Universitat Oberta de Catalunya

uoc.edu

Definición, tipologías y casos de uso de Graph Neural Networks para el aprendizaje basado en relaciones

Alfonso Moure Ortega
Grado de Ingeniería Informática
Inteligencia Artificial

David Isern Alarcón
Carles Ventura Royo

30/12/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	Definición, tipologías y casos de uso de Graph Neural Networks para el aprendizaje basado en relaciones
Nombre del autor:	Alfonso Moure Ortega
Nombre del consultor/a:	David Isern Alarcón
Nombre del PRA:	Carles Ventura Royo
Fecha de entrega (mm/aaaa):	12/2021
Titulación:	Grado de Ingeniería Informática
Área del Trabajo Final:	Inteligencia Artificial
Idioma del trabajo:	Castellano
Palabras clave	deep learning, graph neural network, node classification

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

La evolución del aprendizaje computacional y, en particular, el del aprendizaje profundo, ha acelerado de manera drástica en los últimos años. Esta progresión se ha visto centrada en el uso de datos donde sus entidades no tienen ningún tipo de dependencia entre sí: no existen jerarquías, relaciones de orden o de dependencia. Sin embargo, este proyecto se centra en dicha laguna: en la exploración de la evolución, contexto, casos de uso y técnicas existentes para el aprendizaje sobre estructuras de grafo mediante redes neuronales gráficas.

Para ello, tras repasar la base histórica y teórica de las redes neuronales, se estudian distintas aproximaciones y tipos de problemas que pueden ser resueltos, para lo que se ofrece una clasificación de tipologías de modelos según su aproximación al aprendizaje.

De este modo, se presentan los problemas principales sobre los que aplicar esta particularización de las redes neuronales: clasificar y predecir vértices y nodos de un grafo, catalogar estructuras, predecir cambios dentro del dominio o generar grafos según ejemplos aprendidos.

Una vez se ha hecho este repaso teórico, se ha procedido a implementar con Python modelos de predicción de nodos mediante una aproximación espectral y otra espacial, así como un ejemplo de predicción de enlaces. Además, con el

objetivo de contrastar los resultados, se ha realizado una implementación de clasificación de nodos mediante un algoritmo que no tenga en cuenta la estructura.

Como conclusión, se destaca la alta eficacia de estos modelos para la resolución de problemas sin hacer uso de espacios euclídeos.

Abstract (in English, 250 words or less):

The evolution of machine learning and, in particular, of deep learning, has accelerated drastically in the last decade. However, this development has been focused on the use of datasets where each sample is independent from the rest, without hierarchy, order or any other kind of relationship. This project is focused on exploring the evolution, contexts, use cases and different methods to perform machine learning operations over graphs using graph neural networks.

This work starts by reviewing the history and theoretical knowledge about neural networks, exploring different approximations, frameworks and types of problems. To do so, this work presents a classification of models based on its approach to learning.

Different kinds of problems are presented that can be solved with this particular type of neural networks: node and edge classification, graph classification, predicting changes in the knowledge domain or generating new graphs based on examples.

Following the review of these theoretical aspects, a collection of implementations is presented in a Python notebook using spectral and spatial approximations to graphs to classify nodes, as well as a link prediction model. Also, to be able to compare results, an implementation based on a traditional model that doesn't attend to data structure is used to classify nodes.

At the end, all results are compared and a collection of conclusions is presented to address the motivation of using this type of models, showing that they can outperform those based on euclidean spaces when structured data is present.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

El campo del aprendizaje computacional ha tomado un importante papel en el desarrollo de nuevos productos y soluciones durante los últimos años. Sin embargo, las raíces de esta disciplina nacen de diferentes trabajos realizados durante los años sesenta. Fue en aquellos primeros trabajos cuando se comenzaron a aplicar algoritmos para, entre otros, la clasificación de imágenes y textos [1].

En un inicio, muchos problemas eran reducidos a la transformación de los datos de entrada en elementos dentro de un espacio euclidiano y a la búsqueda de hiperplanos, con el fin de separar las entidades en diferentes grupos o categorías. Este tipo de algoritmos, aunque han evolucionado con el paso de los años, siguen siendo la base de muchas soluciones de aprendizaje automático [2].

Por otro lado, se han desarrollado soluciones que basan su trabajo en la optimización que, si bien también hacen uso de cálculos euclídeos, fundamentan su aprendizaje en la aproximación de una función, ya sea para reducir el coste de error o el éxito de sus predicciones [3,4]. Un buen ejemplo de este tipo de soluciones son aquellas que utilizan redes neuronales, donde destaca el aprendizaje profundo, más conocido por su nombre en inglés: «*deep learning*».

En cualquier caso, estas dos clases de aproximaciones a los problemas buscan identificar patrones en la distribución de los datos en el espacio, lo que exige una extracción y preparación de las entidades para ser representadas mediante vectores. Por ejemplo, la clasificación de los objetos existentes dentro de una colección de documentos puede requerir la extracción de las palabras que lo forman y construir con ellas vectores con su densidad, aparición o no en el texto y otros, siempre mediante valores numéricos [5].

Existen, sin embargo, soluciones que se centran en el aprendizaje sobre estructuras de datos relacionales representadas mediante grafos [2,6,7]. Así, las entidades de los datos se presentan como nodos que pueden ir acompañados de atributos, tales como su identificación o características principales, mientras que las relaciones existentes entre las mismas son simbolizadas como vértices [6,8].

Este tipo de modelos pueden ser usados para resolver diversos tipos de problemas, tales como la clasificación de nodos teniendo en cuenta su entorno de relaciones [2], la predicción de la probabilidad de que dos entidades estén vinculadas [9], la identificación de estructuras completas [8] o, incluso, la generación de otras en base a diferentes ejemplos [10].

Para el trabajo en este tipo de escenarios, es posible el uso de modelos basados en *deep learning* tales como las redes neuronales gráficas (*graph*

neural networks, GNN) [7]. A diferencia de otros modelos, estos están diseñados para resolver problemas concretos mediante el aprendizaje de las relaciones explícitas entre objetos, así como en la tipología de estas vinculaciones y los atributos que las acompañan [2].

Así, pueden ser utilizados para llevar a cabo tareas donde las conexiones entre objetos sean importantes o para resolver problemas caracterizados por la secuencia de elementos o sucesos. Entre ellas, se pueden destacar la generación de recomendaciones (mediante la predicción de enlaces) [11], la generación de escenas sintéticas [10], la clasificación de imágenes y segmentación semántica de su contenido [2], la predicción y anticipación de sucesos mediante el análisis de los caminos que toman los eventos detectados [12], clasificación de nodos en base a sus contenidos y relaciones [2] y otros.

Con todo, el autor de este proyecto busca explorar diferentes problemas que pueden ser resueltos mediante este tipo de métodos, así como estudiar distintas categorías de modelos de GNNs. Para ello, se analizarán una colección de problemas, que serán aplicados sobre un conjunto de datos estructurado para, después, comparar la precisión obtenida en las predicciones frente a otros modelos de aprendizaje profundo donde no se hace uso de información gráfica o relacional.

1.2. Objetivos del Trabajo

El objetivo principal de este proyecto es estudiar el funcionamiento de modelos basados en redes neuronales gráficas (GNNs), la preparación de datos estructurados para las mismas y su nivel de eficacia contrastado frente a otros modelos.

Así, se puede definir la siguiente colección de objetivos, clasificados como principales (obligatorios dentro del marco de este trabajo) y secundarios (opcionales, cuya consecución estará ligada al éxito de los preferentes).

Objetivos prioritarios

1. Describir el funcionamiento conceptual de las redes neuronales gráficas.
2. Definir los distintos tipos de modelos, su funcionamiento y casos de uso.
3. Enmarcar estos modelos dentro del campo del aprendizaje profundo.
4. Preparar una estructura de datos apropiada para el uso de GNNs en base al juego de datos original. Se explicará el proceso completo y cada decisión será razonada.
5. Explorar la implementación de redes neuronales convolucionales gráficas (ConvGNN):
    a. Aproximación espectral: clasificación de nodos.
    b. Aproximación espacial: clasificación de nodos.
    c. Aproximación espacial: predicción de enlaces.
6. Extraer las métricas de precisión de cada implementación.
7. Contrastar los resultados obtenidos y las métricas de precisión frente al uso de una red neuronal tradicional.

8. Desarrollar conclusiones críticas sobre los resultados obtenidos y la comparación entre los distintos modelos.

Objetivos secundarios ordenados por importancia

1. Implementación de los ejemplos anteriores mediante una red neuronal recursiva gráfica (RecGNN).
2. Explorar casos de uso adicionales:
    a. Clasificación de grafos.
    b. Predicción de secuencias.
3. Representar de manera gráfica parte del juego de datos, con el fin de facilitar la comprensión de este tipo de modelos.

1.3. Enfoque y método seguido

Tal y como se ha expresado con anterioridad, el principal objetivo de este proyecto es la creación de un documento didáctico, donde se exploren las distintas soluciones para afrontar problemas asentados sobre datos caracterizados por una estructura relacional. En consecuencia, se seguirá un enfoque centrado en el aprendizaje, donde se empezará por las bases más sencillas de los mismos, para ir ascendiendo poco a poco en complejidad.

Además, puesto que se busca posicionar los modelos de aprendizaje basados en GNN en el marco general del *deep learning*, se presentarán casos prácticos sobre un juego de datos de ejemplo y se razonarán sus implementaciones.

Con todo, a continuación se listan las bases del método seguido en el proyecto:

- Los conceptos serán presentados en orden de complejidad, de manera que se asegure el conocimiento previo necesario para plantear los distintos casos de uso.
- Se asumirá un conocimiento apropiado dentro del campo del aprendizaje computacional y las matemáticas necesarias para trabajar en el mismo.
- Se explicará el funcionamiento de los modelos GNN desde la perspectiva teórica y matemática.
- Se llevarán a cabo las implementaciones de la preparación de datos y de los distintos problemas presentados mediante Python, así como las librerías TensorFlow[1], Keras[2] y PyTorch[3].
- Se hará uso de referencias de textos procedentes de estudios contrastados y que, a ser posible, contengan detalles comparativos de precisión frente a otros modelos.
- Se hará uso de un juego de datos formado por nodos y sus relaciones. Además, cada nodo deberá contar con un identificador único, una clase

1 TensorFlow es una librería propietaria para el lenguaje Python creada por Google para facilitar la implementación de proyectos de aprendizaje computacional: https://www.tensorflow.org/.

2 Keras es una librería para Python de código abierto que ofrece una amplia colección de modelos de redes neuronales: https://keras.io/.

3 PyTorch es una librería de Python de código abierto similar a TensorFlow: https://pytorch.org/.

que lo etiqueta y una colección de características explícitas. Así, se busca que:

- Sea viable llevar a cabo tareas de clasificación de nodos, tanto mediante modelos GNN como la aplicación de otros, tales como CNN.
- Sea posible transformarlo en una estructura de datos apta para su uso en redes neuronales gráficas, esto es, listados de observaciones y características y la matriz de adyacencia del grafo que representan.

Para ello, se ha optado por elegir el juego de datos conocido por el nombre de CORA [13], donde se recogen 2708 referencias de publicaciones científicas clasificables dentro de siete categorías. Además, se aporta una colección de enlaces entre las mismas, donde cada una representa una citación bibliográfica entre publicaciones, así como un vector de palabras existentes en cada documento, donde cada posición del mismo es un valor binario que indica si aparece en el mismo o no.

Como puede verse, estas características hacen que el juego de datos se ajuste a las necesidades ya establecidas para el trabajo.

A modo de nota final, se destaca que el autor tratará en todo momento de trabajar mediante el uso de herramientas libres o de código abierto, con el fin de alinearse mejor con el espíritu de cooperación que caracteriza el campo del aprendizaje automático, el cálculo computacional y la creación de librerías utilizadas dentro del mismo.

1.4. Planificación del Trabajo

A continuación, se presenta el diagrama de gantt del proyecto, que ha sido modificado al finalizar la PEC2 conforme a la motivación expuesta en el informe de seguimiento. Como se puede comprobar, aparece ordenado en base a los plazos de entrega de cada prueba de evaluación continua, empezando por la actual en el momento de redacción de este apartado (PEC1) y terminando con la defensa de proyecto.

Con el fin de reducir la complejidad del esquema se han excluido algunas subtareas demasiado pequeñas para resultar relevantes en el mismo, por lo que han quedado ocultas dentro de su tarea macro. Las acciones presentadas en el plan se centran sobre los objetivos principales y obligatorios del proyecto, por lo que aquellas que hacen alusión a objetivos secundarios no han sido incluidas y serán llevadas a cabo, en todo caso, durante la ejecución de la fase 2 del desarrollo (PEC4).

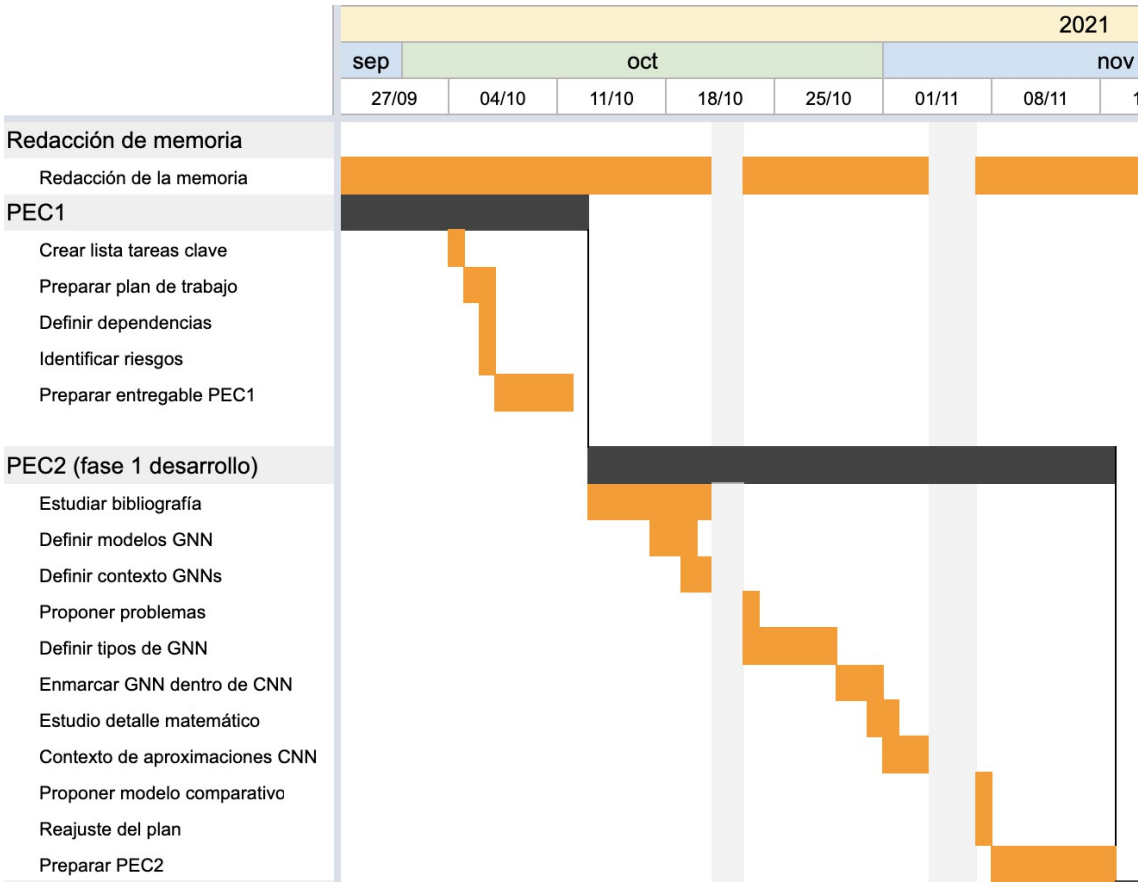
Por otro lado, se ha optado por añadir, sombreado en gris claro, aquellos días en los que el autor tiene claro que no podrá trabajar en el proyecto, debido a diferentes obligaciones profesionales y familiares. Además, se opta por

adelantar algunas de las tareas al inicio de la etapa oficial de las PECs 4 y 5a, con el fin de evitar posibles problemas de tiempos (más abajo se tratarán los riesgos de proyecto).

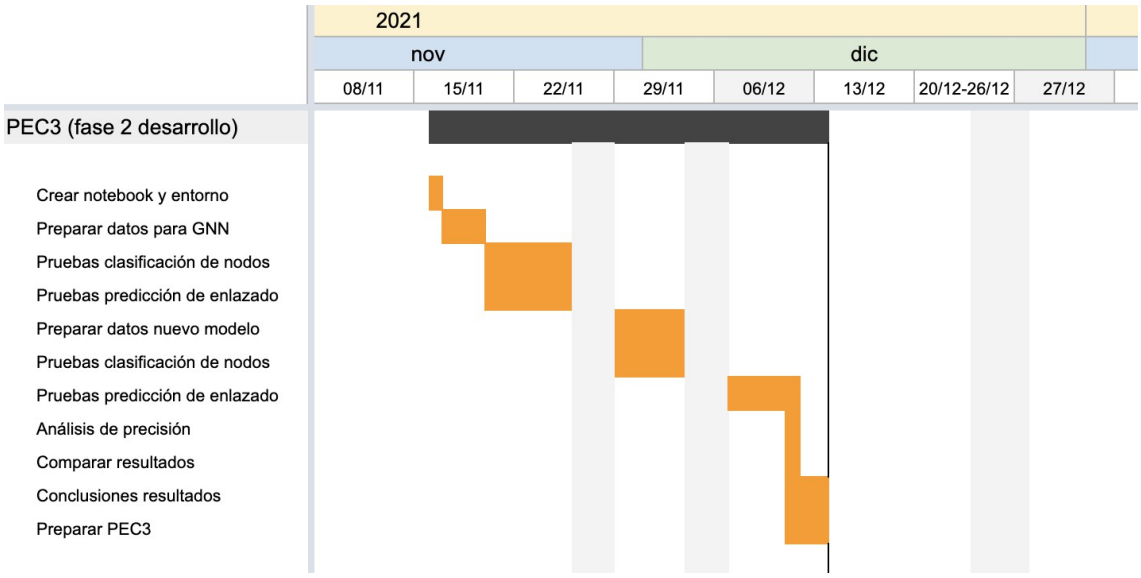
Así, se presenta el plan de trabajo para la PEC1 (en curso al redactar estas líneas) y la PEC2, donde se comenzará a trabajar en el proceso de desarrollo centrado en la primera parte de los objetivos: estudiar el funcionamiento de las GNN, plasmar sus conceptos y repasar sus distintas tipologías. Además, se procederá a entender y explicar los diferentes tipos de problemas que pueden ser resueltos.

Hecho esto, se planteará una introducción detallada a las redes convolucionales y la motivación de su uso como modelo base para trabajar con redes neuronales gráficas, así como sus distintas aproximaciones.

Además, se presenta la tarea de redacción de la memoria, que se llevará a cabo desde el principio del proyecto y hasta el día 10 de enero de 2022.



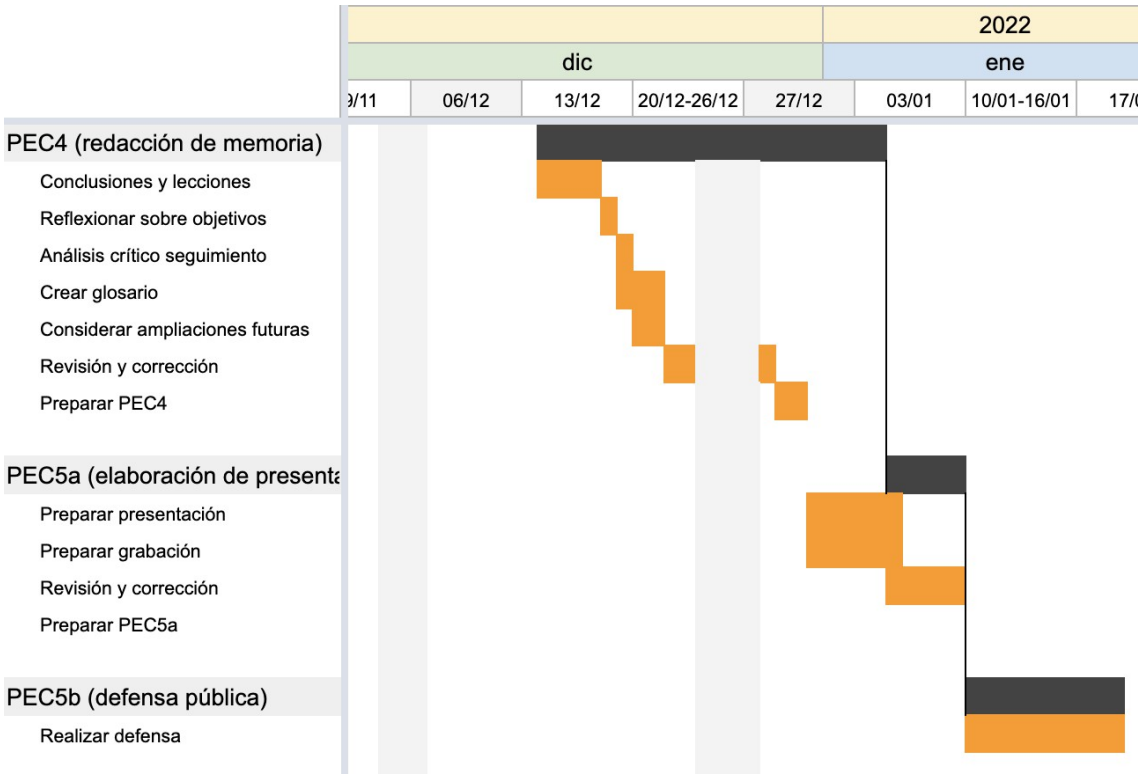
Después, se podrá mover el foco a la fase 2, que estará caracterizada por la implementación de los modelos estudiados en el bloque teórico:



Terminada la PEC3, se procederá a continuar con la preparación de la memoria, realizando una primera revisión. Con todo, se hará uso de lo aprendido para plantear una colección de conclusiones y lecciones aprendidas, algo que permitirá llegar a reflexionar sobre los modelos estudiados y las posibles mejoras que puedan quedar pendientes para el trabajo, entre las que se podrán encontrar, entre otros, los objetivos secundarios.

Llegado a este punto, se deberán realizar sucesivas revisiones y correcciones sobre la memoria de proyecto para evitar cualquier tipo de errata o incorrección.

Así, una vez que se tengan bien claras las ideas, conclusiones y aprendizaje, se procederá a preparar una presentación que resuma los contenidos del proyecto y se grabará una presentación del mismo. Para cerrar el proceso, se procederá a defender los resultados y la presentación ante el tribunal.



Por supuesto, existen una colección de tareas secundarias no menos importantes que han sido resumidas dentro de los bloques de preparación de cada PEC, a saber:

1. Anotar referencias bibliográficas nuevas, de haberlas.
2. Actualizar memoria de proyecto.
3. Preparar informe de seguimiento.
4. Reajustar plan de trabajo.

De esta manera, la memoria de proyecto y el plan de trabajo quedarán actualizados con las nuevas referencias bibliográficas utilizadas y con los plazos ajustados, en caso de ser necesario. Además, se preparará un informe de seguimiento con las impresiones sobre la progresión del mismo.

1.4.1. Justificación de los plazos elegidos

Es importante resaltar que el autor tiene experiencia en la programación con Python y tiene unos conocimientos rudimentarios de algunas de las librerías a utilizar, por lo que se considera que son plazos apropiados.

Se ha optado por no marcar los festivos o fines de semana, pues el autor avanzará diariamente en paralelo a sus otras obligaciones familiares y profesionales.

1.4.2. Riesgos para la planificación

Riesgo	Impacto	Probabilidad	Contramedida
Pérdida de datos	Alto	Baja	Se prepararán la memoria,

			plan de trabajo e informes de seguimiento mediante LibreOffice y se guardará una copia periódica en un servidor privado basado en Nextcloud, así como en la cuenta de Google Drive de la UOC. Todas las implementaciones de código serán guardadas en un repositorio privado en Github.
Problemas con conjunto de datos	Medio	Media	Se hará uso del conjunto de datos CORA [13] como principal del proyecto. Sin embargo, se dispone de otras dos fuentes de datos: Pubmed [14] y CiteSeer [15], a las que se podrá recurrir para hacer mejoras futuras.
Avería del ordenador	Bajo	Baja	Se dispone de un equipo alternativo. Además, al poder acceder al contenido vía NextCloud, Google Drive y Github, una avería no debería impactar de manera seria en el plan.
Carencias de potencia	Baja	Alta	En caso de encontrar problemas de rendimiento, se apostará por implementar las pruebas mediante Colab.

1.5. Recursos necesarios

Para la realización del trabajo se hará uso de los siguientes recursos:

- Editor de texto: LibreOffice, tras convertir la plantilla de Word proveída.
- Entorno de programación: Visual Studio Code.
- Entorno para notebooks interactivos: Jupyter dentro de Visual Studio Code.
- Entorno alternativo: Colab.

1.6. Breve sumario de productos obtenidos

Como resultado de este proyecto, se obtienen los siguientes productos:

1. Memoria de proyecto, donde se presenta una aproximación didáctica a las redes neuronales gráficas, así como ejemplos de implementación de algunos de sus usos comunes.
2. Cuaderno *notebook* creado mediante Jupyter con los ejemplos que se presentan en más detalle dentro de la memoria.
3. Cuaderno *notebook* utilizado para tareas de reducción de dimensionalidad para el experimento tres, que ha sido separado del principal por su elevado coste en tiempo de ejecución, pero que se incorpora como referencia.
4. Repositorio con el código completo de ambos *notebooks*, accesibles en https://github.com/ghostmou/uoc-tfg-gnn.

1.7. Breve descripción de los otros capítulos de la memoria

Esta memoria presenta los siguientes apartados:

- Apartado 1: Introducción, donde se expone la motivación para el proyecto y la temática escogida.
- Apartado 2: Desarrollo del trabajo, lugar en el que se recoge el resultado del proyecto:
    - 2.1: Introducción a las redes neuronales. Se establece el contexto del *deep learning* y sus orígenes inspirados en la biología del cerebro de los vertebrados.
        - 2.1.1: Historia y conceptos básicos. Se repasa su historia, con el fin de centrar la explicación dentro del marco completo de este campo de la inteligencia artificial.
        - 2.1.2: Fundamentos del aprendizaje. Se presenta el concepto de aprendizaje y su funcionamiento.
    - 2.2: Introducción a las redes neuronales gráficas. A modo de especialización de las redes neuronales, se plantea la motivación de su existencia y sus bases matemáticas para representar las estructuras de datos a utilizar. Además:
        - 2.2.1: Tipos de problemas. Descripción de los tipos de problemas

que pueden trabajarse mediante redes gráficas.

- 2.2.2: Representación de grafos y aprendizaje. Se centra en exponer las técnicas utilizadas para representar estructuras de grafo.

- 2.3: Aprendizaje y predicción mediante redes neuronales. Se presentan los distintos tipos de redes neuronales gráficas y se profundiza en las redes convolucionales.
  - 2.3.1: Introducción a las redes neuronales convolucionales gráficas (ConvGNN). Se profundiza en las aplicaciones de este tipo de redes para resolver determinados problemas sobre datos estructurados.
  - 2.3.2: Aproximación a ConvGNN basada en teoría espectral. Se presentan distintos tipos de redes convolucionales que hacen uso de la teoría espectral de grafos para realizar predicciones.
  - 2.3.3: Aproximación a ConvGNN basada en paso de mensajes. Como en el apartado anterior, se exponen diferentes clases de redes convolucionales, en este caso basadas en una aproximación espacial a los grafos y sobre el paso de mensajes.
  - 2.3.4: Resumen de modelos. Cuadro resumen de los modelos estudiados en el apartado 2.3 y sus referencias bibliográficas, con el fin de poder acceder de manera más rápida y eficaz a información adicional.

- 2.4: Implementación de pruebas. Se procede a implementar cuatro pruebas de los modelos expuestos a modo de experimentos.
  - 2.4.1: Preparación del entorno. Explicación sobre la preparción del entorno de implementación.
  - 2.4.2: Conjunto de datos usado en las pruebas. Se presenta el conjunto de datos desde una perspectiva técnica y de estructura. Además, se fijan las bases sobre su proceso de carga.
  - 2.4.3: Pruebas de clasificación de nodos. Se aplican distintas técnicas para clasificar nodos dentro de un grafo.
    - 2.4.3.1. Experimento 1: clasificación de nodos mediante modelo espectral. Se hace uso de TensorFlow para implementar una ConvGNN mediante método espectral aplicada a la clasificación de nodos. Se obtienen mediciones de precisión y error, tanto durante el proceso de aprendizaje como de evaluación de generalización.
    - 2.4.3.2. Experimento 2: clasificación de nodos mediante modelo espacial. Se repite la implementación del punto anterior, en este caso mediante la técnica de paso de

mensajes. Se obtienen mediciones de precisión y error, tanto durante el proceso de aprendizaje como de evaluación de generalización

        - 2.4.3.3. Experimento 3: clasificación mediante modelo no gráfico. Se implementa una red neuronal tradicional para clasificar los nodos de un grafo. Se obtienen mediciones de precisión y error, tanto durante el proceso de aprendizaje como de evaluación de generalización.
    - 2.4.4: Pruebas de predicción de enlaces.
        - 2.4.4.1. Experimento 4: predicción de enlaces mediante WalkPooling. Se hace uso de un modelo de paso de mensajes mediante una capa convolucional donde se aplica walk pooling. Se obtienen mediciones de precisión y error, tanto durante el proceso de aprendizaje como de evaluación de generalización.
    - 2.4.5: Conclusiones sobre las pruebas realizadas. Se comparan los resultados de entrenamiento y generalización de los cuatro experimentos. Se razona y ofrecen conclusiones sobre los mismos.
- Apartado 3: Conclusiones. Se recogen las conclusiones sobre el proyecto llevado a cabo.
- Apartado 4: Glosario. Colección de términos esenciales dentro del proyecto y su definición.
- Apartado 5: Bibliografía. Como se expone en los apartados 1.2 y 1.3, se hace un especial hincapié en la extensión y profundidad de la bibliografía utilizada para la realización del proyecto, cuyo fruto se presenta en este apartado.
- Apartado 6: Anexos. Contenidos adicionales que se entregan por separado.

2. Desarrollo del trabajo

2.1. Introducción a las redes neuronales

2.1.1. Historia y conceptos básicos

Las bases del conocimiento que se tienen sobre el cerebro humano nacen de los avances realizados en diferentes campos durante el siglo XIX: desde la primera visualización mediante microscopio de una célula nerviosa en 1836 de la mano de Gabriel Gustav Valentin, hasta las primeras publicaciones relacionadas con la teoría neuronal planteada por Ramón y Cajal a finales de la década de 1880. Gracias a los trabajos de este último, fue posible llegar a dos conclusiones clave: primero, que las neuronas son células independientes que se interconectan entre sí para permitir el paso de señales, siempre manteniendo su independencia y usando interfaces conocidos como sinapsis; segundo, que pueden especializarse para diferentes funciones y trabajar en base a normas dispares [16,17].

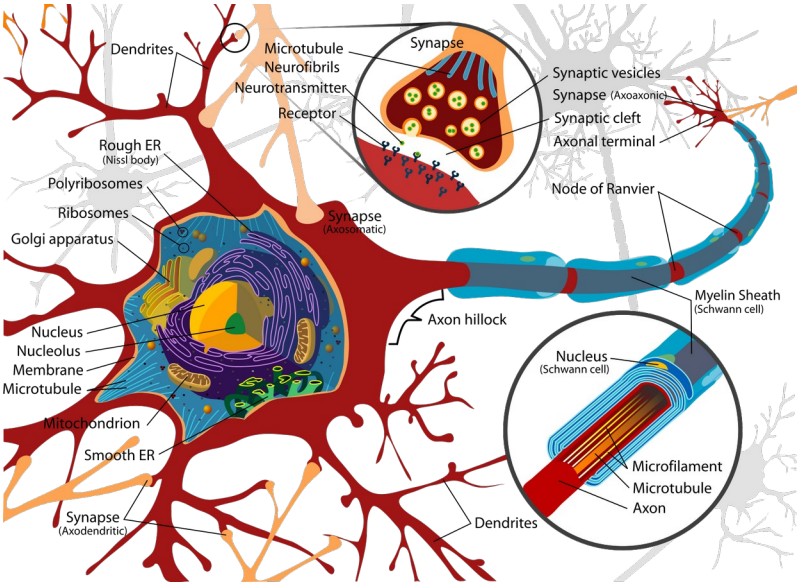


Figure 1: Diagrama de partes de una célula neuronal [70].

Durante las siguientes décadas, distintas técnicas fueron desarrolladas en el terreno del análisis y visualización de tejidos, lo que permitió profundizar en el entendimiento sobre la estructura y relaciones de estas células entre sí. De este modo, fue posible establecer la misión de cada parte de una célula nerviosa y su relación con el funcionamiento principal de la neurona: transformar señales de entrada, procedentes de las dendritas que conectan con el núcleo o soma de la neurona, en información en su salida, que es transmitida por el axón y diseminada mediante los nodos terminales de la

sinapsis mediante el uso de neurotransmisores, que viajan hacia las dendritas de otras células [18,19].

Dentro de este esquema, el soma juega un papel crucial: es la entidad encargada de ajustar la salida en base a la entrada que recibe. Esta misión es llevada a cabo de manera similar al funcionamiento de un transistor en cuanto a que, si las señales de entrada superan un cierto nivel de potencia, la salida se activa con determinada fuerza [19]. Esta relación entre entrada y salida no es inmutable: es flexible y se regula en base al paso del tiempo, la frecuencia en las variaciones de las entradas y otros factores bioquímicos a nivel cerebral [20,21].

Al igual que en otras tantas ramas científicas y tecnológicas, esta visión del sistema nervioso sirvió como inspiración para el desarrollo de un elemento clave dentro de la computación moderna: la invención de la neurona artificial (*artificial neuron* o ANN). Esta primera propuesta planteaba que los cerebros animales eran capaces de realizar cálculos basados en lógica proposicional mediante la combinación de diferentes neuronas [22]. De esta manera, una ANN activaba una salida con un valor lógico (esto es, verdadero o falso) en base a una combinación de entradas.

Gracias a esta forma, Frank Rosenblatt propuso en 1957 la arquitectura de neurona artificial conocida como perceptrón [23], donde entradas y salidas son números reales en lugar de valores binarios y en el que la activación de la salida es función de las entradas y de un peso asignado a cada una de las mismas. Este modelo, conocido como *threshold logic unit*, parece inspirado en el comportamiento adaptativo citado con anterioridad en la lógica de activación del soma neuronal. Su funcionamiento permitió, de manera rudimentaria, aproximar el cálculo del resultado de funciones no lineales mediante el aprendizaje del peso de cada entrada conectada a una neurona [3].

Como planteaba Ramón y Cajal en sus primeros trabajos, las neuronas no funcionan de manera individual, sino que forman parte de una intrincada red con complejas conexiones [16]. Esta idea fundamental inspiró la organización de los perceptrones en múltiples niveles interconectados entre sí: las entradas de un perceptrón se podían conectar a las salidas de otros anteriores. Este concepto llevó a poder aproximar funciones con más elevados grados de libertad, un paso fundamental que había frustrado algunas de las ambiciones y esperanzas depositadas en el modelo original [24]. Dentro del nuevo escenario, el aprendizaje continuó estando basado en la variación del peso de las entradas de cada unidad neuronal.

Aunque este nuevo planteamiento mejoró la precisión de los resultados obtenidos, su capacidad de aprendizaje era limitada y complicada de aplicar, lo que llevó a lo se conoce dentro del campo como el primer *invierno de la inteligencia artificial*, durante el que la inversión se redujo de manera notable [25]. Hasta 1986, estas redes se basaban en el concepto de *feedforward propagation*, donde el aprendizaje se construye mediante el ajuste de los pesos de las entradas de cada neurona de una red donde la información se mueve en una única dirección: desde las entradas hasta las salidas. Solo en el caso de las *recurrent neural networks* se implementa la retroalimentación de parte de las salidas hacia atrás, de manera similar al comportamiento de algunas

neuronas biológicas que conectan su propio axioma con sus dendritas en una especie de recirculación de señales conocida como autapse [3,26].

Así, el aprendizaje se basó en la aproximación de una función que vinculaba las entradas con la salidas. Todo cambió con la publicación uno de los artículos que más impacto han tenido sobre la inteligencia artificial moderna: *Learning Internal Representations by Error Propagation*, que planteaba el concepto de *backpropagation* para optimizar los pesos de las conexiones neuronales, ya no para optimizar una función de salida, sino para hacer lo propio sobre la función de error en las predicciones mediante el uso del error encontrado al *entrenar* la red neuronal [27].

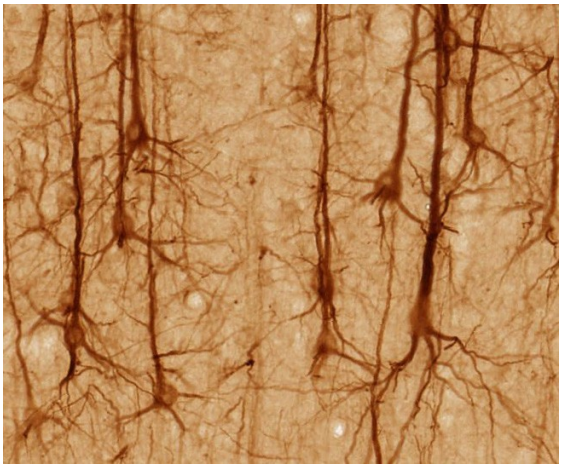


Figure 2: Red de neuronas de un cortex cerebral humano [69].

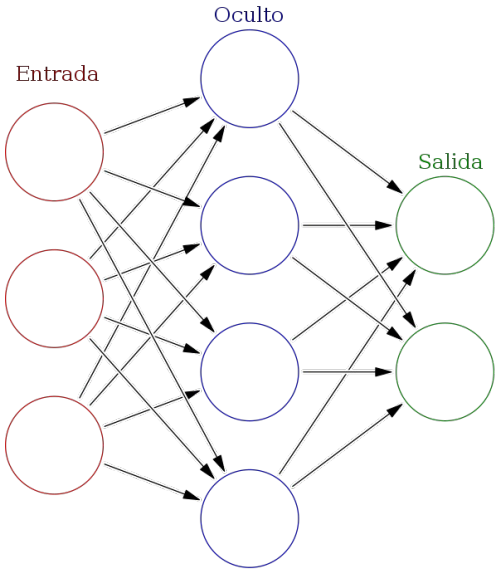


Figure 3: Diagrama esquemático de una red neuronal artificial feedforward [68].

La explotación de estas estructuras por niveles ha llevado a diferentes caminos de aprendizaje, siempre en busca de aproximar el mínimo global de la función de error y tratando de evitar "perderse" dentro de un mínimo local, como veremos más adelante.

La profundidad del encadenamiento de perceptrones ha crecido con el tiempo y llevado al término actual con el que son conocidos este tipo de modelos: *deep learning* o aprendizaje profundo, en clara alusión a su estructura organizada por capas [3]. Estos niveles pueden ser clasificados según su posición como capa de entrada (donde se introducen los datos o contexto que tenemos de una situación abstracta), capa de salida (que devolverán una predicción en base a la entrada mediante una o más salidas de datos) y las capas ocultas o intermedias, encargadas de soportar la optimización del peso de sus entradas para obtener una aproximación a la función de error, además de otras tareas de filtrado que repasaremos más adelante y que dependen del modelo en uso.

Bajo esta colección de métodos es posible realizar cálculos de funciones no lineales para predecir valores, como puede ser la realización de una regresión

o con el fin de clasificar la entrada dentro de una colección de categorías, para lo que se genera un nivel de probabilidad de pertenencia a cada clase [3,28].

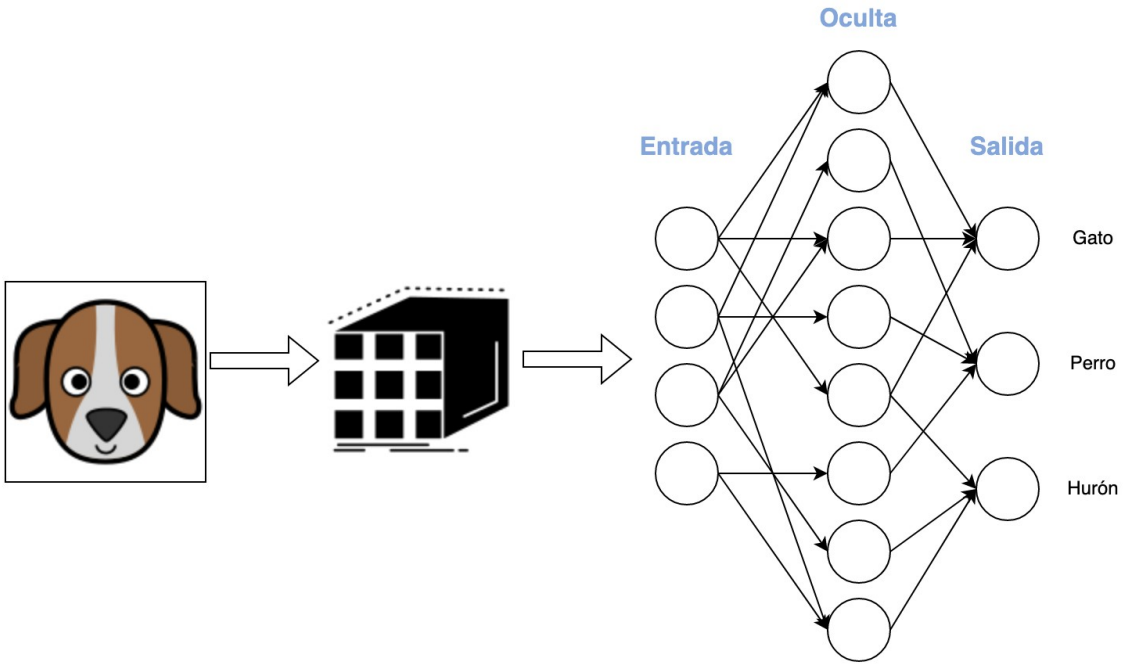


Figure 4: Esquema orientativo de una red neuronal artificial dedicada a la clasificación de imágenes. La entrada de la red es una imagen que ha sido convertida a una matriz que representa la información de la misma. En la salida se presentan tres categorías para clasificar la imagen: perro, gato o hurón.

Como puede intuirse, es preciso convertir la información de entrada en un formato que pueda ser usado en la entrada de la red neuronal. Por ejemplo, en el caso de una imagen, podría convertirse su mapa de píxeles a una matriz donde cada posición represente los colores del mismo. El proceso de preparación de datos, así como el formato de entrada y de salida, dependerán del modelo y de la naturaleza de los datos sobre los que se busca trabajar [3].

2.1.2. Fundamentos del aprendizaje con redes neuronales

El objetivo principal de estas aproximaciones no es otro que el de permitir la predicción de un resultado en base a una entrada de datos. Sin embargo, en lugar de precisar programar cada posible condición del dominio mediante lógica proposicional y programación secuencial, el modelo de aprendizaje automático permite llevar a cabo lo que se conoce como entrenamiento. Así, según el modelo en uso, este entrenamiento supondrá el ajuste de unos parámetros u otros dentro de la red, tales como los valores de activación de entradas y salidas de las neuronas, el peso o relevancia de cada una de las mismas o de los datos de entrada y otros que será expuestos en próximos apartados de este proyecto.

Esta predicción puede tener como objetivo diferentes acciones, desde clasificar

los datos de entrada dentro de una colección de distintas clases, afirmar o negar la pertenencia a un grupo (también conocida como clasificación binaria) o predecir una salida a modo de regresión.

Los datos de entrada pueden tener distintos orígenes, pero su representación suele hacerse, en la mayoría de los casos, mediante una matriz de una o más dimensiones. Como se explicará más adelante, esto presenta algunas limitaciones que pueden ser abordadas mediante la representación de los datos con una estructura de grafo, cuyo estudio es la motivación detrás de este proyecto.

En cuanto a la salida, aunque también puede ser de diferentes tipos, lo más normal es que se devuelvan uno o más valores numéricos. Por ejemplo, en el caso de una regresión, podrán predecirse uno o más valores reales; de igual manera, a la hora de proceder a una clasificación binaria o de entre más clases, se dispondrá, por lo general, de una salida para cada posible clase, donde se indicará la probabilidad de pertenencia a cada una de las mismas.

Con todo, la clave está en el tratamiento que se lleva a cabo sobre la información de entrada para convertirse en la salida. Tal como fue expuesto en el apartado de introducción, las neuronas artificiales tienen como objetivo la aproximación de una función no lineal cuyo resultado dependerá de una colección de valores que el modelo deberá aprender. Gracias a este mecanismo, los datos de entrada son modificados por las unidades neuronales en cada capa, que estarán conectadas de manera secuencial entre sí, hasta llegar a la salida deseada.

Bajo este marco, el proceso de aprendizaje es justo la capacidad del sistema para estimar y optimizar los valores que ha de usar para operar sobre los datos en cada capa de la red neuronal. Estos operandos, conocidos como pesos, son usados para operar sobre los datos de entrada de la neurona y generar una salida. Por lo general, cada capa contará con un vector de pesos característico, aunque, como veremos más adelante, existen tipos de planteamientos que ofrecen variaciones sobre este mecanismo [3].

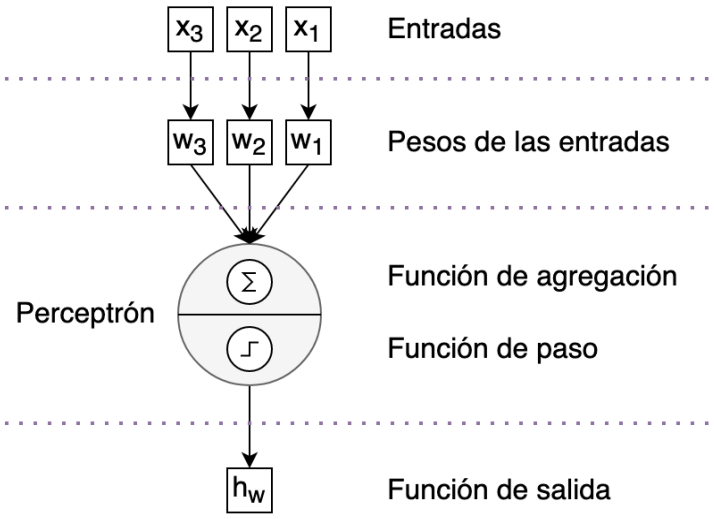


Figure 5: Diagrama de un perceptrón. Se puede ver un vector de entrada de tamaño 3. Sobre cada posición se aplica un valor de peso y el resultado es agregado dentro del perceptrón. Según el resultado de la agregación, la función de paso decide la activación de la neurona y es la función de salida la que genera el resultado final.

Así, cada neurona hará uso de una función de salida que usará la entrada de la misma, el vector de pesos de la capa donde se encuentra y un valor de sesgo que también puede ser aprendido, devolviendo el valor que será pasado a la capa siguiente o que utilizará como salida para la red, según su posición dentro de la misma. La función de salida más básica es la siguiente [28]:

$$h_w(x) = paso(x^T w)$$

- x: matriz de entrada.
- x^T: traspuesta de la matriz de entrada.
- w: matriz de pesos de las entradas.

Las entradas a la neurona, una vez aplicado el peso sobre las mismas, han de ser agregadas para obtener un valor único que pueda ser evaluado por la función de paso que, como se ha explicado con anterioridad, es la encargada de decidir si la neurona activa su salida o no. Por lo general, la función de agregación más común es la suma ponderada [3,28].

Como se puede intuir, la clave del aprendizaje está en la construcción de la matriz de pesos utilizada en cada capa para calcular al función de salida. Como en anteriores ejemplos, esta invención nace como inspiración en un proceso biológico: la llamada regla de Hebb, que afirma que cuando una neurona activa otra con mucha frecuencia, su conexión común se refuerza. Este fenómeno es imitado por las unidades de la red neuronal al reforzar las conexiones de la red y valores del vector de pesos que influyan de manera más positiva en el

resultado [28].

En el contexto de las redes neuronales, este proceso de aprendizaje es conocido como entrenamiento. Para llevarlo a cabo, se realiza una selección de un juego de datos que es conocido como de entrenamiento, donde se dispone de una colección de información de entrada y las salidas esperadas para cada observación. De manera iterativa, se introduce cada ejemplo en la red, se compara el resultado obtenido con el esperado y se calcula la diferencia entre ambos. Con este resultado, se actualiza el peso de cada conexión siguiendo una fórmula conocida como función de actualización que, aunque varía según el algoritmo en uso, suele ser similar a la siguiente [3,28]:

$$w_{i,j}^{(t)} = w_{i,j}^{(t-1)} + \eta \left(y_j - y'_j \right) x_i$$

- $w_{i,j}$: peso de la entrada *i* en la neurona *j*.
- x_i: valor de la posición *i* de la entrada.
- y'_j: salida de la neurona en la iteración de aprendizaje actual.
- y_j: salida esperada en la neurona.
- η: ritmo de aprendizaje.

Una vez disponemos del peso, la neurona puede, como se ha explicado para los perceptrones, activarse o no activarse. Esta condición vendrá dada por la llamada función de activación, que permite o inhibe la salida de la neurona. En sus orígenes, estas redes hacían uso de una función conocida como *step function* que podía generar una salida de valor discreto entre -1 y 1.

Sin embargo, las redes modernas hacen uso de funciones de activación que devuelven números reales, lo que permite aproximar de manera más fiel la función de predicción mediante la aproximación de la función de error. La función de activación podrá estar basada en diferentes tipos, aunque hoy en día la más usada es la conocida como unidad lineal rectificada o ReLu, *rectifier linear unit* [3].

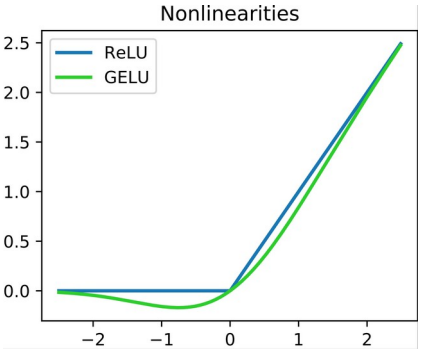


Figure 6: Presentación gráfica de las funciones de activación ReLU y GELU.

$$f(x) = max(0, x)$$

La función de activación ReLU o GELU devolverán 0 cuando el valor de la agregación de la neurona, *x*, sea negativo, lo que dejará a la misma inactiva; por contra, si el valor es positivo, se devolverá el mismo.

En este punto es donde entra en juego el citado avance clave conocido como *backpropagation*, donde el aprendizaje se produce, como se ha indicado, mediante la optimización de la función de error en busca de su mínimo global. Así, durante el proceso de entrenamiento, se recogen los errores de cada predicción y se avanza hacia atrás, ascendiendo desde la salida hasta la entrada de la red, haciendo uso de la función de actualización para recalcular el vector de pesos de cada capa.

De este modo, con cada iteración de entrenamiento se persigue optimizar la función de error para localizar su mínimo global. Uno de los algoritmos de búsqueda de mínimos más usado es el *gradient descent*: se busca el mínimo mediante el seguimiento de la pendiente descendiente de la función de error. Aunque existen diferentes versionas, todas tienen este objetivo de búsqueda [3].

Uno de los parámetros de configuración de modelo más importante es el ritmo de aprendizaje o *learning rate*, encargado de indicar el tamaño del salto dentro de la pendiente descendente de la función para continuar con la búsqueda de un mínimo: a menor sea el salto, más probabilidades de encontrar el mínimo, pero mayor tiempo de aprendizaje; a más grande este parámetro, más probabilidades hay de perder el mínimo global o de perderse en un mínimo local, aunque agiliza el tiempo de entrenamiento [28].

Será en base a esta ruta de búsqueda con la que se calcularán los valores de actualización del vector de pesos y, mediante la repetición del proceso de manera sucesiva con las observaciones del conjunto de entrenamiento, será posible aproximar la predicción en las salidas. Dentro del proceso de aprendizaje se suceden diferentes épocas o *epochs*, donde cada una supone una combinación de los datos de entrenamiento en la entrada y salidas de la red. Como veremos más adelante, este mecanismo permitirá medir la precisión de un modelo para resolver un problema concreto, así como conocer su capacidad para generalizar su aplicación fuera de los datos de entrenamiento [3].

Este proceso ordenado de asimilación puede ser configurado mediante distintos valores que gobernarán su funcionamiento, tales como el ritmo de aprendizaje, el número de capas dentro de la red, la cantidad de neuronas en cada capa, el tamaño de los vectores de pesos y otros. A estos parámetros de configuración los conocemos como hiperparámetros y podrán también ser aprendidos para mejorar la precisión del modelo [28].

Aunque estos algoritmos nos permiten enfrentarnos a distintos tipos de problemas como los citados para clasificar objetos dentro de clases o predecir mediante regresiones, es evidente que la predicción se basa en cálculos sobre

espacios vectoriales euclídeos sobre los que se representan los datos de entrada y se operan sobre ellos mediante cálculo vectorial. Además, se asume que los objetos son descritos de manera completa mediante sus características propias y que son independientes entre sí [29].

2.2. Introducción a las redes neuronales gráficas

Como ha podido verse, los modelos que han sido introducidos hasta el momento se ven limitados al trabajo sobre espacios euclídeos. Son una buena solución para tareas de distintos tipos, como puede ser la clasificación de imágenes o textos, el reconocimiento de voz, procesado de vídeo y otros tipos de datos complejos que cuentan con estructuras espaciales, así como aquellos que pueden ser convertidos a los mismos.

Sin embargo, este tipo de planteamiento asume que no existe relación entre los objetos que conforman los datos de entrada, ya sea un orden, jerarquía, dependencia o relaciones de otros tipos [4]. Esta pérdida de información es relevante en situaciones donde tales vínculos puedan ser importantes, tales como las redes sociales (donde existen relaciones entre sus usuarios y en los que los vínculos pueden ser de distintos tipos: seguidor, amigo, familiar, fan, etc), redes de sensores distribuidos en una red (distancias entre ellos, relevancia de cada estación, agrupaciones en conglomerados, entre otros), la descripción sobre moléculas y sus enlaces atómicos (forma de la molécula, átomos y tipos de interacciones), compatibilidad entre medicamentos u otros compuestos químicos (interacciones complejas entre ellos, tipos de reacción y otros) y una infinidad de ejemplos adicionales [2,4,11].

Estas estructuras relacionales pueden ser representadas mediante un grafo, donde cada vértice será un objeto y las aristas las relaciones que existen entre cada par, los unos y las otras cada uno con sus propios atributos, clases y tipologías. Estos grafos pueden tomar diferentes dimensiones, con colecciones indefinidas de nodos y enlaces, lo que hace que sea complicado utilizarlos como datos de entrada para un modelo de trabajo tradicional como el explicado en el apartado 2.1, lo que los hace inadecuados para trabajar con datos estructurados, salvo que se asuma la potencial pérdida de información valiosa para realizar predicciones [2,3].

Con todo, en 2005 fue propuesto el concepto de red neuronal gráfica (*graph neural network* o GNN), con el objetivo de poder tratar datos representados mediante grafos sin perder información y contexto topológico y, por supuesto, sin precisar convertir sus contenidos y relaciones en vectores para su uso en espacios euclídeos [29]. Estos primeros modelos hacían uso de redes

neuronales recursivas y recibían el nombre de RecGNNs (*recursive graph neural networks*), donde la salida del modelo era retroalimentado a las entradas y, tras sucesivas iteraciones y ajustes de la función de error, se acercaba a converger con la función de salida esperada y se aprendía la construcción del grafo.

Con la llegada y el éxito de los modelos de *deep learning* basados en redes convolucionales (*convolutional neural networks*, CNNs), se propuso el uso de capas neuronales para transformar los grafos de entrada en convoluciones similares a las usadas para clasificar imágenes pero con diferentes grados de éxito, que son conocidas como ConvGNN (*convolutional graph neural networks*). Así, se toman dos aproximaciones principales: por un lado, los basados en teoría espectral de grafos, donde el aprendizaje se basa en la representación de la matriz de adyacencia y los valores propios de la estructura, conocidas como redes neuronales gráficas convolucionales espectrales (*spectral graph convolutional neural networks*, spectral ConvGNN) [30]; por otro lado, se proponen otros que aprenden de la distribución espacial del grafo al completo, lo que permite tener en cuenta información ordinal o relaciones de proximidad de diferente tipos [31,32] y son conocidas como redes neuronales convolucionales gráficas espaciales (*spatial graph convolutional neural networks*, spatial ConvGNN). Se presentarán ambos acercamientos más adelante.

Además de estos sistemas basados en *deep learning* para trabajar con grafos, es posible encontrar otras propuestas que hacen uso de algoritmos tradicionales de aprendizaje computacional, tales como los basados en *network embeddings*, donde se utilizan vectores que conservan la información topológica y la información de etiquetado de los nodos, esto es, la matriz de adyacencia en conjunción con los vectores que caracterizan cada nodo [11,33,34]. Son conocidas como *graph autoencoders* o GAEs. Como en el contexto de otros *encoders*, la codificación de los nodos y sus aristas se hace paso a paso, como una cadena de montaje de descripciones de la estructura, lo que permite, dado un vértice o arista de entrada al modelo, decidir la relación o nodo, respectivamente, que lo sigue. Esto permite la predicción de relaciones u objetos, además de la generación de estructuras sintéticas dada una semilla inicial [35,36].

La estructuración de la información mediante grafos permite representar numerosas situaciones y objetos del mundo natural y del conocimiento humano, pues trabajan con características propias de los mismos y sus relaciones. Sin embargo, hay un punto clave que ha quedado olvidado: en el mundo real, los objetos y vínculos que los enlazan suelen ser dinámicos y cambiar con el tiempo. Para atender este tipo de escenarios, es posible hacer uso de modelos conocidos como espacio-temporales (*spatio-temporal graph neuronal networks*, STGNNs), que permiten no solo aprender sobre una

estructura, sino también en base a cómo sus contenidos y características cambian con el paso del tiempo, tanto a nivel individual, como colectivo. Estos modelos resultan de especial importancia para la predicción de tráfico en redes de carreteras y de comunicaciones [37].

De entre todos estos modelos, este proyecto didáctico se centrará en el estudio de la solución más versátil y estudiada en la actualidad: las redes neuronales gráficas convolucionales o *convolutional graph neural networks*.

2.2.1. Tipos de problemas

Gracias a aprender no solo en base a los contenidos de las observaciones, sino también sobre sus relaciones e incluso topología de las mismas, es posible poner la información dentro de su contexto y ubicarlo dentro de un entorno, de modo que un nodo pueda ser clasificado dentro de una clase diferente según el vecindario con el que se vincula, incluso aunque sus atributos sean distintos a los de las entidades que lo rodean [12,29,38].

Gracias a este planteamiento, es posible crear sistemas que permitan resolver problemas que no podían ser aproximados de manera eficaz con las soluciones tradicionales, ya sea por llevar a una baja eficacia o por la dificultad de hacerlas escalables en un entorno real [11,12,39]. De este modo, es posible trabajar en la predicción o clasificación a tres niveles:

- Nodo, vértice o entidad.
- Arista, relación o pertenencia.
- Grupos de los dos anteriores, es decir, a nivel de grafo o subgrafo.

Clasificación de nodos

El primer uso que es posible destacar es la clasificación de vértices, donde una muestra puede ser asignada a una clase distinta según su contenido y/o su contexto dentro de la estructura [36]. Esto puede ser usado para, por ejemplo, clasificar los usuarios de una red social por sus intereses según las relaciones que tiene (amigos, seguidos y seguidores) y sus características propias (lo que publica).

Clasificación y predicción de relaciones

Un uso muy interesante es la predicción de probabilidad de existir una relación entre dos nodos en base a los vecinos de cada uno de los mismos. Mediante el aprendizaje basado en el contexto de las entidades, es posible describir una nueva observación y la probabilidad de que tenga un vecindario dado o relación con alguno de sus componentes [40,41]. Una posible aplicación sería para la generación de recomendaciones dentro de una tienda online atendiendo a las acciones de los usuarios. También podría ser útil para predecir siguientes pasos en una secuencia de acciones o sugerir nuevas amistades dentro de una

red social.

De igual modo, también es importante resaltar la posibilidad de clasificar las relaciones dentro de una estructura, de manera que pueda describirse cómo se relacionan dos nodos dados, en base a qué características y qué importancia tiene [41]. Esta capacidad resulta de especial importancia para enfrentar problemas tan claves como la resistencia de materiales, la compatibilidad entre moléculas o el diseño de nuevos compuestos químicos [42]. Como veremos, el uso de relaciones con características y tipologías propias tendrá un impacto sustancial en la selección y eficacia del modelo.

Clasificación de grafos

En este caso, el aprendizaje suele ser realizado mediante la identificación de subgrafos que forman parte de otros mayores a modo de patrones, lo que puede ser muy útil para la clasificación de imágenes: pueden ser convertidas en estructuras relacionales donde los nodos del grafo representan los píxeles y sus relaciones con los otros de su entorno [30,32]. De este modo, es posible identificar una imagen completa o una parte de la misma de manera eficaz [43,44].

2.2.2. Representación de grafos y aprendizaje

Aunque se han expuesto diferente tipos de problemas y una definición abstracta del grafo como estructura relacional, no se ha presentado la manera en la que son descritos a nivel explícito. Para ello, es preciso entender cómo es un grafo y la manera de describirlo

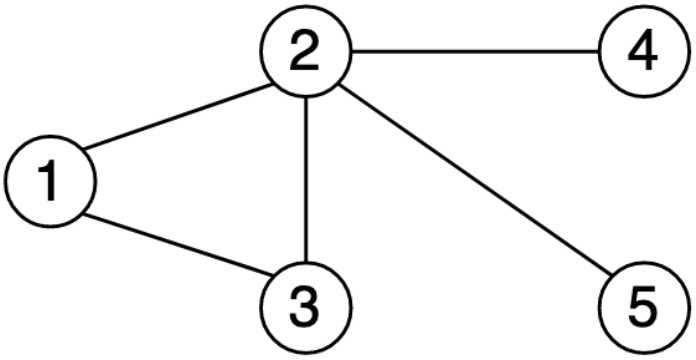


Figure 7: Grafo etiquetado G formado por 5 vértices y 5 aristas.

Por un lado, una de las maneras más sencillas de describir un grafo es mediante las conexiones que existen entre sus vértices. Esto puede hacerse mediante la matriz de adyacencia, que tendrá tantas filas y columnas como vértices haya. Por cada arista que una dos vértices v_i y v_j, se sumará uno a la posición A_{ij}. A continuación, se puede ver la matriz de adyacencia del grafo de

ejemplo:

$$A_G = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Por otro lado, será posible hacer uso de la matriz laplaciana que, entre otras cosas, puede ser usada para encontrar el número de árboles de expansión que posee el grafo: la lista de aristas de la estructura que forman un árbol que cubre todos sus vértices sin generar ciclos. Esta característica puede permitir conocer la pertenencia de un vértice a un árbol concreto, lo que ayuda a la descripción del grafo con suficiente detalle para dirigir el aprendizaje, además de ser una cualidad de capital relevancia para la clasificación de grafos [44]. La matriz laplaciana del grafo anterior quedaría como sigue:

$$L_G = \begin{pmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 4 & -1 & -1 & -1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 1 \end{pmatrix}$$

Mediante estas dos estructuras es posible describir las aristas que relacionan los vértices del grafo. Sin embargo, nos falta justo el valor que constituye cada uno de estos nodos: qué representan y a qué clase pertenecen, así como qué características poseen. Estos atributos aportados sobre cada objeto, que podrían ser usados en un algoritmo de aprendizaje que no fuera sensible al contexto, serán usados junto con las relaciones entre todos ellos para disponer de datos suficientes de cara a entrenar y tomar decisiones sobre la red gráfica [30].

Este tipo de descripción algebraica hace uso del polinomio característico, así como los valores y vectores propios de la estructura, y conforma la base de la llamada teoría espectral de grafos. Con ella, es posible identificar distintas relaciones entre grafos, tales como isomorfismos o coespectralidades. Este planteamiento es usado por las redes neuronales convolucionales gráficas de tipo espectral (*spectral convolutional graph neural networks*) que veremos más adelante. Puesto que las operaciones se realizan dentro de estos datos mediante cálculo euclídeo, es viable aplicar sobre ellos técnicas de reducción de dimensionalidad, tales como PCA, con el fin de adaptarlo al tamaño de la red en uso. En consecuencia, puede llevar a la pérdida de información de la varianza y a problemas a la hora de generalizar el uso del modelo, como veremos más adelante [4].

Así, la representación espectral, aunque solo tenga en cuenta las relaciones pero no sus tipologías, permite la identificación de patrones dentro de modelos

de redes convolucionales, como se explicó en anteriores apartados, lo que supone una alternativa frente a los algoritmos de aprendizaje computacional tradicionales no basados en redes neuronales, aportando la ventaja de hacer uso del contexto de cada objeto para el proceso de aprendizaje e inferencia [30].

Como puede verse, existen distintas maneras de representar un grafo y de hacer uso de la misma para el entrenamiento de un modelo. En los siguientes apartados veremos distintos tipos de modelos neuronales que hacen uso de las mismas.

2.3. Aprendizaje y predicción mediante redes neuronales gráficas convolucionales (ConvGNN)

2.3.1. Introducción a las redes neuronales convolucionales (CNN)

Si los orígenes del campo del *deep learning* provienen de la inspiración en el mundo biológico, sus avances clave también le deben parte del éxito que han cosechado en los últimos años.

La investigación del funcionamiento del cortex visual durante el siglo XX llevó a comprobar que ciertas regiones de neuronas se distribuían el trabajo de procesado en distintos niveles de especialización: por un lado, el espacio captado por los nervios oculares se dividía en una rejilla, donde distintos grupos de células se encargaban de cada una de las celdas; por otro lado, determinadas neuronas se activaban solo bajo ciertos estímulos o, incluso, eran más sensibles a diferentes orientaciones de líneas y perfiles (por ejemplo, verticales u horizontales), e incluso ante diversos tonos e intensidades de color [28,45,46].

Esta idea, conocida como regla de Hubel, fue tomada como base para el análisis y clasificación de imágenes: en lugar de trabajar con los píxeles de una imagen como un todo, se optó por especializar las neuronas de la red artificial para detectar distintos tipos de patrones. De este modo nació el neoconitrón a principios de la década de 1980, hecho que sentó las bases para una nueva familia de modelos de *deep learning* [47,48].

Como resultado de estos razonamientos, se definieron las redes neuronales convolucionales, que permiten la especialización de distintas capas de la red que reciben el nombre, valga la redundancia, de capas convolucionales o *convolutional layers*. Las unidades que las componen, en lugar de estar conectadas a sus niveles previos de manera completa como en otros modelos de *deep learning*, lo hacen de manera parcial y selectiva, de modo que

atiendan solo a ciertas regiones de los datos previos, en una clara imitación del modelo biológico de conexiones neuronales descrito por la regla de Hubel.

A mayor profundidad de capa, más especializada estará y tendrá un tamaño más reducido que las anteriores al disminuir la cantidad de información procesada. Este tipo de reducción del tamaño permite rebajar, además, la complejidad computacional del modelo completo [28].

Además de hacer uso de la especialización por sectores de la capa superior o del dato original, este tipo de redes también hacen uso de la idea de activación selectiva según el estímulo recibido. Así, es posible entrenar las unidades de cada nivel para aprender diferentes patrones procedentes de las capas superiores. Para ello, en lugar de ser entrenadas mediante el uso de un peso escalar dentro de un vector unidimensional, se hará mediante uno bidimensional que hará las veces de matriz de pesos. Esta, al ser operada sobre la entrada de la unidad, dará como resultado un mapa de características o *feature map*, que podrán ser configurados para detectar patrones predefinidos como líneas en distintas direcciones (horizontales, verticales, oblicuas en diferentes ángulos, distintos tipos de contornos, puntos, etc). Para estos casos particulares, denominaremos a las capas con el nombre de filtros [3,28].

Si utilizamos el ejemplo de una red diseñada para clasificar imágenes, se podría describir su funcionamiento mediante el siguiente orden:

1. La capa de entrada será la encargada de recibir la imagen original. Esta capa deberá tener un tamaño predefinido y la imagen habrá que tener, como máximo, las dimensiones de la misma. En caso contrario, la imagen deberá ser comprimida para reducir su tamaño, con la consecuente posibilidad de pérdida de información.
2. A continuación, se presentará una capa convolucional cuyas unidades se centrarán en regiones específicas de la imagen y que han sido configuradas mediante parámetros para tener una serie de filtros disponibles que cuentan con un tamaño concreto.
3. Por debajo de la misma, habrá otra capa convolucional con un juego de parámetros determinados: tamaño, conectividad con la capa previa, filtros por unidad, etc).
4. Se podrán superponer un número indefinido de capas según los parámetros de configuración del modelo. Por supuesto, a más capas, mayor coste computacional en espacio (habrá que almacenar más filtros, mapas de características y matriz de pesos) y tiempo (se deberán realizar más ejecuciones de funciones de activación y actualización).

Con todo, y en base a lo visto en definiciones anteriores, se podrá definir la siguiente fórmula para calcular la salida de una neurona en base a sus filtros

activos, la matriz de pesos y la entrada desde la capa anterior [3,28]:

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \times w_{u,v,k',k}$$

$$\text{donde} \quad \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

- $z_{i,j,k}$: salida de la neurona de la capa *i*, columna *j* de la convolución sobre el mapa *k.*
- *s* y *f*: describen la posición y peso del mapa en uso. En este contexto, $f_{n'}$ es el número de mapas de la capa anterior.
- *x*: entrada a una neurona concreta dada por sus coordenadas *i*, *j* y *k*, de nuevo y respectivamente, según su nivel, columna y mapa.
- *w*: representa la matriz de pesos para el mapa *k*, en el nivel *l* y su entrada en la coliumna *v* nivel *u*.

Como se ha podido apreciar, existen una serie de valores que pueden ser personalizado que hemos citado mediante el nombre *parámetros*. Estos permiten definir distintas características de modelo, tales como el número de capas y unidades para cada una, cantidad de filtros por neurona, tamaño de datos de entrada y otros. Este elevado nivel de parametrización permite aproximar funciones no lineales muy complejas con grados de libertad muy elevados [3].

Como se ha ido mencionando a lo largo de las páginas anteriores, el principal objetivo de este proyecto es el estudio de las redes neuronales que hacen uso de estructuras de datos basadas en grafos. En el caso de las redes convolucionales, esta adaptación puede ser llevada a cabo mediante dos aproximaciones dispares:

1. Por un lado, aquellas que hacen uso de la teoría espectral de grafos y que tienen en cuenta los vértices y sus relaciones, pero que no tiene en cuenta la tipología de las mismas o sus propiedades, así como relaciones de posible nivel o fenómenos transitivos [30]. Este tipo de modelos son conocidos como redes neuronales gráficas convolucionales espectrales o *spectral convolutional neural networks* (Spectral ConvGNN).
2. Por otro lado, las que tienen presente el grafo a nivel espacial, las propiedades de sus relaciones y las de los nodos [40]. En este caso hablaremos de redes neuronales gráficas convolucionales espaciales (*Spatial ConvGNN*).

A continuación, se presenta cada una de estas tipologías en mayor detalle y se estudian sus capacidades y limitaciones, así como las estrategias que se siguen para que puedan adaptarse a diferentes esquemas de conectividad o abordar las limitaciones de tamaño existentes en los modelos tradicionales [41].

2.3.2. Aproximación a ConvGNN basada en teoría espectral de grafos

Las redes convolucionales gráficas espectrales, o *spectral graph convolutional neural networks*, hacen uso de la descripción del grafo en base a la teoría espectral descrita con anterioridad. Así, la entrada de la red será una combinación de su matriz de adyacencia (es decir, las relaciones existentes entre sus nodos) y la matriz laplaciana, de donde se consigue extraer una visión espacial básica de la estructura, al disponer de señales sobre los árboles de expansión que pueden ser formados.

Este tipo de aproximación presenta algunas deficiencias obvias. Para empezar, hasta ahora se ha hecho referencia al uso de atributos en los vértices, pero no en sus aristas: se considera si los nodos están relacionados o no, así como los patrones vecinales que se forman, pero no la manera en la que lo hacen o bajo qué circunstancias [49].

De igual modo, la descripción espectral no permite plantear si un grafo es o no dirigido, de modo que es inviable considerar la importancia de la existencia de relaciones de jerarquía (por ejemplo, indicar que el nodo n_i es padre de n_j), ordenación (n_i es mayor o más relevante que n_j) o pertenencia entre los vértices (n_i pertenece a n_j).

En conclusión, el uso de la descripción espectral del grafo lleva a depender sobre la forma del mismo en base a las relaciones entre sus nodos, lo que hace que los modelos de este tipo puedan alcanzar cierto grado de eficacia para trabajar con grafos isomorfos entre sí, pero pueden ser sensibles a cambios en la estructura a nivel global y, en consecuencia, perder precisión al tratar de generalizar su uso [38,49].

2.3.3. Aproximación a ConvGNN basada en el paso de mensajes

Como puede comprobarse, el uso de redes espectrales presenta limitaciones importantes, en especial cuando la generalización del modelo va destinada a trabajar con estructuras muy complejas donde, además de aparecer distintos tipos de nodos, también pueden encontrarse diferentes clases de relaciones. De igual manera, su sensibilidad a cambios estructurales las hace poco idóneas para tareas dentro de entornos muy cambiantes [49].

A modo de ejemplo, es posible imaginar una estructura que representa al

personal y alumnado de una universidad. Así, se tendrán:

- Nodos que representan profesores, alumnas y asignaturas. Por lo tanto, cada nodo puede ser de una categoría diferente y contar con atributos distintos.
- Las relaciones podrán ser de distintas clases y ser usadas para relacionar variopintos vértices:
    - Las asignaturas podrán tener uno o más profesores. Además, cada uno de estos podrá estar vinculado a distintas asignaturas.
    - Los alumnos podrán estar matriculados en diferentes materias. De igual manera, estas personas podrán relacionarse mediante enlaces de amistad.

    Como se puede comprobar, la estructura puede tener tres tipos de relaciones: “estudiante estudia asignatura”, “profesor enseña asignatura” y “alumna es amiga de alumna”. Además, cada alumna posee un atributo: el grado que está estudiando.

Si el objetivo es construir un modelo destinado a clasificar a cada persona del alumnado dentro de un grafo según sus relaciones, nos sería imposible bajo el modelo espectral pues, de entrada, no sería viable diferenciar cada tipo de relación.

Como solución a este problema, en lugar de basar el entrenamiento solo en las estructuras que describen el grafo en base a sus nodos y las relaciones entre ellos (conocida como *node-wise*), se hace uso de las propiedades derivadas de sus aristas (*edge-wise*) mediante la difusión de valores entre los vértices que unen [31,50]. Mediante el uso de redes neuronales convolucionales que basan su funcionamiento en el paso de mensajes, es posible aprender de información no explícita que no varía cuando los grafos analizados no son isomorfos [51].

MPNN: Message-passing neural networks

El tipo de modelo de red neuronal más utilizado que hace uso de esta lógica son las conocidas como MPNN o *message-passing neural networks*, que permiten, de manera iterativa, transformar los atributos que posee cada nodo en base a los que tienen sus vértices vecinos, que podrán haber sido también modificados en iteraciones previas para adaptarse a su entorno [49,52].

Sea el nodo n_i, que tiene un vector de atributos h_i. Si tomamos el ejemplo anterior, este vértice podría representar a un estudiante y el atributo que indica el grado que está cursando. Así, este nodo contará con una serie de conexiones a otras entidades de manera directa, tales como asignaturas u otros estudiantes, que es conocido como vecindario del nodo n_i.

Puesto que hablamos de un modelo basado en el paso de mensajes, podemos definir un mensaje como la función de los atributos de cada uno de los dos

nodos que se unen por medio de una arista dada y los atributos que la componen. La arista que une el nodo n_i con el n_j, *ij*, será caracterizada por los atributos e_{ij}, Así, la función de mensaje o *messege function* para cada arista será la siguiente [49]:

$$\vec{m_{ij}} = f_e(\vec{h_i}, \vec{h_j}, \vec{e_{ij}})$$

- f_e: función de mensaje de la arista.
- h_i: vector atributos del vértice *i*.
- h_j: vector atributos del vértice *j*.
- e_{ij}: vector atributos de la arista e_{ij}.

De cara a actualizar los atributos de cada nodo, se computará el mensaje de cada vértice que conecta con el mismo, de manera que se vea influido por las características de su entorno. Así, se hará uso de un operador de agregación, como puede ser la suma, para incorporar los valores del entorno y obtener un resultado función de estos y de los atributos del nodo actual [53]. La función siguiente calculará el nuevo valor de los mismos:

$$\vec{h'}_i = f_v(\vec{h_i}, \sum_{j \in N_i}^{n} \vec{m_{ji}})$$

- h_i: atributos del vértice *i*.
- N_i: nodos vecinos del nodo *i*.
- *n*: cardinal de N_i.
- m_{ij}: mensaje de la arista *ji*.

Mediante sucesivas iteraciones de transformación de los atributos de cada nodo según la función de paso de mensaje, cada nodo podrá verse modificado según la información procedente de nodos más y más lejos: en una primera iteración, sus características se actualizan mediante una función que tiene en cuenta su vecindario inmediato; en una segunda, aquellos nodos vecinos contendrán información de los propios, por lo que se tendrá en cuenta las características de nodos y aristas a dos saltos de distancia; en una tercera iteración, se tendrán en cuenta a aquellos a tres saltos, y así sucesivamente.

Mediante el uso de la información pasada, estos modelos resultan más eficaces a la hora de enfrentarse a información que no ha sido observada y, por tanto, facilitan su generalización [38,54]. Sin embargo, pese a que ya podemos tener en cuenta los atributos de las relaciones y nodos por igual, se continúa sin disponer de información que transmita la importancia de una relación, su tipología o su orden. A continuación, se presentarán dos aproximaciones para atender esta limitación: las redes neuronales gráficas de atención y las relacionales.

GAT: graph attention convolutional neural networks

Uno de los principales problemas del planteamiento presentado es que requiere calcular los atributos de cada nodo en base a una operación por cada una de las relaciones con las que conecta. Además, a más niveles de relaciones se quiera tener en cuenta para realizar la inferencia de la salida del

modelo, más iteraciones serán necesarias y más cálculos será oportuno llevar a cabo. Con todo, el coste computacional de entrenar el modelo crece de manera pronunciada con el número de vértices y sus grados, por lo que su uso solo es recomendable sobre estructuras pequeñas, donde se tienen en cuenta solo las aristas más importantes y se descartan el resto antes de iniciar el entrenamiento [52,53].

Este problema es la motivación principal del uso de una mejora sobre el paso de mensajes: las redes neuronales de atención o *graph attention neural networks* (GAEs) aplican un coeficiente de importancia para cada tipo de relación o a cada relación individual.

De esta manera, es posible modificar la función de actualización propuesta para las MPNN para incorporar este coeficiente de importancia, lo que permite establecer un umbral durante el proceso de entrenamiento que lleve a no tener en cuenta la información de las relaciones que no superen dicho nivel [52]. Este umbral puede ser aprendido y se calcula para cada arista *ij*, de manera similar a como se hizo para el paso de mensajes:

$$\alpha_{ij} = a(\vec{h}_i, \vec{h}_j, \vec{e}_{ij})$$

Y es usado para calcular la función de actualización de paso de mensaje, donde se aprende a atender o no a una determinada relación:

$$\vec{h'}_i = \sigma\left(\sum_{j \in N_i}^{n} \alpha_{ij} W \vec{h}_j\right)$$

- W: matriz de pesos aprendida.
- h_j: vector de valores actuales.
- α: resultado de atención para cada arista *ij*.
- N_i: vecindario del nodo *i*.

Aunque las pruebas llevadas a cabo por los autores de este planteamiento demuestran que se produce una reducción significativa del coste de entrenamiento, no ha sido posible, hasta ahora, alcanzar una mejora sustancial en el nivel de precisión obtenido en la generalización del modelo frente a aquel fruto de aplicar redes de tipo MPNN [52].

RGCNs: relational graph convolutional networks

Con anterioridad, se presentaron las redes neuronales convolucionales y se expuso que cada capa cuenta con un vector de pesos diferente que se aplica sobre sus unidades. Así, cuando se hace uso de un modelo basado en paso de mensajes, el nuevo valor siempre será función de un mismo vector de pesos y, salvo que se haga uso de coeficientes de atención como los propuestos para las GAEs, todas las relaciones tendrán el mismo nivel de importancia.

Como alternativa, las RGCNs proponen la utilización de un vector de pesos para cada capa y para cada tipo de relación se esté computando mediante la función de actualización y, en este caso, sin atender a las propiedades de la

arista. Esta operativa permite simplificar el cálculo y sustituye el uso de operaciones vectoriales sobre las aristas por un resultado escalar, de modo que el coste computacional se reduce de manera drástica.

$$\overrightarrow{h_i^{(l+1)}} = \sigma\left(W_o^l h_i^l + \sum_{r \in R} \sum_{j \in N_i^r} \frac{1}{c_{i,r}}\right) W_r^l \vec{h}_j^l$$

- $h_i^{(l+1)}$: actualización en la siguiente capa de la red.
- W_i^l: pesos en la capa actual para una relación concreta. Cuando $i=0$, hace referencia a una relación reflexiva; cuando se usa r, se hace referencia a los pesos para un tipo de relación concreta.
- h_i^l: *salida de la capa l* en el nodo i.
- $c_{i,r}$: constante de normalización que depende del problema y que puede ser aprendida o seleccionada de manera manual.
- N_i^r: conjunto de vecinos del nodo i según el tipo de relación r dentro del conjunto de relaciones de interés R.

Este peso de relación por cada capa puede ser introducido de manera previa según el problema que se está abordando o en base al dominio en uso, pero también puede ser aprendido mediante la adaptación de la función de actualización [2,55].

2.3.4. Resumen de modelos

A continuación, se presenta un cuadro con un resumen de los modelos presentados en este apartado 2.3 y otros modelos más concretos que han sido revisados, así como sus referencias bibliográficas:

Arquitectura	**Aproximación**	**Modelo**	**Referencias**
RecGNN: redes recursivas			[29]
ConvGNN: redes convolucionales	Espectrales	GCN: Graph Convolutional Network	[38,49]
	Espaciales		[31,50,51], generales sobre aproximación espacial
		MPNN: Message-passing neural networks	[38,49,52–54]

		GAT: graph attention convolutional neural networks	[52,53]
		RGCNs: relational graph convolutional networks	[2,55]
STGNN: redes espacio-temporales			[37]
GAE: autoencoders de grafos		Network embeddings y generación de grafos	[11,33,34]

2.4. Implementación de pruebas

Con el fin de ilustrar de una manera más detallada el uso de redes neuronales gráficas, en este apartado se introducirá a la implementación de dos tipos de aplicaciones de las mismas:

1. Pruebas de clasificación de nodos con redes convolucionales, que serán realizadas mediante dos aproximaciones: espectral y espacial.
2. Un experimento de predicción de enlazado, en este caso gracias al uso de métodos de paso de mensajes.

Además, se utilizará también un método de clasificación tradicional basado en el uso de las características de los nodos del grafo, con el objetivo de contrastar las métricas de precisión alcanzadas y justificar, así, la motivación del desarrollo de redes gráficas.

Con el fin de permitir un mayor entendimiento de estas pruebas, se acompaña esta memoria con un *notebook* de Jupyter construido con Python que puede ser ejecutado para probar el código llevado a cabo.

Puesto que el objetivo principal del uso de estas técnicas es la de clasificar o predecir sucesos o relaciones, todos los ejemplos irán acompañados de un análisis de precisión.

2.4.1. Preparación del entorno

Como primer paso, se deberá preparar el entorno para las pruebas. Como se verá en el código adjunto, se procede a importar las librerías clave que serán usadas:

1. numpy [56], un paquete de Python que permite el uso de estructuras de datos para la realización de cálculos computacionales sobre las mismas, con un enfoque muy centrado en el cálculo vectorial sobre distintas dimensiones: vectores, matrices y tensores. Será usada para los cálculos básicos sobre los datos de entrada y el cómputo de la salida de las unidades de cada capa de las redes neuronales.
2. TensorFlow [57,58] es una librería, también implementable mediante Python, que ofrece un entorno de trabajo para aprendizaje computacional, creación de caminos de preparación de datos y el acceso a librerías como Keras, que ofrecen utilidades dentro del campo del *deep learning* para su uso ágil y práctico.
3. Spektral es una librería diseñada para facilitar la explotación de redes neuronales gráficas, para lo que presenta distintos modelos y herramientas para trabajar con datos estructurados mediante grafos [59].
4. PyTorch está compuesto por una colección de módulos diseñados para el aprendizaje computacional con distintas opciones para la optimización de procesos sobre distintas infraestructuras de computación. En este sentido, es bastante similar al ya mencionado TensorFlow [60].
5. De igual modo, se instalan las librerías Scikit-learn [61] y Pandas [62], que serán usadas para algunas tareas de preparación y comprobación de datos. La primera es usada para tareas de aprendizaje computacional y la segunda ofrece diferentes herramientas para la preparación, manipulación y análisis de datos.

Con el fin de poder obtener una visualización clara de la progresión del entrenamiento de los modelos en uso, así como facilitar el análisis de su precisión, se prepara un entorno de trabajo mediante TensorBoard, una herramienta muy útil ofrecida dentro de TensorFlow que permite ser conectada a los modelos para obtener información en tiempo real sobre las acciones que llevan a cabo [63]. Como veremos más adelante y como se presenta en el cuaderno adjunto, se hará referencia a esta herramienta como *callback* desde los procesos de entrenamiento.

Antes de pasar a cargar y preparar los datos, además, se fijan las semillas de aleatoriedad de *numpy* y *tensorflow*, con el objetivo de poder replicar los resultados desde el cuaderno adjunto:

```
np.random.seed(42)
tf.random.set_seed(42)
```

2.4.2. Conjunto de datos usado en las pruebas

Dentro de este proyecto se hace uso de un juego de datos concreto, CORA, el

cuál está compuesto de una colección de entidades que representan publicaciones científicas y las diferentes citas, lo que conforma un grafo formado por nodos, que hacen alusión a cada artículo, y enlaces, que representan las referencias entre cada uno de ellos [13]. Así, la estructura resultante es un grafo dirigido sin enlaces reflexivos y que puede presentar ciclos, donde cada vértice contiene características para indicar el uso de diferentes palabras en cada publicación. Además, cada nodo vendrá etiquetado por su pertenencia a una categoría de entre las siete siguientes:

- Case_Based
- Genetic_Algorithms
- Neural_Networks
- Probabilistic_Methods
- Reinforcement_Learning
- Rule_Learning
- Theory

Gracias a Spektral, es posible cargar el conjunto de datos de manera directa desde un repositorio y cargarla en memoria de manera ordenada. Para ello, dicha librería cuenta con una clase llamada *Citation* que permite la descarga del conjunto y su reorganización.

Una vez hecho esto, es posible explorar sus contenidos:

```
Dataset: Citation(n_graphs=1)
First graph: Graph(n_nodes=2708, n_node_features=1433,
n_edge_features=None, n_labels=7)
Node labels: Case_Based, Genetic_Algorithms, Neural_Networks,
Probabilistic_Methods, Reinforcement_Learning, Rule_Learning,
Theory
```

Por un lado, es posible comprobar que ha sido descargado conjunto de datos compuesto por un único grafo de citaciones con 2708 nodos, 1433 características por nodo, 7 posibles etiquetas (clasificaciones o clases) para los vértices y ninguna característica en sus aristas.

Por otro lado, la librería también construye las colecciones de entidades a usar como juego de entrenamiento, validación y test, que son accesibles desde la misma estructura de datos:

```
Training samples: 140
Validation samples: 500
Test samples: 1000
```

Se comprueba, así, que un subconjunto de los 2708 nodos han sido repartidos en tres grupos: conjunto de entrenamiento, con 140 entidades; conjunto de

validación, con 500 vértices; conjunto de pruebas, con 1000 observaciones. Estos conjuntos vienen representados mediante estructuras de enmascaramiento: vectores con una posición por cada nodo y un valor 1 o 0 según si cada uno de ellos pertenecen al conjunto dado.

A modo de añadido, la estructura de datos permite acceder a los datos espectrales del grafo:

Contenido de los nodos mediante un numpy array

```
dataset.graphs[0].x
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Matriz de adyacencia mediante un sparse matrix

```
dataset.graphs[0].a
```

```
(0, 0)      0.25
(0, 633)    0.25
(0, 1862)   0.2236068
(0, 2582)   0.25
(1, 1)      0.25
(1, 2)      0.20412415
(1, 652)    0.28867513
(1, 654)    0.35355338
(2, 1)      0.20412415
(2, 2)      0.16666667
...
```

Características de las aristas que está vacío en este conjunto de datos

```
dataset.graphs[0].e
```

```
None
```

Etiquetas de cada nodo, en formato numpy array

```
dataset.graphs[0].y
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

2.4.3. Pruebas de clasificación de nodos

Como ha sido expuesto en el apartado 2.3, las redes neuronales gráficas pueden ser utilizadas mediante dos aproximaciones básicas: espectral, para lo que se hará uso de la teoría del mismo nombre para describir grafos, y espacial, gracias al concepto de paso de mensajes.

Estas dos maneras de trabajar el problema permiten el aprendizaje en base a dos factores:

- Contenido de los nodos, esto es, sus características.
- Atributos de los vértices que forman el vecindario de cada observación.

Con todo, una de las mayores diferencias entre ambos modos de trabajar es el nivel de profundidad dentro de la estructura a la hora de aprender: dentro de los modelos espectrales el aprendizaje vendrá dado por la manera en la que los nodos se relacionan con sus vecinos directos y las características de las entidades que lo forman, mientras que el espacial, gracias a la técnica de paso de mensajes ya estudiada, permitirá una comprensión más profunda, al tener en cuenta varios niveles de vecindarios, con nodos cada vez más alejados.

Así, el conjunto de datos escogido nos permitirá el uso de modelos que, entrenados mediante el conjunto de entrenamiento, puedan predecir la categoría de una entidad dado su vecindario.

A continuación, se presentan tres experimentos:

1. Clasificación de nodos mediante un modelo espectral.
2. Clasificación de nodos mediante un modelo espacial.
3. Clasificación de nodos mediante el uso de un modelo tradicional no gráfico, para lo que se atenderá solo a las características de los nodos y no a sus relaciones.

Los objetivos de estos tres experimentos son, por un lado, mostrar la manera de hacer uso de estos modelos y, por otro, comparar sus niveles de precisión.

Como paso previo antes de los experimentos, se computará el peso inicial de cada unidad de entrada a la red neuronal. Para ello, se asignará a cada nodo el mismo peso dentro de cada conjunto de datos mediante el siguiente código Python que, si bien aparece en el *notebook*, es importante explicarlo en esta memoria:

```
weighted_mask = [
    mask.astype(np.float32) / np.count_nonzero(mask)
    for mask in (dataset.mask_tr, dataset.mask_va,
dataset.mask_te)
]
```

Con este procedimiento, se genera un *numpy array* para cada conjunto, donde

el peso de cada nodo viene dado por 1 dividido entre el total de muestras dentro del conjunto, de modo que se obtienen los siguientes valores:

```
Training samples weights: 0.0071428571827709675
Validation samples weights: 0.0020000000949949026
Test samples weights: 0.0010000001639127731
```

2.4.3.1. Experimento 1: clasificación de nodos mediante modelo espectral

En este experimento se va a proceder a clasificar los nodos mediante una red convolucional gráfica que hace uso de la información espectral del grafo: matriz de adyacencia y contenido de los nodos, tal y como se expuso en el apartado 2.2.2 de esta memoria.

Para empezar, gracias a Spektral es posible preparar un objeto que describe los contenidos de los conjuntos de entrenamiento y validación, así como sus pesos de entrada. Puesto que nuestro conjunto de datos está formado por un único grafo, se hace uso de la clase *SingleLoader* de la librería [59].

Para continuar, dentro de la librería se dispone de una colección de modelos ya programados para su uso. Dado que se desea probar la aplicación espectral, se hará uso de la clase *GCN* que ofrece [64] y que configuraremos mediante los siguientes hiperparámetros:

- Ritmo de aprendizaje (*learning rate*) de 0,01.
- Función de agregación (*reduction function*) mediante suma.
- Optimizador Adam como función de optimización para la búsqueda de mínimos en la función de error, pues permite trabajar de una manera eficiente para conjuntos de datos muy heterogéneos que pueden llevar a generar funciones de error con numerosos mínimos locales [28]. Esta función de optimización permitirá al modelo localizar los mínimos locales y globales de la función de error, tal y como se ha expuesto en el apartado 2.1 del proyecto.

Dado que es un experimento cuyo código podría ser interesante tener disponible para otras pruebas sobre otros conjuntos, se opta por configurar el proceso de aprendizaje para que pueda pasar por un número de épocas (*epochs*) elevado: 400. Sin embargo, para evitar hacer repeticiones más allá de lo necesario según el éxito del aprendizaje, se hace uso de la opción de *early stopping* ofrecida por Keras, para lo que se configura una paciencia de 10: si pasado dicho número de épocas la precisión no evoluciona de manera clara, el proceso se dará por finalizado.

Configurado el modelo, es posible definirlo y entrenarlo:

```
model_gcn = GCN(n_labels=dataset.n_labels,
n_input_channels=dataset.n_node_features)

model_gcn.compile(
     optimizer=optimizer, loss=loss_function,
```

```
    weighted_metrics=metrics_to_evaluate)

model_gcn.fit(
    loader_training.load(),
    steps_per_epoch=loader_training.steps_per_epoch,
    validation_data=loader_validation.load(),
    validation_steps=loader_validation.steps_per_epoch,
    epochs=max_number_of_epochs,
    callbacks=[
        EarlyStopping(patience=early_stopping_patience,
              restore_best_weights=True),
        TensorBoard(get_run_logdir('gcn_spectral'))
    ]
)
```

Así, ha sido posible definir el flujo de datos y la red neuronal necesarias para probar el modelo:

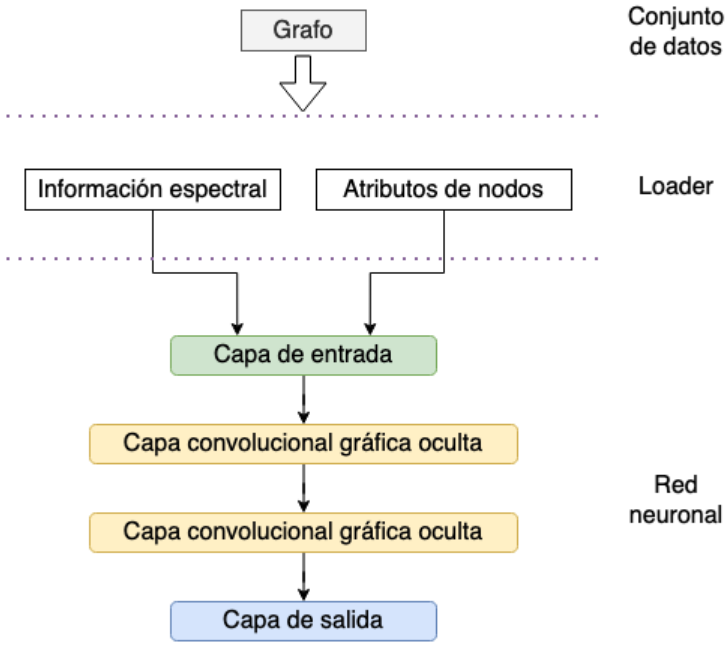


Figure 8: Diagrama de red neuronal convolucional espectral. Se puede comprobar cómo el cargador (Loader) convierte el grafo en la información espectral necesaria para ser usada en la entrada a la red, que está formada por dos capas convolucionales ocultas.

Al ejecutarlo, se puede ver en la salida los resultados de precisión y pérdida en cada época:

```
Epoch 1/400
1/1 [==============================] - 0s 433ms/step - loss:
1.9541 - acc: 0.1071 - val_loss: 1.9512 - val_acc: 0.2660
Epoch 2/400
```

```
1/1 [==============================] - 0s 41ms/step - loss:
1.9482 - acc: 0.3071 - val_loss: 1.9482 - val_acc: 0.3800
Epoch 3/400
1/1 [==============================] - 0s 47ms/step - loss:
1.9422 - acc: 0.5571 - val_loss: 1.9451 - val_acc: 0.4020
Epoch 4/400
1/1 [==============================] - 0s 47ms/step - loss:
1.9372 - acc: 0.5571 - val_loss: 1.9420 - val_acc: 0.3980
Epoch 5/400
1/1 [==============================] - 0s 47ms/step - loss:
1.9288 - acc: 0.6143 - val_loss: 1.9391 - val_acc: 0.3760
...
Epoch 255/400
1/1 [==============================] - 0s 56ms/step - loss:
0.5319 - acc: 0.9786 - val_loss: 1.0503 - val_acc: 0.7860
Epoch 256/400
1/1 [==============================] - 0s 59ms/step - loss:
0.5524 - acc: 0.9571 - val_loss: 1.0501 - val_acc: 0.7840
```

Como se puede ver, en la ejecución de ejemplo que ha sido realizada el proceso de aprendizaje se para de manera temprana en la época número 256 gracias al citado *early stopping*. Se alcanza un nivel de precisión máximo de un 97,86% sobre el conjunto de entrenamiento y de un 78,60% sobre el de validación en la época 164. El error acumulado sobre el juego de validación es de 1,05.

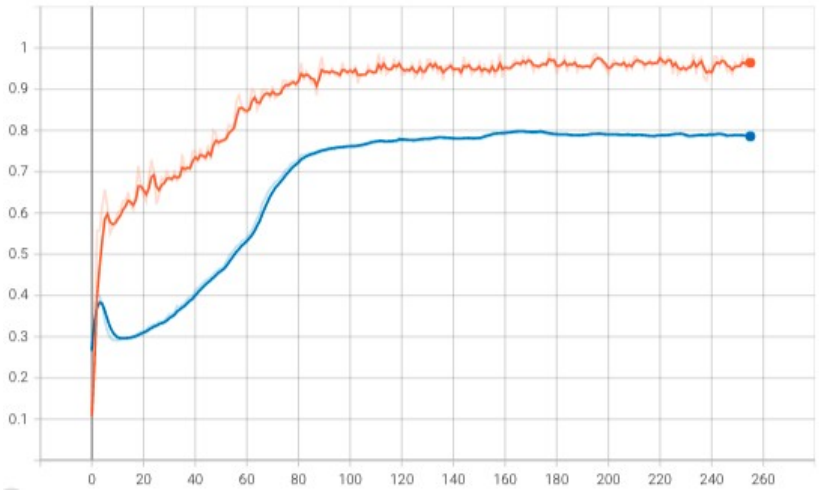


Figure 9: Evolución de la precisión sobre los conjuntos de entrenamiento (rojo) y de validación (azul).

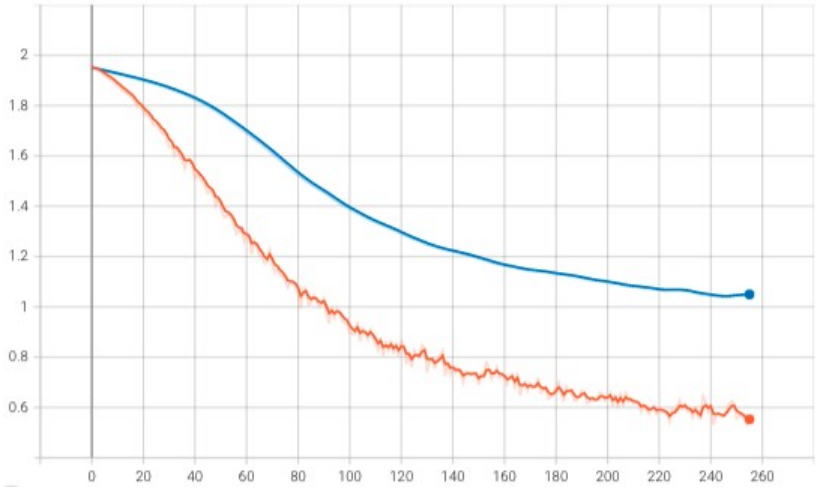


Figure 10: Evolución de error sobre los conjuntos de entrenamiento (rojo) y de validación (azul).

Entrenado el modelo, puede ser usado para evaluar la precisión obtenida frente al conjunto de datos de prueba:

```
loader_te = SingleLoader(dataset,
    sample_weights=weighted_mask[2])
model_gcn.evaluate(loader_te.load(),
    steps=loader_te.steps_per_epoch)
```

Lo que arroja el siguiente resultado:

```
1/1 [==============================] - 0s 25ms/step - loss:
1.0007 - acc: 0.8090
Test loss: 1.0007295608520508
Test accuracy: 0.8090001940727234
```

Es decir, una precisión del 80,9% sobre el juego de prueba, con un error acumulado que ronda el 1.

2.4.3.2. Experimento 2: clasificación de nodos mediante modelo espacial

Los modelos espaciales, tal y como se expone en el apartado 2.3.3, hacen uso de la técnica de paso de mensaje para aprender ya no solo según el vecindario inmediato de un nodo, sino en base al vecindario de sus vecinos. Para ello, el modelo deberá iterar por cada nivel adicional que se desee tener en cuenta, de modo que vaya pasando “mensajes” de un nodo al siguiente.

En este experimento se hará uso de la clase ofrecida por Spektral con nombre *MessagePassing*, ya preparada para este modelo.

Para poder preparar un modelo semejante será preciso primero tomar dos decisiones básicas, que, para esta prueba, estarán fundamentadas sobre, como la propia documentación de Spektral, la propuesta de Kipf y Welling [40]:

- Método matemático para calcular el peso valor del mensaje a pasar de una arista a otra según los nodos que tiene en sus extremos. Para este

experimento, se ha optado por pasar las características de cada nodo vecino sin realizar modificaciones

Dentro de la API ofrecida por la clase *MessagePassing* se esta acción será definida dentro del método *message*:

```
def message(self, x):
        return self.get_j(x)
```

- Qué función de agregación será usada en cada nodo para operar sobre los mensajes que llegan desde sus aristas y actualizarse según el mismo. En base al citado artículo, se hará uso de la media agregada:

```
def aggregate(self, messages):
        return spektral.layers.ops.scatter_mean(messages,
               self.index_i, self.n_nodes)
```

El método *scatter_mean* será el encargado de aplicar dicha agregación de media según los mensajes llegados desde los nodos vecinos.

- Método a usar para actualizar las características de cada nodo según el resultado de la función de agregación, que en este caso será el producto de la matriz de características original por el resultado de la misma, todo ello implementado dentro de *call*, cuya misión es propagar el mensaje en cada iteración, tal como define el propio paper:

```
def call(self, inputs):
        x, a = inputs
        x = tf.matmul(x, self.kernel)
        return self.propagate(x=x, a=a)
```

- Como todo modelo, es necesario definir la función de activación para las unidades que forman la capa de paso de mensajes. Para el ejemplo se ha optado, de nuevo, por hacer uso de lo definido en el artículo: *Softmax* [3].

El modelo se define con una entrada compuesta por dos elementos: la matriz de adyacencia, que tendrá tantas columnas y filas como nodos y que, para asegurar una correcta escalabilidad, será de tipo *sparse*; un vector con las características de los nodos, también de tipo *sparse*. Mediante las clases *Input* de Spektral, se definen los tamaños y tipos de contenido de las dos estructuras de entrada:

```
from keras.layers import Input
adjacency_input = Input(
    (number_of_nodes,), sparse=True, dtype=dataset[0].a.dtype,
    name='adjacency_matrix_input'
)
x_input = Input(
    shape=(number_of_node_features,),
    name='nodes_input'
)
```

Con todo, se procede a definir el modelo compuesto por las dos entradas descritas y cuya capa de salida será la capa de paso de mensajes:

```
# Define output layer based on the MPNN layer defined previously
as MPNNLayer
mpnn_output_layer = MPNNLayer(
    number_of_labels, activation=keras.activations.softmax,
    kernel_regularizer=keras.regularizers.l2(l2_regularization_ra
    te), use_bias=False, name='mpnn_layer'
)([x_input, adjacency_input])

model_mpnn = Model(
     name='mpnn_gcn',
    inputs=[x_input, adjacency_input], outputs=mpnn_output_layer
)
model_mpnn.compile(

    optimizer=keras.optimizers.Adam(learning_rate=learning_rate),
    loss=keras.losses.CategoricalCrossentropy(),
    weighted_metrics=['acc']
)
```

Gracias al método *summary*, se puede representar el modelo que ha sido construido mediante Keras y la capa de paso de mensajesdefinida mediante Spektral:

Model: “mpnn_gcn”

Layer (type)	Output Shape	Param #	Connected to
nodes_input (InputLayer)	[(None, 1433)]	0	[]
adjacency_matrix_input (InputLayer)	[(None, 2708)]	0	[]
mpnn_layer (MPNNLayer)	(None, 7)	10031	['nodes_input[0][0]', 'adjacency_matrix_input[0][0]']

Total params:	10,031
Trainable params:	10,031
Non-trainable params:	0

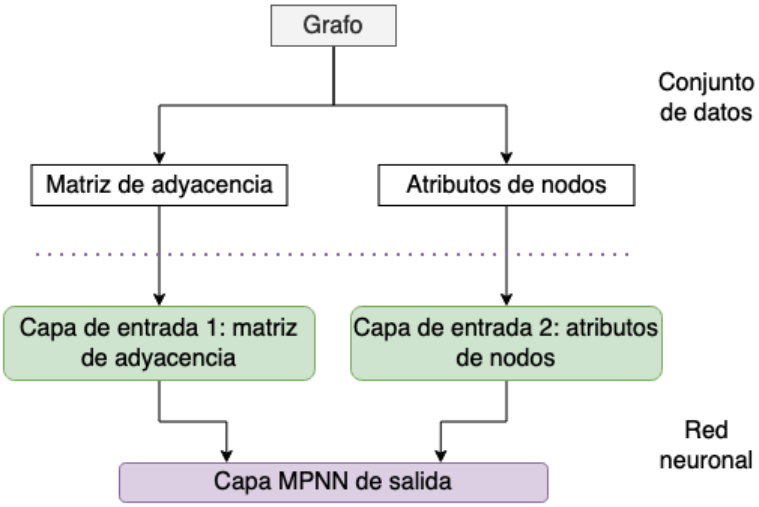


Figure 11: Diagrama del proceso de carga y red neuronal de paso de mensajes. Puede verse que consta de dos capas de entrada: una para la matriz de adyacencia, otra para los atributos de los nodos. Cuenta, además, con una capa de paso de mensajes que funciona también como salida.

Con el modelo definido, se procede a configurar los cargadores de datos como en el ejemplo anterior y se llama a la función de entrenamiento, a la que se le pasan los parámetros de configuración ya comentados. Como en el caso anterior, se hace uso de *early stopping* para detener el proceso si no hay una mejora significativa y se prueba un entrenamiento con 400 épocas:

```
Epoch 1/400
1/1 [==============================] - 1s 524ms/step - loss:
1.9466 - acc: 0.1143 - val_loss: 1.8721 - val_acc: 0.7020
Epoch 2/400
1/1 [==============================] - 0s 37ms/step - loss:
1.7838 - acc: 0.9929 - val_loss: 1.7995 - val_acc: 0.7040
Epoch 3/400
1/1 [==============================] - 0s 40ms/step - loss:
1.6265 - acc: 0.9929 - val_loss: 1.7294 - val_acc: 0.7100
Epoch 4/400
1/1 [==============================] - 0s 40ms/step - loss:
1.4764 - acc: 0.9929 - val_loss: 1.6620 - val_acc: 0.7160
Epoch 5/400
1/1 [==============================] - 0s 40ms/step - loss:
1.3345 - acc: 0.9929 - val_loss: 1.5976 - val_acc: 0.7200
...
Epoch 67/400
1/1 [==============================] - 0s 26ms/step - loss:
0.0283 - acc: 1.0000 - val_loss: 0.8606 - val_acc: 0.7180
Epoch 68/400
1/1 [==============================] - 0s 25ms/step - loss:
0.0278 - acc: 1.0000 - val_loss: 0.8608 - val_acc: 0.7180
```

Así, se detiene el entrenamiento en la época 68. Sin embargo, podemos ver algo extraño: el nivel de precisión sobre el conjunto de validación parece quedarse estancado, algo que se ve más claro mediante la representación gráfica que se extrae de TensorBoard:

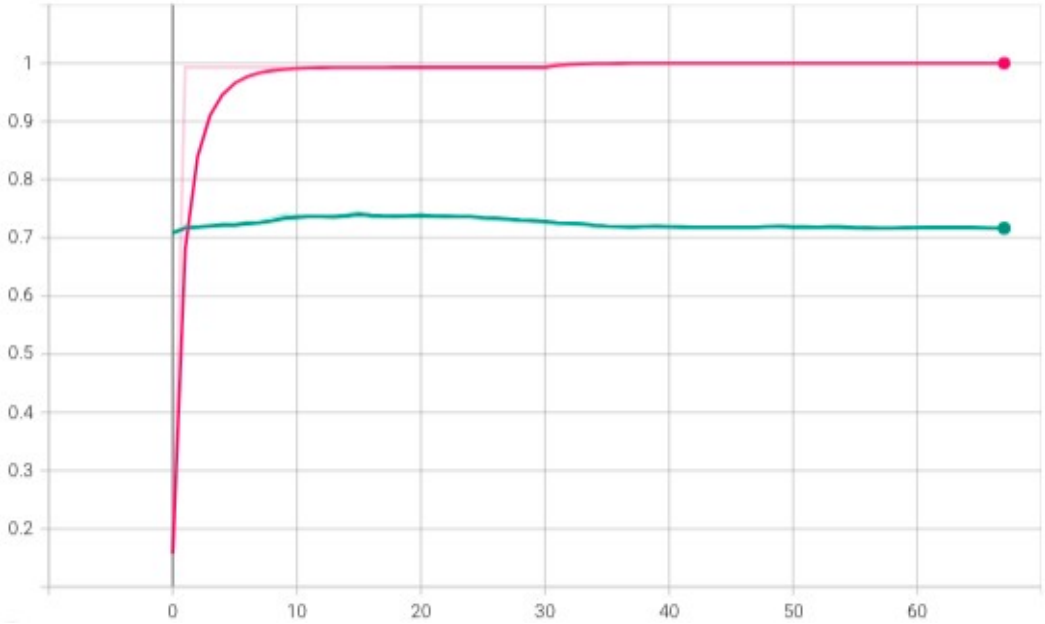


Figure 12: Evolución de precisión sobre el conjunto de entrenamiento (en rojo) y sobre el de validación (en verde)

En vista de estos datos, parece que puede haber un riesgo de *overfitting* o sobreoptimización sobre el juego de entrenamiento: esto ya es algo que se propone como posibilidad en la propuesta original de redes con paso de mensaje y es algo a vigilar cuando el juego de datos es pequeño.

Dado que nos encontramos ante un modelo espacial con paso de mensajes, cada época o iteración de entrenamiento ampliará el rango de acción del paso de mensajes. Por otro lado, puesto que el conjunto de datos es de un tamaño reducido, de manera intuitiva se puede deducir que puede bastar con 2 o 3 iteraciones para cumplir con el objetivo de arrojar resultados buenos a la hora de generalizar el modelo.

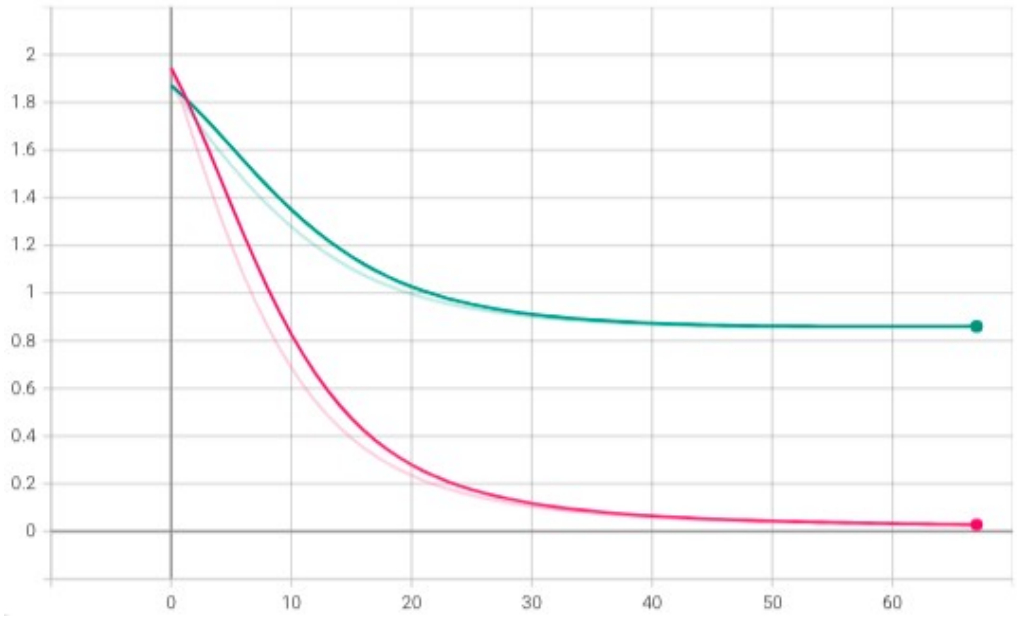


Figure 13: Evolución de error sobre el conjunto de entrenamiento (en rojo) y sobre el de validación (en verde)

En cuanto a la evolución del error acumulado, es posible comprobar algo similar: tiende a estabilizarse durante las primeras iteraciones.

Con todo, se procede a comprobar la precisión final sobre el mismo, tras cargar el conjunto desde el enmascarado:

```
model_mpnn.evaluate(loader_te.load(),
      steps=loader_te.steps_per_epoch)
```

Lo que arroja los siguientes resultados:

```
1/1 [==============================] - 0s 15ms/step - loss:
0.8307 - acc: 0.7490
Test loss: 0.8307310938835144
Test accuracy: 0.7490002512931824
```

Como se puede comprobar, se ha obtenido una precisión de 74,9% y un error acumulado de 0,83 sobre el conjunto de pruebas.

2.4.3.3. Experimento 3: clasificación mediante modelo no gráfico

Tras las dos pruebas anteriores, se va a proceder a llevar a cabo un experimento adicional: comprobar el nivel de precisión que puede alcanzarse mediante el uso de un modelo tradicional, esto es, uno que no haga uso de la estructura del grafo para aprender y llevar a cabo predicciones.

Como paso inicial, se deberá preparar los datos para no hacer uso de la estructura. Tal y como se ha presentado con anterioridad, el juego de datos

descargado cuenta con una matriz de características de los nodos que puede ser filtrada mediante enmascaramiento. Así, será posible construir tres subconjuntos de entre los nodos: entrenamiento, validación y pruebas. Además, se construirán los mismos para obtener la clasificación original de cada vértice dentro de su clase:

```
from  sklearn.model_selection  import  train_test_split

X = dataset[0].x
y = dataset[0].y
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)
X_train, X_validation, y_train, y_validation = train_test_split(
    X_train, y_train, test_size=0.2, random_state=42,
    stratify=y_train)

print(f'Training: X {X_train.shape}, y {y_train.shape}, from a
    source of {np.sum(dataset.mask_tr)} samples')
print(f'Validation: X {X_validation.shape}, y
    {y_validation.shape}, from a source of
    {np.sum(dataset.mask_va)} samples')
print(f'Test: X {X_test.shape}, y {y_test.shape}, from a source
    of {np.sum(dataset.mask_te)} samples')
```

Lo que permite obtener la siguiente colección de datos:

```
Training: X (1516, 1433), y (1516, 7), 1516 nodes
Validation: X (379, 1433), y (379, 7),  379 nodes
Test: X (813, 1433), y (813, 7), 813 nodes
```

Después, se procede a definir un modelo de red neuronal construido mediante la API secuencial de Keras incluida con TensorFlow. EN este caso, se hace uso de una única capa oculta con 100 unidades. La entrada estará preparada para aceptar observaciones del conjunto de entrenamiento que ha sido preparado con anterioridad:

```
sequential_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=X_train[0].shape,
          name='nodes_input'),
    keras.layers.Dense(100, activation=keras.activations.relu,
          name='hidden_dense'),
    keras.layers.Dense(7, activation=keras.activations.relu,
          name='output')
], name='non_gnn')
```

Así, la red queda definida como sigue:

```
Model: "non_gnn"
```

Layer (type)	Output Shape	Param #
nodes_input (Flatten)	(None, 1433)	0
hidden_dense (Dense)	(None, 100)	143400
output (Dense)	(None, 7)	707
Total params:	144,107	

```
Trainable params:    144,107
Non-trainable        0
params:
```

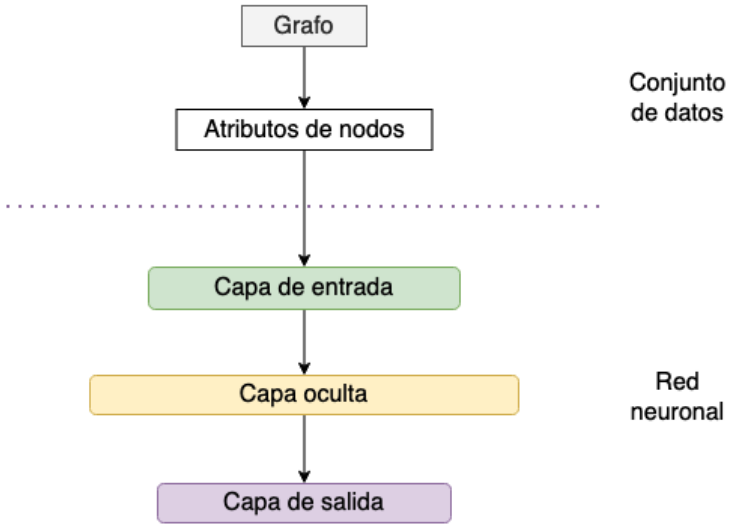


Figure 14: Diagrama de la red neuronal densa que ha sido planteada. La entrada a la red solo hace uso de los atributos de los nodos, de modo que se ignora la estructura relacional del grafo.

Creada la red, se puede proceder a compilar y entrenar. Para esta prueba, se hace uso de un ritmo de aprendizaje por defecto para el optimizador Adam. Además, se opta por un máximo de 30 épocas:

```
sequential_model.compile(
    optimizer=Adam(),
    loss=CategoricalCrossentropy(reduction=reduction_function),
    weighted_metrics=['acc']
)

sequential_model.fit(
    X_train, y_train, epochs=30,
    validation_data=(X_validation, y_validation),
    callbacks=[
        EarlyStopping(patience=20,  restore_best_weights=True),
        TensorBoard(get_run_logdir('non_gnn'))
    ]
)
```

Con el modelo listo, se procede a ejecutar el entrenamiento:

```
Epoch 1/30
48/48 [==============================] - 1s 6ms/step - loss:
120.9424 - acc: 0.3687 - val_loss: 61.4952 - val_acc: 0.5040
Epoch 2/30
48/48 [==============================] - 0s 4ms/step - loss:
49.6267 - acc: 0.7447 - val_loss: nan - val_acc: 0.6121
Epoch 3/30
48/48 [==============================] - 0s 5ms/step - loss: nan
- acc: 0.2460 - val_loss: nan - val_acc: 0.1293
```

```
Epoch 4/30
48/48 [==============================] - 0s 6ms/step - loss: nan
- acc: 0.1299 - val_loss: nan - val_acc: 0.1293
Epoch 5/30
48/48 [==============================] - 0s 4ms/step - loss: nan
- acc: 0.1299 - val_loss: nan - val_acc: 0.1293
...
Epoch 20/30
48/48 [==============================] - 0s 5ms/step - loss: nan
- acc: 0.1299 - val_loss: nan - val_acc: 0.1293
Epoch 21/30
48/48 [==============================] - 0s 4ms/step - loss: nan
- acc: 0.1299 - val_loss: nan - val_acc: 0.1293
```

Tras la ejecución, se procede a estudiar la evolución de las curvas de precisión y de error.

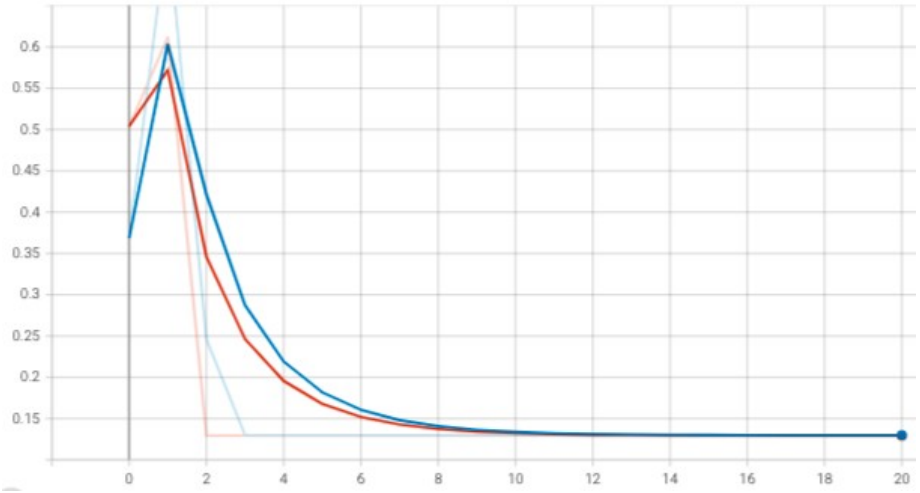


Figure 15: Evolución de precisión sobre el conjunto de entrenamiento (en azul) y sobre el de validación (en rojo)

Se procede a evaluar el modelo frente a los distintos conjuntos de entrenamiento, validación y pruebas:

```
Evaluation over training data:
      Loss: 73.60014343261719
      Accuracy: 68.47%
Evaluation over validation data:
      Loss: 81.18524932861328
      Accuracy: 50.40%
Evaluation over test data:
      Loss: 73.11640167236328
      Accuracy: 54.49%
```

Se comprueba, así, una precisión del 68,47% sobre el conjunto de entrenamiento y un 50,40%. Al generalizar sobre el conjunto de test, sin embargo, se obtiene una precisión del 54,49%. Con el fin de profundizar en esta problemática, se genera un informe de clasificación y la matriz de

confusión sobre el resultado obtenido:

```
                        precision    recall  f1-score   support

             Case_Based       0.46      0.37      0.41       105
    Genetic_Algorithms        0.71      0.23      0.35        65
       Neural_Networks        0.66      0.53      0.59       126
 Probabilistic_Methods        0.50      0.87      0.63       246
Reinforcement_Learning        0.62      0.44      0.51       128
         Rule_Learning        0.72      0.38      0.50        89
                Theory        0.50      0.33      0.40        54

              accuracy                           0.54       813
             macro avg        0.60      0.45      0.48       813
          weighted avg        0.58      0.54      0.53       813
```

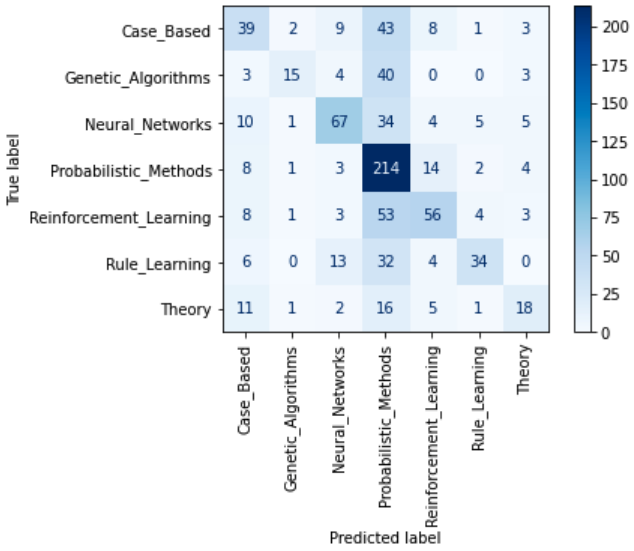


Figure 16: Matriz de confusión sobre el resultado del experimento 3, donde se puede comprobar la relación entre clasificaciones esperadas y las obtenidas.

De entrada, llama la atención el elevado *recall* de la clase *Probabilistic_Methods*, aunque su precisión es baja. No parece que destaque la precisión de ninguna en particular.

Puesto que el conjunto de datos cuenta con una cantidad de 1433 atributos por muestra, dentro del *notebook* se ha procedido a llevar a cabo una reducción de dimensionalidad mediante *FeatureAgglomeration* de Scikit-Learn. Tras llevar a cabo un proceso de búsqueda de agrupaciones apropiado, se han generado tres posibles versiones del conjunto de datos: 732, 995 y 1242 atributos.

	Entrenamiento		**Validación**		**Pruebas**	
#	**Precisión**	**Pérdida**	**Precisión**	**Pérdida**	**Precisión**	**Pérdida**
732	0.77	50.61	0.64	70.31	0.62	64.56
995	0.83	50.79	0.64	79.10	0.66	58.59
1242	0.86		0.63	63.10	0.65	64.17

De entre los tres, el que ha ofrecido una mejor precisión a la hora de generalizar tras aplicar el mismo modelo de red neuronal anterior ha sido el de 995 atributos, con una precisión sobre el conjunto de pruebas del 66,05% y una pérdida de 58,59. A continuación, puede verse el informe de clasificación y la matriz de confusión para dicha configuración.

	precision	recall	f1-score	support
Case_Based	0.47	0.63	0.54	105
Genetic_Algorithms	0.66	0.51	0.57	65
Neural_Networks	0.78	0.67	0.72	126
Probabilistic_Methods	0.71	0.78	0.75	246
Reinforcement_Learning	0.67	0.65	0.66	128
Rule_Learning	0.66	0.58	0.62	89
Theory	0.60	0.50	0.55	54
accuracy			0.66	813
macro avg	0.65	0.62	0.63	813
weighted avg	0.67	0.66	0.66	813

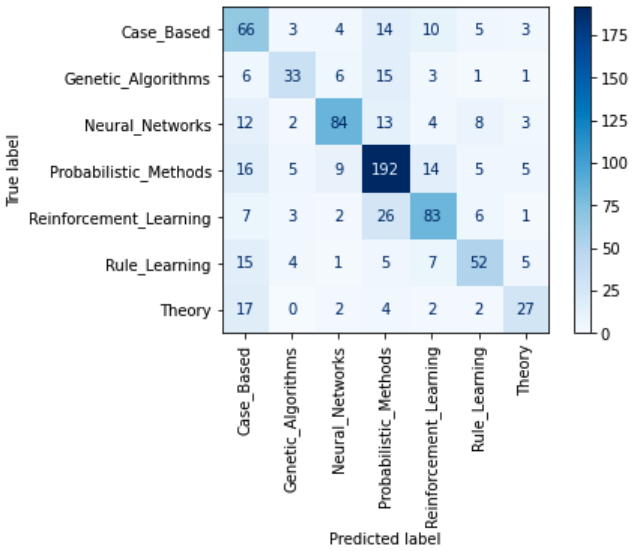


Figure 17: Matriz de confusión enriquecida tras la reducción de dimensionalidad a 995 atributos.

Pueden verse los detalles en la implementación del cuaderno de código adjunto con el proyecto (Anexo 2), así como el

2.4.4. Pruebas de predicción de enlaces

Hasta el momento, las implementaciones han tenido el objetivo de predecir la clasificación de los nodos del grafo. De entrada, los dos primeros ejemplos han hecho uso tanto de las características de los vértices como de la estructura del

grafo, cada uno con un enfoque profundidad diferentes. El tercero, por su lado, ha hecho uso únicamente de los contenidos de cada entidad para decidir su clasificación. Como se presentó en el apartado 2.3.1, existe otro caso de uso muy útil para las redes neuronales gráficas: la predicción de enlaces.

2.4.4.1. Experimento 4: predicción de enlaces mediante WalkPooling

Para esta prueba de implementación, en lugar de hacer uso de Spektral y TensorFlow, se le da una oportunidad a PyTorch y al código propuesto en el artículo oficinal para el algoritmo WalkPooling [65,66].

Dentro de PyTorch se presenta una librería específica conocida como PyTorch Geometric o PyG, la cuál tiene como principal objetivo, al igual que Spektral, el facilitar el trabajo con redes neuronales gráficas [67]. Para este caso, se hará uso de la técnica de paso de mensajes aplicada a la predicción de enlaces.

El primero paso antes de poder realizar la prueba es, por supuesto, la preparación del entorno. Para ello, el *notebook* adjunto está preparado para descargar el repositorio de WalkPooling e importar sus módulos de predicción de enlaces. Tras esto, se deben configurar dos variables clave:

- *walk_length*, que indica la longitud en aristas que podrá recorrer el algoritmo por el grafo desde cada nodo en las iteraciones de aprendizaje. Se sitúa, como indican los datos de configuración por defecto del artículo, en un valor de *2*.
- *epoch_num*, que configura la cantidad de épocas que serán realizadas para el aprendizaje.

El siguiente paso preparatorio será la construcción del conjunto de datos, algo que ya permite el código del repositorio. Con todo, se hará uso de un 20% del conjunto de datos para ser usado como validación y otro ratio equivalente para el conjunto de test.

Tras descargar y preparar los datos, se procede a explorar su contenido:

```
train_loader.dataset
      [Data(x=[168, 1433], edge_index=[2, 440], z=0,
      edge_mask=[440], label=1.0),
       Data(x=[159, 1433], edge_index=[2, 388], z=0,
      edge_mask=[388], label=1.0),
       Data(x=[16, 1433], edge_index=[2, 34], z=0,
      edge_mask=[34], label=1.0),
       Data(x=[39, 1433], edge_index=[2, 96], z=0,
      edge_mask=[96], label=1.0),
       Data(x=[28, 1433], edge_index=[2, 74], z=0,
      edge_mask=[74], label=1.0),
       Data(x=[125, 1433], edge_index=[2, 350], z=0,
      edge_mask=[350], label=1.0),
       Data(x=[122, 1433], edge_index=[2, 364], z=0,
      edge_mask=[364], label=1.0),
       Data(x=[36, 1433], edge_index=[2, 80], z=0,
      edge_mask=[80], label=1.0),
```

```
 Data(x=[10, 1433], edge_index=[2, 20], z=0,
edge_mask=[20], label=1.0),
…
 Data(x=[302, 1433], edge_index=[2, 830], z=0,
edge_mask=[830], label=1.0),
 Data(x=[43, 1433], edge_index=[2, 104], z=0,
edge_mask=[104], label=1.0),
 ...]
```

Como se puede comprobar, el método de preparación de datos a dividido el grafo original de CORA en subgrafos que puedan ser usados durante el proceso de entrenamiento iterativo.

Con los datos listos, se procede a implementar el modelo mediante la clase incluida en el repositorio de WalkPooling *LinkPred* que hereda de la clase *MessagePassing* proveída por PyTorch Geometric. Se elige, además, el algoritmo *Adam* como optimizador y como función de pérdida *MSE*.

Una vez que los conjuntos de datos y el modelo están listos, se procede a implementar las funciones de entrenamiento y pruebas, ambas conectadas a TensorBoard para poder obtener un seguimiento más detallado de su progresión.

Como se podrá comprobar en el *notebook*, la ejecución de este modelo es más lenta, pues lleva a cabo sucesivas iteraciones de entrenamiento, validación y pruebas sobre los subgrafos que han sido generados durante la preparación de datos. Esto lleva a realizar 198 iteraciones de entrenamiento, 66 de validación y las mismas de prueba por cada época:

```
train: 100%|██████████| 198/198 [01:02<00:00,  3.15it/s]
test:val: 100%|██████████| 66/66 [00:10<00:00,  6.16it/s]
test:test: 100%|██████████| 66/66 [00:08<00:00,  7.56it/s]
Epoch: 1/10  Loss: 0.2355
Validation accuracy: 0.8321   Validation loss: 0.1553
Test accuracy: 0.8245   Test loss: 0.0962
train: 100%|██████████| 198/198 [00:58<00:00,  3.38it/s]
test:val: 100%|██████████| 66/66 [00:09<00:00,  7.01it/s]
test:test: 100%|██████████| 66/66 [00:08<00:00,  7.49it/s]
Epoch: 2/10  Loss: 0.2022
Validation accuracy: 0.8539   Validation loss: 0.1379
Test accuracy: 0.8468   Test loss: 0.1229
train: 100%|██████████| 198/198 [00:59<00:00,  3.33it/s]
test:val: 100%|██████████| 66/66 [00:09<00:00,  7.11it/s]
test:test: 100%|██████████| 66/66 [00:08<00:00,  7.77it/s]
Epoch: 3/10  Loss: 0.1874
…
train: 100%|██████████| 198/198 [01:01<00:00,  3.22it/s]
test:val: 100%|██████████| 66/66 [00:11<00:00,  5.66it/s]
test:test: 100%|██████████| 66/66 [00:09<00:00,  7.28it/s]
Epoch: 9/10  Loss: 0.1522
Validation accuracy: 0.8364   Validation loss: 0.2399
Test accuracy: 0.8360   Test loss: 0.1134
train: 100%|██████████| 198/198 [01:07<00:00,  2.92it/s]
test:val: 100%|██████████| 66/66 [00:09<00:00,  6.86it/s]
test:test: 100%|██████████| 66/66 [00:09<00:00,  7.12it/s]Epoch:
```

```
10/10  Loss: 0.1480
Validation accuracy: 0.8187   Validation loss: 0.1584
Test accuracy: 0.8148   Test loss: 0.1423
From loss: Final Test AUC: 0.8085, Final Test AP: 0.8468
From AUC: Final Test AUC: 0.8104, Final Test AP: 0.8468
```

Tras ejecutar las 10 épocas de entrenamiento, validación y test, se obtiene el valor máximo de precisión sobre los conjuntos de validación y pruebas en la época 4:

- Sobre conjunto de validación en época 3: precisión de 85,39% y pérdida de 0,14.
- Sobre conjunto de prueba en época 4: precisión de 84,86% y pérdida de 0,13.

Mediante TensorBoard podemos observar la evolución del proceso al detalle:

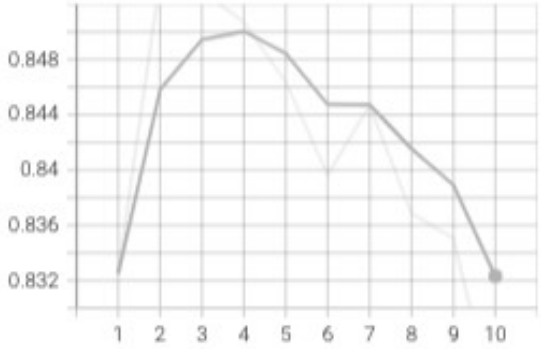


Figure 18: Evolución de precisión sobre conjunto de validación.

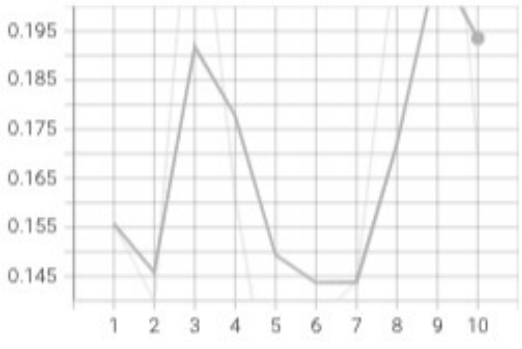


Figure 19: Evolución de error sobre conjunto de validación.

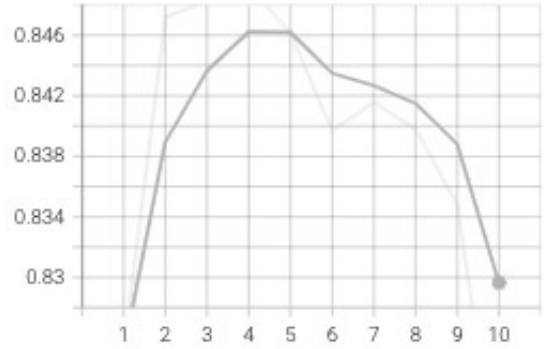


Figure 20: Evolución de precisión sobre conjunto de prueba.

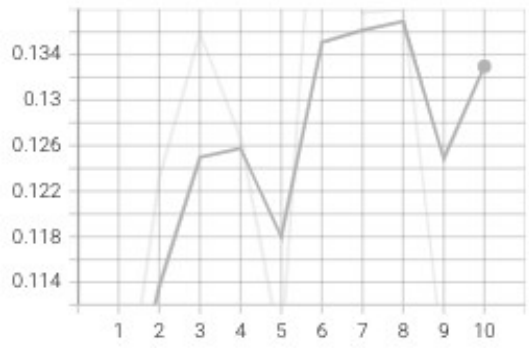


Figure 21: Evolución de error sobre conjunto de prueba.

Tal y como se puede comprobar, el proceso de entrenamiento llega a un nivel

de precisión máximo en la época 4 sobre el conjunto de validación y de pruebas , respectivamente, a partir de la cuál se percibe una pérdida progresiva de precisión, tal y como establece el artículo [66]. Es importante remarcar que no se ha hecho uso de técnicas de *early stopping* en esta implementación, con el fin de ceñirse mejor al artículo original.

2.4.5. Conclusiones sobre las pruebas realizadas

Como se adelantaba al principio de este apartado, el principal objetivo ha sido comprobar la complejidad de implementación de distintos modelos de predicción de enlaces y clasificación de nodos, así como medir su nivel de precisión mediante distintas aproximaciones.

A continuación, se presenta un resumen de los resultados obtenidos:

		Entrenamiento		**Validación**		**Pruebas**	
Experimento	**Épocas**	**Precisión**	**Error**	**Precisión**	**Error**	**Precisión**	**Error**
Clasificación de nodos mediante modelo espectral	255/40 mejor: 164	97,86%	0,5319	78,60%	1,05	80,9%	1
Clasificación de nodos mediante modelo espacial	68/400 mejor: 13	99%	0,48	75%	1,16	74,9%	0,83
Clasificación de nodos mediante modelo no gráfico (sobre reducción a 995 atributos)	24/30 mejor: 6	83,84%	50,79	64,11%	79,01	66,05%	58,59
Predicción de enlaces mediante paso de mensajes	10/10 mejor: 4	-	-	85,19%	0,23	84,74%	0,14

Se puede apreciar como los modelos de clasificación de nodos que hacen uso de información estructural del grafo obtienen un nivel de precisión notablemente superior al de aquel que solo usa las características de los nodos, lo que ya en sí supone una buena justificación para el uso de estos modelos. De entre los dos, el espacial puede ser considerado el más rápido en converger hasta una solución óptima: como se visualiza en el apartado 2.4.3.2, con menos de 10 épocas de entrenamiento se puede alcanzar una precisión sobre el conjunto de validación cercana a la máxima registrada en la iteración 68.

Además, merece una mención especial el muy pobre resultado arrojado por el modelo tradicional que no ha hecho uso de la información gráfica. Como ya se ha presentado en el apartado 2.4.4.3, la información aportada para los atributos de los nodos está demasiado dispersa y no parece la más adecuada para

aplicar un algoritmo de clasificación, al no ser posible aproximar una función que los separe dentro del espacio.

En este sentido y tras comparar este resultado con el comportamiento de los modelos gráficos, parece claro que la información relacional (esto es, las citaciones aportadas para cada publicación) sí ha sido suficiente para poder clasificar las entidades del grafo, lo que justifica, una vez más, el uso de redes neuronales gráficas para resolver este tipo de problemas.

Por otro lado, el modelo de predicción de enlaces mediante *walk pooling* destaca por su precisión sobresaliente sobre el juego de test y, a tenor de estos datos, parece un buen candidato para la generalización de la solución. Por contra, su tiempo de aprendizaje es mayor, al llevar a cabo épocas de entrenamiento donde el aprendizaje se realiza sobre subgrafos generados de manera sucesiva.

3. Conclusiones

Punto de partida y su influencia en el proyecto

Al abordar este proyecto, se ha hecho con la intención de explorar un tipo particular de las redes neuronales artificiales utilizadas dentro del campo del *deep learning*: aquellas que trabajan sobre datos estructurados y, más en concreto, grafos.

Para ello, se ha seguido la estrategia planteada en un inicio: explorar su historia y evolución, examinar las razones que motivan el interés por este tipo de modelos y sus diferentes variaciones.

En un momento inicial, se tenía un conocimiento rudimentario sobre las mismas: se comprendía de manera intuitiva que existen dominios de datos que son descritos de manera más eficaz mediante un grafo, tales como las redes sociales o, en general, cualquiera donde sus entidades puedan organizarse mediante jerarquías o conexiones.

Con el progreso del proyecto, la lectura de nuevas fuentes bibliográficas y la visualización de vídeos divulgativos (que han sido debidamente referenciados dentro del apartado 5, “Bibliografía”), se pudo acceder a nuevos puntos de vista para comprender este área de conocimiento: cómo se clasifican los problemas que pueden querer ser resueltos, cómo las redes gráficas encajan dentro del contexto del aprendizaje computacional y su importancia para la toma de decisiones en un escenario donde los objetos tienen relaciones o pueden evolucionar con el tiempo.

Conocimiento adquirido

Aunque la respuesta sencilla sería que “se ha podido chapotear en el conocimiento sobre redes gráficas”, se procede a listar algunos de los conocimientos clave:

- Conocer el concepto de red neuronal gráfica y su posición dentro del contexto del aprendizaje computacional profundo.
- Comprender la manera en la que la representación de un grafo como una estructura de datos influye en el proceso de aprendizaje.
- Entender los distintos tipos de problemas que pueden ser resueltos mediante este tipo de redes: clasificación de nodos y aristas, predicción de nodos y aristas, clasificación de grafos, generación de grafos o subgrafos e, incluso, predicción de acciones con el paso del tiempo.
- Las redes neuronales gráficas pueden alcanzar niveles de precisión más elevados que los basados en espacios euclídeos cuando se les presenta

la suficiente información estructural.

- Conocer distintas aproximaciones algorítmicas a los distintos problemas que se han catalogado.
- Descubrir nuevas referencias bibliográficas que invitan a buscar inspiración, tales como las propuestas por Gilmer et al [49] sobre las redes *MPNN*, el cual se ha encontrado de especial interés, o las redes gráficas de atención planteadas por Veličković et al [52], de especial interés por su novedoso planteamiento, con un recorrido por delante todavía por explorar.
- Se ha adquirido soltura al trabajar con TensorFlow y Keras para la implementación de redes neuronales secuenciales y mediante modelos ya creados de manera previa, como los ofrecidos por Spektral.
- Se ha podido realizar una primera introducción a la librería PyTorch.

Consecución de objetivos

Todos los objetivos de primer orden, que eran aquellos que se habían fijado como obligatorios, han sido cumplidos:

1. Se ha descrito el comportamiento conceptual de las redes neuronales gráficas.
2. Se han explorado distintos tipos de modelos y se ha repasado su funcionamiento y casos de uso claves.
3. Se ha enmarcado este conocimiento dentro del marco del aprendizaje profundo.
4. Se han preparado los datos de entrada para su trabajo con redes gráficas.
5. Se han implementado ejemplos de uso de redes gráficas espectrales y espaciales para la clasificación de nodos. Además, se ha implementado un modelo de predicción de enlaces mediante una aproximación especial.
6. Se han extraído las métricas de precisión apropiadas para cada caso y se han contrastado los resultados obtenidos.

Sin embargo, no ha sido viable abordar los objetivos secundarios opcionales, fracaso que es justificado mediante los restos que se presentan más adelante en este documento. Los objetivos que han quedado en el aire son los siguientes:

1. Implementación de un experimento mediante una red neuronal gráfica recursiva.
2. Exploración de usos para clasificación de grafos y predicción de

secuencias, aunque este último podría suponer una generalización del experimento número cuatro del apartado 2.4.4.1.

3. Realizar una representación gráfica de uno de los grafos, opción que ha sido descartada por haber subestimado el tamaño que podría tomar y por encontrarlo, por ahora, poco útil.

Retos encontrados

Durante la realización del presente trabajo se han encontrado numerosos retos y problemáticas que han tenido que ser resueltos:

- Al principio del proyecto, se consideró la posibilidad de comenzar por una descripción del funcionamiento rudimentario de las redes gráficas. No obstante, durante la realización de la PEC2 se comprobó la necesidad de aportar un contexto más detallado sobre la historia e influencia de estos modelos.
- De igual manera, durante el periodo de realización de la PEC2 también se optó, tras contrastarlo con el director de proyecto, que podría ser oportuno profundizar en las redes gráficas que hacen uso de convoluciones y, desde estas, descender a aquellas de tipo espectral y espacial, al considerarlas de mayor interés.
- Como añadido, se dejó de lado la profundización en tipologías más complejas, tales como las redes gráficas espacio-temporales, de atención o generación, al considerar que no habría tiempo suficiente. Se dejan, así, como un trabajo a ampliar en el futuro.

Así, estos retos tienen más relación con la toma de decisiones sobre la profundidad del trabajo. Además, se encontraron otros retos en el camino de la implementación:

- El trabajo experimental para la predicción de enlaces del apartado 2.4.4.1 tuvo que ser abordado mediante PyTorch y haciendo uso del propio repositorio original del artículo que proponía el algoritmo de *walk pooling*, aunque la intención original era poder implementarlo bajo TensorFlow y Keras.
- Los resultados del experimento 3 en el apartado 2.4.4.3 arrojaron, de entrada, resultados bastante pobres. Tras contrastarlo con el director de proyecto, se optó por aplicar una reducción de dimensionalidad para mejorar la precisión en la generalización.

Seguimiento de la planificación y metodología

No se han encontrado problemas particulares para hacer seguimiento a la planificación propuesta, aunque ésta fue modificada durante la realización de la PEC2 para reflejar el cambio de discurso divulgativo.

Tampoco han surgido problemas con las herramientas elegidas para llevar a

cabo el proyecto o con los recursos iniciales.

En consecuencia, no ha sido preciso realizar cambios en la metodología de trabajo.

Líneas de trabajo futuro

Además de los objetivos secundarios (opcionales) que no han podido ser abordados, el autor de este trabajo considera que el proyecto podría ser expandido mediante algunas acciones extra:

- Explorar las redes espacio-temporales en profundidad, pues plantean opciones muy interesantes para resolver problemas de optimización en dominios relacionales. Un caso de uso excelente sería el de optimizar los tiempos de duración de los semáforos de una ciudad de manera adaptativa al tráfico existente y a la predicción que pueda hacerse sobre su futura evolución en diferentes cruces.
- Llevar a cabo tareas de clasificación de vértices y aristas en escenarios donde existan conjuntos de datos más amplios.
- Estudiar ejemplos adicionales dentro del campo de la generación de grafos. Un buen caso de uso sería la generación de esquemas de acciones sobre patrones aprendidos para relatar historias, generar puzzles para videojuegos u otros.

4. Glosario

Aunque se explica cada término dentro de su contexto a lo largo de la memoria, a continuación se presenta una colección de los más importantes y sus definiciones, así como los acrónimos más importantes.

- Aprendizaje computacional o aprendizaje automático: parte del campo de la inteligencia artificial enfocado en la creación de algoritmos que aprenden de la experiencia y entrenamiento para resolver problemas.
- Conjunto de datos: colección de información que, dentro del contexto de este proyecto, aparece estructurada.
- Conjunto de entrenamiento: subconjunto del conjunto de datos dedicado a tareas de entrenamiento de un algoritmo de aprendizaje computacional.
- Conjunto de test: subconjunto del conjunto de datos dedicado a tareas de evaluación de precisión de un algoritmo de aprendizaje computacional tras su proceso de entrenamiento.
- Conjunto de validación: subconjunto del conjunto de datos dedicado a validar la precisión parcial durante las distintas etapas o épocas de un proceso de entrenamiento.
- *ConvGNN*: *convolutional graph neural network o* red neuronal convolucional gráfica.
- Convolución: denotada como $(f*g)$, donde f y g son dos funciones, una convolución es un proceso matemático que transforma dos funciones en una tercera que incluye su superposición.
- Cálculos euclídeos: operaciones geométricas sobre espacios euclideanos.
- *Deep learning*: término en inglés para referirse a aprendizaje computacional. Ver «aprendizaje computacional».
- Dendrita: terminaciones de una neurona que la conectan, vía las sinapsis, con otras células del mismo tipo. Son usadas para recibir y emitir estímulos desde y a sus vecinas.
- Época: fase o iteración del proceso de entrenamiento de un algoritmo de aprendizaje computacional. Ver «aprendizaje computacional».
- Entrada de datos: punto donde se reciben datos para su procesado.
- Espacio euclidiano o espacio eucídeo: espacio geométrico cuyas

posiciones son descritas mediante coordenadas respecto a un punto dereferencia.

- Función de actualización: función que calcula y actualiza los pesos aprendidos por una neurona artificial tras una época de entrenamiento.
- Función de agregación: definición matemática que hace uso de una colección de elementos para calcular un valor único. Un ejemplo claro es una suma de dos o más operandos, lo que devuelve un valor único.
- Función de paso: definición matemática que establece el umbral a partir del cuál una neurona artificial devolverá su resultado o permanecerá inactiva.
- Función de salida: método que calcula el valor de salida de una neurona artificial.
- *GAE*: *graph auto-encoder*, es un tipo de *GNN* especializada en la creación de representaciones vectoriales de grafos para su uso, por ejemplo, en algoritmos de aprendizaje computacional no gráficos o de tipo *GAT*.
- *GAT*: *graph attention network* o red neuronal gráfica de atención. Es un tipo particular de *GNN* que hace uso de indicadores de atención para decidir a qué tipo de relaciones de un grafo atender con el fin de reducir el coste computacional.
- *GNN*: *graph neural network* o red neuronal gráfica. Es un tipo particular de red neuronal especializada en el aprendizaje sobre estructuras de datos de tipo grafo.
- GNN espectral: subclase de redes gráficas que hace uso de la teoría espectral de grafos para sus procesos de aprendizaje.
- Grafo: estructura de representación donde una colección de objetos (conocidos como vértices, nodos o entidades) se conectan (aristas o arcos) para representar relaciones dentro de un conjunto o dominio.
- Hiperparámetro: parámetro usado para configurar el comportamiento de aprendizaje de una red neuronal.
- Hiperplano: estructura geométrica de una o más dimensiones que divide un espacio euclídeo en dos mitades.
- Keras: librería de Python especializada en modelos de aprendizaje computacional.
- Matriz: representación matemática de colecciones de valores numéricos organizados en dos dimensiones.
- Matriz de adyacencia: matriz que representa la conexión entre vértices dentro de un grafo. Es cuadrada y cuenta con tantas filas y columnas

como nodos hay en la estructura. En el cruce de cada fila y columna aparece como valor el número de conexiones entre los vértices correspondientes.

- Matriz laplaciana: matriz que representa un grafo.
- MPNN: *message-passing neural network* o red neuronal de paso de mensajes. Es un planteamiento de *GNN* espectral.
- Mínimo global: punto de una función matemática donde el valor resultante es el mínimo de todo el dominio.
- Mínimo local: punto de una función matemática donde la derivada de la misma es igual a cero, lo que indica que se encuentra en un valle.
- Neurona: célula del sistema nervioso que se conecta con otras del mismo tipo mediante sinapsis.
- Neurona artificial: construcción computacional que genera una salida en base a una entrada sobre la que se aplica una función de paso.
- Neurotransmisor: señal química emitida por una neurona vía una sinapsis para comunicarse con otra célula adyacente.
- Nivel de red neuronal: capa dentro de una red neuronal formada por varias unidades.
- *Numpy*: librería de Python especializada en la realización de cálculos matemáticos.
- Peso de entrada: valor numérico que indica la importancia que se da a una entrada de datos.
- Python: lenguaje de programación de alto nivel y de propósito general.
- PyTorch: librería de Python especializada en aprendizaje computacional.
- RecGN: *recursive graph network* o red gráfica recursiva. Es un tipo particular de red neuronal donde las neuronas se conectan de manera recursiva.
- RecGNN: tipo particular de *RecGN* especializada en el trabajo sobre estructuras de grafo.
- Red neuronal: estructura de neuronas artificiales interconectadas entre sí para llevar a cabo aproximaciones de funciones no lineales.
- Red neuronal convolucional: tipo particular de red neuronal que hace uso de convoluciones para simplificar los procesos de aprendizaje mediante la especialización de neuronas sobre la detección de determinados patrones.
- *RGCN* o *relational graph convolutional network* o red convolucional

gráfica relacional. Es un tipo particular de *GNN* especializada en el aprendizaje sobre estructuras de grafo donde las aristas cuentan con atributos o diferentes tipologías.

- Sinapsis: punto de conexión entre dos neuronas que facilita el intercambio de señales eléctricas y químicas.
- *STGNN* o *spatio-temporal graph neural network* o red neuronal gráfica espacio-temporal: tipo particular de *GNN* especializada en el aprendizaje sobre estructuras de grafo cuyas relaciones cambian con el paso del tiempo.
- Tensor: representación matemática de colecciones de valores numéricos organizados en tres o más dimensiones.
- TensorFlow: librería de Python que facilita el trabajo de cálculo sobre matrices y tensores de manera distribuida y orientada a su aplicación en tareas de aprendizaje automático.
- Teoría espectral de grafos: teoría matemática dedicada a la descripción explícita de una estructura de grafo mediante, entre otros, sus matrices de adyacencia y laplaciana.
- Unidad: dentro del contexto de las redes neuronales artificiales, una unidad es una neurona dentro de una capa o nivel de la red.
- Vector: representación matemática de colecciones de valores numéricos organizados en una dimensión.
- Vértice o nodo: objeto o entidad fundamental dentro de un grafo.

5. Bibliografía

[1] FRADKOV, Alexander L., 2020a. Early History of Machine Learning. *IFAC-PapersOnLine*. 1 enero 2020. Vol. 53, no. 2, p. 1385-1390. DOI 10.1016/j.ifacol.2020.12.1888.

[2] WU, Zonghan, PAN, Shirui, CHEN, Fengwen, LONG, Guodong, ZHANG, Chengqi y YU, Philip S., 2021b. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*. enero 2021. Vol. 32, no. 1, p. 4-24. DOI 10.1109/TNNLS.2020.2978386.

[3] GOODFELLOW, Ian, BENGIO, Yoshua y COURVILLE, Aaron, 2016c. *Deep learning*. Cambridge, Massachusetts: The MIT Press. Adaptive computation and machine learning. ISBN 978-0-262-03561-3. Q325.5 .G66 2016

[4] BRONSTEIN, Michael M., BRUNA, Joan, LECUN, Yann, SZLAM, Arthur y VANDERGHEYNST, Pierre, 2017d. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine*. julio 2017. Vol. 34, no. 4, p. 18-42. DOI 10.1109/MSP.2017.2693418.

[5] VOLPI, Riccardo, THAKUR, Uddhipan y MALAGÒ, Luigi, 2021e. Changing the Geometry of Representations: α-Embeddings for NLP Tasks. *Entropy*. 26 febrero 2021. Vol. 23, no. 3, p. 287. DOI 10.3390/e23030287.

[6] SANCHEZ-LENGELING, Benjamin, REIF, Emily, PEARCE, Adam y WILTSCHKO, Alex, 2021f. A Gentle Introduction to Graph Neural Networks. *Distill*. 17 agosto 2021. Vol. 6, no. 8, p. 10.23915/distill.00033. DOI 10.23915/distill.00033.

[7] SCARSELLI, F., GORI, M., AH CHUNG TSOI, HAGENBUCHNER, M. y MONFARDINI, G., 2009g. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*. enero 2009. Vol. 20, no. 1, p. 61-80. DOI 10.1109/TNN.2008.2005605.

[8] ZHANG, Muhan, CUI, Zhicheng, NEUMANN, Marion y CHEN, Yixin, sin fecha. An End-to-End Deep Learning Architecture for Graph Classification. . P. 8.

[9] QIU, Jiezhong, TANG, Jian, MA, Hao, DONG, Yuxiao, WANG, Kuansan y TANG, Jie, 2018i. DeepInf: Social Influence Prediction with Deep Learning. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 19 julio 2018. P. 2110-2119. DOI 10.1145/3219819.3220077.

[10] CS224W: Machine Learning with Graphs | 2021 | Lecture 15.1 - Deep Generative Models for Graphs - YouTube, sin fecha. [en línea]. [Accedido 16 octubre 2021 j]. Recuperado a partir de: https://www.youtube.com/watch?v=IMpkHvQ0LA4

[11] YING, Rex, HE, Ruining, CHEN, Kaifeng, EKSOMBATCHAI, Pong, HAMILTON, William L. y LESKOVEC, Jure, 2018k. Graph Convolutional Neural Networks for Web-Scale Recommender Systems. *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 19 julio 2018. P. 974-983. DOI 10.1145/3219819.3219890.

[12] BATTAGLIA, Peter W., PASCANU, Razvan, LAI, Matthew, REZENDE, Danilo y KAVUKCUOGLU, Koray, 2016l. Interaction Networks for Learning about Objects, Relations and Physics. *arXiv:1612.00222 [cs]* [en línea]. 1 diciembre 2016. [Accedido 16 octubre 2021]. Recuperado a partir de: http://arxiv.org/abs/1612.00222

[13] CORA Dataset, sin fecha. [en línea]. [Accedido 16 octubre 2021 m]. Recuperado a partir de: https://relational.fit.cvut.cz/dataset/CORA

[14] PubMed Diabetes Dataset, sin fecha. [en línea]. [Accedido 16 octubre 2021 n]. Recuperado a partir de: https://relational.fit.cvut.cz/dataset/PubMed_Diabetes

[15] Citeseer Dataset, sin fecha. [en línea]. [Accedido 16 octubre 2021 o]. Recuperado a partir de: https://relational.fit.cvut.cz/dataset/Citeseer

[16] LÓPEZ-MUÑOZ, Francisco, BOYA, Jesús y ALAMO, Cecilio, 2006p. Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal. *Brain Research Bulletin*. octubre 2006. Vol. 70, no. 4-6, p. 391-405. DOI 10.1016/j.brainresbull.2006.07.010.

[17] NEWMAN, Eric A., ARAQUE, Alfonso, DUBINSKY, Janet M., SWANSON, Larry W., KING, Lyndel Saunders y HIMMEL, Eric (eds.), 2017q. *The beautiful brain: the drawings of Santiago Ramón y Cajal*. New York: Abrams. ISBN 978-1-4197-2227-1. QM455 .B39 2017

[18] STANGOR, Charles y WALINGA, Jennifer, 2014r. 4.1 The Neuron Is the Building Block of the Nervous System. [en línea]. 17 octubre 2014. [Accedido 30 octubre 2021]. Recuperado a partir de: https://opentextbc.ca/introductiontopsychology/chapter/3-1-the-neuron-is-the-building-block-of-the-nervous-system/

[19] ZOLNIK, Timothy A. y CONNORS, Barry W., 2016s. Electrical synapses and the development of inhibitory circuits in the thalamus. *The Journal of Physiology*. 15 mayo 2016. Vol. 594, no. 10, p. 2579-2592. DOI 10.1113/JP271880.

[20] BEAN, Bruce P., 2007t. The action potential in mammalian central neurons. *Nature Reviews Neuroscience*. junio 2007. Vol. 8, no. 6, p. 451-465. DOI 10.1038/nrn2148.

[21] LETERRIER, Christophe, 2018u. The Axon Initial Segment: An Updated Viewpoint. *The Journal of Neuroscience*. 28 febrero 2018. Vol. 38, no. 9, p. 2135-2145. DOI 10.1523/JNEUROSCI.1922-17.2018.

[22] PICCININI, Gualtiero, 2004v. The First Computational Theory of Mind and

Brain: A Close Look at Mcculloch and Pitts's "Logical Calculus of Ideas Immanent in Nervous Activity". *Synthese*. 1 agosto 2004. Vol. 141, no. 2, p. 175-215. DOI 10.1023/B:SYNT.0000043018.52445.3e.

[23] Rosenblatt's Contributions, sin fecha. [en línea]. [Accedido 31 octubre 2021 w]. Recuperado a partir de: http://csis.pace.edu/~ctappert/srd2011/rosenblatt-contributions.htm

[24] STRAWN, George y STRAWN, Candace, 2016x. Masterminds of Artificial Intelligence: Marvin Minsky and Seymour Papert. *IT Professional*. noviembre 2016. Vol. 18, no. 6, p. 62-64. DOI 10.1109/MITP.2016.116.

[25] AI winter, 2021y. *Wikipedia* [en línea]. [Accedido 30 octubre 2021]. Recuperado a partir de: https://en.wikipedia.org/w/index.php?title=AI_winter&oldid=1052089413

[26] SEUNG, H. Sebastian, LEE, Daniel D., REIS, Ben Y. y TANK, David W., 2000z. The Autapse: A Simple Illustration of Short-Term Analog Memory Storage by Tuned Synaptic Feedback. *Journal of Computational Neuroscience*. 1 septiembre 2000. Vol. 9, no. 2, p. 171-185. DOI 10.1023/A:1008971908649.

[27] RUMELHART, D E, HINTON, G E y WILLIAMS, R J, sin fecha. Learning Internal Representations by Error Propagation. . P. 23.

[28] GÉRON, Aurélien, 2019ab. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems*. Second edition. Beijing [China] ; Sebastopol, CA: O'Reilly Media, Inc. ISBN 978-1-4920-3264-9. QA76.73.P98 G45 2019

[29] GORI, M., MONFARDINI, G. y SCARSELLI, F., 2005ac. A new model for learning in graph domains. En: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* julio 2005. p. 729-734 vol. 2.

[30] BRUNA, Joan, ZAREMBA, Wojciech, SZLAM, Arthur y LECUN, Yann, 2014ad. Spectral Networks and Locally Connected Networks on Graphs. *arXiv:1312.6203 [cs]* [en línea]. 21 mayo 2014. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1312.6203

[31] MICHELI, Alessio, 2009ae. Neural Network for Graphs: A Contextual Constructive Approach. *IEEE Transactions on Neural Networks*. marzo 2009. Vol. 20, no. 3, p. 498-511. DOI 10.1109/TNN.2008.2010350.

[32] DANEL, Tomasz, SPUREK, Przemysław, TABOR, Jacek, ŚMIEJA, Marek, STRUSKI, Łukasz, SŁOWIK, Agnieszka y MAZIARKA, Łukasz, 2020af. Spatial Graph Convolutional Networks. *arXiv:1909.05310 [cs, stat]* [en línea]. 2 julio 2020. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1909.05310

[33] CAI, Hongyun, ZHENG, Vincent W. y CHANG, Kevin Chen-Chuan, 2018ag. A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. *arXiv:1709.07604 [cs]* [en línea]. 2 febrero 2018. [Accedido 6 noviembre 2021]. Recuperado a partir de:

http://arxiv.org/abs/1709.07604

[34] ZHANG, Daokun, YIN, Jie, ZHU, Xingquan y ZHANG, Chengqi, 2018ah. Network Representation Learning: A Survey. *arXiv:1801.05852 [cs, stat]* [en línea]. 19 julio 2018. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1801.05852

[35] KUSNER, Matt J., PAIGE, Brooks y HERNÁNDEZ-LOBATO, José Miguel, 2017ai. Grammar Variational Autoencoder. [en línea]. 6 marzo 2017. [Accedido 6 noviembre 2021]. Recuperado a partir de: https://arxiv.org/abs/1703.01925v1

[36] KIPF, Thomas N. y WELLING, Max, 2016aj. Variational Graph Auto-Encoders. *arXiv:1611.07308 [cs, stat]* [en línea]. 21 noviembre 2016. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1611.07308

[37] SHUMAN, David I., NARANG, Sunil K., FROSSARD, Pascal, ORTEGA, Antonio y VANDERGHEYNST, Pierre, 2013ak. The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and Other Irregular Domains. *IEEE Signal Processing Magazine*. mayo 2013. Vol. 30, no. 3, p. 83-98. DOI 10.1109/MSP.2012.2235192.

[38] BATTAGLIA, Peter W., HAMRICK, Jessica B., BAPST, Victor, SANCHEZ-GONZALEZ, Alvaro, ZAMBALDI, Vinicius, MALINOWSKI, Mateusz, TACCHETTI, Andrea, RAPOSO, David, SANTORO, Adam, FAULKNER, Ryan, GULCEHRE, Caglar, SONG, Francis, BALLARD, Andrew, GILMER, Justin, DAHL, George, VASWANI, Ashish, ALLEN, Kelsey, NASH, Charles, LANGSTON, Victoria, DYER, Chris, HEESS, Nicolas, WIERSTRA, Daan, KOHLI, Pushmeet, BOTVINICK, Matt, VINYALS, Oriol, LI, Yujia y PASCANU, Razvan, 2018al. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261 [cs, stat]* [en línea]. 17 octubre 2018. [Accedido 5 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1806.01261

[39] GOYAL, Palash y FERRARA, Emilio, 2018am. Graph Embedding Techniques, Applications, and Performance: A Survey. *Knowledge-Based Systems*. julio 2018. Vol. 151, p. 78-94. DOI 10.1016/j.knosys.2018.03.022.

[40] KIPF, Thomas N. y WELLING, Max, 2017an. Semi-Supervised Classification with Graph Convolutional Networks. *arXiv:1609.02907 [cs, stat]* [en línea]. 22 febrero 2017. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1609.02907

[41] SIMONOVSKY, Martin y KOMODAKIS, Nikos, 2017ao. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. *arXiv:1704.02901 [cs]* [en línea]. 8 agosto 2017. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1704.02901

[42] GÓMEZ-BOMBARELLI, Rafael, WEI, Jennifer N., DUVENAUD, David, HERNÁNDEZ-LOBATO, José Miguel, SÁNCHEZ-LENGELING, Benjamín,

SHEBERLA, Dennis, AGUILERA-IPARRAGUIRRE, Jorge, HIRZEL, Timothy D., ADAMS, Ryan P. y ASPURU-GUZIK, Alán, 2018ap. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*. 28 febrero 2018. Vol. 4, no. 2, p. 268-276. DOI 10.1021/acscentsci.7b00572.

[43] PAN, Shirui, WU, Jia, ZHU, Xingquan, LONG, Guodong y ZHANG, Chengqi, 2017aq. Task Sensitive Feature Exploration and Learning for Multitask Graph Classification. *IEEE Transactions on Cybernetics*. marzo 2017. Vol. 47, no. 3, p. 744-758. DOI 10.1109/TCYB.2016.2526058.

[44] PAN, Shirui, WU, Jia, ZHU, Xingquan, ZHANG, Chengqi y YU, Philip S., 2016ar. Joint Structure Feature Exploration and Regularization for Multi-Task Graph Classification. *IEEE Transactions on Knowledge and Data Engineering*. marzo 2016. Vol. 28, no. 3, p. 715-728. DOI 10.1109/TKDE.2015.2492567.

[45] HUBEL, D. H., 1959as. Single unit activity in striate cortex of unrestrained cats. *The Journal of Physiology*. 2 septiembre 1959. Vol. 147, no. 2, p. 226-238. DOI 10.1113/jphysiol.1959.sp006238.

[46] HUBEL, D H y WIESEL, T N, sin fecha. RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX. . P. 18.

[47] FUKUSHIMA, Kunihiko, 1980au. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*. abril 1980. Vol. 36, no. 4, p. 193-202. DOI 10.1007/BF00344251.

[48] LECUN, Yann, BOTTOU, Leon, BENGIO, Y. y HAFFNER, Patrick, 1998av. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*. 1 diciembre 1998. Vol. 86, p. 2278-2324. DOI 10.1109/5.726791.

[49] GILMER, Justin, SCHOENHOLZ, Samuel S., RILEY, Patrick F., VINYALS, Oriol y DAHL, George E., 2017aw. Neural Message Passing for Quantum Chemistry. *arXiv:1704.01212 [cs]* [en línea]. 12 junio 2017. [Accedido 7 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1704.01212

[50] WU, Ning, ZHAO, Xin Wayne, WANG, Jingyuan y PAN, Dayan, 2020ax. Learning Effective Road Network Representation with Hierarchical Graph Neural Networks. En: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* [en línea]. New York, NY, USA: Association for Computing Machinery. 23 agosto 2020. p. 6-14. [Accedido 6 noviembre 2021]. KDD '20. ISBN 978-1-4503-7998-4. Recuperado a partir de: https://doi.org/10.1145/3394486.3403043

[51] LI, Yaguang, YU, Rose, SHAHABI, Cyrus y LIU, Yan, 2018ay. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. *arXiv:1707.01926 [cs, stat]* [en línea]. 22 febrero 2018. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1707.01926

[52] VELIČKOVIĆ, Petar, CUCURULL, Guillem, CASANOVA, Arantxa, ROMERO, Adriana, LIÒ, Pietro y BENGIO, Yoshua, 2018az. Graph Attention Networks. *arXiv:1710.10903 [cs, stat]* [en línea]. 4 febrero 2018. [Accedido 7 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1710.10903

[53] PETAR VELIČKOVIĆ, 2021ba. *Theoretical Foundations of Graph Neural Networks* [en línea]. 22 febrero 2021. [Accedido 7 noviembre 2021]. Recuperado a partir de: https://www.youtube.com/watch?v=uF53xsT7mjc

[54] HAMILTON, William L., YING, Rex y LESKOVEC, Jure, 2018bb. Inductive Representation Learning on Large Graphs. *arXiv:1706.02216 [cs, stat]* [en línea]. 10 septiembre 2018. [Accedido 6 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1706.02216

[55] SCHLICHTKRULL, Michael, KIPF, Thomas N., BLOEM, Peter, BERG, Rianne van den, TITOV, Ivan y WELLING, Max, 2017bc. Modeling Relational Data with Graph Convolutional Networks. *arXiv:1703.06103 [cs, stat]* [en línea]. 26 octubre 2017. [Accedido 7 noviembre 2021]. Recuperado a partir de: http://arxiv.org/abs/1703.06103

[56] What is NumPy? — NumPy v1.21 Manual, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bd]. Recuperado a partir de: https://numpy.org/doc/stable/user/whatisnumpy.html

[57] Why TensorFlow, sin fecha. [en línea]. [Accedido 11 diciembre 2021 be]. Recuperado a partir de: https://www.tensorflow.org/about

[58] Introduction to TensorFlow, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bf]. Recuperado a partir de: https://www.tensorflow.org/learn

[59] Spektral, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bg]. Recuperado a partir de: https://graphneural.network/

[60] PyTorch, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bh]. Recuperado a partir de: https://www.pytorch.org

[61] scikit-learn: machine learning in Python — scikit-learn 1.0.2 documentation, sin fecha. [en línea]. [Accedido 29 diciembre 2021 bi]. Recuperado a partir de: https://scikit-learn.org/stable/

[62] pandas - Python Data Analysis Library, sin fecha. [en línea]. [Accedido 29 diciembre 2021 bj]. Recuperado a partir de: https://pandas.pydata.org/

[63] TensorBoard | TensorFlow, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bk]. Recuperado a partir de: https://www.tensorflow.org/tensorboard

[64] Models - Spektral, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bl]. Recuperado a partir de: https://graphneural.network/models/

[65] PAN, Liming, SHI, Cheng y DOKMANIĆ, Ivan, 2021bm. Neural Link Prediction with Walk Pooling. *arXiv:2110.04375 [cs]* [en línea]. 8 octubre 2021. [Accedido 5 diciembre 2021]. Recuperado a partir de: http://arxiv.org/abs/2110.04375

[66] SHI, Cheng, 2021bn. *WalkPooling* [en línea]. Python. [Accedido 11 diciembre 2021]. Recuperado a partir de: https://github.com/DaDaCheng/WalkPooling

[67] PyG Documentation — pytorch_geometric 2.0.2 documentation, sin fecha. [en línea]. [Accedido 11 diciembre 2021 bo]. Recuperado a partir de: https://pytorch-geometric.readthedocs.io/en/latest/

[68] EN:USER:CBURNETT, 2019bp. *Español: Esquema de red neuronal* [en línea]. 7 mayo 2019. [Accedido 27 diciembre 2021]. Recuperado a partir de: https://commons.wikimedia.org/wiki/File:Colored_neural_network_es.svgFile:Colored neural network uk.svg

[69] CAMPUS, UC Regents Davis, sin fecha. *English: SMI32-stained pyramidal neurons in cerebral cortex.* [en línea]. [Accedido 27 diciembre 2021 bq]. Recuperado a partir de: https://commons.wikimedia.org/wiki/File:Smi32neuron.jpghttp://brainmaps.org

[70] File:Complete neuron cell diagram en.svg, 2010br. *Wikipedia* [en línea]. [Accedido 27 diciembre 2021]. Recuperado a partir de: https://en.wikipedia.org/w/index.php?title=File:Complete_neuron_cell_diagram_en.svg&oldid=370915184

6. Anexos

1. Cuaderno de código *tfg-gnn-alfonsomoure.ipynb*: tal y como se ha mencionado en distintas partes de esta memoria, se anexa un *notebook* de Jupyter con el código utilizado para realizar las pruebas. El mismo cuaderno incluye, además, las diferentes secuencias necesarias para instalar los módulos utilizados.

2. Cuaderno de código *featureselectionexp3.ipynb*: dedicado a labúsqueda de la combinación de columnas más adecuada para el experimento 3 del apartado 2.4.3.3.

3. Repositorio en Github con el código de las implementaciones: https://github.com/ghostmou/uoc-tfg-gnn.