Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**

Grado en Ingeniería Informática

Trabajo Fin de Grado

Optimización y Evaluación de Graph Neural Networks para la Predicción de Efectos Secundarios en el Conjunto de Datos ogbl-ddi

Autor: David Cano Rosillo
Tutor: Damiano Zanardini

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Optimización y Evaluación de Graph Neural Networks para la Predicción de Efectos Secundarios en el Conjunto de Datos ogbl-ddi

Junio 2024

Autor: David Cano Rosillo
Tutor: Damiano Zanardini
Departamento de Inteligencia Artificial
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Este trabajo se centra en las Graph Neural Networks (GNN), una clase de redes neuronales capaces de modelar eficazmente problemas cuya estructura subyacente se representa como un grafo. En este estudio, se aplican las GNN al descubrimiento de efectos secundarios de medicamentos utilizando el conjunto de datos ogbl-ddi. Para ello, se realiza una exhaustiva revisión del estado del arte, implementando las mejores variantes y las técnicas más avanzadas en predicción de aristas. Además, se emplean técnicas de vanguardia durante el entrenamiento, como la optimización bayesiana para la búsqueda de hiperparámetros. Finalmente, se analiza en profundidad el comportamiento de la métrica Hits@K y se proponen nuevas líneas de investigación basadas en las conclusiones obtenidas.

Abstract

This work focuses on Graph Neural Networks (GNN), a class of neural networks capable of effectively modeling problems whose underlying structure is represented as a graph. In this study, GNN are applied to the discovery of drug side effects using the ogbl-ddi dataset. A comprehensive review of the state of the art is conducted, implementing the best variants and the latest techniques in edge prediction. Additionally, cutting-edge techniques such as Bayesian optimization for hyperparameter tuning are employed during training. Finally, the behavior of the Hits@K metric is thoroughly analyzed, and new research directions are proposed based on the conclusions drawn.

Tabla de contenidos

Capítulo 1

Introducción

1.1. Contexto y Motivación

Actualmente, el descubrimiento de efectos secundarios de medicamentos es un área crítica en la farmacología, por su impacto en la salud pública y el desarrollo de fármacos. Los grafos, como modelos de datos, permiten representar relaciones complejas entre entidades, lo que los hace especialmente útiles para este tipo de problemas. Los avances recientes en Graph Neural Networks (GNN) han demostrado su capacidad para manejar datos estructurados, presentando una oportunidad única para mejorar el análisis y la predicción de efectos secundarios en medicamentos.

1.2. Objetivos

Este trabajo tiene como objetivo principal aplicar Graph Neural Networks (GNN) al descubrimiento de efectos secundarios de medicamentos utilizando el conjunto de datos ogbl-ddi. Los objetivos específicos incluyen:

- Realizar una exhaustiva revisión del estado del arte de las GNN.
- Implementar y comparar varias variantes de GNN para la predicción de aristas.
- Utilizar técnicas avanzadas de optimización, como la optimización bayesiana, para mejorar el rendimiento de los modelos.
- Analizar en profundidad el comportamiento de la métrica Hits@K y proponer nuevas líneas de investigación basadas en los resultados obtenidos.

1.3. Graph Neural Networks

Las Graph Neural Networks (GNN) son una clase de redes neuronales diseñadas para trabajar con datos estructurados en forma de grafos. Estas redes extienden las capacidades de las redes neuronales tradicionales al permitir la modelación de relaciones y dependencias complejas entre entidades representadas como nodos y aristas. Las GNN han sido aplicadas a diversos campos, desde la generación de tácticas de fútbol [1] al descubrimiento de nuevos materiales [2], pasando por la recomendación de vídeos cortos [3].

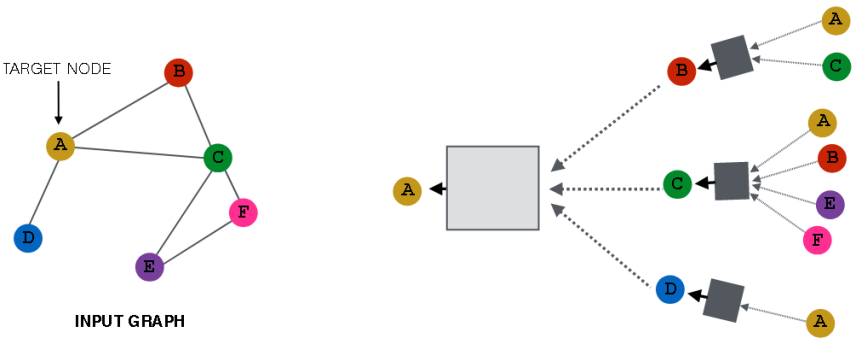


Figura 1.1: Paso de mensajes en una GNN con dos capas. Las representaciones del vecindario se agregan repetidamente para obtener las representaciones finales de cada nodo.

El núcleo del funcionamiento de las GNN radica en el mecanismo de propagación de mensajes, donde la información de un nodo se actualiza mediante la agregación de información de sus nodos vecinos. Este proceso se realiza en múltiples capas, permitiendo que la información se propague a través del grafo y se capture la estructura global.

- **Inicialización**: Cada nodo comienza con una representación inicial, que puede ser una característica específica del nodo, como un vector de características. Estas representaciones iniciales se denotan como h_v para cada nodo v. Estas representaciones también son referidas por su término en inglés «embedding».
- **Agregación de Mensajes**: En cada capa k, cada nodo v recibe mensajes de nodos en su vecindario $\mathcal{N}_v$. Estos mensajes se agregan mediante una función de agregación, que puede ser una suma, media o una operación más compleja, sea m_v el mensaje recibido por el nodo v, este se calcula así:

$$m_v = \text{AGGREGATE}\left(\{h_u : u \in \mathcal{N}_v\}\right) \tag{1.1}$$

- **Actualización del Nodo**: Después de la agregación, el nodo actualiza su representación h_v combinando el mensaje agregado con su propia representación previa mediante una función de actualización, que generalmente es una red neuronal:

$$h_v = \text{UPDATE}(h_v, m_v) \tag{1.2}$$

- **Iteración**: Este proceso de agregación y actualización se repite durante varias capas, permitiendo que la información fluya a través del grafo y las representaciones de los nodos capturen las dependencias y relaciones complejas en el grafo.

Una vez aprendidas las representaciones de cada nodo, estas se pueden utilizar para diversas tareas sobre grafos:

- **Clasificación de Nodos:** Usando directamente la representación del nodo h_v. Por ejemplo, el tema de un artículo en función de sus citaciones en un grafo de citación.
- **Predicción de Aristas:** Usando las representaciones h_u y h_v para predecir la existencia de una arista (u, v). Este es el método usado por los modelos implementados en este trabajo.

- **Clasificación de grafos:** Agregando todas las representaciones de los nodos del grafo. Una posible aplicación sería clasificar grafos que representen moléculas o proteínas.

La tarea de este trabajo es la predicción de aristas en el grafo ogbl-ddi. En este grafo los nodos representan medicamentos, siendo las aristas la existencia de un efecto secundario entre dos medicamentos. Por lo tanto, al predecir una arista en este grafo estaremos prediciendo si tomar dos medicamentos conjuntamente generará un efecto secundario.

1.4. Estructura de la Memoria

La estructura de este documento se organiza de la siguiente manera:

- **Capítulo 1: Introducción**: Presenta el contexto, los objetivos, una introducción a las GNN y la estructura del trabajo.
- **Capítulo 2: Estado del Arte**: Revisión exhaustiva de las técnicas y modelos más avanzados en el campo de las GNN.
- **Capítulo 3: Técnicas Aplicadas**: Descripción de las técnicas utilizadas en el desarrollo y entrenamiento de los modelos GNN.
- **Capítulo 4: Contexto**: Detalles sobre el conjunto de datos utilizado, las métricas de evaluación y las tecnologías empleadas.
- **Capítulo 5: Resultados**: Presentación y análisis de los resultados obtenidos.
- **Capítulo 6: Conclusiones**: Resumen de los hallazgos y su importancia.
- **Capítulo 7: Líneas Futuras**: Propuestas para futuras investigaciones basadas en las conclusiones del trabajo.
- **Capítulo 8: Análisis de Impacto**: Evaluación del impacto potencial de los resultados y su relación con los objetivos de desarrollo sostenible de la Unión Europea.

Capítulo 2

Estado del arte

La revisión del estado del arte se ha centrado en los métodos con mejores métricas en el conjunto de datos ogbl-ddi [4]. En la página de Open Graph Benchmark [5] mantienen un ranking actualizado con los mejores métodos. En la medida que las limitaciones computacionales han permitido, ya que algunas variantes no caben en la memoria de la GPU usada, se han reproducido los métodos más exitosos.

2.1. Graph Convolutional Networks

Las Graph Convolutional Networks (GCN) [6] son una variante de las GNN que implementan una agregación con normalización de los embeddings en el vecindario. La agregación es la siguiente, sea h_i el embedding del nodo a actualizar y h_j el embedding de un vecino en $\mathcal{N}_i$, el vecindario del nodo i,

$$h_i = \sum_{j \in \mathcal{N}_i} \frac{1}{\sqrt{\deg(i) * \deg(j)}} W h_j \tag{2.1}$$

donde $deg(i)$ y $deg(j)$ son los grados de los nodos i y j. La red aprenderá la matriz W, con la cual actualizará todos los embeddings. La normalización usada, dividir por $\sqrt{\deg(i) * \deg(j)}$, reduce el peso de los vecinos con más grado. Esto es útil, ya que estos vecinos suelen tener valores muy altos al haber sumado sus numerosos vecinos.

Las GCN pueden computar el embedding de todos los nodos eficientemente mediante operaciones por matrices. Esta cualidad es necesaria para una implementación eficiente que haga uso de la rápida multiplicación de matrices de los ordenadores.

2.2. GraphSAGE

Mientras que las GCN usaban todos los vecinos de un nodo para calcular el nuevo embedding de ese nodo, GraphSAGE (SAmple and aggreGatE) [7] solo usa una parte del vecindario. Para ello primero muestrea un subconjunto de los vecinos $\mathcal{N}_i$ y luego aplica la siguiente fórmula,

$$h_i = W \cdot CONCAT(h_i, AGGREGATE(\{h_j | j \in \mathcal{N}_i\})) \tag{2.2}$$

Donde $CONCAT$ es una función de concatenación y $AGGREGATE$ alguna función que agregue los embeddings de los vecinos. Para la función $AGGREGATE$ hay varias opciones, por ejemplo, la suma, la media, el máximo... Cada una tiene sus ventajas, por ejemplo, con la media ganaremos estabilidad, ya que no importara el número de vecinos. Sin embargo, si elegimos la suma como función de agregación, la red podrá saber el número de vecinos en base al embedding recibido. Por ejemplo, un nodo con más vecinos tendrá un valor alto, mientras que uno con un grado muy pequeño tendrá un valor más reducido.

Además, proponen normalizar el embedding obtenido por su norma $L2$ para mantener la estabilidad numérica. Por lo tanto, después de la aplicación de la fórmula 2.2 se aplicaría la siguiente división,

$$h_i = h_i / ||h_i||_2 \tag{2.3}$$

2.3. Graph Attention Networks

Las Graph Attention Networks (GAT) [8] incorporan el mecanismo de atención sobre los embeddings del vecindario del nodo a computar. Estos coeficientes de atención permiten a la red aumentar la influencia de un nodo vecino en concreto para el cálculo de un embedding h_i. Esto permite a la red prestar «atención» a aquellos embeddings que considera más importantes. Sea α_{ij} el peso que le da el nodo i al vecino j, este se calcula mediante:

$$\alpha_{ij} = exp(LeakyReLU(a(Wh_i, Wh_j))) \tag{2.4}$$

Donde W es una matriz que convierte el vector h_i a un vector de dimensión R^F y a una función $a : R^F \times R^F \mapsto R$. En este caso los autores usan un MLP como a. Por lo tanto, los coeficientes de atención se calculan multiplicando ambos embeddings por W, concatenándolos, pasándolos por a para después aplicarles una función de activación LeakyReLU y exponenciarlos. A continuación, se normalizan los coeficientes por la suma de todos los demás coeficientes del vecindario del nodo en cuestión i:

$$\alpha_{ij} = \frac{\alpha_{ij}}{\sum_{j \in \mathcal{N}_j} \alpha_{ij}} \tag{2.5}$$

A continuación, se le aplica una matriz aprendida W a cada embedding para después agregarlos pesándolos por sus coeficientes. Finalmente, se aplica una función de activación σ, a menudo una ReLU, para obtener la representación final h_i.

$$h_i = \sigma(\sum_{j \in \mathcal{N}_j} \alpha_{ij} W h_j) \tag{2.6}$$

En la figura 2.1 se puede ver un diagrama del proceso. A la izquierda se muestra cómo se obtiene el coeficiente α_{ij} mediante la función a. A la derecha se muestra cómo se agregan los embeddings del vecindario pesándolos por sus coeficientes de atención. Un detalle

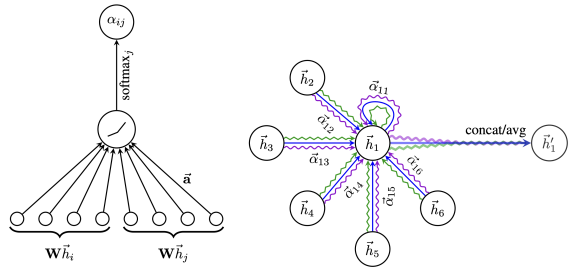


Figura 2.1: Cálculo de los coeficientes de atención y agregación pesada de los embeddings en la arquitectura GAT

interesante es que el propio embedding sin actualizar, h_i, se usa en la agregación. Esto es equivalente a añadir una arista de cada nodo a sí mismo, transformando la matriz de adyacencia A a $\hat{A} = A + I_n$ donde I_n es la matriz de identidad del grafo con n nodos.

2.4. Network in Graph Neural Network

La propuesta de [9], Network in Graph Neural Network (NGNN) busca aliviar el problema conocido como «oversmoothing». El oversmoothing ocurre cuando se utilizan demasiadas capas GNN seguidas. Los embeddings de todos los nodos van convergiendo a un mismo valor debido a utilizar el paso de mensajes muchas veces. Estos embeddings son muy parecidos para zonas lejanas del grafo, y, por lo tanto, no reflejan información sobre el entorno del nodo correctamente. Esto anula el propósito de las GNN, que buscan incorporar información local sobre el vecindario mediante el paso de mensajes.

Por lo tanto, con las GNN no se puede seguir la exitosa práctica de simplemente hacer las redes más profundas. Para aumentar la capacidad de las GNN aliviando el oversmoothing se propone NGNN. La propuesta consiste en utilizar capas Multilayer Perceptron (MLP) entre capas de GNN. Estas capas MLP no utilizan paso de mensajes, por lo tanto, no resultarán en oversmoothing. Los autores demuestran empíricamente que esta modificación, que es agnóstica al tipo de capa GNN, mejora los resultados tanto para GraphSAGE como GAT en diversos conjuntos de datos como ogbl-ddi. Los embeddings de cada vecino se actualizarían de la siguiente manera para una GAT modificada:

$$h_i = GAT(MLP(GAT(h_i))) \tag{2.7}$$

No se necesitan muchas capas MLP intercaladas entre GAT para obtener buenos resultados. Concretamente en ogbl-ddi los autores proponen usar solo una, como en la ecuación 2.7. Es importante destacar que las capas MLP no tienen acceso a la información de los vecinos y solo modifican el propio embedding h_i.

2.5. GraphSAGE + Edge Attributes

De momento, la información de las aristas solo se ha usado implícitamente, permitiendo el flujo de información entre un nodo y sus vecinos durante el paso de mensajes. En [10] se propone incorporar embeddings en las aristas a la red con información sobre estadísticos.

Para ello usan la técnica primero definida por [11]. Esta técnica consiste en elegir unos nodos aleatorios del subgrafo de entrenamiento, denotados como a_i. Una vez escogidos estos nodos, a los que los autores se refieren como «nodos ancla», se calcula la longitud del camino más corto de cada nodo u a cada nodo ancla a_i. Con esto se obtiene un vector $d_{u,A}$ de la distancia del nodo u a cada uno de los nodos anclas a_i. A continuación, se define la distancia de una arista (u, v) a los nodos anclas sumando las distancias de los nodos u y v a los nodos ancla:

$$d_{u,v} = d_{u,A} + d_{v,A} \tag{2.8}$$

Después los autores incorporan esta información a la red, previamente multiplicándola por una matriz aprendida X. Con esto se obtiene el embedding $h_{u,v}$ asociado a la arista (u, v).

$$h_{u,v} = X \cdot d_{u,v} \tag{2.9}$$

Finalmente, modifican la arquitectura de GraphSAGE para hacer uso de los vectores generados en cada arista. Sean W_0, W_1 y W_2 matrices aprendidas por la red, el embedding h_v se calcula como:

$$h_v = \sigma(W_0 h_v + W_1 mean_{u \in \mathcal{N}(v)}(h_u + W_2 h_{u,v})) \tag{2.10}$$

Donde σ es una función de activación como ReLU. En la ecuación 2.10 se incorpora la información de las aristas de cada nodo del vecindario con $mean_{u \in \mathcal{N}(v)}(h_u + W_2 h_{u,v})$. Curiosamente, la información de la arista se suma a la del embedding del nodo antes de agregar, en vez de ir por separado. También se le da mucha importancia al embedding sin actualizar h_v, al que se le dedica la transformación aprendida W_0.

2.6. Pairwise Learning for Neural Link Prediction

La aportación de [12] consiste en reemplazar la función de pérdida usada. Para un problema de clasificación, en nuestro caso la clasificación de si una arista existe, se suele usar la entropía cruzada binaria (BCE). Sean y las etiquetas reales (0 si la arista no existe y 1 si existe) e $\hat{y}$ las predicciones del modelo:

$$BCE(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}) + (1 - y_i) \log(1 - \hat{y})] \tag{2.11}$$

Esta función busca que se etiqueten aristas que existen como 1 y aristas que no existen como 0. Sin embargo, citando a [12]:

> La mayoría de las tareas de predicción de enlaces no tienen como objetivo etiquetar pares positivos como 1 y pares negativos como 0, sino que requieren clasificar los pares positivos por encima de los pares negativos en términos de ranking.

De hecho, las métricas con las que se mide el rendimiento de estos modelos son AUC e Hits@K. A estas dos métricas les da igual si la predicción es 0 o 1, solamente les importa que las predicciones de aristas que no existen estén por debajo de las predicciones de las aristas que sí existen. Lo ideal sería usar AUC como función de pérdida de la red, para que la red la optimizase directamente. Pero AUC no es diferenciable, así que hay que buscar una alternativa que se comporte de manera parecida durante la optimización.

Los autores proponen la siguiente fórmula. Sea $(v_i, v_j) \in E$ una arista positiva y $(v_i, v_k) \in E^-$ una arista negativa (se suele llamar a las aristas que no existen aristas negativas) y nuestra función que asigna valores f_θ, se define la función sustituta O_{AUC} como,

$$O_{AUC} = (1 - f_\theta(v_i, v_j) + f_\theta(v_i, v_k))^2 = (1 - pred\_pos + pred\_neg)^2 \tag{2.12}$$

Donde $pred\_pos$ son las predicciones positivas y $pred\_neg$ las negativas. Para ganar una intuición sobre cómo se comporta podemos comprobar algunos casos. Por ejemplo, si la predicción positiva, $f_\theta(v_i, v_j)$, es 1 y la negativa, $f_\theta(v_i, v_k)$, es 0 el resultado será la penalización mínima, 0. De manera más general, la función es 0 cuando la predicción positiva es mayor que la negativa por 1. Los autores destacan que esta función maximiza aproximadamente la métrica AUC. Por lo tanto, el modelo entrenado buscará que las predicciones positivas sean mayores que las negativas.

¿Por qué poner el margen en 1 y no dejar que la distancia entre las positivas y las negativas sea mayor? Realmente los autores prueban más funciones de pérdida aparte de la descrita en 2.12. Por ejemplo, prueban una que el margen sea mayor que uno, y otra que no fija ningún margen. Simplemente encuentran que en el conjunto de datos ogbl-ddi la que mejor funciona es la ecuación 2.12.

2.7. Ensemble Learning for Graph Neural Networks

En [13] se propone hacer una media pesada de la predicción de varios modelos para así mejorar las predicciones finales. Para determinar por cuanto pesar cada modelo se usan los pesos que mejor resultado dan en el conjunto de validación. Por lo tanto, se busca un vector $X \in R^N$ de pesos con el que pesar cada una de las predicciones $preds_i$ de los N modelos. Sean $preds$ las predicciones finales, estas se obtendrían mediante la siguiente media pesada,

$$preds = \sum X_i * preds_i \tag{2.13}$$

Como existen muchas configuraciones posibles, los autores proponen buscar el vector de pesos X con optimización bayesiana. Los algoritmos de optimización bayesiana, en este caso usan Tree-Structured Parzen Estimator (TPE), permiten optimizar cualquier función, ya que no requieren de conocer la derivada. En la sección 3.4.1 se describe con más detalle este algoritmo.

Actualmente, para la predicción de aristas para identificación de efectos secundarios en el conjunto ogbl-ddi, el estado del arte es este método. Los autores utilizan tanto una media con el mismo peso a todos los modelos como una con los pesos aprendidos mediante TPE.

Ambas técnicas obtienen buenos resultados, pero conducen un estudio de ablación en el que se encuentra superior el uso de TPE.

Capítulo 3

Técnicas aplicadas

En este capítulo se discuten algunas técnicas de entrenamiento usadas por todos los autores mencionados en el capítulo de estado del arte. Estas técnicas son comunes al Deep Learning y se pueden usar independientemente de la arquitectura que se esté entrenando.

3.1. Gradient clipping

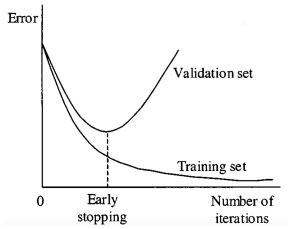
Esta técnica busca mejorar la estabilidad de los gradientes durante el entrenamiento. El gradiente es la derivada de los pesos con respecto a la función de pérdida y determina qué partes de la red deben actualizarse más durante el entrenamiento. El gradiente debe mantenerse dentro de un rango razonable: si es demasiado pequeño, la red no aprenderá; si es demasiado grande, no convergerá. En [14], los autores exploraron cómo lidiar con este problema durante el entrenamiento de redes neuronales recurrentes y encontraron que limitar el gradiente puede ser de ayuda.

La técnica de «gradient clipping» limita la norma del gradiente a un máximo para evitar que se incremente excesivamente durante el entrenamiento. En el entrenamiento de las GNN, generalmente se opta por limitar el gradiente a una norma de 1. Esta técnica demuestra ser bastante efectiva en la práctica, sin tener ningún inconveniente en términos de eficiencia de cómputo.

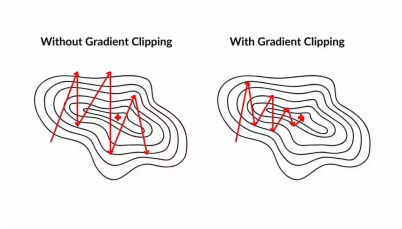
3.2. Inicialización de pesos

La inicialización de los pesos de una red neuronal tiene consecuencias en su rendimiento y velocidad de convergencia. Los autores de las GNN implementadas usan la inicialización de Xavier [15]. Esta inicializa los pesos de la red siguiendo una normal definida como $\mathcal{N}(0, \sigma^2)$ donde σ, la desviación estándar, es calculada basándonos en el perceptrón inicializado de la siguiente forma,

$$\sigma = \sqrt{\frac{2}{fan\_in + fan\_out}} \tag{3.1}$$



(a) Visualización de early stopping.



(b) Visualización de gradient clipping.

Figura 3.1: Visualización de algunas de las técnicas aplicadas.

Donde fan_in es el número de entradas del perceptrón y fan_out el número de perceptrones a los que se propaga el resultado. Por lo tanto, la varianza de los pesos es inversamente proporcional a la conectividad del perceptrón. Esta técnica ayuda a mantener la varianza de los gradientes más constante a través de las capas.

3.3. Early stopping

El early stopping consiste en parar el entrenamiento de la red neuronal en cuanto el rendimiento en el conjunto de validación empiece a bajar. Esta técnica es una manera de aplicar regularización a la red, parando el entrenamiento antes de que se produzca el sobreajuste de la red a los datos de entrenamiento.

Se puede configurar cierta paciencia para que adicionalmente espere un número de épocas desde que empieza a bajar el rendimiento por si mejorase la validación al final. Si no se mejora, se restaurarán los pesos que mejor métricas obtuvieron en validación. De esta manera se asegura que los pesos finales de la red son los que mejor rendimiento tuvieron en validación.

La razón por la que la curva de validación suele tener la forma descrita en 3.1, bajando hasta que en cierto punto entrenar más solo empeora, tiene que ver con los patrones aprendidos. Al principio la red aprende los patrones más generales, que generalizan bien al conjunto de validación, mientras que más tarde aprende las particularidades del conjunto de train. Que primero se aprendan los patrones que generalizan mejor no es una casualidad: se aprenden primero porque estos reducen más (en el conjunto de train) la función de pérdida que el propio ruido del conjunto de train.

3.4. Optimización de hiperparámetros

El entrenamiento de una red tiene muchos hiperparámetros como por ejemplo learning rate, número de capas o tamaño de cada capa. Todos estos hiperparámetros son parámetros que no pueden ser optimizados directamente durante el entrenamiento, ya que no hay una derivada asociada a ellos con la que aplicar descenso de gradiente (el método por el que aprenden las redes neuronales).

En este contexto en el que no se sabe que hiperparámetros son los mejores, se puede buscar una configuración que de buenos resultados de diversas maneras. Una alternativa popular es el «grid search» que explora una «rejilla» predefinida de configuraciones de hiperpa-

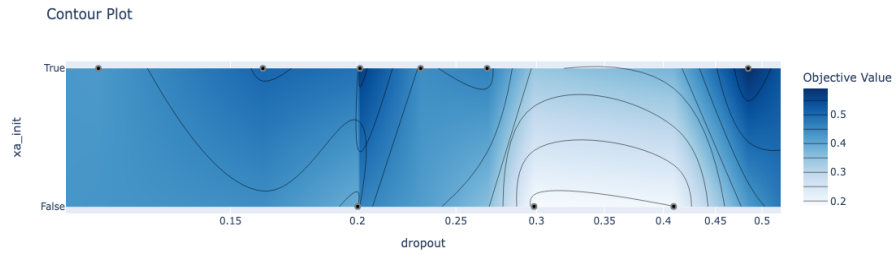


Figura 3.2: Distribución modelada durante el entrenamiento del rendimiento en función de dos hiperparámetros. Se pintan con un color más oscuro las regiones que obtienen mejores resultados.

rámetros y devuelve la que mejor rendimiento obtiene en validación. Esta alternativa, si se predefine una rejilla lo suficientemente exhaustiva, devolverá un resultado muy bueno. Sin embargo, en la realidad no se tiene el cómputo necesario para explorar una rejilla muy grande, por lo que es interesante considerar opciones que hagan una búsqueda más inteligente.

3.4.1. Optimización bayesiana

La idea detrás de la optimización bayesiana es utilizar la información de qué configuraciones han tenido buenos resultados para dedicar más tiempo de cómputo a explorar regiones prometedoras. Esta técnica es muy atractiva de cara a entrenar redes neuronales, donde probar una configuración lleva mucho tiempo y existen muchos hiperparámetros a optimizar.

A nivel de implementación se puede usar la librería Optuna [16]. Esta permite utilizar diversos algoritmos de optimización bayesiana, como por ejemplo el algoritmo Tree-Structured Parzen Estimator (TPE) [17]. Para una exposición a fondo se recomienda [18] donde se investiga el comportamiento de los distintos parámetros de TPE, este es un resumen de su funcionamiento:

1. Evalúa una x_i configuración de hiperparámetros.
2. Particiona el histórico de configuraciones previas D en D_g y D_b, dependiendo de su rendimiento. Siendo D_g las mejores y D_b las peores.
3. Modeliza la probabilidad de que una configuración pertenezca al mejor conjunto $p(x|D_g)$, o al peor $p(x|D_b)$, con alguna técnica (e.g. KDE, gaussianas...).
4. Escoge la configuración a probar x_{i+1} que maximice la función de adquisición definida como:

$$r(x|D) := \frac{p(x|D_g)}{p(x|D_b)} \tag{3.2}$$

Las distribuciones $p(x|D_g)$ y $p(x|D_b)$ se suelen estimar mediante Kernel Density Estimation (KDE) con kernels gaussianos. Además, se suelen pesar las configuraciones en función de su antigüedad para dar más importancia en explorar las regiones descubiertas recientemente.

La librería Optuna [16] permite también descartar algunas configuraciones durante la optimización, basándose en sus valores intermedios. De esta manera se puede parar el entrenamiento de una red neuronal si desde el principio se observan resultados por debajo de la media en validación. Esta es quizás la mayor ventaja de la optimización bayesiana con Optuna, permitiendo probar muchas configuraciones en muy poco tiempo. Aunque esta reducción de cómputo no es gratuita, conlleva una suposición fuerte que no siempre se cumple. La suposición es que todas las redes convergen a la misma velocidad, por lo que una red mala al principio seguirá siendo mala en una época más avanzada.

Capítulo 4

Contexto

4.1. Conjunto de datos

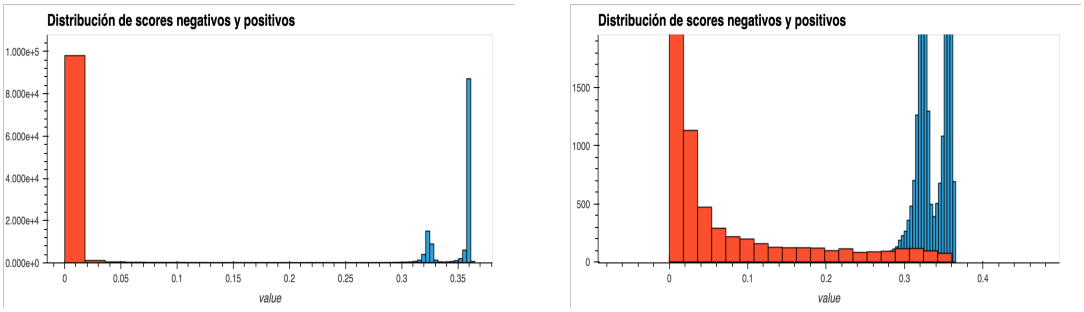
El conjunto de datos [4] consiste en un grafo no dirigido de 4267 nodos y 1334889 aristas. Cada uno de estos nodos representa un medicamento y las aristas efectos secundarios entre ellos. Por lo tanto, dos medicamentos estarán conectados por una arista si tomarlos a la vez genera efectos secundarios.

Este conjunto de datos solo tiene información estructural sobre el grafo, no contiene información sobre el tipo de medicamento ni sobre el tipo de efecto secundario generado. Esta propiedad lo hace interesante para estudiar la efectividad de las GNN, ya que el rendimiento dependerá solo de lo bien que se extraiga la información estructural. Intuitivamente, podemos imaginar a la red aprendiendo relaciones transitivas o más complejas durante el entrenamiento, que permiten descubrir nuevos efectos secundarios no descubiertos antes.

Para poder comparar la efectividad de los modelos, Open Graph Benchmark [19] proporciona una partición predeterminada del conjunto de entrenamiento, validación y test. Los resultados proporcionados deberán darse en test, habiendo usado solo el conjunto de entrenamiento y validación para entrenar la red y estimar los hiperparámetros. Esta partición está hecha basándose en las proteínas a las que afectan cada medicamento, evitando que haya medicamentos de la misma familia tanto en validación como en test. Esta característica hace las cosas considerablemente más difíciles porque la parte del grafo en validación no se comporta igual que la parte de test. Esto genera un cambio de distribución que puede resultar en una pérdida de correlación entre los resultados en validación y test. En el apartado de resultados se explora más este problema que será clave en determinar qué arquitecturas implementadas funcionan mejor.

4.2. Métrica de evaluación

Cuando evaluamos el modelo esperamos que las aristas negativas (i.e. aristas que no existen) tengan una probabilidad menor asignada por nuestro modelo que las aristas positivas. Esta intuición es reflejada por la métrica usada para evaluar su rendimiento, $Hits@k$, que para un conjunto de aristas positivas $\mathcal{K}_{pos}$ y de aristas negativas $\mathcal{K}_{neg}$ se define como:



(a) Histograma de probabilidades sin ampliar (b) Histograma ampliado en la intersección

Figura 4.1: Visualización de las probabilidades asignadas por un modelo. La distribución de las aristas negativas (en rojo), está superpuesta con la de las aristas positivas (en azul). Como podemos observar, la distribución de negativos tiene una larga cola.

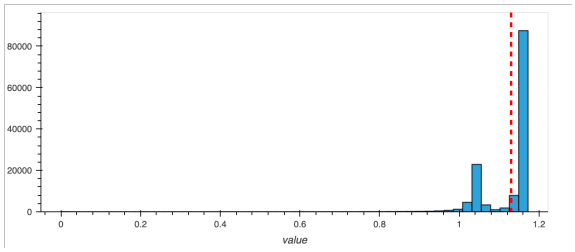


Figura 4.2: En azul se muestra la distribución de probabilidades asignadas a aristas positivas. La línea discontinua indica el límite que establecería el vigésimo error más alto de las aristas negativas.

$$Hits@k = \frac{|\{pos \in \mathcal{K}_{pos} \mid model(pos) \geq top(k, \mathcal{K}_{neg})\}|}{|\mathcal{K}_{pos}|} \tag{4.1}$$

Donde $top(k, \mathcal{K}_{neg})$ es una función que devuelve la k-ésima probabilidad más grande asignada a una arista negativa y $model(pos)$ la probabilidad asignada por el modelo a la arista positiva pos. $top(k, \mathcal{K}_{neg})$ es la vigésima arista negativa en la que el modelo se equivoca más. Por lo tanto, $Hits@20$ será el porcentaje de aristas positivas con una probabilidad asignada mayor a la vigésima probabilidad más grande asignada a una arista negativa. Esta métrica es mucho más estricta que el AUC porque el modelo necesita puntuar consistentemente los bordes positivos más alto que a casi todos los bordes negativos.

Esta métrica se puede entender mejor describiendo el algoritmo para calcularla:

1. Asignar una probabilidad a las aristas negativas $\mathcal{K}_{neg}$ y positivas $\mathcal{K}_{pos}$.
2. Seleccionar la k-ésima mayor probabilidad $top(k, \mathcal{K}_{neg})$, de todas las asignadas a aristas negativas.
3. Calcular el porcentaje de positivos mayores que $top(k, \mathcal{K}_{neg})$ respecto al número total de positivos.

Por lo tanto, un Hits@20 de 50 % significa que el 50 % de las aristas positivas tienen una probabilidad asignada mayor que la vigésima probabilidad mayor asignada a una arista

negativa.

En la figura 5.3 se muestran las distribuciones de predicciones asociadas a aristas positivas y negativas, en azul y rojo respectivamente. Entender cómo se comporta esta métrica es muy importante porque es la establecida para decidir qué modelos son mejores que otros. Las arquitecturas que se ha decidido implementar en este trabajo son las que mejor puntuación consiguen en esta métrica, y que figuran en lo alto del ranking de Open Graph Benchmark.

4.3. Tecnologías usadas

El uso de librerías en el entrenamiento de modelos de Deep Learning resulta clave para una implementación eficiente. Algunas librerías clásicas son Pytorch [20] y Tensorflow [21], estas se encargan principalmente de la propagación de los errores durante el entrenamiento. Además, contienen implementaciones del estado del arte y son actualizadas cada muy poco tiempo. De las dos librerías, Pytorch permite más flexibilidad especificando la funcionalidad a un nivel más bajo. Se ha elegido Pytorch para tener más control y un conocimiento más profundo de las arquitecturas al tener que implementarlas desde bloques más elementales.

Por la particularidad de las GNN de actualizar los valores según los vecinos, se requiere una librería que implemente esta operación eficientemente usando la GPU. Una alternativa con una sintaxis parecida a Pytorch y compatible con este es Pytorch Geometric [22]. Esta librería permite usar capas de GNN populares como GAT o GCN. Además, permite crear nuevas capas, solo teniendo que definir la agregación de los vecinos. Esta última funcionalidad fue necesaria para implementar la agregación con información de las aristas, según propone [10].

Capítulo 5

Resultados

En este capítulo se describen los resultados de los modelos probados. Durante el capítulo de estado del arte se describen en profundidad los modelos implementados. A continuación, se enumeran los modelos usados, escribiéndose entre paréntesis el acrónimo que se usará para referirse al modelo en las tablas y discusión.

- Graph Convolutional Network [6] (GCN)
- GraphSAGE [7]
- Graph Attention Network [8] (GAT)
- Network in Graph Neural Network [9] (NGNN)
- GraphSAGE + Edge attributes [10] (EDGE)
- Pairwise Learning for Neural Link Prediction [12] (AUC_LOSS)

5.1. Resultados de cada modelo

Todos los modelos se han entrenado usando early stopping, siendo la puntuación en el conjunto de test reportado el obtenido al usar el estado del modelo que mejor puntuación obtiene en validación.

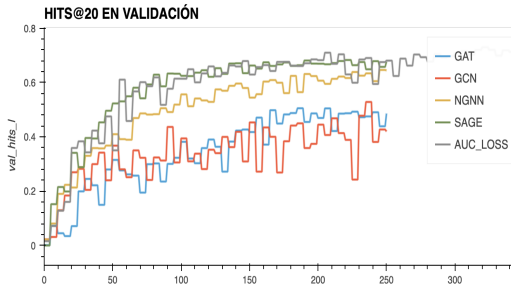
El entrenamiento de estos modelos ha supuesto una gran carga computacional. Por suerte, se ha usado una GPU que tenía suficiente memoria para entrenar la mayoría de los modelos con la configuración reportada por los autores. Por esta limitación, se entrenó EDGE con una configuración subóptima, para el que se redujo el tamaño de la red.

En la figura 5.1 se reportan los resultados obtenidos para cada modelo. Las métricas de interés son Hits@20 y el área bajo la curva ROC. Observando esta se llega a las siguientes conclusiones:

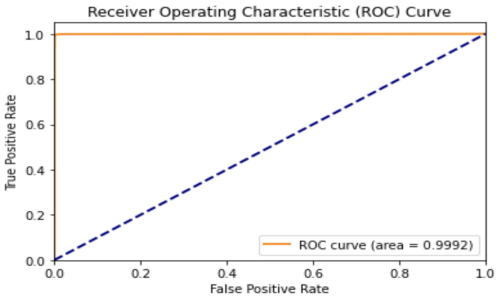
1. El problema está en gran parte resuelto. Hasta el AUC del modelo más simple (GCN) es mayor que 0,99. Este es un resultado extremadamente bueno, siendo 1 la puntuación de un clasificador perfecto.

	Hits@20 val	AUC val	Hits@20 test	AUC test
GAT	0.540800	0.998300	0.303800	0.998100
GCN	0.555900	0.997600	0.405300	0.997300
NGNN	0.673600	0.998800	0.396100	0.999000
EDGE	0.686800	0.999300	0.390900	0.999400
SAGE	0.695400	0.999200	0.333800	0.999500
AUC_LOSS	0.729900	0.999400	0.871800	0.999200

Figura 5.1: Hits@20 y AUC de cada modelo en los conjuntos de validación y test



(a) Hits@20 en validación de distintos modelos en función de la época



(b) Curva ROC con 0,99 de área

Figura 5.2: Mientras que el AUC es casi ideal para todos los modelos, la métrica Hits@20 muestra diferencias en el rendimiento de los modelos.

2. Un buen Hits@20 en validación no se traduce a un buen Hits@20 en test. Sin embargo, el AUC en validación se traslada muy bien a test. Esto se puede explicar por el comportamiento de Hits@20, que se fija en las 20 peores predicciones. Como se explica en la sección 4.2 esto hace que sea muy sensible al ruido.

3. El mejor modelo es AUC_LOSS obteniendo el mejor resultado tanto en validación como en test en términos de Hits@20. Este resultado está muy por encima del resto, y resulta impresionante considerando que se trata de GraphSAGE con la función de pérdida cambiada para adaptarse a Hits@20. En realidad, se debe a una combinación de buenos hiperparámetros y función de pérdida.

Estos resultados se pueden comparar con los obtenidos en [23]. En este artículo los autores comentan cómo les fue difícil replicar los resultados obtenidos por otros autores, siendo peores para algunos modelos y mejores para otros. Los resultados inferiores que se obtienen en este trabajo se pueden explicar por el máximo de épocas establecido durante el entrenamiento, que paraba los experimentos en 250 épocas. Por el buen resultado inicial de AUC_LOSS se decidió entrenar a este modelo por más tiempo, lo que en parte también explica la diferencia de rendimiento de este.

5.2. Resultados de técnicas de ensemble

De todos los métodos implementados, las mejores métricas (en términos de Hits@20) se obtienen haciendo una media de las predicciones de varios modelos (ensemble). En este

	Test Hits@20
SAGE_1	0.334000
SAGE_2	0.463000
SAGE_3	0.535000
SAGE_4	0.427000
SAGE_5	0.514000
TPE	0.869000
VotingAverage	0.900000

(a) Resultados de distinta semilla

	Test Hits@20
SAGE	0.334000
GAT	0.304000
GCN	0.405000
NGNN	0.396000
EDGE	0.391000
AUC_LOSS	0.872000
TPE	0.882000
VotingAverage	0.916000

(b) Resultados de distinta arquitectura

Figura 5.3: Resultados de las técnicas de ensemble. Simplemente cambiar de semilla es suficiente para obtener resultados muy buenos. La media simple obtiene mejores resultados que la optimización bayesiana (TPE).

trabajo se propone que se debe a que mejoran la estabilidad, suavizando el efecto de las peores predicciones. Esto explota las características de la métrica Hits@20, ya que Hits@20 se fija principalmente en la vigésima peor predicción de arista negativa. Primero se describe la notación con la que se referirá a los distintos métodos:

- TPE: Se utiliza el algoritmo de optimización bayesiana Tree Parzen Estimator en el conjunto de validación para decidir por cuánto pesar las predicciones de cada modelo a la hora de hacer la media pesada en test [13].
- VotingAverage: Se hace la media de todos los modelos pesándolos por igual. Esta alternativa se estudia para comprobar si los buenos resultados son por TPE o simplemente por hacer ensemble.

Con estas técnicas se combinarán las predicciones de los siguientes ensembles:

- Un ensemble de todas las arquitecturas implementadas en este trabajo.
- Un ensemble de la misma arquitectura (GraphSAGE) entrenada con 5 semillas distintas.

El resultado de los experimentos se puede visualizar en la figura 5.4. Como se puede observar, la métrica Hits@20 se mejora drásticamente haciendo ensemble. Simplemente cambiar la semilla y hacer la media consigue pasar de un Hits@20 de alrededor de 0,40 en cada modelo individual a 0,9. Para ambos experimentos, el ensemble cambiando semilla y el ensemble cambiando arquitectura, mejoran el Hits@20 de las predicciones por mucho. Curiosamente, la media simple (VotingAverage) funciona mejor que la optimización bayesiana (TPE) para los modelos entrenados en este trabajo, al contrario de lo encontrado por los autores de [13] en su estudio de ablación.

Aunque hacer ensemble es una técnica que suele mejorar los resultados, las mejoras obtenidas en este caso son desproporcionadas. El ensemble de modelos GraphSAGE entrenados con semillas distintas es el más drástico con una mejora de alrededor del 80 % en Hits@20. En la siguiente sección se propone una nueva teoría con la que explicar esta mejora y entender con más profundidad el comportamiento de Hits@K.

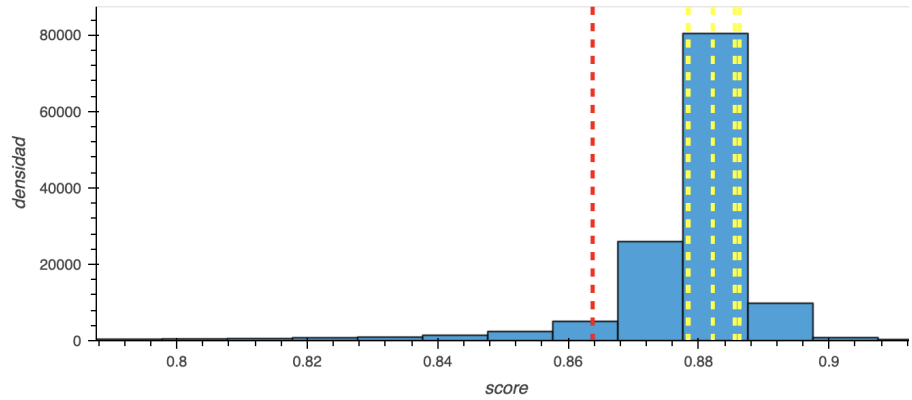


Figura 5.4: En azul se muestra la distribución de predicciones para aristas positivas. La línea discontinua roja indica el límite que establecería la vigésima predicción con mayor error de las aristas negativas haciendo ensemble. En amarillo se muestran los límites que establecería cada uno de los modelos por separado.

5.2.1. Hipótesis propuesta

Ante la rotunda efectividad observada de los métodos de ensemble se busca una explicación que los relacione con la métrica Hits@20. Suponiendo la siguiente hipótesis se llega a una explicación satisfactoria de por qué funcionan tan bien:

Las 20 peores predicciones negativas son diferentes para cada modelo entrenado. Incluso entrenar la misma arquitectura con distinta semilla es suficiente para que no sean las mismas aristas.

Suponiendo que las 20 peores predicciones de cada modelo son distintas, entonces al calcular la media el error de estas se reducirá. Las predicciones del resto de modelos compensarán las peores predicciones de cada modelo. Por lo tanto, el vigésimo error más alto será más pequeño, moviendo el límite establecido por Hits@20 y dando una mejor puntuación.

Investigando la distribución de predicciones del ensemble de modelos con distinta semilla (figura 5.4) se confirma esta hipótesis. Como se puede ver en la gráfica, el límite establecido por la media de las predicciones es mucho más favorable. Aunque la distancia entre los límites de los modelos por separado respecto al límite del ensemble es de apenas 0,02, gran parte de la distribución de predicciones positivas está en esta franja. En conclusión:

- Hits@20 es una métrica muy sensible al ruido, fijándose en la vigésima predicción de arista negativa con más error.
- El uso de ensemble reduce este vigésimo error, haciendo que muchos más positivos queden por encima del límite establecido.
- Afortunadamente, no se equivocan todos los modelos en las mismas aristas negativas, si esto fuese así, los métodos de ensemble no mejorarían drásticamente Hits@20.

5.2.2. Explorando las distribuciones de predicciones

En esta subsección se van a investigar las distribuciones de predicciones de aristas positivas y negativas. Para ello se van a estudiar los resultados de entrenar GraphSAGE con distintas

semillas y la media de estas, el ensemble. El objetivo es ver qué diferencias existen entre las predicciones haciendo ensemble y las predicciones de los modelos por separado. Las preguntas que se busca responder son las siguientes:

- ¿Hacer ensemble mejora todas las predicciones por igual, o solo suaviza el error de las peores predicciones?

	Reducción general	Reducción 20 peores
SAGE_1	0.000688	0.097285
SAGE_2	0.000916	0.100843
SAGE_3	-0.000500	0.134938
SAGE_4	-0.000893	0.130324
SAGE_5	-0.000211	0.034761

Cuadro 5.1: Reducción general de error y reducción de error en las 20 peores predicciones en aristas negativas al usar ensemble. Valores negativos indican que se aumentó el error.

Para responder a la primera pregunta se compara la reducción de error ganada al hacer ensemble 5.1. Se calcula la reducción de error media de las 20 peores predicciones frente a la reducción media de todas las predicciones. Los resultados muestran una reducción consistente en el error de las predicciones al aplicar ensemble. Además, la reducción de error es mucho mayor en las peores predicciones. Esta mejora, aunque parezca poca, es la razón por la que se mueve la frontera de Hits@20 en 5.4.

- ¿Es verdad que las 20 peores predicciones de aristas negativas son distintas para cada modelo?

(a) En el conjunto de validación

(b) El conjunto de test

Figura 5.5: Número de aristas negativas que se repiten en el top 20 de peores predicciones para cada par de modelos.

Para investigar esto podemos limitarnos a contar las aristas que se repiten en el top 20 peores predicciones negativas de cada pareja de modelos. Como podemos ver en la figura 5.5 no se repiten todas las aristas. Esta es la suposición central de la hipótesis propuesta.

- ¿Es la distribución de predicciones de ensemble considerablemente diferente a las demás?

Para ello vamos a usar la divergencia de Kullback-Leibler (KL). La divergencia KL mide la similitud entre dos distribuciones. Se puede ver como la esperanza del exceso de sorpresa

de usar Q para modelar P cuando la distribución real es P. Siendo la diferencia de sorpresa (también llamada información) para un mismo evento x bajo las dos distribuciones,

$$I_q(x) - I_p(x) = -logQ(x) + logP(x) = log\frac{p(x)}{q(x)} \tag{5.1}$$

Entonces la esperanza de esta diferencia sobre P(X) será la divergencia KL,

$$D_{\mathrm{KL}}(P\|Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)}\, dx \tag{5.2}$$

La divergencia KL no es simétrica, pero se puede modificar la fórmula para tener una versión simétrica. Esta es conocida como la divergencia Jensen Shannon y siendo $M = \frac{1}{2}(P + Q)$ se define como,

$$D_{\mathrm{JS}}(P\|Q) = \frac{1}{2}D_{\mathrm{KL}}(P\|M) + \frac{1}{2}D_{\mathrm{KL}}(Q\|M) \tag{5.3}$$

Este estadístico se usa sobre distribuciones de probabilidad, por lo que habrá que convertir el vector de predicciones en una distribución. Para esto se genera un histograma del vector y se asignan probabilidades en función del número de elementos en cada rango. Una vez modeladas las distribuciones de predicciones de cada modelo para aristas positivas y negativas se compara con las demás. El resultado es la tabla 5.2, que reporta la divergencia JS media entre las predicciones de un modelo a las predicciones de los demás modelos para las mismas aristas.

	val_pos	**val_neg**	**test_pos**	**test_neg**
SAGE_1	0.107301	0.004903	0.032597	0.004936
SAGE_2	0.063503	0.004861	0.069909	0.004648
SAGE_3	0.054015	0.004818	0.024728	0.004898
SAGE_4	0.071341	0.005087	0.028834	0.005040
SAGE_5	0.057848	0.005024	0.051733	0.005137
ENSEMBLE	0.119109	0.007157	0.032363	0.007172

Cuadro 5.2: Divergencia JS media de las predicciones de cada modelo con el resto. Se comparan las distribuciones de predicciones de aristas positivas y negativas en validación y test. Comparar las distribuciones de predicciones negativas con las positivas resulta en una divergencia JS de 0,979.

Como podemos ver, no hay apenas diferencia entre las distribuciones. Además, la distribución de predicciones de ensemble tiene los mismos valores que las demás. Por lo tanto, podemos afirmar que la distribución de ensemble es muy parecida a las demás. Esto apunta a que la mejora de rendimiento de Hits@K se da por una mejora en las 20 peores predicciones, más que por una mejora general en todas las predicciones.

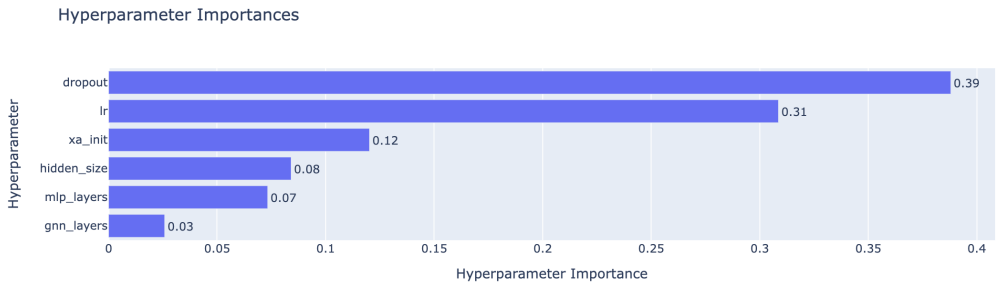


Figura 5.6: Estimación de la importancia de hiperparámetros durante la optimización bayesiana.

Hiperparámetro	**Rango**
Capas GNN	[2,4]
Capas MLP	[2,4]
Dropout	[0.05,0.5]
Hidden Size	[64, 128, 256, 512]
Learning Rate	$[10^{-4}, 10^{-2}]$
Inicialización Xavier	[True, False]

Cuadro 5.3: Espacio de búsqueda de hiperparámetros

5.3. Resultado de optimización bayesiana

Como se explica en más profundidad en el apartado 3.4.1, la optimización bayesiana permite optimizar funciones de caja negra. Esto permite optimizar mucho más eficientemente la búsqueda de hiperparámetros como el número de capas óptimo de una GNN. En este trabajo se aplica la búsqueda a los hiperparámetros del modelo implementado siguiendo [12], referido con el nombre AUC_LOSS. Este modelo es simplemente un GraphSAGE con una función derivable que aproxima AUC.

Para optimizar con Optuna hay que definir el espacio de búsqueda y la función a optimizar. En nuestro caso, la función a optimizar será la métrica Hits@20 obtenida por el modelo en el conjunto de validación, excluyendo el conjunto de test hasta el final. Para el espacio de búsqueda se ha escogido un espacio amplio para demostrar las ventajas de usar esta técnica. Este espacio se describe en la tabla 5.3.

Una vez definido el espacio y la función, comienza la optimización siguiendo el algoritmo Tree-Structured Parzen Estimator. Como las configuraciones a probar no están predefinidas, sino que se van decidiendo para probar las más prometedoras, se puede explorar más a fondo un hiperparámetro que otro. En la figura 5.5. se visualizan los hiperparámetros más importantes, a los que más experimentos se les ha dedicado.

	Hits@20 validación	Hits@20 test
Optimización bayesiana	0.781000	0.814000
Hiperparámetros de autores	0.729000	0.871000

Figura 5.7: Resultados de la mejor hiperparametrización encontrada frente a los resultados con la hiperparametrización encontrada por los autores. Se consigue un mejor resultado en validación y un resultado comparable en test.

Hiperparámetro	Opt. Bayesiana	Autores
Capas GNN	2	2
Capas MLP	4	2
Dropout	0.21	0.3
Hidden Size	512	512
Learning Rate	0.00036	0.001
Inicialización Xavier	False	True

Cuadro 5.4: Mejor configuración encontrada durante la búsqueda

Adicionalmente, Optuna utiliza la siguiente técnica que nos permitirá probar muchas configuraciones en poco tiempo: la pruna de experimentos. Aprovechando que se puede extraer información del rendimiento de una configuración antes de terminarla, por ejemplo, midiendo su rendimiento en validación, Optuna para los experimentos poco prometedores. La heurística usada es la siguiente: cortar los experimentos que estén por debajo de la mediana del resto para la época actual. De las 127 configuraciones probadas solo se completaron 20, apenas un 16 %. En total se tardaron 5 horas y media en completar la optimización. De todo el tiempo, 4 horas y 20 minutos corresponden a los experimentos que se completaron, por lo que de no parar los experimentos tendríamos alrededor de 5 veces más tiempo de cómputo.

A continuación, se comparan los hiperparámetros obtenidos respecto a los de los autores. Como podemos ver, los hiperparámetros son parecidos, aunque no del todo iguales. Haber encontrado un mínimo distinto al que encontraron los autores es interesante, ya que refleja la existencia de múltiples mínimos. Los resultados, reportados en la figura 5.7, son satisfactorios, obteniendo una mejor métrica en validación y generalizando al conjunto de test con un resultado comparable al de los autores.

5.4. Resultados de usar la función de pérdida focal

A luz de los resultados de cambiar la función de pérdida entropía cruzada binaria (BCE) por O_{AUC} (la propuesta de [12]), se decide experimentar con otra función de pérdida. Se propone usar la función de pérdida focal, que penaliza más las peores predicciones. Esto ayudaría al modelo a centrarse en esas 20 peores predicciones de aristas que marcan el límite de Hits@20. Esta se define para los objetos de la clase t, siendo p_t la predicción de que el objeto pertenezca a t, como:

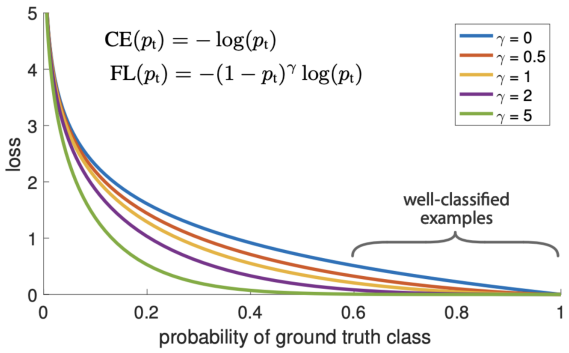


Figura 5.8: Visualización del efecto del parámetro γ en la función de pérdida focal. Un mayor γ penaliza más las peores predicciones.

$$\mathrm{FL}(p_t) = -(1-p_t)^{\gamma}\log(p_t) \tag{5.4}$$

Se trata de una modificación a BCE en la que se añade un parámetro γ que penaliza más las peores predicciones. Si γ es 0, se tiene la función BCE y a medida que este parámetro aumenta se penalizan más las predicciones con más error. En consecuencia, la red se centrará en reducir el error de las peores predicciones, en vez de tratar de mejorar todas por igual.

	Hits@20 val	**Hits@20 test**
$\gamma = 0$	0.722307	0.853434
$\gamma = 0{,}5$	0.728524	0.716007
$\gamma = 1$	0.731896	0.729596
$\gamma = 1{,}5$	0.725123	0.656983
$\gamma = 2$	0.727026	0.689615
$\gamma = 3$	0.724584	0.587022
$\gamma = 4$	0.721655	0.487666
$\gamma = 5$	0.711796	0.624995
ENSEMBLE	**0.737027**	**0.962454**

Cuadro 5.5: Hits@20 en validación y test de distintos parámetros de γ y el ensemble de todos los modelos. Extrañamente, los resultados en validación se mantienen estables, mientras que los del conjunto de test cambian en función de γ.

Para probar esta función se entrena el modelo GraphSAGE con distintos valores de γ y usando los mismos hiperparámetros que en el artículo de [12]. Los resultados son muy sorprendentes, se pueden hacer dos observaciones:

1. El parámetro gamma está correlacionado negativamente con el rendimiento. Esto indica que la función de pérdida focal no es una buena modificación.

2. Todos los resultados son superiores a los obtenidos para GraphSAGE en el apartado 5.1. por un amplio margen.

Incluso cuando γ es 0, en cuyo caso la función de pérdida es idéntica a BCE, se obtienen un Hits@20 muy superior al obtenido previamente. Esto en un principio no tiene sentido, deberían de obtenerse el mismo resultado que en el apartado 5.1. donde se prueba con BCE. Después de buscar un bug en el código de experimentos en vano, se encontró con la clave: Los hiperparámetros usados no son los mismos a los usados en 5.1. La causa de la mejora general en Hits@K no es el cambio de función, sino usar los hiperparámetros óptimos.

Aunque queda descartado el uso de la función de pérdida focal, estos experimentos han revelado la gran influencia de los hiperparámetros en el resultado. Seguramente la mejora drástica vista en 5.1 al usar O_{AUC} se deba no solo al cambio de función, sino que también a una mejor hiperparametrización.

Finalmente, se prueba a hacer ensemble de todos estos modelos. Aunque ninguno de ellos es mejor que el obtenido usando O_{AUC} son todos bastante buenos. Esto permite hacer un ensemble competitivo y con mucha diversidad en sus miembros. Este ensemble obtiene los mejores resultados del trabajo, con un Hits@k de 0,96, incluso cuando aparentemente la incorporación de los parámetros γ es subóptima.

Capítulo 6

Conclusiones

Este trabajo ha aplicado las arquitecturas y técnicas de entrenamiento del estado del arte a la predicción de aristas con Graph Neural Networks. En el proceso de implementación y revisión del estado del arte se ha ganado un mayor entendimiento sobre el funcionamiento de las GNN y la motivación detrás de cada mejora propuesta por la literatura científica.

Las conclusiones obtenidas de los experimentos y lectura del estado del arte son las siguientes:

- Las GNN son una arquitectura estado del arte en procesamiento de grafos. La clave de su éxito es la invarianza a la permutación mediante el paso de mensajes.
- El estado del arte en predicción de aristas en el conjunto ogbl-ddi son los métodos de ensemble. Estos métodos consiguen resultados muy buenos, incluso solo cambiando la semilla de arquitecturas simples como GraphSAGE. Los ensembles obtenidos son muy competitivos, el mejor de ellos obteniendo un 0,96 de Hits@20 en test.
- La hipótesis propuesta en este trabajo vincula la efectividad de los métodos de ensemble con el comportamiento de Hits@K. La estrategia de mejorar Hits@K mediante reducir el error de la k-ésima predicción negativa con más error sugiere nuevos enfoques en el diseño de las arquitecturas.
- La optimización bayesiana es actualmente una herramienta clave en la búsqueda de hiperparámetros. Utilizar la información de los experimentos para parar el entrenamiento tempranamente permite explorar muchas configuraciones en una fracción del tiempo usual.

Un motivo recurrente durante este trabajo es el entendimiento y explotación del comportamiento de la métrica Hits@K. Gran parte de las técnicas implementadas como el uso de ensemble [13] o la función de pérdida modificada de [12] están íntimamente relacionadas con esta. Aunque estas aportaciones se puedan ver como trucos ad hoc, mejorar una métrica bien alineada implica mejorar el funcionamiento de estos modelos en situaciones reales. Estos modelos tienen emocionantes aplicaciones capaces de transformar el mundo en ámbitos como la biomedicina.

Capítulo 7

Líneas futuras

El trabajo realizado aporta información sobre el comportamiento de Hits@K y su relación con las técnicas de ensemble. La hipótesis propuesta podría ser investigada en mayor profundidad con las siguientes líneas de investigación. Algunas ideas que surgen son las siguientes:

Investigar como afecta la diversidad de los modelos que componen los ensembles a los resultados de estos. Quizás usar modelos que sean muy distintos entre sí hace que se equivoquen en aristas negativas diferentes, mejorando las métricas de Hits@K en el proceso. Incluso se podría probar a entrenar modelos que busquen ser distintos al resto, por ejemplo, maximizando la divergencia KL de sus predicciones con respecto a las del ensemble.

Conducir un estudio de ablación para cuantificar la ganancia obtenida por el uso de O_{AUC} respecto a usar solo GraphSAGE con los mejores hiperparámetros. Lo que parece una ganancia notable podría ser principalmente el resultado de entrenar durante muchas épocas con los mejores hiperparámetros.

Si nos centramos en tratar de mejorar el rendimiento de los modelos, también queda mucho por hacer. Los siguientes puntos serían claves para la mejora del rendimiento:

- Entrenar durante más épocas; para este conjunto de datos se suelen utilizar 500. Esto mejoraría todos los modelos y no se ha llevado a cabo por la carga computacional.
- Realizar ensemble del mejor modelo, GraphSAGE con la función de pérdida O_{AUC} de [12], con diferentes semillas. Debido a los buenos resultados de hacer ensemble con diferentes semillas de GraphSAGE, esto podría dar un muy buen resultado.
- Implementar algunos modelos que, por limitaciones en la memoria de la GPU, no se han podido replicar. Usar estos modelos en los ensembles podría explicar los buenos resultados de [13].

También sería interesante buscar en la literatura porque las GNN convergen tan lento. A diferencia de otras arquitecturas de redes neuronales, en los experimentos durante este trabajo nunca han llegado a sufrir un sobreajuste. Como se comenta previamente, no llegan a alcanzar su máximo de rendimiento hasta la época 500, lo cual se traduce en 3 horas de cómputo en el hardware usado durante este trabajo. De habilitar una convergencia

más rápida, se podrían probar muchas más variantes, por lo que se acabarían encontrando mejores técnicas y modelos.

Capítulo 8

Análisis de impacto

La implementación del estado del arte en predicción de efectos secundarios al tomar varios medicamentos a la vez, utilizando Graph Neural Networks (GNN), representa un avance significativo hacia el logro del Objetivo 3 de Desarrollo Sostenible (ODS): garantizar una vida sana y promover el bienestar para todos en todas las edades.

El uso de GNN en este contexto permite modelar de manera más precisa las interacciones entre diferentes medicamentos y cómo estas interacciones pueden afectar la salud de los pacientes. Al predecir con mayor precisión los posibles efectos secundarios de la polifarmacia, este trabajo contribuye directamente a mejorar la seguridad y eficacia de los tratamientos médicos, lo que se alinea perfectamente con el ODS 3.

Al reducir el riesgo de efectos secundarios adversos debido a la interacción de medicamentos, esta investigación ayuda a promover una vida más saludable y un bienestar general para todas las personas, independientemente de su edad o condición. Además, al proporcionar herramientas más efectivas para la gestión de la salud y la medicación, se fomenta un sistema de atención médica más eficiente y centrado en el paciente, lo que a su vez contribuye a la realización del ODS 3 y al progreso hacia una sociedad más saludable y sostenible en su conjunto.

Bibliografía

[1] Z. Wang, P. Veličković, D. Hennes et al., *Nature Communications*, vol. 15, pág. 1906, 2024. DOI: 10.1038/s41467-024-45965-x. dirección: https://doi.org/10.1038/s41467-024-45965-x.

[2] A. Merchant, S. Batzner, S. S. Schoenholz, M. Aykol, G. Cheon y E. D. Cubuk, *Nature*, vol. 624, págs. 80-85, 2023. DOI: 10.1038/s41586-023-06735-9. dirección: https://doi.org/10.1038/s41586-023-06735-9.

[3] Y. Liu, Q. Liu, Y. Tian et al., *Concept-Aware Denoising Graph Neural Network for Micro-Video Recommendation*, 2021. arXiv: 2109.13527 [cs.IR].

[4] D. S. Wishart, Y. D. Feunang, A. C. Guo et al., «DrugBank 5.0: a major update to the DrugBank database for 2018», *Nucleic Acids Res*, vol. 46, n.º D1, págs. D1074-D1082, ene. de 2018. DOI: 10.1093/nar/gkx1037.

[5] W. Hu, M. Fey, M. Zitnik et al., *Open Graph Benchmark: Datasets for Machine Learning on Graphs*, 2021. arXiv: 2005.00687 [cs.LG].

[6] T. N. Kipf y M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, 2017. arXiv: 1609.02907 [cs.LG].

[7] W. L. Hamilton, R. Ying y J. Leskovec, *Inductive Representation Learning on Large Graphs*, 2018. arXiv: 1706.02216 [cs.SI].

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò e Y. Bengio, *Graph Attention Networks*, 2018. arXiv: 1710.10903 [stat.ML].

[9] X. Song, R. Ma, J. Li, M. Zhang y D. P. Wipf, *Network In Graph Neural Network*, 2021. arXiv: 2111.11638 [cs.LG].

[10] B. Li, Y. Xia, S. Xie, L. Wu y T. Qin, «Distance-Enhanced Graph Neural Network for Link Prediction», 2021. dirección: https://api.semanticscholar.org/CorpusID:236448552.

[11] S. Lu y J. Yang, «Link Prediction with Structural Information», 2021.

[12] Z. Wang, Y. Zhou, L. Hong, Y. Zou, H. Su y S. Chen, *Pairwise Learning for Neural Link Prediction*, 2022. arXiv: 2112.02936 [cs.LG].

[13] Z. H. Wong, L. Yue y Q. Yao, *Ensemble Learning for Graph Neural Networks*, 2023. arXiv: 2310.14166 [cs.LG].

[14] R. Pascanu, T. Mikolov e Y. Bengio, *On the difficulty of training Recurrent Neural Networks*, 2013. arXiv: 1211.5063 [cs.LG].

[15] X. Glorot e Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks», en *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh y M. Titterington, eds., ép. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May de 2010, págs. 249-256. dirección: `https://proceedings.mlr.press/v9/glorot10a.html`.

[16] T. Akiba, S. Sano, T. Yanase, T. Ohta y M. Koyama, «Optuna: A Next-generation Hyperparameter Optimization Framework», en *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[17] J. Bergstra, R. Bardenet, Y. Bengio y B. Kégl, «Algorithms for Hyper-Parameter Optimization», en *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira y K. Weinberger, eds., vol. 24, Curran Associates, Inc., 2011. dirección: `https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf`.

[18] S. Watanabe, *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*, 2023. arXiv: `2304.11127 [cs.LG]`.

[19] W. Hu, M. Fey, M. Zitnik et al., «Open Graph Benchmark: Datasets for Machine Learning on Graphs», *arXiv preprint arXiv:2005.00687*, 2020.

[20] A. Paszke, S. Gross, F. Massa et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, 2019. arXiv: `1912.01703 [cs.LG]`.

[21] Martín Abadi, Ashish Agarwal, Paul Barham et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. dirección: `https://www.tensorflow.org/`.

[22] M. Fey y J. E. Lenssen, «Fast Graph Representation Learning with PyTorch Geometric», en *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[23] J. Li, H. Shomer, H. Mao et al., *Evaluating Graph Neural Networks for Link Prediction: Current Pitfalls and New Benchmarking*, 2023. arXiv: `2306.10453 [cs.LG]`.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Sat Jun 01 20:20:08 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)