Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Redes Neuronales sobre Grafos (GNN): una Prueba de Concepto para sistemas recomendadores

"Graph Neural Networks (GNN): A Proof of Concept for Recommender Systems"

Cristian Vilanova González

La Laguna, *14* de *julio* de 2023

D. **Marcos Colebrook Santamaría**, con N.I.F. 43.787.808-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

“Redes Neuronales sobre Grafos (GNN): una Prueba de Concepto para sistemas recomendadores”

ha sido realizada bajo su dirección por D. **Cristian Vilanova González**,

con N.I.F. 79.097.272-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

Quiero aprovechar este espacio para agradecer a todas aquellas personas que han hecho posible la realización de esta memoria de prácticas: En primer lugar, quiero agradecer a mi tutor académico, Marcos Colebrook Santamaría, por su orientación y apoyo. También, me gustaría agradecer a todos los profesores y profesionales del campo, cuya experiencia y conocimientos compartidos han enriquecido mi aprendizaje y han contribuido al desarrollo de este trabajo. Finalmente, también quiero agradecer a mi familia y amigos, quienes siempre han estado ahí para apoyarme en cada etapa de mi formación académica y profesional.

Licencia

Resumen

En los últimos años, los avances en Inteligencia Artificial (IA) han revolucionado diversos campos, desde el procesamiento del lenguaje natural hasta la visión por computadora. En este contexto, los grafos han surgido como una herramienta fundamental para representar y modelar las complejas relaciones entre entidades.

Los grafos son estructuras compuestas por nodos interconectados mediante aristas, y su aplicación en la inteligencia artificial ha demostrado un potencial extraordinario para abordar problemas complejos y mejorar los resultados obtenidos. En lugar de tratar los datos de manera aislada, los grafos permiten capturar la interacción y la dependencia entre las entidades, lo que brinda una visión más completa y enriquecida de los sistemas que se están modelando.

El objetivo principal de este trabajo de investigación es explorar el uso de los grafos en la inteligencia artificial, enfatizando su importancia y destacando su capacidad para mejorar diversos aspectos de los sistemas de IA. Nos centraremos específicamente en cómo los grafos pueden potenciar la precisión y la eficacia de los algoritmos de aprendizaje automático y los sistemas recomendadores. Además, estudiaremos las redes neuronales basadas en grafos (*Graph Neural Networks*, GNN), que son modelos de aprendizaje automático que operan directamente sobre estructuras de grafos y han demostrado su eficacia en una amplia gama de problemas de IA en diferentes dominios.

Para ilustrar de manera concreta la aplicación de los grafos en la IA, presentaremos una prueba de concepto de un modelo de Red Neuronal de Grafos (GNN) para desarrollar un sistema recomendador basado en datos de Amazon. Este ejemplo concreto nos permitirá mostrar cómo el enfoque basado en grafos puede capturar las relaciones complejas entre los usuarios, los productos y otros factores relevantes, y así generar recomendaciones personalizadas y más precisas.

Además, exploramos diversas técnicas y enfoques utilizados en la aplicación de grafos en la IA, desde la representación de datos en forma de grafos hasta los algoritmos de aprendizaje que aprovechan esta estructura. También analizaremos los beneficios y desafíos asociados con el uso de grafos, así como las áreas de investigación y las tendencias futuras en este campo en constante evolución.

Palabras clave: Grafos, Inteligencia Artificial, Red Neuronal sobre Grafos (GNN), Sistema Recomendador.

Abstract

In recent years, advances in Artificial Intelligence (AI) have revolutionized various fields, from natural language processing to computer vision. In this context, graphs have emerged as a fundamental tool for representing and modeling complex relationships between entities.

Graphs are structures composed of interconnected nodes and edges, and their application in artificial intelligence has shown extraordinary potential for addressing complex problems and improving results. Instead of treating data in isolation, graphs allow us to capture the interaction and dependency among entities, providing a more comprehensive and enriched view of the systems being modeled.

The main objective of this research work is to explore the use of graphs in artificial intelligence, emphasizing their importance and highlighting their ability to enhance various aspects of AI systems. We will specifically focus on how graphs can boost the accuracy and effectiveness of machine learning algorithms and recommender systems.

To concretely illustrate the application of graphs in AI, we will present a proof of concept of a Graph Neural Network (GNN) model to develop a recommender system based on Amazon data. This specific example will showcase how the graph-based approach can capture the complex relationships between users, products, and other relevant factors, thereby generating personalized and more accurate recommendations.

Furthermore, we will explore various techniques and approaches used in the application of graphs in AI, from representing data in graph form to learning algorithms that leverage this structure. We will also analyze the benefits and challenges associated with using graphs, as well as the areas of research and future trends in this rapidly evolving field.

Keywords: *Graphs, Artificial Intelligence, Graph Neural Networks (GNN), Recommender System.*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En los últimos años, el *Deep Learning*[1] (aprendizaje profundo) ha surgido como un enfoque poderoso en el campo de la Inteligencia Artificial (IA). Ha demostrado una notable capacidad para abordar problemas complejos y ha impulsado avances significativos en áreas como el procesamiento del lenguaje natural, la visión por computadora y el reconocimiento de patrones. Uno de los aspectos clave que ha contribuido al éxito del *Deep Learning* es su capacidad para modelar y aprender representaciones de datos altamente complejas. En este sentido, los Grafos han surgido como una herramienta esencial en el contexto del *Deep Learning*.

Los grafos son estructuras fundamentales en la teoría de grafos y se componen de una tupla ordenada de elementos. En su forma más básica, un grafo consta de un conjunto de nodos V (también llamados vértices) y un conjunto de aristas E que conectan los nodos. Las aristas pueden tener direcciones y pesos, lo que agrega mayor flexibilidad en la representación de las relaciones entre los nodos.

$$G = (V, E) \quad V = \{v_1, v_2, \ldots, v_n\} \quad E = \{(v_i, v_j), (v_k, v_l), \ldots\}$$

$$V = \text{Conjunto de vértices del grafo}$$

$$E = \text{Conjunto de aristas del grafo}$$

Figura 1.1: *Tupla de un grafo*

Existen varios tipos de grafos, cada uno con características únicas y utilidades específicas:

- **Grafo Dirigido:** en este tipo de grafo, las aristas tienen una dirección específica, denominándose en este caso como *arcos*, lo que refleja relaciones asimétricas entre los nodos. Es decir, se puede viajar de un nodo de origen a un nodo de destino a lo largo de la dirección del arco, pero no necesariamente en la dirección opuesta.

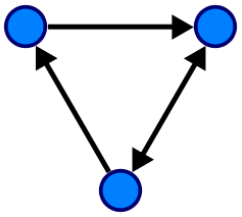

Figura 1.2: *Grafo dirigido*

- **Grafo No Dirigido:** en este tipo de grafo, las aristas no tienen una dirección específica, lo que indica relaciones simétricas o bidireccionales entre los nodos. La información puede fluir en ambas direcciones a lo largo de las aristas.

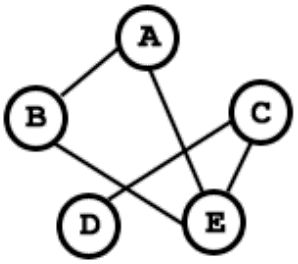

Figura 1.3: *Grafo no dirigido*

- **Grafo Ponderado:** en un grafo ponderado, las aristas tienen asignados valores numéricos llamados pesos. Estos pesos representan la fuerza, la distancia, la similitud u otra métrica relevante de la relación entre los nodos conectados por la arista. Los grafos ponderados son útiles cuando se desea capturar la intensidad o la importancia relativa de las conexiones entre los nodos. Las redes neuronales de grafo (GNN), de las cuales hablaremos más adelante, utilizan este tipo de grafos.

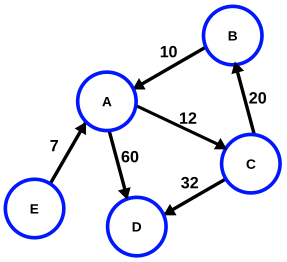

Figura 1.4: *Grafo ponderado*

- **Grafo Bipartito:** en un grafo bipartito, los nodos se dividen en dos conjuntos distintos y las aristas solo pueden conectar nodos de diferentes conjuntos. Este tipo de grafo es útil para representar relaciones entre dos conjuntos de entidades diferentes, como usuarios y productos, actores y películas, o estudiantes y cursos. Este tipo es el más usado para realizar sistemas de recomendación basados en usuarios y productos asociados, siendo el que he utilizado para elaborar la prueba de concepto, la cual se explicará más adelante.

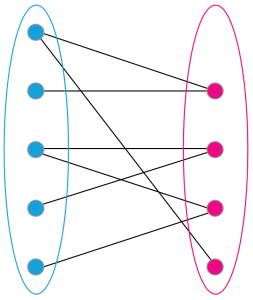

Figura 1.5: *Grafo bipartito*

Estas representaciones permiten capturar de manera eficiente la información estructural y las interacciones complejas presentes en muchos conjuntos de datos del mundo real. A diferencia de los enfoques tradicionales, que tratan los datos de manera aislada, los grafos permiten representar la información en un contexto más completo, teniendo en cuenta las interacciones entre las entidades, lo que brinda una visión más rica y enriquecida de los sistemas que se están modelando.

Los grafos han demostrado ser una herramienta fundamental en la teoría de grafos y en la resolución de problemas en inteligencia artificial, ofreciendo una amplia gama de aplicaciones en diversos campos, como redes sociales, sistemas recomendadores, análisis de redes, biología computacional, optimización y muchos más. Su estructura y flexibilidad permiten modelar relaciones complejas y dependencias entre entidades, lo que resulta invaluable en el análisis de datos y la toma de decisiones.

En los últimos años, las redes neuronales basadas en grafos (GNN) han surgido como una maravillosa forma de aprovechar las características de los grafos en la IA, revolucionando la manera en que aplicamos los grafos en ella. Estas redes neuronales son capaces de aprender y generalizar a partir de datos estructurados en forma de grafos, lo que las convierte en una herramienta poderosa para abordar problemas complejos. Pueden propagar información a través de los nodos y las aristas de un grafo, permitiendo que cada nodo actualice su representación en función de sus vecinos y las relaciones que existen entre ellos. Esta capacidad de capturar información contextual y relaciones complejas ha llevado a avances significativos en una amplia gama de aplicaciones de IA.

Por ejemplo, en el análisis de imágenes, las GNN se utilizan para modelar las relaciones espaciales y semánticas entre los elementos de una imagen representados como un grafo. Estas redes pueden capturar la estructura jerárquica y las interacciones entre las regiones de la imagen, lo que mejora el rendimiento en tareas como la segmentación semántica, la detección de objetos y el seguimiento de objetos en videos.

En el procesamiento del lenguaje natural, las GNN han demostrado ser eficaces para el análisis sintáctico y semántico de textos. Al representar las palabras o entidades como nodos y las relaciones gramaticales o semánticas como aristas, estas pueden capturar las dependencias entre las partes del texto y mejorar el rendimiento en tareas como el

análisis de sentimientos, la extracción de información y la generación de resúmenes.

Además de estas aplicaciones, las GNN se utilizan en otros dominios de la IA, como las redes sociales, los sistemas recomendadores y la biología computacional. En las redes sociales, ayudan a comprender las interacciones entre los usuarios y a descubrir comunidades. En los sistemas recomendadores, modelan las relaciones entre los elementos recomendados y los usuarios, mejorando la precisión de las recomendaciones. En biología computacional, permiten representar interacciones moleculares y redes de regulación génica, lo que facilita la comprensión de los mecanismos biológicos subyacentes.

Como vemos, los grafos demuestran ser una herramienta fundamental en la teoría de grafos y en la resolución de problemas en inteligencia artificial, ofreciendo una amplia gama de aplicaciones en diversos campos, como análisis de redes, optimización y muchos más. A medida que los avances en inteligencia artificial continúan, se espera que jueguen un papel cada vez más importante en el análisis y la comprensión de datos complejos. Su capacidad para modelar relaciones y dependencias entre entidades los convierte en una herramienta valiosa en la toma de decisiones basada en datos y en la resolución de problemas complejos. Sin embargo, a pesar de los beneficios que ofrecen, también enfrentan desafíos ya que la escalabilidad en conjuntos de datos masivos y la complejidad computacional en algoritmos de grafos más complicados son consideraciones importantes, además de que el diseño adecuado de un grafo y la selección de algoritmos óptimos requieren conocimientos especializados.

A lo largo de este proyecto, se analizará detalladamente la aplicabilidad de los grafos en la inteligencia artificial. En primer lugar, se realizará un recorrido por los antecedentes de las GNN en los que se abordarán campos como el *Machine Learning*, *Deep Learning* y los principales avances relacionados con el estudio de redes neuronales. A continuación, se realizará una investigación del estado del arte de las distintas áreas de conocimiento relacionado con los grafos y las redes neuronales basadas en grafos para finalizar con una prueba de concepto (PoC, *Proof of Concept*) en la que se aplicarán técnicas de *Data Science*, *Machine Learning* y *Deep Learning,* con motivo de elaborar un sistema recomendador de productos de Amazon.

Capítulo 2

Antecedentes

2.1 Inteligencia Artificial

La inteligencia artificial (IA) ha evolucionado a lo largo de varias décadas, gracias a importantes contribuciones de diversos campos y científicos. Durante la Segunda Guerra Mundial, los avances en la computación y la electrónica sentaron las bases para los primeros desarrollos en IA. En 1943, Warren McCulloch y Walter Pitts propusieron un modelo matemático de una neurona artificial, sentando los cimientos de las redes neuronales. Luego, en 1950, Alan Turing publicó el famoso artículo "*Computing Machinery and Intelligence*", donde propuso la Prueba de Turing para evaluar la capacidad de una máquina de exhibir comportamiento inteligente.

El verdadero impulso para la IA se produjo en la llamada "edad de oro" (1956-1974). En 1956, se celebró la conferencia de Dartmouth[2], considerada el punto de partida oficial del campo de estudio de la IA. Durante este período, se realizaron avances significativos, como la creación del programa de ajedrez de Arthur Samuel y el desarrollo del sistema de resolución de problemas de Edward Feigenbaum. Sin embargo, a mediados de la década de 1970, surgieron dificultades técnicas y falta de financiamiento, lo que llevó a un período de desilusión conocido como el "invierno de la IA".

A medida que los enfoques basados en el procesamiento de conocimiento simbólico se hicieron más prominentes, los sistemas expertos se convirtieron en una aplicación destacada de la IA en la década de 1970, cuando los primeros microprocesadores permitieron avances en su programación. Estos sistemas capturaban el conocimiento de expertos humanos en bases de reglas y lo utilizaban para tomar decisiones. Además, el desarrollo de lenguajes de programación como PROLOG y LISP impulsó la investigación en esta área.

En la década de 1980, el enfoque se desplazó hacia el aprendizaje automático. Se desarrollaron algoritmos y técnicas, como el algoritmo de retropropagación, para entrenar redes neuronales artificiales. Sin embargo, a mediados de la década de 1990, el progreso en el aprendizaje automático se volvió a estancar debido a la falta de conjuntos de datos adecuados y a limitaciones computacionales.

No fue hasta finales de los 90, que se produjo un hito significativo con Deep Blue, el sistema de IA desarrollado por IBM, ya que, en mayo de 1997, Deep Blue ganó una partida de ajedrez contra el campeón Garry Kasparov. Basado en un algoritmo sistemático de fuerza bruta, Deep Blue evaluó y ponderó todos los movimientos posibles. Aunque este logro marcó un hito importante en la historia de la IA, se reconoce que Deep

Blue solo abarcaba un aspecto concreto, el de las reglas del ajedrez, y aún estaba lejos de la capacidad de modelar la complejidad del mundo.

El nuevo boom de la IA comenzó a partir del año 2010, impulsado por dos factores clave. En primer lugar, el acceso a grandes volúmenes de datos se hizo posible, lo que permitió utilizar algoritmos de clasificación y reconocimiento de imágenes de manera más eficiente. En segundo lugar, la alta eficiencia de los procesadores de tarjetas gráficas (*Graphics Processing Unit*, GPU) aceleró el cálculo de los algoritmos de aprendizaje, reduciendo el tiempo requerido para procesar datos masivos. Este cambio de paradigma completo llevó al enfoque inductivo, donde las máquinas descubren reglas y patrones por sí mismas a través de correlación y clasificación basadas en grandes conjuntos de datos, también conocidos como *Big Data*. Entre las técnicas de aprendizaje automático, el aprendizaje profundo (*Deep Learning*) destacó como especialmente prometedor para diversas aplicaciones.

En los últimos años, la IA ha experimentado un mayor crecimiento debido a los avances en el poder computacional y la disponibilidad de grandes conjuntos de datos. El enfoque en el aprendizaje profundo y las redes neuronales convolucionales ha revolucionado campos como la visión por computadora y el procesamiento del lenguaje natural. Además, técnicas como el procesamiento del lenguaje natural basado en transformadores (*Bidirectional Encoder Representations from Transformers*, BERT) y el aprendizaje por refuerzo han demostrado resultados sobresalientes en diversas aplicaciones.

A medida que la tecnología continúa avanzando, la IA sigue evolucionando y tiene un impacto significativo en una amplia gama de campos y aplicaciones, sin embargo aún queda mucho trabajo para que puedan igualar la capacidad humana de pensamiento.

2.2 Machine Learning

El *Machine Learning*, o aprendizaje automático, ha experimentado un desarrollo progresivo a lo largo de los años, con la aparición de diferentes modelos y enfoques que han contribuido a su evolución. Estos modelos se han construido sobre la base de investigaciones y experimentos que han permitido avanzar en el campo del aprendizaje automático y expandir sus aplicaciones en diversos dominios. A continuación, se presentan algunos de los modelos y enfoques más destacados y cómo han surgido a lo largo del tiempo.

Uno de los primeros enfoques del aprendizaje automático es el "*Self Learning[3]*" o aprendizaje autónomo, que se basa en la idea de que una máquina puede aprender por sí misma sin intervención humana directa. Este enfoque se inspira en el concepto de la plasticidad cerebral y la capacidad de los seres vivos para adaptarse y aprender a partir de la experiencia. Aunque los primeros experimentos en este sentido se llevaron a cabo

en la década de 1950, su desarrollo fue limitado debido a las limitaciones tecnológicas de la época.

Posteriormente, el enfoque del "*Aprendizaje Supervisado*[4]" ganó popularidad. En este enfoque, se proporciona a la máquina un conjunto de ejemplos de entrada y salida esperada, y el objetivo es aprender una función que pueda mapear nuevas entradas a salidas correctas. Este enfoque se ha utilizado con éxito en tareas como la clasificación de imágenes, el reconocimiento de voz y el análisis de texto. Uno de los hitos significativos fue el desarrollo del perceptrón por Frank Rosenblatt en 1957, que se considera el primer modelo de aprendizaje automático basado en el aprendizaje supervisado.

A medida que el volumen de datos y la capacidad de cómputo aumentaron, surgieron modelos más complejos y poderosos. Uno de ellos es el "Aprendizaje Profundo" o *Deep Learning*, que se basa en redes neuronales artificiales con múltiples capas interconectadas. Estas redes tienen la capacidad de aprender representaciones de datos altamente complejas y han demostrado un rendimiento excepcional en tareas como el reconocimiento de imágenes, el procesamiento del lenguaje natural y el análisis de secuencias. El surgimiento del *Deep Learning* se atribuye a avances clave en la teoría de las redes neuronales, así como a mejoras en el *hardware* y el acceso a grandes conjuntos de datos.

Otro enfoque importante es el "Aprendizaje No Supervisado[5]", que se centra en el descubrimiento de patrones y estructuras ocultas en los datos sin la necesidad de etiquetas o salidas esperadas. Este enfoque ha sido utilizado en tareas como la agrupación de datos, la reducción de dimensiones y la generación de representaciones latentes. Uno de los modelos destacados en este campo es el algoritmo de agrupamiento K-means, que fue desarrollado en la década de 1950 y ha sido ampliamente utilizado en diversas aplicaciones.

Además de estos enfoques principales, también ha habido investigaciones en otras áreas, como el "Aprendizaje por Refuerzo[6]" y el "Aprendizaje Semi-Supervisado[7]". El aprendizaje por refuerzo se centra en que una máquina aprenda a través de la interacción con un entorno, recibiendo recompensas o castigos según sus acciones. Por otro lado, el aprendizaje semi-supervisado combina elementos del aprendizaje supervisado y no supervisado, aprovechando tanto datos etiquetados como no etiquetados para mejorar el rendimiento de los modelos.

A lo largo de los años, estos diferentes enfoques y modelos de *Machine Learning* han sido objeto de experimentación y pruebas en diversos dominios. Se han llevado a cabo experimentos para mejorar la clasificación de imágenes, la detección de fraudes, el diagnóstico médico, la traducción automática, entre otros. Estas pruebas han demostrado el potencial y la versatilidad del *Machine Learning* en la resolución de

problemas complejos y en la automatización de tareas que anteriormente requerían la intervención humana.

Los avances del *Machine Learning* se basan en la evolución de modelos y enfoques a lo largo del tiempo. Desde los primeros intentos de aprendizaje autónomo hasta los modelos más sofisticados y potentes del *Deep Learning*, cada avance ha contribuido a ampliar las capacidades de las máquinas para aprender y tomar decisiones basadas en datos. Estos avances se han respaldado con experimentos y pruebas en diversos dominios, demostrando el impacto y el potencial del *Machine Learning* en una amplia gama de aplicaciones.

Capítulo 3

Estado del arte

3.1 Redes Neuronales y Deep Learning

Antes de entender lo que es una red neuronal basada en grafos, es necesario comprender qué es y cómo funciona una red neuronal convencional. Las redes neuronales (*Neural Networks,* NN), se han convertido en uno de los avances más revolucionarios en el campo del aprendizaje automático. Estos modelos computacionales, inspirados en el intrincado funcionamiento de los cerebros biológicos, han demostrado su eficacia en una amplia gama de aplicaciones.

Las redes neuronales consisten en unidades interconectadas llamadas neuronas, que se organizan en capas y procesan información a través de conexiones ponderadas. Cada neurona en una red neuronal realiza una operación de regresión lineal, donde las entradas se multiplican por sus respectivos pesos, se suman y se les agrega un sesgo (o umbral). El resultado se pasa a través de una función de activación, que determina si el nodo se activa o no. Esta salida se convierte en la entrada para el siguiente nodo en la red, y así sucesivamente, hasta que se obtiene la salida final. Este proceso de pasar datos de una capa a la siguiente define la red neuronal como una red de propagación hacia delante o *feedforward* [8].

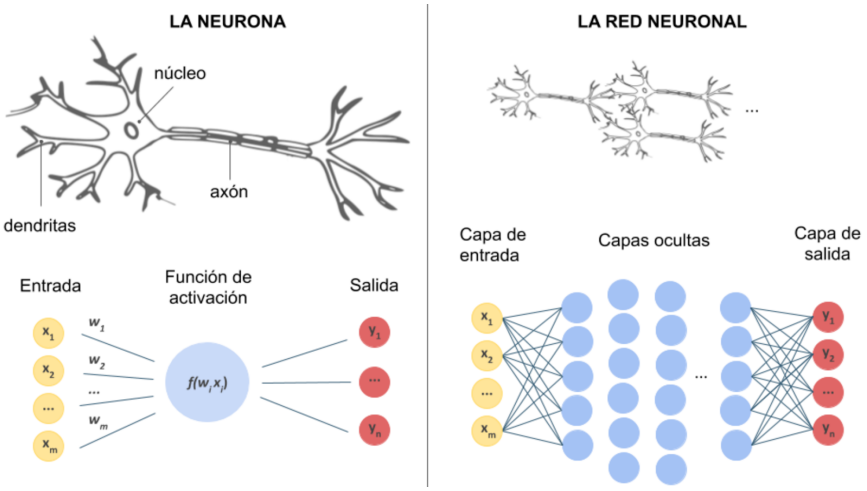

Figura 3.1: *Comparación entre neuronas reales y artificiales*

Fuente: cebebelgica.es/blog/10/que-es-una-red-neuronal-artificial

Las redes neuronales profundas, también conocidas como modelos de aprendizaje profundo o Deep Learning, llevan el concepto de las redes neuronales convencionales al siguiente nivel. A diferencia de las redes neuronales tradicionales, las redes neuronales profundas constan de múltiples capas ocultas entre la capa de entrada y la capa de salida (ver Figura 3.2). Esta estructura jerárquica permite que el modelo aprenda representaciones de los datos en diferentes niveles de abstracción.

Cada capa oculta en una red neuronal profunda realiza transformaciones no lineales complejas de las entradas recibidas. Es decir, cada capa extrae y aprende características cada vez más abstractas y de mayor nivel a medida que se profundiza en la red. Esto permite que el modelo capture patrones y estructuras más complejas en los datos.

Este tipo de redes neuronales se han vuelto especialmente populares en el campo del aprendizaje automático debido a su capacidad para abordar problemas más complejos y manejar grandes conjuntos de datos. Al aprender representaciones jerárquicas de los datos, estas redes pueden descubrir características relevantes de manera automática, sin requerir una intervención humana directa en la definición de características específicas.

Además, las redes neuronales profundas pueden aprovechar técnicas avanzadas, como la regularización, el *dropout* y la normalización de lotes, para mejorar el rendimiento y la capacidad de generalización del modelo. Estas técnicas ayudan a prevenir el sobreajuste y permiten que el modelo generalice mejor a datos nuevos y no vistos anteriormente

El aprendizaje profundo ha demostrado ser extremadamente exitoso en una variedad de tareas, como el procesamiento de imágenes, el procesamiento del lenguaje natural, la visión por computadora, la traducción automática, el reconocimiento de voz y muchos otros campos. Estos modelos han logrado avances significativos en áreas como la detección de objetos, el reconocimiento de patrones, la generación de texto y la toma de decisiones basada en datos.

La ventaja del *Deep Learning* radica en su capacidad para automatizar la extracción de características, lo que significa que no es necesario realizar un procesamiento previo de los datos o definir manualmente características específicas. El modelo aprende de manera automática y progresiva, utilizando las múltiples capas ocultas para aprender representaciones de niveles cada vez más abstractos. Esto permite que el *Deep Learning* sea especialmente efectivo en problemas complejos y de gran escala, donde los patrones y las relaciones pueden ser difíciles de capturar mediante métodos tradicionales de aprendizaje automático.

Sin embargo, el uso del *Deep Learning* también plantea desafíos. Estos modelos requieren grandes conjuntos de datos para un entrenamiento adecuado y su entrenamiento puede ser computacionalmente intensivo. Además, la interpretación de los resultados y la aplicabilidad de los modelos de *Deep Learning* pueden ser difíciles debido a su complejidad y la falta de transparencia en el proceso de toma de decisiones.

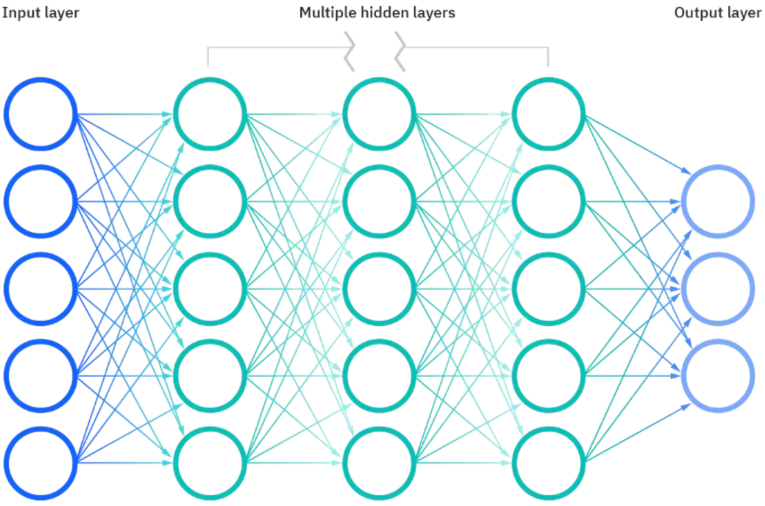

Figura 3.2: *Ejemplo de capas de una red neuronal profunda*
[Fuente: www.ibm.com/es-es/topics/neural-networks]

3.2 Grafos y dónde encontrarlos

Los grafos están presentes a nuestro alrededor; los objetos del mundo real a menudo se definen en términos de sus conexiones con otras cosas, por lo que un conjunto de objetos y las conexiones entre ellos se pueden expresar de manera natural como un grafo.

Es posible que ya estemos familiarizados con algunos tipos de datos en forma de grafos, como las redes sociales, donde los usuarios son los nodos y las aristas son las relaciones entre ellos Sin embargo, los grafos son una representación poderosa y general de datos, por lo que existen diversos tipos de datos que quizás no creas que podrían modelarse como grafos como pueden ser las imágenes y el texto. Aunque pueda parecer sorprendente, al ver las imágenes y el texto como grafos, podemos comprender mejor las simetrías y estructuras que poseen, permitiéndonos analizar su contenido de una manera más profunda.

- **Imágenes como grafos:** normalmente pensamos en las imágenes como cuadrículas rectangulares con canales de imagen, representándolas como matrices (por ejemplo, *floats* de tamaño 244x244x3). Otra forma de pensar en las imágenes es como grafos con una estructura regular, donde cada píxel representa un nodo y está conectado mediante una arista a los píxeles adyacentes. Cada píxel no situado en el borde tiene exactamente 8 vecinos, y la información almacenada en cada nodo es un vector tridimensional que representa el valor RGB del píxel.

  Una forma de visualizar la conectividad de un grafo es a través de su matriz de adyacencia. Ordenamos los nodos, en este caso cada uno de los 25 píxeles en una imagen simple de 5x5 de una cara sonriente, y llenamos una matriz de n nodos×n nodos con una entrada si dos nodos comparten una arista. Es importante destacar que cada una de estas tres representaciones a continuación son diferentes vistas de los mismos datos (ver Figura 3.3).

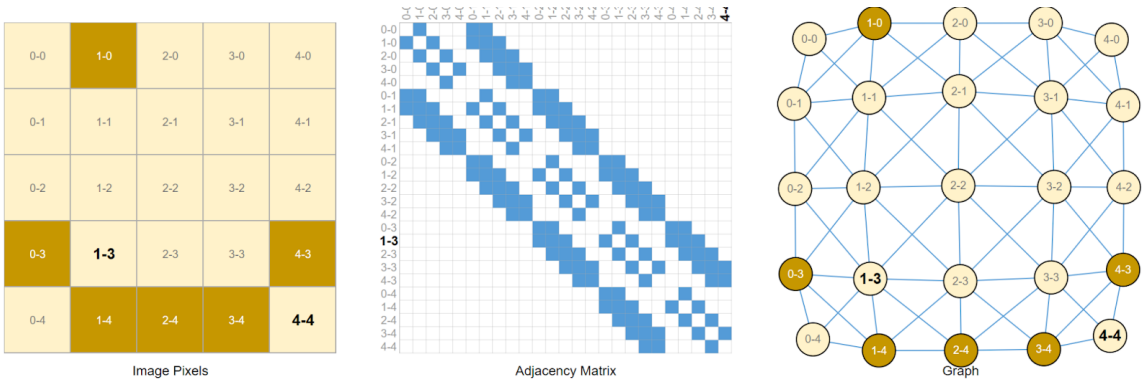

Figura 3.3: *Ejemplo de representación de imagen usando un grafo*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

- **Texto como grafo:** podemos digitalizar el texto asociando índices a cada carácter, palabra o token, y representar el texto como una secuencia de estos índices. Esto crea un grafo dirigido simple (ver figura 1.1), donde cada carácter o índice es un nodo y está conectado mediante una arista al nodo que le sigue.

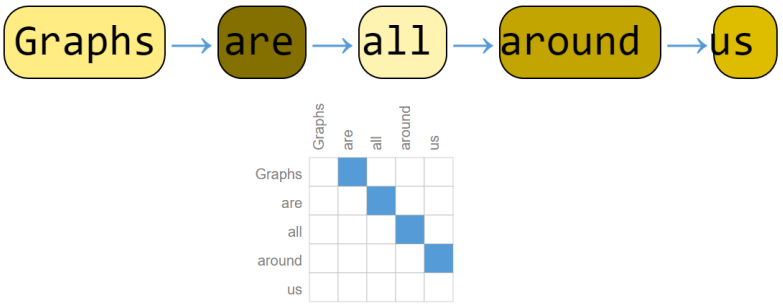

Figura 3.4: *Ejemplo de representación de texto usando un grafo*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

Por supuesto, en la práctica, esto no es usualmente cómo se codifican los textos e imágenes: estas representaciones gráficas son redundantes ya que todas las imágenes y todos los textos tendrán estructuras muy regulares. Por ejemplo, las imágenes tienen una estructura en forma de bandas en su matriz de adyacencia debido a que todos los nodos (píxeles) están conectados en una cuadrícula; y la matriz de adyacencia para el texto es simplemente una línea diagonal, ya que cada palabra solo se conecta con la palabra anterior y la siguiente.

Pasemos ahora a datos que tienen una estructura más heterogénea, es decir, datos en donde el número de vecinos de cada nodo es variable (a diferencia del tamaño de vecindario fijo de imágenes y texto). Estos datos son difíciles de describir de otra manera que no sea mediante un grafo, por ejemplo:

- **Moléculas como grafos:** las moléculas son los bloques fundamentales de la materia y están compuestas por átomos y electrones en el espacio tridimensional.

Todas las partículas interactúan, pero cuando un par de átomos se encuentran a una distancia estable entre sí, decimos que comparten un enlace covalente. Diferentes pares de átomos y enlaces tienen diferentes distancias (por ejemplo, enlaces simples, enlaces dobles). Por ello, es una abstracción muy conveniente y común describir este objeto tridimensional como un grafo, donde los nodos son átomos y las aristas son enlaces covalentes. A continuación se muestra un ejemplo de molécula común y su grafo y matriz de adyacencia asociados (ver Figura 3.5).

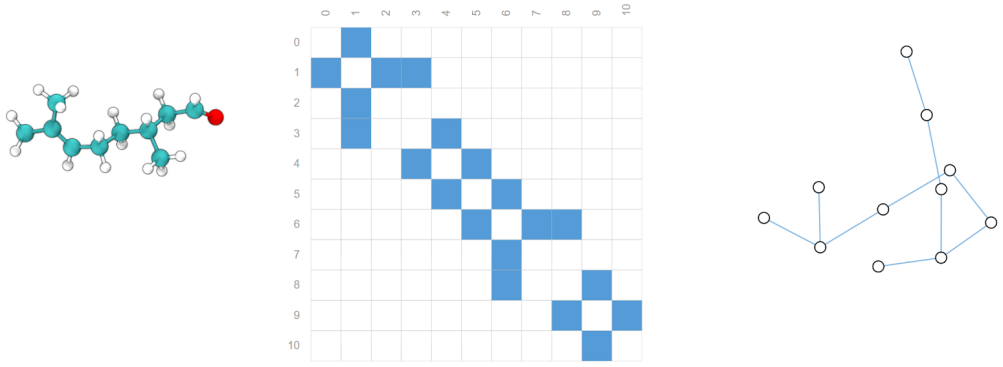

Figura 3.5: *Ejemplo de representación de una molécula usando un grafo*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

- **Redes sociales como grafos:** las redes sociales son herramientas para estudiar los patrones en el comportamiento colectivo de personas, instituciones y organizaciones. Podemos construir un grafo que represente grupos de personas al modelar a los individuos como nodos y sus relaciones como aristas. Por ejemplo para el caso de la red social Twitter podemos usar un grafo en donde los nodos son los usuarios y los arcos las relaciones de seguimiento entre ellos. En este caso sería un grafo dirigido ya que un usuario puede seguir a otro pero que el recíproco no sea necesariamente cierto. En el caso de la red social Facebook podríamos usar un grafo no dirigido para referirnos a la relación de amistad entre usuarios, ya que cuando un usuario es amigo de otro, es una relación bidireccional.

- **Redes de citas como grafos:** los científicos rutinariamente citan el trabajo de otros científicos al publicar artículos. Podemos visualizar estas redes de citas como un grafo, donde cada artículo es un nodo y cada arista dirigida es una cita entre un artículo y otro. Además, podemos agregar información sobre cada artículo en cada nodo, como una incrustación de palabras del resumen.

- **Otros ejemplos:** En visión por computadora, a veces queremos etiquetar objetos en escenas visuales. Podemos construir grafos tratando estos objetos como nodos y sus relaciones como aristas. Modelos de aprendizaje automático, código de programación y ecuaciones matemáticas también se pueden expresar como grafos, donde las variables son nodos y las aristas son operaciones que tienen estas variables como entrada y salida.

La estructura de los grafos del mundo real puede variar ampliamente entre diferentes tipos de datos: algunos grafos tienen muchos nodos con pocas conexiones entre ellos, o viceversa. Los conjuntos de datos de grafos pueden

variar ampliamente (tanto dentro de un conjunto de datos dado como entre conjuntos de datos) en términos del número de nodos, aristas y la conectividad de los nodos.

3.3 Uso de Grafos en inteligencia artificial

Una vez hemos descrito algunos ejemplos de grafos que podemos encontrar, debemos decidir qué tareas realizar con estos datos. Debido a la estructura y la capacidad de los grafos para vincular información, la principal tarea que podemos abordar en inteligencia artificial es la predicción. Existen tres tipos generales de tareas de predicción en grafos: a nivel de grafo, a nivel de nodo y a nivel de arista.

Estos tres niveles de problemas de predicción descritos anteriormente, se pueden resolver con una sola clase de modelo, el GNN, que explicaremos en el siguiente apartado. Pero primero y antes de pasar a explicarlo, haremos un recorrido más detallado por las tres clases de problemas de predicción en los grafos proporcionando ejemplos concretos de cada uno.

3.3.1 Predicción a nivel de grafo

En una tarea a nivel de grafo, nuestro objetivo es predecir la propiedad de un grafo completo. Por ejemplo, para una molécula representada como un grafo (ver Figura 3.6), podríamos querer predecir cómo huele la molécula o si se unirá a un receptor implicado en una enfermedad. Esto es análogo a los problemas de clasificación de imágenes con MNIST y CIFAR, donde queremos asociar una etiqueta a una imagen completa. En el caso del texto, un problema similar es el análisis de sentimientos [9], donde queremos identificar el estado de ánimo o la emoción de una oración completa de una vez.

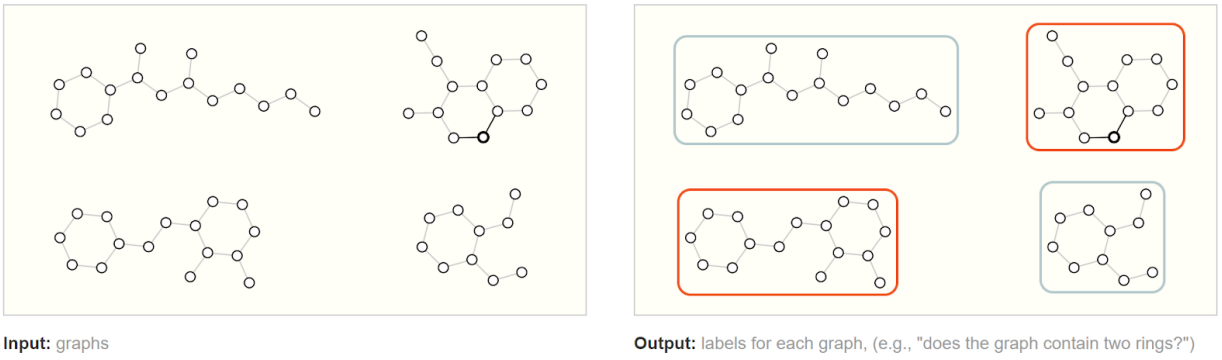

Figura 3.6: *Ejemplo de predicción a nivel de grafo*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

3.3.2 Predicción a nivel de nodo

Las tareas a nivel de nodo se enfocan en predecir una propiedad para cada nodo individual dentro de un grafo.

Un ejemplo clásico de un problema de predicción a nivel de nodo es el famoso "Club de

Karate de Zach" [10] . En este ejemplo, el grafo representa una red social donde los nodos son los practicantes de karate que pertenecen a uno de los dos clubes de karate después de una disputa política y las aristas son las interacciones entre ellos fuera del karate. La tarea consiste en clasificar a cada miembro del club y predecir a cuál de los dos líderes, el Sr. Hi (instructor) o John H (administrador), serán leales después de la disputa. Aquí, la proximidad de un nodo a los líderes del club está altamente correlacionada con la lealtad hacia ellos. En términos de imágenes, esta tarea se asemeja a la segmentación de imágenes, donde intentamos etiquetar el papel de cada píxel en una imagen. En el caso del texto, un ejemplo similar sería predecir las partes del discurso de cada palabra en una oración (sustantivo, verbo, adverbio, etc.).

Es decir, en una tarea a nivel de nodo, nuestro objetivo es asignar etiquetas o roles a cada nodo en el grafo (ver Figura 3.7), en función de sus características individuales y su relación con otros nodos.

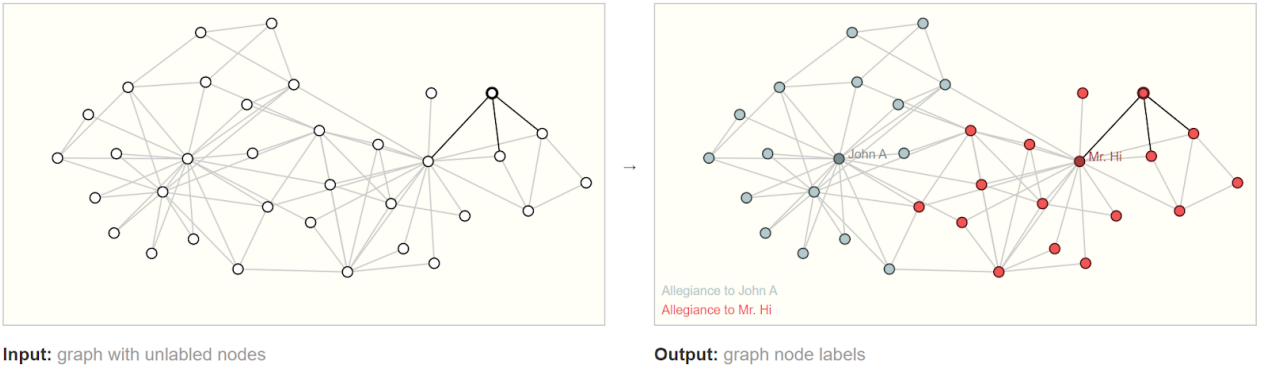

Figura 3.7: *Ejemplo de predicción a nivel de nodo*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

3.3.3 Predicción a nivel de arista

En una tarea a nivel de arista, el objetivo es predecir las propiedades o la presencia de aristas en el grafo. Por ejemplo, en el campo de la comprensión de escenas en imágenes, los modelos de aprendizaje profundo pueden predecir la relación entre los objetos en una imagen. Esto se puede formular como una clasificación a nivel de arista, donde dado un conjunto de nodos que representan los objetos en la imagen, queremos predecir cuáles de estos nodos comparten una arista y cuál es el valor de esa arista. Para ello, podríamos considerar el grafo completamente conectado y, basándonos en los valores predichos, eliminar las aristas para obtener un grafo más disperso.

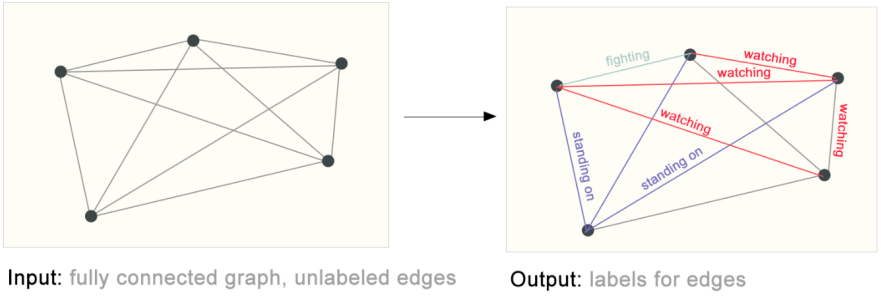

Figura 3.8: *Ejemplo de predicción a nivel de arista*

Fuente: distill.pub/2021/gnn-intro/#into-the-weeds

3.4 Redes Neuronales basadas en Grafos

Ahora que ya conocemos el poder expresivo de los grafos y lo que son las redes neuronales convencionales, podemos continuar comentando un punto clave en este proyecto: las redes neuronales sobre grafos (GNN).

Las redes neuronales basadas en grafos, o *Graph Neural Networks[11] (GNN)*, surgen con el objetivo de aprovechar la potencia de representación de los grafos usándolos en redes neuronales. Son un tipo especial de red neuronal que funciona directamente sobre grafos y que, a diferencia de las redes neuronales típicas, reciben de entrada información codificada como vectores, matrices o tensores. Esta diferencia, permite que las GNN, puedan aprender a partir de información más compleja, ya que no son sólo números aislados, si no que se incorporan las relaciones entre estos.

El funcionamiento básico de una GNN se basa en el concepto de propagación de mensajes[12]. En cada capa de la red, se calcula una nueva representación para cada nodo combinando la información de sus vecinos. Esto se logra mediante una combinación lineal de las representaciones de los nodos vecinos y una función de activación no lineal. A medida que se propagan los mensajes a través del grafo, las representaciones de los nodos se actualizan y se capturan las características locales y globales del grafo (ver Figura 3.9).

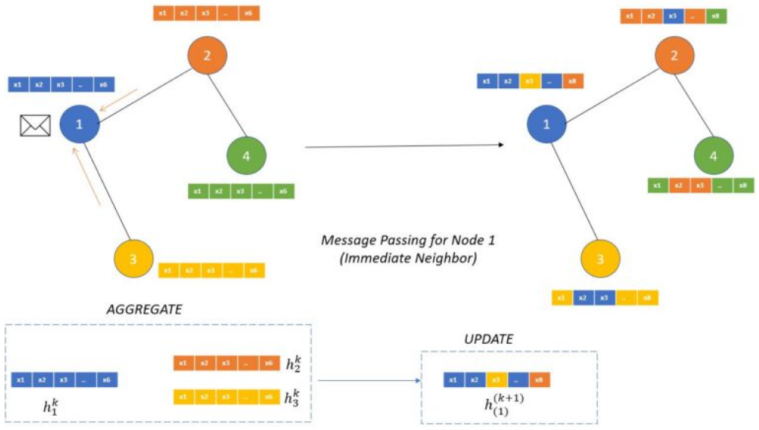

Figura 3.9: *Ejemplo de paso de mensajes en una GNN*
Fuente: http://vinija.ai/recsys/GNN/#embedding-generation-neural-message-passing

Una característica clave de las GNNs es su capacidad para aprender de forma *end-to-end*, es decir, aprender tanto las representaciones de los nodos como los parámetros de las capas de la red a través de la retropropagación del error. Esto permite que las GNNs se adapten a los datos y aprendan a modelar las relaciones específicas presentes en el grafo.

Además, las GNNs pueden incorporar información adicional, como atributos de nodos o aristas, para enriquecer las representaciones y mejorar el rendimiento predictivo. También existen variantes de las GNNs que pueden manejar grafos dinámicos o grafos con estructuras variables.

Aunque todo esto puede sonar muy bien, también hemos de tener en cuenta que el uso de grafos en el aprendizaje automático puede presentar algunos desafíos debido a la representación de los grafos en formatos compatibles con las redes neuronales. Mientras que las matrices de características de los nodos y las aristas son relativamente simples de representar, la conectividad del grafo es más complicada.

Además, el uso de matrices de adyacencia puede ser ineficiente debido a su dispersión y a la posibilidad de diferentes matrices que codifiquen la misma conectividad. El problema de la invariancia a las permutaciones también es relevante, ya que diferentes matrices de adyacencia pueden producir resultados diferentes en una red neuronal profunda, siendo este problema un área de investigación activa.

Para abordar de forma elegante y eficiente estos problemas en las GNN, se utiliza el enfoque de matrices dispersas o listas de adyacencia. Es decir, en lugar de utilizar una matriz de adyacencia completa, que puede ocupar mucho espacio y ser computacionalmente costosa, se utiliza una representación más compacta que describe la conectividad de cada arista en forma de tuplas (i, j) almacenadas en una lista. El uso de esta técnica permite que el modelo capture de manera eficiente y efectiva los patrones y relaciones entre los nodos, sin verse afectado por cambios en el orden de los nodos, consiguiendo así la invariancia a las permutaciones.

A medida que se investiga más en este tipo de redes neuronales se descubren diversos usos y utilidades. Entre sus principales usos en la actualidad se encuentran:

- **Diagnóstico de enfermedades:** si organizamos las enfermedades en un grafo de acuerdo a sus síntomas, podemos utilizar una GNN para que aprenda las relaciones entre síntomas y enfermedades, lo que puede utilizarse para generar diagnósticos más precisos.

- **Síntesis de compuestos químicos:** entrenando una red sobre una base de datos de moléculas, representadas como grafos, es posible aprender a sintetizar nuevas moléculas o determinar nuevas formas de generarlas.

- **Modelado de diseminación de enfermedades:** combinando la información de geolocalización de las personas así como síntomas potenciales, es posible generar modelos que puedan utilizarse para predecir la propagación de enfermedades como el COVID-19. En este caso, la información de posición es organizada como grafos que alimentan una GNN para aprender las relaciones de propagación.

- **Predicción de tráfico:** combinando información espacial de las carreteras

(distancias) e información variable en el tiempo (como es el estado de la meteorología), es posible predecir el flujo que habrá en las vías.

- **Recomendación de productos:** codificando la información de los productos como nodos, y relacionando los que más se compran en conjunto, es posible entrenar una GNN para poder generar recomendaciones dadas las relaciones existentes entre productos (ver Capítulo 5).

- **Clasificación de textos:** aunque es posible clasificar textos utilizando otros métodos del área de la Inteligencia Artificial, las GNN también se pueden utilizar si representamos las palabras como nodos, y las relaciones entre ellas como aristas.

- **Generación de imágenes:** es posible aprender la relación entre los objetos en imágenes, para poder generarlas automáticamente a partir de una descripción o incluso realizar el proceso contrario, es decir, generar una descripción de una imagen a partir de los objetos que la componen.

Las redes neuronales sobre grafos (GNN) han experimentado un crecimiento significativo en los últimos años, y se han convertido en un campo de investigación activo en el aprendizaje automático y la inteligencia artificial. Uno de los últimos avances más importantes en las GNN ha sido su capacidad para modelar y analizar grafos más grandes y complejos gracias al aumento de la disponibilidad de datos masivos y el desarrollo de técnicas de procesamiento en paralelo.

Además, hoy en día se sigue investigando y mejorando la capacidad de las GNN para manejar diferentes tipos de estructuras de grafos. Esto incluye el análisis de hipergrafos [13], donde los nodos pueden estar conectados por hiperaristas que involucran conjuntos de nodos, lo que permite un modelado más flexible y preciso de relaciones complejas. También se siguen desarrollando extensiones de las GNN para grafos dinámicos, donde las conexiones y las características de los nodos pueden cambiar con el tiempo, lo que permite capturar la evolución y los patrones temporales en los datos.

Por todo esto, la aplicabilidad de grafos en redes neuronales es un campo que sigue en continua expansión y que promete ser uno de los campos que puede cambiar el futuro de la Inteligencia Artificial.

3.5 Tipo de Redes Neuronales de Grafo

Existen varios tipos de redes neuronales sobre grafos. A continuación, vamos a comentar los principales tipos.

3.5.1 Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales de Grafos[14] (GCN) son similares a las CNN tradicionales. Aprenden características examinando los nodos vecinos, agregan vectores

de nodos, pasan el resultado a una capa densa, y aplican una función de activación no lineal.

El objetivo principal de una GCN es aprender representaciones de nodos en un grafo, de manera similar a como las redes convolucionales aprenden representaciones de píxeles en imágenes. Una GCN logra esto propagando y combinando información a través de las aristas del grafo en cada capa de la red. El funcionamiento básico de una GCN se compone de los siguientes pasos:

- **Inicialización de las representaciones de los nodos:** en la capa inicial, se asignan representaciones iniciales a cada nodo del grafo. Estas representaciones pueden ser vectores de características asociados a cada nodo o incluso simplemente la identidad del nodo.

- **Propagación de mensajes:** en cada capa, los nodos propagan mensajes a sus vecinos en el grafo. Esto implica combinar las representaciones de los nodos vecinos con la representación del nodo actual, y actualizarla posteriormente. Esta operación se realiza mediante una convolución basada en la estructura del grafo.

- **Agregación de información:** después de propagar los mensajes, se realiza una agregación de información en cada nodo. Esto implica combinar las representaciones actualizadas de los nodos vecinos con la representación del nodo actual para obtener una nueva representación del nodo (convolución). Esta operación se puede comparar con una convolución en redes neuronales convolucionales (CNN), donde se realiza una operación similar de combinación y procesamiento de características vecinas para obtener una nueva representación (ver Figura 3.10 y Figura 3.11).

- **Capas adicionales:** el proceso de propagación de mensajes y agregación de información se repite en capas adicionales para capturar información más compleja y de mayor alcance en el grafo. Cada capa puede tener parámetros de aprendizaje que se ajustan durante el proceso de entrenamiento.

- **Tarea específica:** finalmente, las representaciones de los nodos obtenidas en la última capa se utilizan para realizar una tarea específica, como clasificación, regresión o detección de anomalías, dependiendo del problema que se esté abordando.

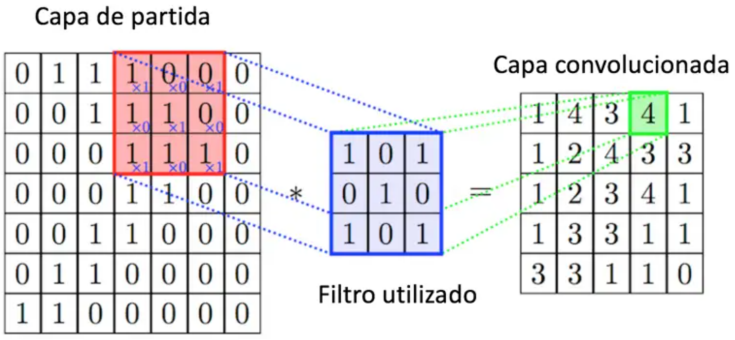

Figura 3.10: *Ejemplo de convolución*

Fuente: https://www.diegocalvo.es/red-neuronal-convolucional/

3.5.2 Redes de Auto-Codificador de Grafos (*Graph Auto-Encoder Networks*)

Las redes de auto-codificador de grafos (*Graph Auto-Encoders*, GAE) son arquitecturas de redes neuronales diseñadas para el aprendizaje no supervisado en grafos. Su objetivo es aprender una representación latente comprimida de los nodos en un grafo, capturando las características y estructuras importantes de los datos.

La GAE consta de un codificador (*encoder*) y un decodificador (*decoder*). En el codificador, se asignan representaciones iniciales a los nodos y se propagan mensajes entre ellos para actualizar sus representaciones. Luego, se agrega información de los nodos vecinos a la representación del nodo actual. A continuación, las representaciones de los nodos se mapean a un espacio latente de menor dimensionalidad. En el decodificador, se reconstruye el grafo a partir de las representaciones latentes, generando una matriz de adyacencia o similitud. Durante el entrenamiento, se utiliza una función de pérdida para comparar el grafo reconstruido con el original y optimizar los parámetros del modelo. Las GAE se entrenan de manera no supervisada, lo que significa que no requieren etiquetas. Aprenden automáticamente las estructuras y características esenciales del grafo.

3.5.3 Redes Neuronales de Grafos Recurrentes (*Recurrent Graph Neural Networks*, RGNN)

Este tipo de redes aprenden el mejor patrón de difusión y pueden manejar grafos multi-relacionales[15] en los que un solo nodo tiene múltiples relaciones. Este tipo de red neuronal de grafos utiliza regularizadores para mejorar la suavidad y eliminar la sobre-parametrización, por lo que utilizan menos potencia de cálculo para producir mejores resultados.

El funcionamiento de una RGNN[16] se basa en la propagación de mensajes entre nodos. En cada paso de propagación, los nodos envían mensajes a sus nodos vecinos y actualizan sus propias representaciones en función de los mensajes recibidos. Estos

mensajes contienen información sobre las características de los nodos emisores, las aristas que los conectan y cualquier información adicional relevante. Utilizando funciones de agregación, la representación de cada nodo se actualiza al combinar la información de los mensajes entrantes con su representación actual.

A medida que se repite el proceso de propagación de mensajes, las representaciones de los nodos se enriquecen al capturar la información de su entorno y sus interacciones con otros nodos. Estas representaciones enriquecidas se pueden utilizar para diversas tareas, como clasificación, predicción o agrupación de nodos, dependiendo del objetivo específico.

Durante el entrenamiento, las RGNN utilizan la retropropagación del error para ajustar los parámetros del modelo. La función de pérdida se define según la tarea que se esté abordando y se utiliza para calcular los gradientes y actualizar los pesos de las conexiones en el grafo.

Las RGNN han demostrado ser eficaces en una amplia gama de aplicaciones, como el análisis de redes sociales, la recomendación de productos, la predicción de enlaces y la detección de anomalías en datos de grafo.

3.5.4 Redes Neuronales de Grafos con Puertas (*Gated Graph Neural Networks*, GGNN)

Son una variante de las RGNN que se han desarrollado para abordar el desafío de modelar dependencias a largo plazo en datos estructurados en forma de grafos. Las GGNN mejoran las RGNN al introducir puertas de nodos, aristas y tiempo, similares a las utilizadas en las Unidades Recurrentes con Puertas (*Gated Recurrent Unit*, GRU).

El funcionamiento de las GGNN se basa en la propagación de mensajes, al igual que las RGNN. Sin embargo, en las GGNN, se añaden puertas que permiten controlar el flujo de información en diferentes estados. Estas puertas se utilizan para recordar y olvidar información relevante en la propagación de mensajes, permitiendo capturar dependencias a largo plazo.

En cada paso de propagación, las GGNN calculan un estado oculto para cada nodo en función de los mensajes recibidos de sus vecinos. Las puertas de nodos y aristas determinan qué información se debe agregar o eliminar en cada estado oculto. Además, se introduce una puerta de tiempo para considerar dependencias temporales en la propagación de mensajes.

El proceso de propagación de mensajes se repite varias veces para permitir que la red capture dependencias a largo plazo. Al finalizar, se obtiene una representación enriquecida para cada nodo, que contiene información relevante considerando tanto las

interacciones locales como las dependencias a largo plazo.

Las GGNN han demostrado ser efectivas en diversas aplicaciones, como el procesamiento de lenguaje natural, el análisis de redes sociales y la predicción de enlaces en grafos.

Capítulo 4

Diseño e implementación de GNNs

Una vez que hemos visto el estado del arte de las redes neuronales como sus tipos y utilidades, a continuación, y antes de pasar a explicar la prueba de concepto, es necesario explicar las distintas tecnologías que pueden ser utilizadas en la elaboración de redes neuronales basadas en grafos.

4.1 Conceptos tecnológicos: CPU vs GPU

CUDA (*Compute Unified Device Architecture, Arquitectura Unificada de Dispositivos de Cómputo*)[17] es una plataforma de computación paralela desarrollada por NVIDIA que utiliza el poder de procesamiento de las *GPU* (*Graphics Processing Unit*, *Unidades de Procesamiento Gráfico*) para realizar cálculos intensivos en paralelo, permitiendo a los desarrolladores ejecutar tareas en ella y aprovechar su capacidad de procesamiento masivo. Entre la principales ventajas que proporciona *CUDA* se encuentran:

Rendimiento: Las *GPU* están diseñadas para realizar operaciones matemáticas intensivas en paralelo, lo que las hace altamente eficientes en tareas de computación de alto rendimiento. *CUDA* permite aprovechar este potencial y acelerar significativamente los cálculos en comparación con la *CPU* tradicional.

Paralelismo masivo: Las *GPU* cuentan con miles de núcleos de procesamiento que pueden trabajar simultáneamente en tareas independientes. Esto permite realizar un alto grado de paralelismo en el procesamiento de datos, lo que resulta en un rendimiento mucho más rápido para aplicaciones que requieren realizar múltiples cálculos simultáneamente.

Programación flexible: *CUDA* proporciona un conjunto de herramientas y APIs que permiten a los desarrolladores escribir código en lenguajes de programación como *C, C++* y *Python* para aprovechar la potencia de las *GPU*. Esto facilita la programación y el desarrollo de aplicaciones de alto rendimiento.

Amplia adopción: *CUDA* es una plataforma ampliamente utilizada y respaldada por NVIDIA, lo que significa que hay una gran comunidad de desarrolladores y una amplia disponibilidad de recursos, bibliotecas y frameworks que facilitan el desarrollo de aplicaciones aceleradas por *GPU*.

4.2 Librerías de implementación de GNN

4.2.1 Pytorch

PyTorch[18] es un framework de aprendizaje profundo ampliamente utilizado y altamente popular en la comunidad de investigación y desarrollo de Inteligencia Artificial. Se destaca por su enfoque intuitivo y flexible para construir y entrenar modelos de machine learning, incluidos los modelos de *Graph Neural Networks* (GNN).

Una de las principales ventajas de *PyTorch* es su facilidad de uso. Proporciona una interfaz amigable y expresiva que facilita la implementación de algoritmos y modelos complejos. PyTorch permite a los desarrolladores escribir código en un estilo imperativo, lo que significa que pueden definir y ejecutar operaciones en tiempo real, lo que resulta en un flujo de trabajo más intuitivo y depuración más sencilla.

Otra ventaja clave de *PyTorch* es su flexibilidad. Permite a los usuarios personalizar y adaptar fácilmente los modelos de acuerdo con sus necesidades específicas. Ofrece una gran cantidad de herramientas y funciones para el procesamiento de tensores, manipulación de datos y construcción de redes neuronales. Además, su arquitectura modular facilita la construcción de modelos complejos y la reutilización de componentes.

También se beneficia de una comunidad activa y en crecimiento, lo que significa que hay una amplia gama de recursos disponibles, desde documentación detallada y tutoriales hasta bibliotecas y paquetes complementarios desarrollados por la comunidad. Esto facilita el aprendizaje y la resolución de problemas, ya que puedes obtener ayuda y encontrar soluciones a desafíos comunes.

En cuanto a la implementación de GNN, *PyTorch* ofrece varias bibliotecas y herramientas específicas para trabajar con grafos, como *PyTorch Geometric* y *DGL*. Estas bibliotecas proporcionan capas, funciones y algoritmos especializados para el procesamiento de grafos y la construcción de modelos de GNN de manera eficiente.

Estas características lo convierten en una elección popular para la implementación de modelos de GNN y otras aplicaciones de aprendizaje automático.

4.2.2 TensorFlow

TensorFlow[19] es un framework de aprendizaje automático que ofrece un conjunto completo de herramientas para desarrollar modelos de *Machine Learning*, incluidos los modelos de GNN.

Una de las principales ventajas de *TensorFlow* es su capacidad para escalar desde aplicaciones de nivel de investigación hasta implementaciones de producción en diferentes plataformas, desde dispositivos móviles hasta grandes clústeres de servidores.

Esto se debe a su arquitectura flexible y altamente optimizada, que permite un rendimiento eficiente y distribuido en diferentes configuraciones de hardware.

TensorFlow ofrece una amplia gama de herramientas y APIs que facilitan la construcción, entrenamiento y despliegue de modelos de GNN. Proporciona una interfaz de programación intuitiva y expresiva que permite a los desarrolladores definir y ejecutar operaciones en tensores de manera eficiente. Además, también ofrece un seguimiento automático de gradientes para facilitar la optimización de modelos mediante algoritmos de aprendizaje automático.

Otra ventaja clave de TensorFlow es su gran comunidad y ecosistema, lo que se traduce en una amplia gama de recursos disponibles, como documentación detallada, tutoriales, ejemplos de código y modelos pre-entrenados. Además, cuenta con el respaldo y soporte de Google, lo que garantiza su desarrollo continuo y actualizaciones regulares.

TensorFlow también ofrece herramientas específicas para el procesamiento de grafos, como *TensorFlow Graph Neural Network* (TF-GNN), que proporciona capas y algoritmos optimizados para el entrenamiento de modelos de GNN. Estas herramientas facilitan la construcción y el entrenamiento de modelos de GNN de manera eficiente y escalable.

4.2.3 *Deep Graph Library* (DGL)

Deep Graph Library[20] (*DGL*) es una biblioteca de aprendizaje automático especializada en el procesamiento y modelado de datos estructurados en forma de grafos. *DGL* proporciona una amplia gama de herramientas y funcionalidades para la construcción, entrenamiento y despliegue de modelos de *Graph Neural Networks* (GNN).

Una de las principales ventajas de *DGL* es su enfoque centrado en grafos y su capacidad para manejar grafos de gran escala de manera eficiente. *DGL* optimiza el procesamiento de grafos utilizando técnicas de muestreo y paralelismo que permiten acelerar el entrenamiento y la inferencia en datos de grafos de gran tamaño.

Ofrece una interfaz fácil de usar que permite a los desarrolladores definir y ejecutar operaciones en grafos de manera intuitiva. Proporciona una abstracción de alto nivel que permite trabajar con nodos y aristas de grafos, así como con estructuras de datos relacionadas, como matrices de adyacencia y características de nodos. Esto simplifica el proceso de construcción y entrenamiento de modelos de GNN.

Una ventaja destacada de *DGL* es su compatibilidad con múltiples *frameworks* de aprendizaje automático, como *PyTorch*, *TensorFlow* y *MXNet*. Esto significa que los usuarios pueden aprovechar la flexibilidad y las características únicas de cada *framework* mientras utilizan *DGL* para tareas específicas de grafos. *DGL* facilita la integración de modelos de GNN en el flujo de trabajo existente de los desarrolladores.

Además, ofrece una variedad de algoritmos de grafos predefinidos y listos para usar, como agregaciones de vecinos, muestreo de grafos y cálculo de centralidad. Estos algoritmos facilitan la implementación de tareas comunes en el procesamiento de grafos,

como clasificación, regresión y predicción de enlaces.

Su enfoque centrado en grafos, su eficiencia en el manejo de grafos de gran escala y su compatibilidad con múltiples *frameworks* de aprendizaje automático hacen de DGL una herramienta poderosa para construir y entrenar modelos de GNN.

4.2.4 *StellarGraph* (DGL)

StellarGraph[21] es una biblioteca de *Python* especializada en el análisis y modelado de datos estructurados en forma de grafos. Está diseñada específicamente para trabajar con datos de grafos heterogéneos y grandes, y proporciona una amplia gama de herramientas para la construcción, visualización y análisis de grafos.

Una de las principales ventajas de *StellarGraph* es su capacidad para manejar datos de grafos heterogéneos, es decir, grafos que contienen diferentes tipos de nodos y relaciones. Esto permite modelar relaciones complejas y capturar información diversa en el análisis de grafos. Además, *StellarGraph* proporciona algoritmos específicos para grafos heterogéneos que pueden ser aplicados directamente a los datos.

Ofrece una variedad de algoritmos de aprendizaje automático listos para usar, como *Graph Convolutional Networks (GCNs)*, *GraphSAGE* y *Gated Graph Neural Networks (GGNN)*. Estos algoritmos permiten realizar tareas comunes en el análisis de grafos, como clasificación, regresión y predicción de enlaces. También ofrece herramientas para la visualización interactiva de grafos, lo que facilita la comprensión y exploración de los datos.

Una ventaja destacada de *StellarGraph* es su integración con bibliotecas de aprendizaje automático como *TensorFlow* y *Keras*. Esto permite a los usuarios aprovechar las capacidades de estas bibliotecas para construir y entrenar modelos de grafos de manera eficiente y escalable.

Además, *StellarGraph* proporciona funcionalidades para el preprocesamiento y transformación de datos de grafos, como la generación de embeddings de nodos y la normalización de características. Estas funcionalidades son fundamentales para preparar los datos antes de aplicar algoritmos de aprendizaje automático.

4.2.5 *Spektral*

Spektral[22] es una biblioteca de *Python* diseñada específicamente para el procesamiento y aprendizaje automático en grafos. Está construida sobre *TensorFlow* y *Keras*, lo que la hace compatible con las capacidades y herramientas de estas bibliotecas.

Una de las principales ventajas de *Spektral* es su enfoque en la representación y manipulación eficiente de datos de grafos en el contexto del aprendizaje automático, proporcionando una amplia gama de operaciones y funciones para realizar tareas comunes en el procesamiento de grafos, como la construcción de matrices de adyacencia y matrices de características, el cálculo de medidas de centralidad y la extracción de subgrafos.

Spektral también ofrece una variedad de modelos de aprendizaje automático listos para usar en grafos, incluyendo *Graph Convolutional Networks (GCN)*, *Graph Attention Networks (GAT)* y *GraphSAGE*. Estos modelos están diseñados específicamente para aprovechar la estructura de los grafos y capturar las relaciones y características complejas presentes en los datos.

Otra ventaja de *Spektral* es su capacidad para trabajar con grafos de gran escala. Utiliza técnicas eficientes de procesamiento y cómputo distribuido para manejar grafos con millones de nodos y aristas. Esto la hace adecuada para aplicaciones de grafos a gran escala en campos como la biología, la química y las redes sociales.

Además, *o*frece una interfaz fácil de usar y bien documentada que facilita la implementación y experimentación con modelos de aprendizaje automático en grafos, proporcionando una sintaxis similar a la de *Keras*, lo que hace que el proceso de construcción y entrenamiento de modelos sea más intuitivo para los usuarios familiarizados con esta biblioteca.

Capítulo 5

Prueba de Concepto (PoC)

Una vez que hemos explicado y recorrido los principales conceptos de los grafos y las redes neuronales, podemos pasar a explicar la prueba de concepto que se ha llevado a cabo. En ella, creamos un modelo de GCN (*Graph Convolutional Network*) para realizar predicciones de productos, dando como entrada un grafo construido a partir de un *dataset* de reseñas de Amazon. El objetivo de esta prueba de concepto es mostrar cómo las GNN pueden ser grandes herramientas para recomendar productos gracias a su poder de capturar relaciones entre entidades.

Para la implementación de esta prueba de concepto, he utilizado la librería *Pytorch* anteriormente explicada (ver Capítulo 4), ya que proporciona diversas funcionalidades y, posiblemente, más intuitiva y clara que el resto de librerías para el contexto de este proyecto. Además, para llevar a cabo los entrenamientos, pruebas y análisis hemos utilizado un servidor de cómputo (UDIGEN) perteneciente a la Universidad de la Laguna. La máquina en cuestión, está dotada con 376 GB de RAM, 2 CPUs y 40 cores físicos, lo cual facilitó las costosas operaciones de entrenamiento de la GNN, así como también agilizó todos los procesos de análisis y pruebas.

A continuación se explicará detalladamente cómo ha sido el proceso de elaboración del sistema recomendador, abordando desde la elección, transformación y estructuración de los datos hasta el modelo, métricas y parámetros utilizados. Posterior a ello, se mostrarán los resultados y las conclusiones obtenidas.

5.1 Datasets

5.1.1 Elección de los *datasets*

Antes de empezar cualquier tarea, para realizar un sistema recomendador, es de vital importancia buscar el *dataset* correcto que más se adapte a las necesidades del problema. En la elaboración de este tipo de sistemas, lo ideal es buscar datos estructurados de tal forma que estén relacionados, o que puedan ser relacionados de alguna manera, así como también disponer de los suficientes metadatos para facilitar y enriquecer el proceso de recomendación. Además, es recomendable buscar que los datos permitan construir fácilmente grafos bipartitos que relacionen dos conjuntos de nodos (usuario y elemento a recomendar)

La primera idea que surgió fue buscar *datasets* de redes sociales, ya que como bien se

sabe y hemos explicado a lo largo de la memoria, este tipo de datos suele funcionar muy bien aplicados en redes neuronales de grafo. Siguiendo este criterio se realizó una búsqueda exhaustiva sobre *datasets* de redes sociales por diferentes repositorios de datos como Kagel [23], o *Network Repository* [24], entre otros.

Si bien es cierto que se encontraron diversos conjuntos de datos interesantes, estos eran bastante poco genéricos, y en su mayoría implicaba que la entrada a la red neuronal fuera demasiado compleja, al no dejar claro qué tipo de características se podían predecir en la GNN. Además, no eran datos reales y, al disponer de únicamente usuarios y relaciones entre ellos, no favorecía su estructuración como grafo bipartito (solo permite predecir relaciones entre usuarios pero sin productos); por lo que bajo esa perspectiva no era un buen ejemplo genérico para aplicar en un sistema recomendador.

Por ello, tras proseguir con la búsqueda, se encontró un prometedor *dataset* de Amazon el cual estaba compuesto por diferentes usuarios, cada uno junto con sus atributos (nombre, id, etc) y un array con los id de los productos que habían comprado dichos usuarios. Si bien es cierto que ahora se disponían de usuarios y productos, el problema de usar este *dataset radicaba* en que no se disponía de metadatos para los productos, por lo que solo se podía vincular a los usuarios con productos ficticios de los cuales no se tendría información al hacer las recomendaciones. Además, a pesar de que se podía vincular usuarios con los productos que compraban, la información estaba estructurada de tal manera que era bastante complejo sacar las relaciones entre ellos.

Con el objetivo de facilitar esta tarea de relacionar elementos, se llegó a la conclusión de que lo que se debería de buscar era algún tipo de *dataset* en el que los datos contuvieran usuarios y productos que, de alguna manera tuvieran una relación o pudieran ser relacionados entre ellos fácilmente. En este contexto, el mejor *dataset* que se podía aplicar a un sistema recomendador era uno compuesto de reseñas de usuarios sobre productos, ya que de esta manera el usuario está relacionándose con un producto al darle su reseña. Además, esto favorecía usar una estructura de grafo bipartita, pudiendo usar el rate del usuario como características de las aristas a predecir por la GNN.

Es por ello que finalmente, y tras buscar nuevamente por múltiples repositorios de datos, se determinó que, para entrenar la red neuronal del sistema recomendador se utilizaría un repositorio de diversos datasets en formato JSON formados por reseñas de usuarios de Amazon en 2018.

En este repositorio se encontraron diversos *datasets* de utilidad formados por reseñas de múltiples tipos de productos, que iban desde productos de belleza, hasta productos deportivos o tarjetas de regalo. Además, se proporcionan tanto los datos de las reseñas como los metadatos de los productos y usuarios, lo que nos permitía elaborar recomendaciones de calidad (ver Figura 5.1).

AMAZON FASHION	reviews (883,636 reviews)	metadata (186,637 products)
All Beauty	reviews (371,345 reviews)	metadata (32,992 products)
Appliances	reviews (602,777 reviews)	metadata (30,459 products)
Arts Crafts and Sewing	reviews (2,875,917 reviews)	metadata (303,426 products)
Automotive	reviews (7,990,166 reviews)	metadata (932,019 products)
Books	reviews (51,311,621 reviews)	metadata (2,935,525 products)
CDs and Vinyl	reviews (4,543,369 reviews)	metadata (544,442 products)
Cell Phones and Accessories	reviews (10,063,255 reviews)	metadata (590,269 products)
Clothing Shoes and Jewelry	reviews (32,292,099 reviews)	metadata (2,685,059 products)
Digital Music	reviews (1,584,082 reviews)	metadata (465,392 products)
Electronics	reviews (20,994,353 reviews)	metadata (786,868 products)
Gift Cards	reviews (147,194 reviews)	metadata (1,548 products)
Grocery and Gourmet Food	reviews (5,074,160 reviews)	metadata (287,209 products)
Home and Kitchen	reviews (21,928,568 reviews)	metadata (1,301,225 products)
Industrial and Scientific	reviews (1,758,333 reviews)	metadata (167,524 products)
Kindle Store	reviews (5,722,988 reviews)	metadata (493,859 products)
Luxury Beauty	reviews (574,628 reviews)	metadata (12,308 products)
Magazine Subscriptions	reviews (89,689 reviews)	metadata (3,493 products)
Movies and TV	reviews (8,765,568 reviews)	metadata (203,970 products)
Musical Instruments	reviews (1,512,530 reviews)	metadata (120,400 products)
Office Products	reviews (5,581,313 reviews)	metadata (315,644 products)
Patio Lawn and Garden	reviews (5,236,058 reviews)	metadata (279,697 products)
Pet Supplies	reviews (6,542,483 reviews)	metadata (206,141 products)
Prime Pantry	reviews (471,614 reviews)	metadata (10,815 products)
Software	reviews (459,436 reviews)	metadata (26,815 products)
Sports and Outdoors	reviews (12,980,837 reviews)	metadata (962,876 products)
Tools and Home Improvement	reviews (9,015,203 reviews)	metadata (571,982 products)
Toys and Games	reviews (8,201,231 reviews)	metadata (634,414 products)
Video Games	reviews (2,565,349 reviews)	metadata (84,893 products)

Figura 5.1: *Datasets del repositorio de datos*

5.1.2 Preprocesamiento de los datos

Una vez seleccionado el conjunto de datos, es necesario transformarlo y prepararlo para, posteriormente, convertirlo a un grafo. Estos procedimientos han de ser genéricos para cualquier *dataset* del repositorio, permitiendo cambiar entre ellos fácilmente.

Lo primero a realizar, es descargar y descomprimir los datos en formato JSON. Para ello, se puede elaborar una función en donde utilizando librerías como *urllib.request*, *gzib* y *shutil* podemos descargar y descomprimir fácilmente desde el código a partir del enlace a los *datasets* del repositorio. Una vez descargamos y descomprimimos los archivos de los datos de las reseñas y los metadatos asociados, podemos proseguir con las conversiones pertinentes.

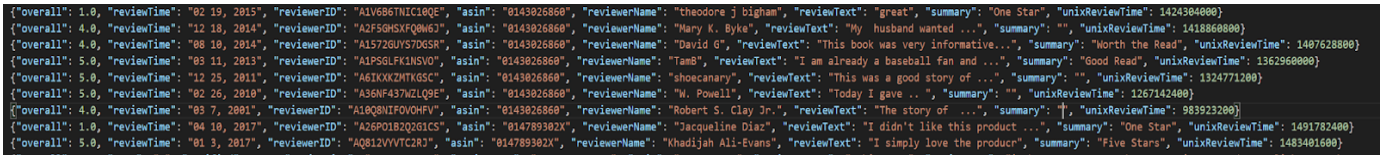

Figura 5.2: *Fragmento de un dataset de reviews*

Ahora que tenemos, por un lado el archivo JSON de las *reviews*, donde cada línea representa una reseña de un usuario (ver Figura 5.2); y por otro el archivo JSON para los metadatos de cada producto, donde cada línea representa cada uno de los productos existentes en las *reviews*; el siguiente paso antes de pasar a tratar los datos, es proceder a limpiarlos y prepararlos. Esto es, verificar que no haya caracteres extraños, elementos vacíos, etc. Para este caso concreto, hubo que cambiar algunas comillas dobles por comillas simples.

Una vez realizada la limpieza de los datos, y puesto que no vamos a necesitar todos los datos de los archivos descargados, tenemos que filtrarlos en función de nuestros intereses descartando todo lo innecesario. Como hemos mencionado antes, nos interesa obtener todos los usuarios y productos junto con su *rate* asociado, además de los

metadatos.

Por ello, en nuestro caso creamos dos nuevos archivos CSV filtrados a partir de los JSON descargados: uno para las *reviews* denominado *amazon_reviews.csv*, en el que por cada línea del *dataset* de *reviews*, obtenemos el nombre de usuario, su identificador, el identificador del producto asociado y la valoración; y otro para los metadatos denominado *amazon_metadatos.csv*, en el que por cada línea obtenemos datos interesantes asociados a los productos reseñados como pueden ser, el nombre, la descripción, el precio, o la categoría, entre otros. La función fundamental de estos metadatos es la de enriquecer las predicciones del modelo.

Además de esto, aprovechando que se nos proporciona, podemos incluir en *amazon_reviews.csv* el tiempo UNIX de cuando se realizó cada reseña, ya que contar con este dato puede ser interesante para entrenar el modelo de tal manera que tenga en cuenta las tendencias temporales, pero eso lo comentaremos más adelante.

Una vez tenemos los nuevos archivos con los datos filtrados, ya podemos empezar a prepararlos. Para trabajar más cómodamente con los datos, y simplificar todas las operaciones, creamos un *Dataframe* de la librería *Pandas,* el cual se carga a partir de *amazon_reviews.csv*, que recordemos, es donde se encuentran los datos relacionados y filtrados. Por lo que nos queda una tabla de este estilo:

User id	Product id	Rate	Unix time
A26PO1B2Q2G1CS	014789302X	3.0	1491782400
A9KCT397IWK97	014789002X	1.0	1491782430
A3I0VZC187KN4V	672289002U	5.0	1491785300
. . .	. . .	. . .	. . .
B3I0VZCHG7KN4V	67228373UI	4.0	1491782219
AIOU89C187KN4V	972T7E002U	5.0	1491783462

Tabla 5.1: *Tabla del Dataframe usado en el proyecto*

5.2 Estructura y creación del grafo

Otra cuestión a tener en cuenta, fue crear la estructura necesaria para nuestro grafo. Lo primero en lo que deberíamos pensar es qué entidades o nodos participarán o formarán parte de nuestro grafo. Dado que lo que queremos es crear un sistema recomendador a partir de reseñas de Amazon, lo que se nos viene a la cabeza es que debemos tener usuarios y productos como nodos. Ahora bien, *¿cómo se deben relacionar estos nodos?*, *¿deberían tener relaciones entre nodos de su mismo tipo?* Y si es así, *¿qué tipo de relaciones pueden tener?*.

La respuesta es más sencilla de lo que parece; como lo que realmente queremos es relacionar productos con usuarios, y usuarios con productos, bastaría con que relacionemos usuarios con productos y productos con usuarios de manera bidireccional, es decir, creando un grafo no dirigido. Además dado que únicamente nos interesa predecir qué producto le gustaría a un usuario en función de los gustos generales, no necesitamos relacionar productos con productos ni usuarios con usuarios.

Sintetizando todo lo anterior, tenemos dos tipos de nodos: usuarios y productos y relaciones (aristas) bidireccionales entre ellos. Si recuerdas los diferentes tipos de grafos comentados al principio (ver Introducción) tenemos un grafo bipartito en el que un conjunto son todos los usuarios y el otro todos los productos, que es justo lo que buscamos en el proceso de elección de los *datasets*. De tal manera que aplicando esto a nuestro dataframe, nos quedaría, de manera resumida algo así:

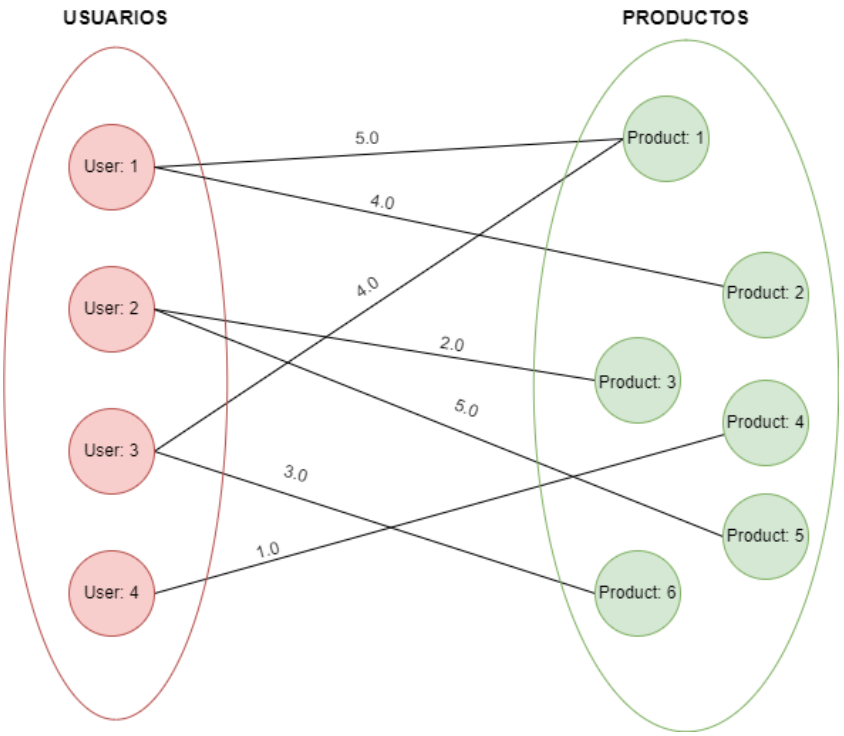

Figura 5.3: *Muestra de grafo bipartito creado a partir del dataset*

Como vemos, por un lado tenemos los nodos de los usuarios, los cuales tienen como atributo el identificador numérico asociado (no es el real); y por otro lado tenemos los nodos productos que, de igual manera, también tienen como atributo su identificador numérico. Las aristas enlazan los usuarios con los productos bidireccionalmente teniendo en cuenta si un usuario ha hecho una reseña del mismo dándole el valor del *rate* asociado. Viendo este esquema general y por relaciones indirectas, ya se podría deducir cosas como que al *user 1* es probable que le guste el *product 6*, esto muestra el poder expresivo de los grafos y el por qué las GNN son el futuro del *Deep Learning.*

5.3 Transformación para el procesamiento de los datos

Como hemos comentado anteriormente, las GNN pueden recibir como entrada, tensores

o listas de matrices dispersas.

Por ello, lo siguiente que debemos hacer, es obtener la matriz de interacción de nuestro grafo. A partir de esta matriz de interacción podremos crear la matriz de adyacencia y posteriormente, convertirla a tensor/matriz dispersa.

La matriz de interacción la conseguimos creando una lista para usuarios, otra para productos y otra para los valores. Es decir, tenemos las listas *users*, *products* y *rates*, las cuales relacionan cada fila de nuestro dataframe proporcionandonos un par del tipo *(i,j)* = *rate*, donde *i = id del usuario* y *j = id producto*. Por ejemplo siguiendo el ejemplo de la tabla anterior (ver Tabla 5.1) tenemos:

users = [A26PO1B2Q2G1CS, …, B3I0VZCHG7KN4V, AIOU89C187KN4V]
products = [014789302X, …, 67228373UI, 972T7E002U]
rates = [3.0, …, 4.0, 5.0]

Un detalle de vital importancia, es que las GNN solo operan claves numéricas, por lo que este tipo de identificadores alfanuméricos no servirían. Para solucionar esto podemos crear 2 nuevas columnas en nuestro *dataframe* que contendrán los nuevos ids: una que relacione los id de los usuarios de *0* a *N* y otra que relacione los id de los productos de la misma forma. quedándonos el *dataframe* de la siguiente manera:

User id	User key	Product id	Product key	Rate	Unix time
A26PO1B2Q2G1CS	0	014789302X	0	3.0	1491782400
A9KCT397IWK97	1	014789002X	1	1.0	1491782430
A3I0VZC187KN4V	2	672289002U	2	5.0	1491785300
. . .	. . .	. . .	. . .	. . .	. . .
B3I0VZCHG7KN4V	N - 1	67228373UI	N - 1	4.0	1491782219
AIOU89C187KN4V	N	972T7E002U	N	5.0	1491783462

Tabla 5.2: *Tabla del dataframe con las nuevas claves añadidas*

Tras realizar esto y modificar las listas con los nuevos identificadores, ya podemos convertir estas listas a tensores para representar la matriz de interaccion *R*. De esta forma, tenemos un tensor bidimensional, *edge_index* y otro unidimensional *edge_values*. El tensor *edge_index* almacenará los valores de la columna *User key* contenidos en la lista users en *edge_index[0]* y los valores de la columna *Product key* contenidos en la lista productos en *edge_index[1]*. El tensor *edge values* contendrá los valores de los *rates* de la lista *rates*.

Otro detalle a tener en cuenta que puede mejorar considerablemente la eficiencia, es contar con un archivo *conversions.csv* que guarde como diccionario las conversiones entre los antiguos id y las nuevas claves. De esta manera no necesitamos recorrer el *dataframe* para volver a transformar de clave a id. Por ejemplo cuando hagamos las recomendaciones y vayamos a usar los metadatos, la GNN solo conocerá los *Products keys* y los *User keys*, pero para vincularlos con los metadatos necesitamos los id. Esto se

comentará más adelante

Ahora que ya tenemos la matriz de interacción en forma de tensores, para conseguir la matriz de adyacencia *A* utilizamos la siguiente fórmula algebraica (ver Figura 5.3), donde: R = Matriz de Interacción y Rt = matriz de interacción traspuesta

$$A = \begin{pmatrix} 0 & R \\ R^T & 0 \end{pmatrix}, \quad R \in \mathbb{R}^{M \times N}$$

Figura 5.4: *Fórmula de conversión de matriz de iteración a matriz de adyacencia*

Lo que se consigue aplicando esta fórmula es básicamente, obtener la matriz de interacción de forma simétrica. Es decir, si en nuestra matriz de interacción tenemos un par *i,j = k*, ahora también tendríamos el mismo par a la inversa, *j,i = k*, consiguiendo así bidireccionalidad en la matriz de adyacencia.

Una vez hemos aplicado la fórmula y hemos creado la matriz de adyacencia, tenemos dos opciones: o convertirla a una lista de matriz dispersa, o como en nuestro caso, convertirla a tensor disperso. (Básicamente es lo mismo)

Pongamos un ejemplo simple del proceso para visualizarlo:

Aplicado a nuestro ejemplo de usuarios y productos tenemos:

User id	Product id	Rate
0	*0*	*1*
0	*1*	*2*
1	*0*	*3*
1	*1*	*4*

Tensores:

edge_index[0] = [0,0,1,1]
edge_index[1] = [0,1,0,1]
edge_value = [1,2,3,4]

Tenemos la siguiente matriz de interacción *M*:

$$M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

Aplicando la fórmula obtenemos la siguiente matriz de adyacencia *A*:

$$A = \begin{pmatrix} 0 & 0 & 1 & 2 \\ 0 & 0 & 3 & 4 \\ 1 & 3 & 0 & 0 \\ 2 & 4 & 0 & 0 \end{pmatrix}$$

Como vemos, lo que hace es crear índices "artificiales" para que cuando hagamos la conversión a matriz dispersa, se eviten los pares repetidos que puedan dificultar la conversión. Es decir, en lugar de obtener los índices dispersos: *Matriz A dispersa* = (*(0,0) = 1, (0,1)/(1,0) = 2+3 , (1,0)/(0,1) = 3+2 , (1,1) = 4)* lo cual podría producir conflictos en la GNN al mezclar valores, obtenemos: *Matriz A dispersa = ((0,2)/(2,0) = 1 | (0,3)/(3,0) = 2 | (1,2)/(2,1) = 3 | (1,3)/(3,1) = 4).*

Uno de los problemas que se encuentran es que, para aplicar esta fórmula, se tienen que hacer operaciones internas en formato de matriz densa, antes de pasarlo a matriz dispersa (en nuestro caso tensores dispersos) que recordemos, es lo mejor para las GNN; por lo que para problemas grandes requiere muchísima memoría, al duplicar en anchura y largura la matriz de iteracción. Para solucionar este problema se puede trabajar totalmente en formato disperso desde el principio, creando la matriz de adyacencia dispersa a partir de la matriz de iteracción dispersa. Sin embargo, esto puede complicarse demasiado debido a la dificultades secundarias que implica como puede ser la duplicidad de claves, mencionada anteriormente. (Ver apéndice A.1 y A.3)

5.4 Modelo

Ahora que disponemos de los datos de entrada preparados, pasemos a explicar el modelo de GNN utilizado: *LightGCN*. *LightGCN[25]* es un modelo de recomendación basado en grafos que, a diferencia de otros modelos más complejos que incorporan capas adicionales y técnicas sofisticadas, se enfoca en el uso de la propagación de mensajes para obtener representaciones de nodos efectivas.

En lugar de utilizar técnicas avanzadas de agregación de información o capas adicionales, se utiliza una función de propagación de mensajes simple y eficiente. En cada paso de propagación, los embeddings de los nodos se actualizan tomando en cuenta únicamente las conexiones directas entre nodos y sin tener en cuenta información adicional como atributos o características de los nodos. Esta simplicidad permite que el modelo sea fácil de implementar y computacionalmente eficiente. Además, al centrarse únicamente en la estructura de grafo y la propagación de mensajes, este modelo puede ser especialmente efectivo en situaciones donde los datos son escasos o las características de los nodos no están disponibles. A pesar de esta simplicidad, *LightGCN* ha demostrado obtener resultados competitivos en tareas de recomendación, mostrando que un enfoque más liviano y centrado en la propagación de mensajes puede ser efectivo en ciertos contextos de recomendación.

En el Apéndice A.3 se puede observar la implementación de la clase, en donde puede apreciar el método *forward*, en el que se realiza un proceso iterativo donde los *embeddings* se propagan a lo largo de las conexiones del grafo, y se combinan con la

información de los vecinos para actualizar su representación. Esto es lo que permite capturar la información relevante y las relaciones en el grafo de manera efectiva. Una vez que se han actualizado los *embeddings*, se utilizan para calcular la similitud entre usuarios y elementos, lo que permite realizar recomendaciones personalizadas. Para esto, se utiliza una capa lineal que combina los *embeddings* de usuario y elemento, y se genera la salida final.

5.5 Entrenamiento

Lo que buscamos con este entrenamiento es que, dadas unas etiquetas, en nuestro caso *rates*, el modelo sea capaz de aprender qué *rate* pondría un usuario al conjunto de productos existentes, de tal manera que podamos realizar recomendaciones al obtener los primeros *k* productos con más *rate* para cada usuario (*supervised-learning*). Para entrenar el modelo, se ha dividido el *dataset* en tres partes: entrenamiento, testeo y validación. Cabe destacar que, si se dispone de *GPU*, el modelo puede utilizar *CUDA (ver Capítulo 4)* para usarla en lugar de la *CPU*, lo cual puede favorecer los tiempos y eficiencia del entrenamiento. A continuación, vamos a comentar los puntos clave del entrenamiento.

El modelo se entrena utilizando un optimizador *Adam[26]*, que se encarga de actualizar los parámetros del mismo durante el entrenamiento. El optimizador en cuestión, se define con una tasa de aprendizaje (*Learning Rate*, *LR*) específica y un coeficiente de decaimiento de peso (*weight_decay*) para evitar el sobreajuste.

La función de pérdida utilizada en este caso es el error cuadrático medio (*MSELoss*), que compara las predicciones del modelo con los valores reales de las interacciones entre usuarios y productos con el objetivo de minimizar esta pérdida durante el entrenamiento.

El proceso de entrenamiento se realiza a través de un bucle que itera un número determinado de veces (*epoch*). En cada iteración, se realiza una propagación hacia adelante (*forward propagation*) para obtener las predicciones del modelo en los datos de entrenamiento. Luego, se calcula la pérdida mediante la función de pérdida y se realiza una propagación hacia atrás (*backward propagation*) para actualizar los parámetros del modelo utilizando el optimizador.

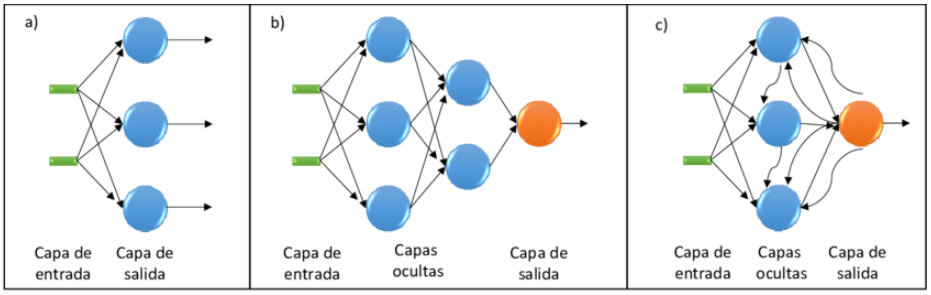

Figura 5.5: *Ejemplo de propagación*

Fuente:[www.researchgate.net/figure/Red-Neuronal-monocapa-b-Red-Neuronal-multicapa-de-propagacion-hacia_fig5_315762548]

Cada cierto número de iteraciones (*ITERS_PER_EVAL*), se evalúa el rendimiento del modelo en el conjunto de validación. Esto se hace cambiando el modelo al modo de evaluación *model.eval()* y calculando la pérdida y otras métricas de evaluación, como la

precisión y recuperación en *k* (*recall_at_k* y *precision_at_k*), de las cuales hablaremos en el siguiente apartado.

Al final del proceso, se guarda el estado del modelo entrenado en un archivo .pth para su posterior uso, de tal manera que podamos hacer comparaciones entre diferentes modelos fácilmente.

5.6 Resultados y análisis

A continuación, se mostrarán los distintos resultados obtenidos, así como también las recomendaciones realizadas por el modelo, utilizando distintos parámetros y datasets.

Estos resultados se han obtenido definiendo una única capa y 4000 *epoch* en el modelo, que traducido en tiempo, usando nuestra máquina *UDIGEN* y sin utilizar las funciones de CUDA (puesto que no se disponía de *GPU*) ha significado hora y media de entrenamiento, para un dataset de 300.000 reseñas de 40.000 productos. Si bien es cierto, que para las GNN suele ser recomendable utilizar de 3 a 5 capas, el motivo de definir una única capa en el modelo se ha debido a que, si recordamos lo dicho anteriormente, el modelo *LightGCN* tiende a funcionar mejor a mayor simplicidad. Esto último se pudo verificar comprobando que si definíamos tres capas en el modelo, empeoraban considerablemente las métricas obtenidas.

Para medir la eficacia de nuestro entrenamiento sobre el modelo, se han usado las siguientes métricas:

- ***Precision:*** en el contexto de una GNN, la precisión se refiere a la proporción de predicciones positivas correctas en relación con todas las predicciones positivas realizadas por el modelo. Es decir, mide cuántos de los nodos o aristas que el modelo ha clasificado como positivos son realmente positivos en el conjunto de datos.

  La fórmula para calcular la precisión es:

  Precisión = Número de verdaderos positivos / (Número de verdaderos positivos + Número de falsos positivos)

  Donde los verdaderos positivos son los casos en los que el modelo ha clasificado correctamente un ejemplo positivo, y los falsos positivos son los casos en los que el modelo ha clasificado incorrectamente un ejemplo negativo como positivo.

  Esta métrica, es importante para evaluar la capacidad del modelo para realizar predicciones precisas en problemas de clasificación en grafos. Una alta precisión indica que el modelo tiene una baja tasa de errores de falsos positivos y es capaz de identificar correctamente los casos positivos en el conjunto de datos. Sin embargo, es importante tener en cuenta que la precisión debe evaluarse junto con otras métricas, como el recall, para obtener una imagen completa del rendimiento del modelo.

- ***Recall:*** en el contexto de una GNN, el recall mide la proporción de ejemplos

positivos que el modelo ha identificado correctamente en relación con todos los ejemplos positivos presentes en el conjunto de datos. Es decir, mide cuántos de los nodos o aristas que realmente son positivos han sido identificados como positivos por el modelo.

La fórmula para calcular el recall es:

Recall = Número de verdaderos positivos / (Número de verdaderos positivos + Número de falsos negativos)

Donde los verdaderos positivos son los casos en los que el modelo ha clasificado correctamente un ejemplo positivo, y los falsos negativos son los casos en los que el modelo ha clasificado incorrectamente un ejemplo positivo como negativo.

El recall es una métrica importante para evaluar la capacidad del modelo para capturar y detectar de manera efectiva los ejemplos positivos en el conjunto de datos. Un alto recall indica que el modelo tiene una baja tasa de errores de falsos negativos y es capaz de identificar la mayoría de los casos positivos presentes.

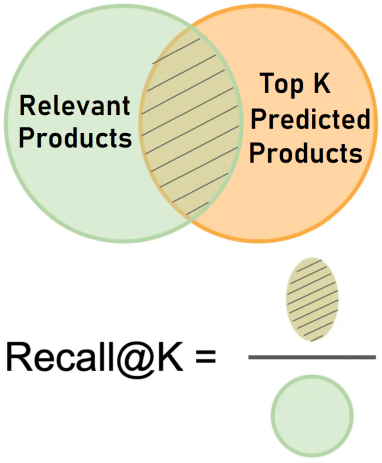

Figura 5.6: *Ilustración del recall*

Fuente: [medium.com/stanford-cs224w/track-neural-recommender-system-]

Recapitulando lo anterior, los resultados obtenidos para un entrenamiento con una capa y 4000 *epoch* han sido los siguientes:

```
Final Precision: 0.75951
Final Recall: 0.75754
```

Figura 5.7: *Resultados finales del entrenamiento*

```
100% 4000/4000 [1:28:45<00:00, 1.32s/it]
[Iteration 0/4000], train_loss: 22.02955, val_loss: 21.98312,  recall_at_k 0.0, precision_at_k 0.0
[Iteration 200/4000], train_loss: 5.77462, val_loss: 10.98968,  recall_at_k 0.0, precision_at_k 0.0
[Iteration 400/4000], train_loss: 3.37604, val_loss: 7.22481,  recall_at_k 0.0156, precision_at_k 0.01801
[Iteration 600/4000], train_loss: 2.57112, val_loss: 6.09825,  recall_at_k 0.13599, precision_at_k 0.13959
[Iteration 800/4000], train_loss: 2.14752, val_loss: 5.54263,  recall_at_k 0.17612, precision_at_k 0.18012
[Iteration 1000/4000], train_loss: 1.88545, val_loss: 5.18292,  recall_at_k 0.20498, precision_at_k 0.20914
[Iteration 1200/4000], train_loss: 1.70888, val_loss: 4.91777,  recall_at_k 0.22344, precision_at_k 0.22771
[Iteration 1400/4000], train_loss: 1.58328, val_loss: 4.70873,  recall_at_k 0.24665, precision_at_k 0.25057
[Iteration 1600/4000], train_loss: 1.4902, val_loss: 4.5369,  recall_at_k 0.26092, precision_at_k 0.2645
[Iteration 1800/4000], train_loss: 1.41915, val_loss: 4.39114,  recall_at_k 0.29627, precision_at_k 0.29962
[Iteration 2000/4000], train_loss: 1.36344, val_loss: 4.26429,  recall_at_k 0.33003, precision_at_k 0.33308
[Iteration 2200/4000], train_loss: 1.31889, val_loss: 4.15104,  recall_at_k 0.38127, precision_at_k 0.38414
[Iteration 2400/4000], train_loss: 1.28254, val_loss: 4.04787,  recall_at_k 0.46677, precision_at_k 0.46941
[Iteration 2600/4000], train_loss: 1.25152, val_loss: 3.95353,  recall_at_k 0.53712, precision_at_k 0.53966
[Iteration 2800/4000], train_loss: 1.22549, val_loss: 3.86427,  recall_at_k 0.62306, precision_at_k 0.62536
[Iteration 3000/4000], train_loss: 1.203, val_loss: 3.77952,  recall_at_k 0.64424, precision_at_k 0.64628
[Iteration 3200/4000], train_loss: 1.1832, val_loss: 3.69943,  recall_at_k 0.67999, precision_at_k 0.68188
[Iteration 3400/4000], train_loss: 1.16581, val_loss: 3.62049,  recall_at_k 0.70593, precision_at_k 0.70779
[Iteration 3600/4000], train_loss: 1.15002, val_loss: 3.54527,  recall_at_k 0.73143, precision_at_k 0.73325
[Iteration 3800/4000], train_loss: 1.13584, val_loss: 3.47247,  recall_at_k 0.74315, precision_at_k 0.74475
Tiempo transcurrido: 5325.246889829636 seconds
```

Figura 5.8: *Muestra de Iterations (epochs) durante el entrenamiento*

Se ha alcanzado una *precisión* final de *0,75951* lo que significa que de todas las predicciones positivas realizadas por el modelo, el *75%* han sido verdaderamente positivas. En cuanto al *recall final*, este ha sido de *0,75754* significando en que el modelo es capaz identificar y recuperar el *75%* de todos los casos positivos presentes en los datos.

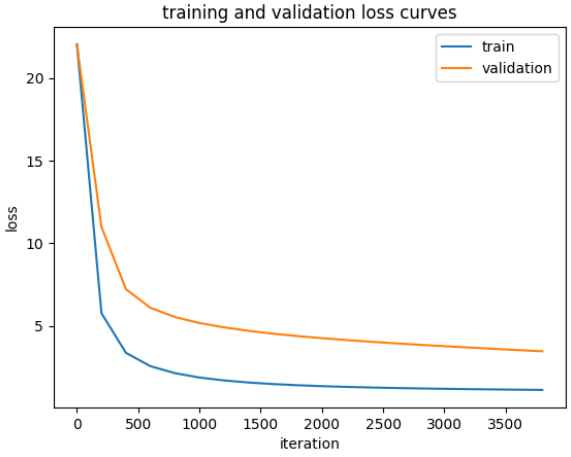

Figura 5.9: *Gráfico de representación de la pérdida*

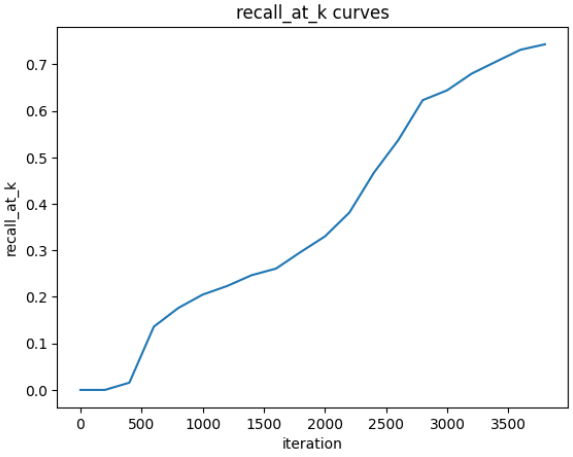

Figura 5.10: *Gráfico de representación del recall*

Como vemos, los resultados obtenidos demuestran la eficacia de las GNN en sistemas recomendadores. Sin embargo, observando las gráficas podemos llegar a la conclusión de que a partir de 1500 *epoch*, el modelo no mejora demasiado la pérdida, mientras que si que mejora considerablemente el recall. Esto último puede ser debido a que a partir de este punto el modelo empiece a sobreentrenar. Este sobreentrenamiento puede estar producido por diversos factores entre los que se encuentran:

- **Tamaño insuficiente del conjunto de entrenamiento:** si el conjunto de entrenamiento es pequeño, es posible que el modelo memorice los datos en lugar de aprender patrones generales.

- **Complejidad del modelo:** si el modelo de GNN tiene demasiados parámetros en comparación con la cantidad de datos disponibles, existe un mayor riesgo de sobreajuste. El modelo puede aprender ruido o detalles irrelevantes en lugar de patrones útiles.

- **Falta de regularización:** la falta de técnicas de regularización, como la regularización *L1* o *L2*, la técnica de abandono (*dropout*) o la normalización del lote (*batch normalization*), puede permitir que el modelo se ajuste demasiado a los datos de entrenamiento.

- **Número excesivo de *epochs* de entrenamiento:** si se entrena durante un número excesivo de épocas, el modelo puede ajustarse demasiado a los datos de entrenamiento y perder su capacidad de generalización en nuevos datos. En este caso, vemos que a partir de 1500 *epoch* el modelo no mejora demasiado la pérdida.

Para solucionar esto podemos aplicar soluciones como la *validación cruzada*, evaluando múltiples modelos con diferentes parámetros para seleccionar el mejor; hasta la *parada temprana*, en donde monitorizamos las métrica de evaluación para detener el entrenamiento cuando el rendimiento deja de mejorar, evitando así el sobreajuste.

Dicho todo esto, pasemos a mostrar el objetivo real de este entrenamiento y de los sistemas recomendadores: las recomendaciones. A continuación se puede observar (ver Figura 5.9) las primeras *10* recomendaciones realizadas para el usuario *A5QQOOZJOVPSF*. En este punto es donde utilizamos los metadatos anteriormente mencionados, de tal manera que podemos mostrar el nombre y precio de los productos en cuestión.

```
Recomendations for user A5QQOOZJOVPSF: (ordered by preference):
Product 1 = Name: Amazon.com Gift Card in a Holiday Sprig Box   price:
Product 2 = Name: Amazon.com Gift Card in a Santa Smile Tin   price: $25.00
Product 3 = Name:  Amazon Gift Card - Print - Happy Birthday (Balloons)   price:
Product 4 = Name:  Amazon eGift Card - Thank You (Smile)   price:
Product 5 = Name: Amazon.com Gift Card in a Kraft Paper Reveal with Jingle Bells   price: $20.00
Product 6 = Name: Amazon.com Gift Card in a Polka Dot Reveal (Classic Black Card Design)   price: $15.00
Product 7 = Name:  Amazon eGift Card - Birthday (Make a Wish)   price:
Product 8 = Name: Amazon.com Gift Card with a Holiday Teddy Bear - Limited Edition   price:
Product 9 = Name:  Amazon eGift Card - Happy Birthday (Balloons)   price:
Product 10 = Name:  Amazon eGift Card - Amazon Boxes   price:
```

Figura 5.11: *Ejemplo de recomendaciones realizadas para un usuario*

Una de las conclusiones a las que se llegó a partir de la visualización de múltiples recomendaciones para diversos usuarios, es que siempre giraban en torno a los mismos productos. Esto puede deberse a que se produce un sesgo (*bias*) que puede estar provocado o bien por el sobreentrenamiento anteriormente mencionado, o porque las reseñas del *dataset* están acotadas a una categoría concreta de producto, por lo que sí hay productos con muy buena puntuación general, estos pueden ser recomendables para todos los usuarios.

Capítulo 6

Conclusiones y líneas futuras

6.1 Conclusiones

En este trabajo de fin de grado se ha explorado el campo de las redes neuronales de grafos (GNN) y se ha estudiado, a través de un ejemplo, su aplicación en sistemas recomendadores. A lo largo del proyecto, hemos demostrado el potencial de los grafos para representar y analizar relaciones complejas entre usuarios y productos, y cómo las GNN pueden aprovechar esta estructura para mejorar la precisión y efectividad de los sistemas recomendadores.

El desarrollo de esta prueba de concepto completa ha permitido comprender y aplicar los pasos clave para construir un sistema recomendador basado en GNN, desde la adquisición y transformación de datos hasta el entrenamiento y evaluación del modelo. Los resultados obtenidos en la prueba de concepto respaldan la eficacia de las GNN en términos de rendimiento y capacidad para capturar relaciones no lineales en datos de grafos. Además, se ha destacado la importancia de elegir la arquitectura de GNN adecuada y de explorar técnicas de preprocesamiento y regularización para optimizar el rendimiento del modelo.

Por todo ello, hemos demostrado que las Redes Neuronales de Grafos son una herramienta poderosa y prometedora en el campo de la inteligencia artificial. Su capacidad para capturar relaciones complejas en datos de grafos los convierte en un enfoque invaluable para mejorar los sistemas recomendadores y abordar otros problemas de aprendizaje automático y, a medida que avanzamos hacia un futuro impulsado por datos masivos y estructuras complejas, las GNN seguirán siendo una área de investigación activa y prometedora, siendo el futuro del aprendizaje automático y jugando un papel crucial en la evolución de la IA.

6.1 Líneas futuras

Las *Graph Neural Networks* (GNN) han mostrado un gran potencial en diversos campos de aplicación. Estas son algunas de las principales líneas futuras de investigación y desarrollo en el campo de las GNN:

- **Mejora de la arquitectura y la capacidad de modelado:** se busca diseñar arquitecturas de GNN más sofisticadas y eficientes, que sean capaces de capturar

patrones aún más complejos en los grafos. Esto incluye la exploración de nuevas capas de propagación de mensajes, técnicas de atención y mecanismos de agregación de información más avanzados.

- **Manejo de grafos dinámicos:** los grafos que evolucionan en el tiempo presentan un desafío interesante para las GNN. La investigación se centra en desarrollar métodos que puedan adaptarse y actualizar las representaciones de los nodos y las aristas a medida que el grafo cambia, lo que permitirá abordar escenarios de datos en constante cambio.

- **Incorporación de información multimodal:** la integración de múltiples modalidades de datos en las GNN es un área de investigación prometedora. Se busca desarrollar modelos que puedan fusionar y utilizar de manera efectiva información proveniente de diferentes fuentes, como texto, imágenes, audio, etc., para realizar tareas de predicción más precisas y completas.

- **Interpretabilidad y explicabilidad:** a medida que las GNN se utilizan en aplicaciones críticas, como la medicina o la toma de decisiones en tiempo real, se vuelve fundamental comprender y explicar las decisiones tomadas por el modelo. La investigación se enfoca en desarrollar técnicas que permitan comprender y visualizar el proceso de toma de decisiones de las GNN, mejorando así su interpretabilidad y confiabilidad.

- **Escalabilidad y eficiencia:** las GNN pueden enfrentar desafíos de escalabilidad al tratar con grandes grafos o grandes volúmenes de datos. Se busca desarrollar técnicas de entrenamiento y representación más eficientes, así como estrategias de muestreo y agrupamiento inteligentes, para poder aplicar las GNN a conjuntos de datos cada vez más grandes y complejos.

Estas áreas de investigación prometen abrir nuevas oportunidades y aplicaciones para las GNN en diversos campos y contribuir al avance de la inteligencia artificial basada en grafos.

Capítulo 7
Summary and Conclusions

In this undergraduate thesis, we have explored the field of Graph Neural Networks (GNN) and studied their application in recommender systems through an example. Throughout the project, we have demonstrated the potential of graphs to represent and analyze complex relationships between users and products, and how GNNs can leverage this structure to improve the accuracy and effectiveness of recommender systems.

The development of this complete proof of concept has allowed us to understand and apply key steps in building a GNN-based recommender system, from data acquisition and transformation to model training and evaluation. The results obtained in the proof of concept support the effectiveness of GNNs in terms of performance and their ability to capture nonlinear relationships in graph data.

Furthermore, we have highlighted the importance of choosing the appropriate GNN architecture and exploring preprocessing and regularization techniques to optimize model performance.

Therefore, we have demonstrated that Graph Neural Networks are a powerful and promising tool in the field of artificial intelligence. Their ability to capture complex relationships in graph data makes them an invaluable approach for improving recommender systems and addressing other machine learning problems. As we move towards a future driven by massive data and complex structures, GNNs will continue to be an active and promising area of research. They represent the future of machine learning and play a crucial role in the evolution of AI.

Capítulo 8

Presupuesto

En este capítulo se hace una descripción del presupuesto del proyecto. En la Tabla 8.1 se muestra un desglose del presupuesto, dividido en 8 partes diferenciadas: Revisión bibliográfica, estudio de antecedentes, búsqueda y elección de *datasets*, transformación y procesamiento de los datos, investigación sobre implementación de GNN, desarrollo del código, pruebas, análisis y resultados, y redacción de la memoria

8.1 Presupuesto del proyecto

Fases	Horas	Precio
Revisión bibliográfica	40	20€ / hora
Estudio de antecedentes y estado del arte	40	20€ / hora
Búsqueda y elección de datasets	20	10€ / hora
Transformación y procesamiento de los datos	30	20€ / hora
Investigación sobre implementación de GNN	30	25€ / hora
Desarrollo del código	80	30€ / hora
Pruebas, análisis y resultados	15	5€ / hora
Redacción de la memoria	55	30€ / hora
Total	**310**	**5925**

Tabla 7.1: *Tabla de presupuesto*

Apéndice A: Código destacable

A.1 Ejemplo de función para transformar de matriz de interacción a matriz de adyacencia

/***

gnn_amazon_recommender.ipynb

****

DESCRIPCIÓN: Función para crear la matriz de adyacencia a partir de una matriz de interacción de un grafo bipartito. Los parámetros son los tensores

***/

```
def convert_r_mat_edge_index_to_adj_mat_edge_index(input_edge_index,
input_edge_values):
    R = torch.zeros((num_users, num_products))
    for i in range(len(input_edge_index[0])):
        row_idx = input_edge_index[0][i]
        col_idx = input_edge_index[1][i]
        R[row_idx][col_idx] = input_edge_values[i] # assign actual edge value to
Interaction Matrix


    R_transpose = torch.transpose(R, 0, 1)

    # create adj matrix
    adj_mat = torch.zeros((num_users + num_products , num_users + num_products))
    adj_mat[: num_users, num_users :] = R.clone()
    adj_mat[num_users :, : num_users] = R_transpose.clone()

    adj_mat_coo = adj_mat.to_sparse_coo()
    adj_mat_coo_indices = adj_mat_coo.indices()
    adj_mat_coo_values = adj_mat_coo.values()
    del adj_mat
    return adj_mat_coo_indices, adj_mat_coo_values
```

A.2 Ejemplo de función mejorada para transformar de matriz de interacción a matriz de adyacencia

/***

gnn_amazon_recommender.ipynb

```
**********************************************************************************
```

DESCRIPCIÓN: Función para crear la matriz de adyacencia a partir de una matriz de interacción de un grafo bipartito. Los parámetros son los tensores. En este caso la función no requiere ocupar grandes cantidades de memoria RAM ya que se trabaja únicamente con matrices dispersas sin necesidad de trabajar con matrices densas en el proceso

```
**********************************************************************************/

def convert_r_mat_edge_index_to_adj_mat_edge_index(input_edge_index,
input_edge_values):
    num_nodes = input_edge_index.max().item() + 1

    shape = (num_nodes, num_nodes)

    # Build original sparse matrix
    input_edge_index[1] += nodes
    adj_mat_coo = torch.sparse_coo_tensor(input_edge_index, input_edge_values,
shape)

    # Calculate transpose sparse matrix
    adj_mat_transpose_coo = torch.sparse_coo_tensor(input_edge_index.flip(0),
input_edge_values, shape)

    # Merge sparse matrix
    adj_mat_combined = adj_mat_coo.coalesce() + adj_mat_transpose_coo.coalesce()

    adj_mat_combined_coo = adj_mat_combined.to_sparse_coo()

    return adj_mat_combined_coo.indices(), adj_mat_combined_coo.values()
```

A.3 Modelo de GNN LightGCN

```
/**********************************************************************************
```

gnn_amazon_recommender.ipynb

```
**********************************************************************************
```

DESCRIPCIÓN: Definición de la clase LightGCN usando la librería Pytorch

```
**********************************************************************************/

class LightGCN(MessagePassing):
    def __init__(self, num_users, num_items, embedding_dim=64, K=3,
add_self_loops=False, dropout_rate=0.1):
        super().__init__()
        self.dropout_rate = dropout_rate
        self.num_users = num_users
        self.num_items = num_items
        self.embedding_dim = embedding_dim
        self.K = K
```

```
        self.add_self_loops = add_self_loops

        self.users_emb = nn.Embedding(num_embeddings=self.num_users,
embedding_dim=self.embedding_dim) # e_u^0
        self.items_emb = nn.Embedding(num_embeddings=self.num_items,
embedding_dim=self.embedding_dim) # e_i^0

        nn.init.normal_(self.users_emb.weight, std=0.1)
        nn.init.normal_(self.items_emb.weight, std=0.1)

        self.out = nn.Linear(embedding_dim + embedding_dim, 1)

    def forward(self, edge_index: Tensor, edge_values: Tensor):
        edge_index_norm = gcn_norm(edge_index=edge_index,
                                   add_self_loops=self.add_self_loops)
        emb_0 = torch.cat([self.users_emb.weight, self.items_emb.weight])
        embs = [emb_0]
        emb_k = emb_0

        for i in range(self.K):
            emb_k = self.propagate(edge_index=edge_index_norm[0], x=emb_k,
norm=edge_index_norm[1])
            embs.append(emb_k)

        embs = torch.stack(embs, dim=1)
        emb_final = torch.mean(embs, dim=1)
        users_emb_final, items_emb_final = torch.split(emb_final,
                                                       [self.num_users,
self.num_items])
        r_mat_edge_index, _ =
convert_adj_mat_edge_index_to_r_mat_edge_index(edge_index, edge_values)
        src, dest =  r_mat_edge_index[0], r_mat_edge_index[1]

        user_embeds = users_emb_final[src]
        item_embeds = items_emb_final[dest]
        output = torch.cat([user_embeds, item_embeds], dim=1)
        output = self.out(output)

        return output

    def message(self, x_j, norm):
        return norm.view(-1, 1) * x_j
```

Apéndice B: Repositorio del proyecto

B.1 Repositorio de la prueba de concepto

En el siguiente enlace se adjunta el repositorio público con el código de la prueba de concepto del sistema recomendador de Amazon.

Enlace a repositorio del proyecto

Bibliografía

[1] Y. LeCun, Y. Bengio and G. Hinton, '"Deep learning," Nature, vol. 521, no. 7553, pp. 436-444.

[2] R. Kline, '"Cybernetics, Automata Studies, and the Dartmouth Conference on Artificial Intelligence," *IEEE Annals of the History of Computing*, vol. 33, no. 4, pp. 5-16.

[3] N. N. Pise and P. Kulkarni, '"A Survey of Semi-Supervised Learning Methods," 2008 International Conference on Computational Intelligence and Security, vol. 2, pp. 30-34.

[4] V. Gupta, V. K. Mishra, P. Singhal and A. Kumar, '"An Overview of Supervised Machine Learning Algorithm," 2022 11th International Conference on System Modeling & Advancement in Research Trends (SMART), pp. 87-92.

[5] M. Usama, J. Qadir, A. Raza, H. Arif, K. -l. A. Yau, Y. Elkhatib, A. Hussain and A. Al-Fuqaha, '"Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges," *IEEE Access*, vol. 7, pp. 65579-65615.

[6] L. Lyu, Y. Shen and S. Zhang, '"The Advance of Reinforcement Learning and Deep Reinforcement Learning," *2022 IEEE International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA)*, pp. 644-648.

[7] Olivier Chapelle, Bernhard Schölkopf and Alexander Zien, '"Semi-Supervised Learning in Practice,", 2006.

[8] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533-536.

[9] S. Paliwal, S. Kumar Khatri and M. Sharma, '"Sentiment Analysis and Prediction Using Neural Networks," 2018 International Conference on Inventive Research in

[10] W. Zachary, '"An Information Flow Model for Conflict and Fission in Small Groups1," *Journal of anthropological research*, vol. 33.

[11] Z. Ye, Y. J. Kumar, G. O. Sing, F. Song and J. Wang, '"A Comprehensive Survey of Graph Neural Networks for Knowledge Graphs," *IEEE Access*, vol. 10, pp. 75729-75741.

[12] X. Fan, M. Gong, Y. Wu, A. K. Qin and Y. Xie, '"Propagation Enhanced Neural Message Passing for Graph Representation Learning," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 2, pp. 1952-1964.

[13] Y. Gao, Y. Feng, S. Ji and R. Ji, '"HGNN+: General Hypergraph Neural Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 3, pp.

3181-3199.

[14] T. Kasanishi, X. Wang and T. Yamasaki, '"Edge-Level Explanations for Graph Neural Networks by Extending Explainability Methods for Convolutional Neural Networks," 2021 IEEE International Symposium on Multimedia (ISM), pp. 249-252.

[15] Pedro Almagro Blanco, '"Descubrimiento de Conocimiento en Grafos Multi-Relacionales," Abril de 2017, pp. 131-144.

[16] L. Ruiz, F. Gama and A. Ribeiro, '"Spatial Gating Strategies for Graph Recurrent Neural Networks," ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5550-5554.

[17] CUDA: https://docs.nvidia.com/cuda/

[18] Pytorch: https://pytorch.org/

[19] TensorFlow: https://www.tensorflow.org/?hl=es-419

[20] Deep Graph Library: https://www.dgl.ai/

[21] StellarGraph: https://stellargraph.readthedocs.io/en/stable/

[22] Spektral: https://graphneural.network/

[23] Kagel: https://www.kaggle.com/

[24] *Network Repository: https://networkrepository.com/*

[25] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, “LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation.” arXiv, Jul. 07, 2020. doi: 10.48550/arXiv.2002.02126.

[26] Z. Zhang, '"Improved Adam Optimizer for Deep Neural Networks," *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, pp. 1-2.