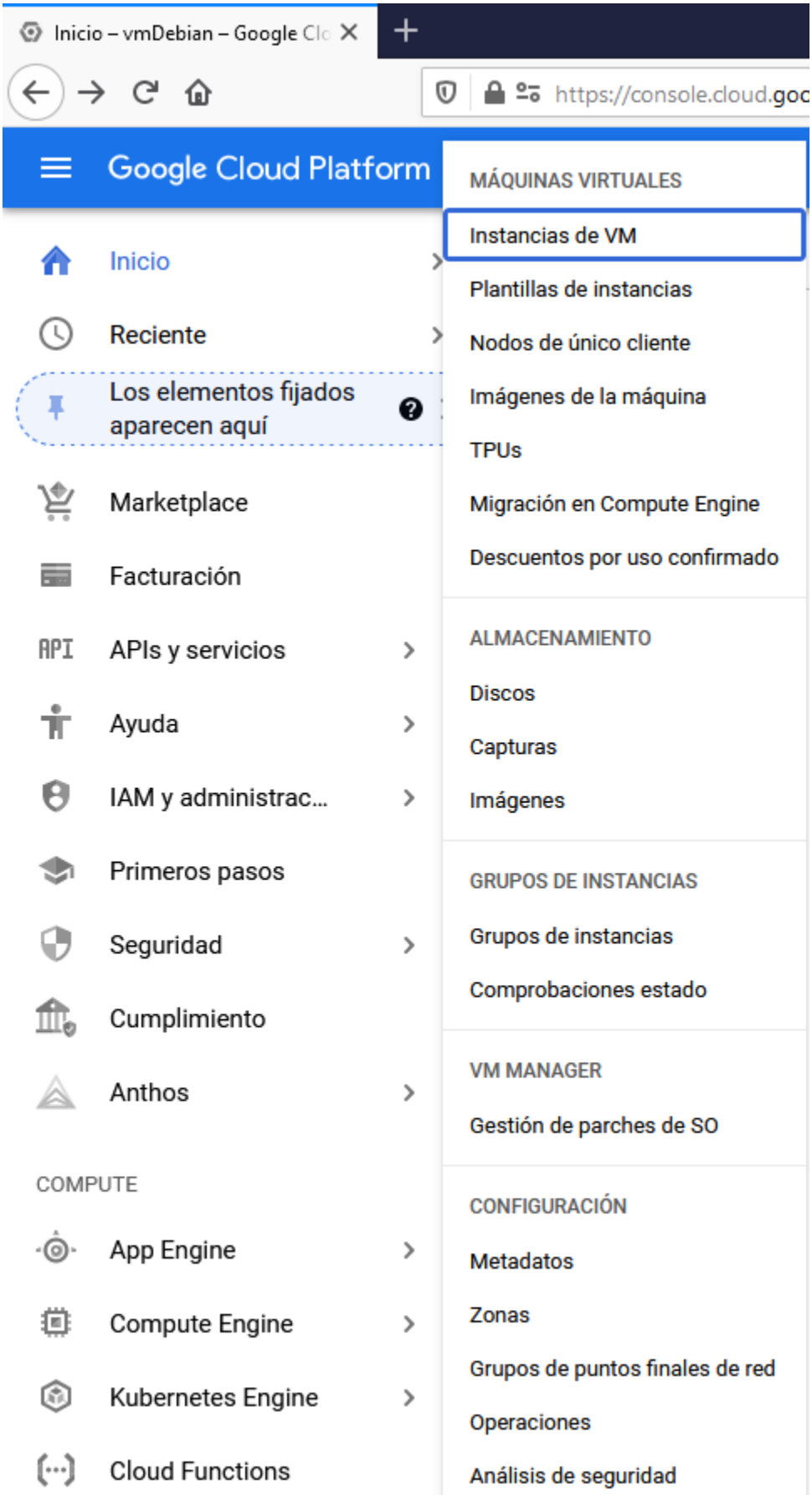


Ilustración A.8: Opción Instancias de VM GCP.

A continuación, aparecerá la página referente a todas aquellas máquinas virtuales que se hayan creado instancia en la plataforma.

Ilustración A.9: Sección Instancias de VM GCP.

Ahora se procederá a crear una instancia dentro de la opción seleccionada haciendo click en el botón 'Crear Instancias'. A continuación, aparecerá la página con el formulario con los distintos componentes y propiedades que presentará la nueva instancia de máquina virtual, como el tipo de CPU de la máquina y su memoria o la región en la que se almacenará y ejecutará esta misma, esta última propiedad es importante puesto que no todas las regiones disponen de los mismos componentes o incluso tienen disponibilidad para tener alguno, por ejemplo, algunas regiones europeas no disponen de máquinas con GPU. Como se puede observar, el sistema operativo a instalar es una distribución GNU/Linux Debian v.10 con una interfaz de línea de comandos o CLI. En el caso de la máquina empleada para realizar las baterías de prueba del estudio y análisis del modelo, presenta las siguientes propiedades:

- Nombre la VM: debian-rbm.
- Tipo de máquina: custom (1 vCPU, 7,5 GB de RAM).
- Región: europe-west2-a.
- Sin GPU.

Una vez rellenado el formulario con los componentes y las propiedades de la máquina virtual hacemos click en el botón 'Crear', una vez creada la instancia se iniciará automáticamente.

En el siguiente apartado se detallarán los pasos necesarios para configurar un entorno de ejecución para ejecutar los scripts de este proyecto.

Instalación anaconda en la Máquina Virtual de GCP

Una vez creada la instancia de la máquina virtual, se podrá comprobar que ahora aparece en el listado de instancias y además estará ejecutándose. A continuación, accederemos al desplegable de la máquina llamado 'SSH' y seleccionaremos la opción 'Abrir en la ventana del navegardor' para que se despliegue un entorno CLI del sistema de la máquina virtual donde configurar el entorno de ejecución.

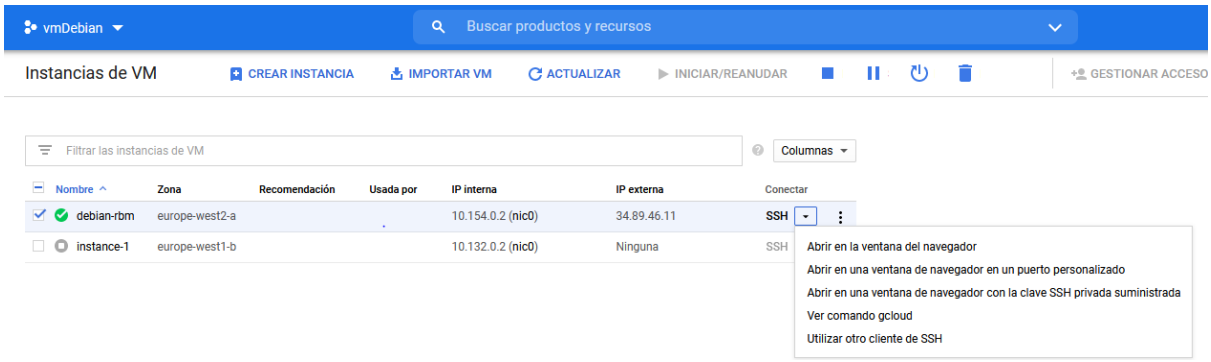


Ilustración A.10: Desplegable SSH de la VM en GCP:

Una vez desplegada la ventana con el entorno CLI, procederemos a descargar e instalar Anaconda en la máquina virtual, para realizar este proceso necesitamos previamente actualizar los paquetes del sistema y descargar las librerías bzip2, libxm12-dev y wget, para ello utilizaremos los siguientes comandos.

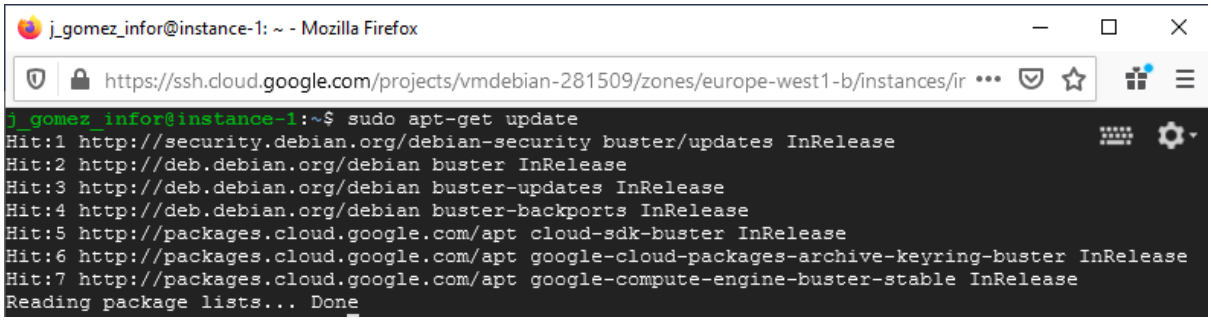


Ilustración A.11: Actualización paquetes instalados de la VM.

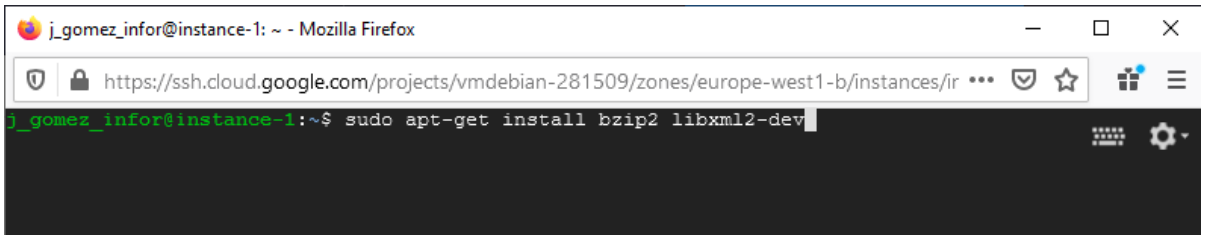


Ilustración A.12: Instalación paquetes bzip2 y libxml2-dev.

Ilustración A.13: Instalación paquetes Git en la VM.

Una vez instalados estos paquetes procederemos a descargar la versión completa de Anaconda, también se puede utilizar la versión MiniConda; en el caso de desear utilizar la versión MiniConda, este ha sido el enlace empleado para realizar la instalación de Anaconda en la máquina virtual, donde también se explica cómo instalar MiniConda (https://medium.com/google-cloud/set-up-anaconda-under-google-cloud-vm-on-windows-f71fc1064bd7). Para descargar el instalador de Anaconda se ha utilizado el siguiente comando:

wget https://repo.anaconda.com/archive/Anaconda3-2018.12-Linux-x86_64.sh

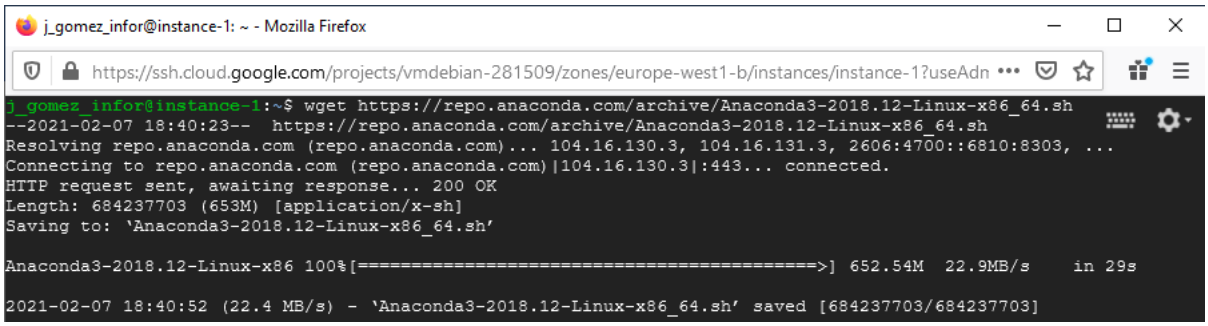


Ilustración A.14: Descargar instalador Anaconda.

Después de ejecutar el comando anterior, se procederá a comprobar si éste ha sido correctamente descargardo y a instalar Anaconda en la máquina virtual y a eliminar el archivo bash tras su correcta instalación.

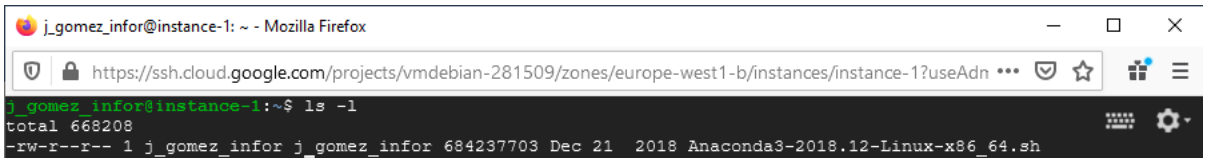


Ilustración A.15: Comprobación correcta descarga archivo .sh de Anaconda.

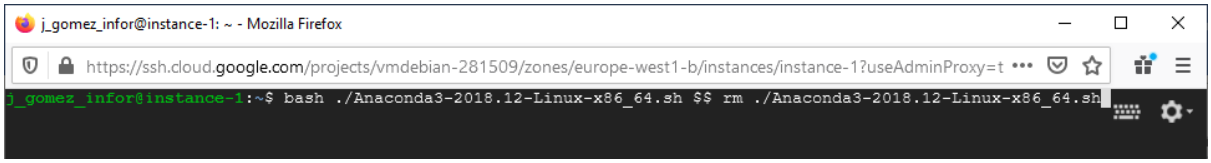


Ilustración A.16: Instalación Anaconda y eliminación archivo .sh.

Tras ejecutar el comando, se iniciará el proceso de instalación de anaconda donde se pedirá confirmar los términos de uso y elegir un directorio de instalación para que este proceso se inicie. Una vez este proceso finalice se comprobará si se ha instalado correctamente.

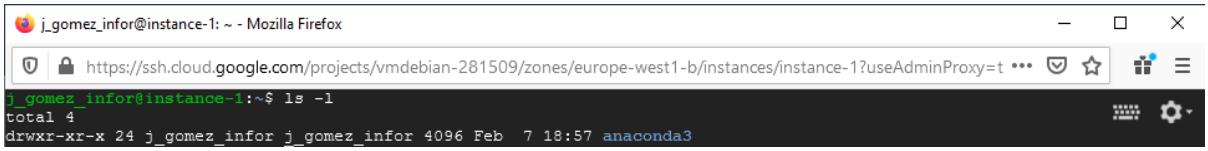


Ilustración A.17: Comprobación instalación Anaconda.

Instalación de Git y descarga del repositorio en Máquina Virtual GCP

Una vez finalizada la correcta instalación de Anaconda en el sistema de la máquina virtual, se debe instalar los paquetes necesarios de Git para instalar el contenido del repositorio en el sistema, para ello se empleará el siguiente comando:

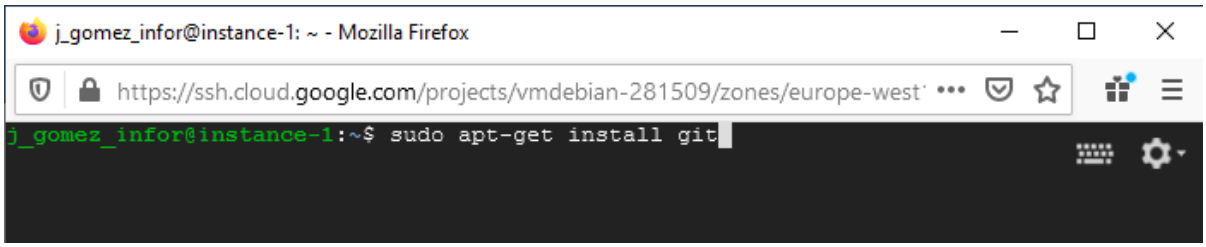


Ilustración A.18: Instalación paquetes Git en la VM.

Una vez finalizada la descarga de la librería de Git, se procederá a inicializar un repositorio local donde poder almacenar el contenido de la rama TitanicRBM del repositorio remoto de GitHub.

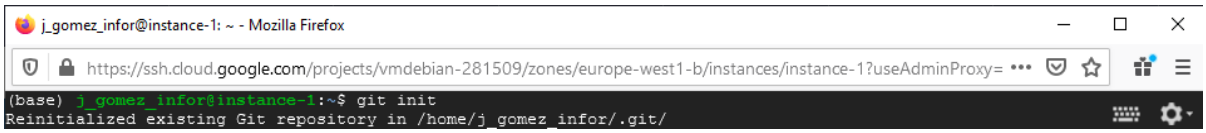


Ilustración A.19: Inicializar repositorio local VM en GCP.

A continuación, se clona el repositorio remoto y se comprueba si se ha creado el directorio con los siguientes comandos:

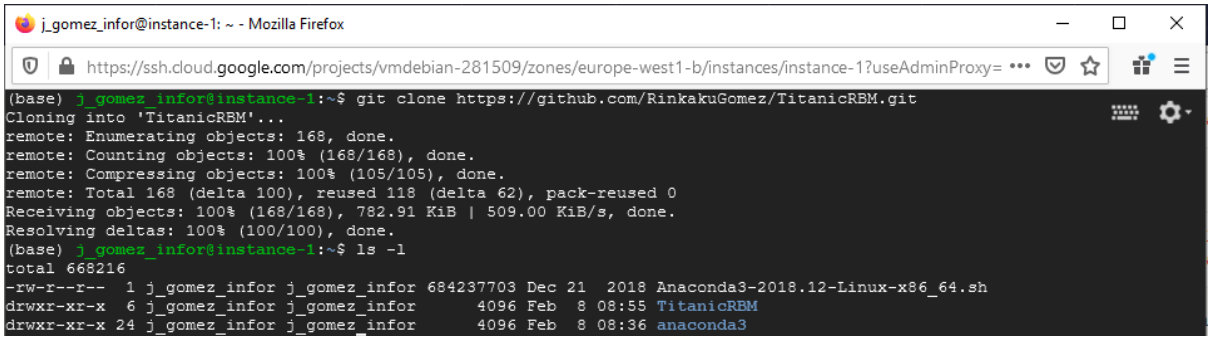


Ilustración A.20: Clonado del repositorio remotoVM en GCP.

El directorio que se ha generado por defecto pertenece a la rama master del repositorio remoto, por lo tanto, ahora se procederá a cambiar la rama actual a la rama TitanicRBM.

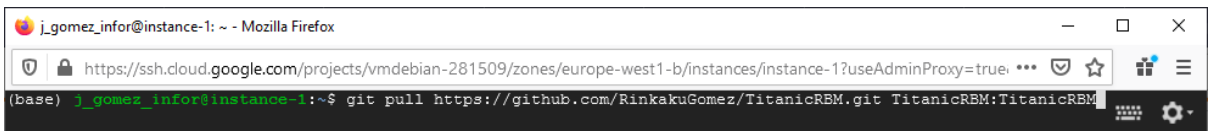


Ilustración A.21: Creación rama local a partir de la rama remota TitanicRBM GCP.

Después, se seleccionará la rama local TitanicRBM como la rama a usar con el siguiente comando:

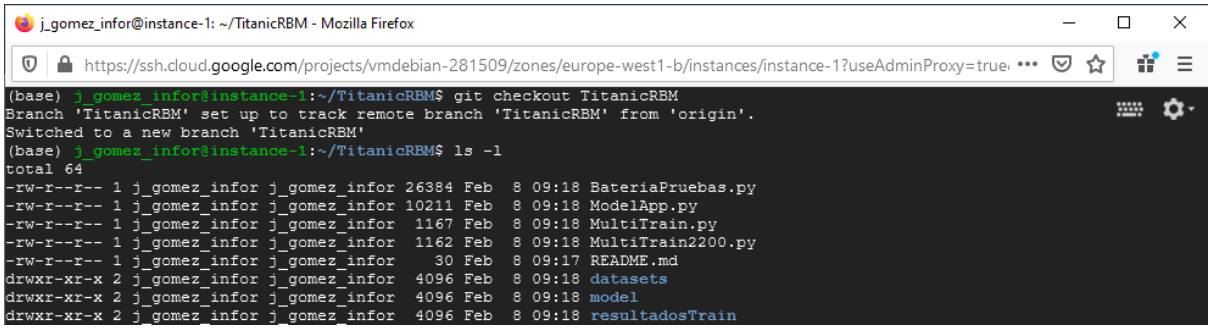


Ilustración A.22: Cambio de rama local a TitanicRBM VM en GCP.

Una vez ejecutado el comando anterior, se ha almacenado el contenido de la rama remota TitanicRBM en el repositorio local de la máquina virtual de GCP. A continuación, si se desea ejecutar los scripts solo se debe confirmar que el entorno de ejecución de Anaconda es el creado para TensorFlow y ejecutar el script ModelApp.py. Se pueden ejecutar los scripts 'MultiTrain.py' y 'MultiTrain2200.py', los cuales se emplearon para realizar las baterías de pruebas del modelo, este script ejecuta las 60 pruebas realizadas para cada uno de los conjuntos de datos empleados.

B. GLOSARIO

Regresión logística: es un tipo de análisis de regresión utilizado para predecir el resultado de una variable categórica en función de las variables independientes o predictoras.

Función de partición $\mathcal{Z}$: es un funcional de un sistema en equilibrio. A partir de esta función se puede derivar a partir las funciones de estado, como la energía libre, energía interna, entropía, etc.

Red neuronal Deep Belief: es un modelo generativo de red neuronal profunda. En este tipo de redes las conexiones de las neuronas se realizan únicamente con todas las neuronas de las capas anterior y posterior, sin permitir conexiones con las neuronas de una misma capa. Este tipo de redes pueden. Debido a la estructura anterior, estos modelos pueden verse como un conjunto de modelos de redes RBM unidos unos con otros donde los estados ocultos de la capa anterior se utilizan como estados visibles de la siguiente. A diferencia de la estructura empleada en las redes Deep Boltzmann, las redes Deep Belief solo utiliza conexiones entre capa bidireccionales en el primer conjunto de neuronas.

Proceso estocástico: es un conjunto de variables aleatorias ordenadas en el tiempo.

Método de estimación Maximun Likehood: es un método de estimación empleado en modelos lineales de Machine Learning. Este método se emplea para determinar los valores de los parámetros de estos modelos, de forma que estos valores maximicen la probabilidad de generar datos observados del modelo.

Métodos de Markov Chain Monte Carlo: son un conjunto de técnicas empleadas para generar un muestreo de datos a partir de distribuciones de probabilidad basadas en la cadena de Markov.

Teorema de Bayes: este teorema formula como actualizar las probabilidades de una hipótesis dada por una evidencia, derivándose de los axiomas de la probabilidad condicional. Además, puede utilizarse para razonar una amplia gama de problemas relacionados con la actualización de creencias.

Inferencia bayesiana: es una técnica MCMC empleado sobre modelos lineales del tipo $y = \theta x + e$, donde 'x' representa los valores de entrada e 'y' representa los valores observados, cuyo objetivo persigue estimar las posteriores distribuciones del parámetro θ basandose en el teorema de Bayes.

Estimación no sesgada: consiste en el cálculo de un valor de la desviación estándar de una población de valores obtenida a partir de una muestra estadística, de forma que la esperanza matemática obtenida del cálculo se igual su valor verdadero.

Coeficientes de Mel-Cepstrum: son un conjunto de coeficientes que emplean la escala de Mel para representar la amplitud del espectro del habla de manera compacta, proporcionando así un coste de computación bajo y una buena robustez

Ruido gaussiano: es un tipo de señal aleatoria no deseada que está asociado a la radiación electromagnética. Este tipo de ruido presenta una densidad de probabilidad que sigue una distribución normal o de Gauss.

Distribución binomial: en términos de teoría de probabilidades y estadística, es una distribución probabilidad discreta formada por una secuencia de N experimentos

independientes. Frecuentemente se emplea para modelar el número de éxitos en una muestra de tamaño n tomada como reemplazo de una población de tamaño N.

Distribución de Poisson: en términos de teoría de probabilidades y estadística, es una distribución de probabilidad discreta que expresa la probabilidad de que un número determinado de eventos ocurran en un intervalo fijo de tiempo o espacio, siempre y cuando estos eventos se produzcan con una tasa media constante conocida e independiente del tiempo transcurrido desde el último evento.

Red neuronal Feedforward: es un tipo de red neuronal que emplea algoritmos de Deep Learning como método de aprendizaje. Su estructura se compone de una capa de neuronas visibles, otra capa que representa las salidas obtenidas y un conjunto de N capas de neuronas ocultas. Este tipo de redes solo presenta conexiones entre las neuronas de distinta capa, es decir, las neuronas de una misma capa están conectadas con todas las neuronas de la capa siguiente, pero no con neuronas de su misma capa. La trasmisión de la información solo se realiza en un único sentido, hacía adelante. Por estos motivos, se considera el modelo más simple de red neuronal profunda.

CSV (Comma-Separeted Values): es un tipo de fichero de texto delimitado que emplea la ‘,’ como separador de los valores, los distintos registros se separan por filas y cada uno de sus campos por columnas separados por comas.

C. ACRÓNIMOS

Acrónimo	Significado en inglés	Acrónimo español	Significado en español
ML	Machine Learning	-	Aprendizaje Automático
SGD	Stochastic Gradient Descent	-	Gradiente Descendente Estocástico
RBM	Restricted Boltzmann Machine	-	Máquinas Restringidas de Boltzmann
CSV	Comma-Separeted Values	-	Valores Separados por Comas
GCP	Google Cloud Platform	-	Plataforma Cloud de Google
NLP	Natural Language Processing	-	Procesamiento del Lenguaje Natural
IDE	Integrated Development Environment	-	Entorno de Desarrollo Integrado
CLI	Command-Line Interface	-	Interfaz de Línea de Comandos
ReLU	Rectified Linear Unit	-	Unidad lineal rectificada

Tabla C.1: Tabla de acrónimos.