ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

LASE: Learned Adjacency Spectral Embeddings

Tesis presentada a la Facultad de Ingeniería de la
Universidad de la República por

María Sofía Pérez Casulo

en cumplimiento parcial de los requerimientos
para la obtención del título de
Magister en Ciencia de datos y Aprendizaje automático.

Directores de Tesis

Dr. Marcelo Fiori ..... Universidad de la República
Dr. Federico Larroca ..... Universidad de la República
Dr. Gonzalo Mateos ..... Universidad de Rochester

Tribunal

Dr. Martín Rocamora ..... Universidad de la República
Dr. Mauricio Velasco ..... Universidad Católica del Uruguay
Dra. Soledad Villar ..... Universidad de Johns Hopkins

Director Académico

Dr. Marcelo Fiori ..... Universidad de la República

Montevideo
viernes 3 mayo, 2024

LASE: Learned Adjacency Spectral Embeddings, María Sofía Pérez Casulo.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 98 páginas.
Compilada el viernes 3 mayo, 2024.

Agradecimientos

Quiero expresar mi más sincero agradecimiento a la Facultad de Ingeniería de la Universidad de la República por darme la oportunidad de acceder a una educación de calidad, lo cual ha sido fundamental para mi desarrollo profesional y personal.

Extiendo un especial reconocimiento a mis tutores, los doctores Marcelo, Federico y Gonzalo, cuyos valiosos comentarios y confianza depositada en mí han sido esenciales para la realización de este trabajo.

Agradezco también al resto del equipo de grafos, Paola y Bernardo, por sus consejos y sugerencias durante nuestras reuniones semanales, que siempre contribuyeron a enriquecer mi investigación. Extiendo mi gratitud al Instituto de Matemática y Estadística Rafael Laguardia por permitirme ser parte de un grupo tan destacado.

Por último, pero no menos importante, agradezco a mi familia y amigos por su incansable apoyo a lo largo de esta etapa. En particular, a mi esposo Santiago, cuyo constante aliento ha sido una fuente de motivación indispensable para alcanzar este punto.

Esta página ha sido intencionalmente dejada en blanco.

A Santi, Pachi y Draco

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Las redes neuronales de grafos (GNNs, por su sigla en inglés) se han convertido en un foco de interés durante los últimos años. Las mismas han sido aplicadas exitosamente en problemas de varios dominios, mostrando ser una metodología particularmente efectiva en lo que respecta al aprendizaje de representaciones de grafos (GRL). Por su parte, los Random Dot Product Graphs (RDPG) son un modelo generativo de grafos muy popular por su simplicidad, interpretabilidad y poder de expresión. El mismo postula que un grafo puede modelarse considerando que existen posiciones latentes (o *embeddings*) para cada nodo y especifica la probabilidad de interconexión entre estos como el producto interno entre los *embeddings* asociados. La tarea de estimar estos *embeddings* se plantea como un problema de factorización de matrices no convexo. Los *Adjacency Spectral Embeddings* (ASE) ofrecen una solución aproximada a este problema obtenida mediante la descomposición espectral de la matriz de adyacencia, la cual si bien goza de garantías estadísticas sólidas puede ser costosa computacionalmente.

El presente trabajo se centra en el uso de GNNs para aprender *embeddings* similares a los de ASE. En particular, se propone una nueva arquitectura, denominada *Learned Adjacency Spectral Embeddings* (LASE), que utiliza la técnica de *Algorithm Unrolling* para modelar el descenso por gradiente (GD) aplicado a ASE. En esta arquitectura, cada iteración de GD se representa como una capa en una red neuronal, compuesta por componentes clásicos de las GNNs como las *Graph Convolutional Networks* (GCN) y los mecanismos de *Graph Attention* (GAT). Los *embeddings* obtenidos con LASE muestran ser de una calidad comparable o incluso superior a los obtenidos mediante otros métodos tradicionales como ser la descomposición en valores singulares (SVD). La arquitectura propuesta permite la incorporación de mecanismos de *sparse attention* los cuales reducen significativamente el costo computacional requerido. A su vez, se introduce una versión generalizada, *Generalized* LASE (GLASE), que expande su aplicación a escenarios con grafos heterofilios. Por su parte, este trabajo propone el uso de los *embeddings* obtenidos con LASE como *Positional Encodings* (PE), que pueden ser utilizados por ejemplo para alimentar modelos basados en Graph Transformers (GTs). A su vez, la incorporación de la información estructural que aportan estos *embeddings*, permite romper la simetría en las representaciones de los nodos, venciendo así algunas de las limitaciones que presentan las GCNs clásicas. En particular, se propone la creación de un módulo basado en LASE que pueda integrarse como parte de la arquitectura de un modelo más general destinado a tareas específicas, como ser clasificación de nodos o predicción de enlaces. Los experimentos realizados durante este trabajo consideran grafos homofilios y heterofilios, y buscan validar la efectividad y adaptabilidad de los modelos en una variedad de escenarios. En particular, se utilizan conjuntos de datos sintéticos generados mediante modelos *Stochastic Block Model* (SBM), así como conjuntos datos reales comúnmente utilizados en la literatura.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Tabla de contenidos

Capítulo 1

Introducción

Las redes neuronales de grafos (GNNs, por sus siglas en inglés) se han convertido en un foco de interés en los últimos años. Las mismas han sido aplicadas a problemas de numerosos dominios, desde sistemas de recomendación (RecSys, por sus siglas en inglés) [54] [25], hasta grafos de conocimientos (KG, por sus siglas en inglés) [26] [57] y procesamiento de lenguaje natural (NLP, por sus siglas en inglés) [48] [44]. En particular, han mostrado ser una metodología efectiva en lo que respecta al aprendizaje de representaciones de grafos (GRL, por sus siglas en inglés) [23].

La mayoría de las GNNs hacen uso de un mecanismo de pasaje de mensajes o *Neural Message Passing* (NMP) donde la representación de los nodos se construye mediante la agregación de información de vecinos locales. Este tipo de arquitectura es fundamentalmente estructural, es decir, la representación de los nodos depende únicamente de la estructura local del grafo. Como consecuencia, este tipo de GNNs (MP-GNNs) fallan en diferenciar dos nodos con la misma estructura local. La equivalencia entre el clásico algoritmo de Weisfeiler-Lehman (WL) [55] el cual brinda una condición necesaria para determinar si dos grafos son isomorfos y las MP-GNNs resalta la incapacidad de este tipo de GNNs para distinguir entre grafos con estructuras locales idénticas [56]. Una alternativa para abordar este problema consiste en adicionar señales en cada nodo que permitan romper la simetría, mediante la incorporación de Positional Encodings (PE) o Structural Encodings (SE) [50]. Dada su popularidad, es de particular interés para este trabajo el uso de PE basados en descomposición espectral de la matriz de adyacencia [33], asumiendo que la estructura del grafo proviene de un modelo *Random Dot Product Graph* (RDPG) [6].

La estimación de los *embeddings* de un RDPG suele plantearse como un problema de factorización de matrices no convexo. Los *Adjacency Spectral Embeddings* (ASE) ofrecen una solución aproximada a este problema obtenida mediante la descomposición espectral de la matriz de adyacencia, la cual, si bien goza de garantías estadísticas sólidas, puede ser costosa computacionalmente y formalmente resuelve un problema sustituto. Por su parte, este problema también puede resolverse utilizando algoritmos de optimización iterativos como ser el descenso por gradiente (GD) [17]. Como contrapartida, este tipo de soluciones iterativas conllevan a un re-cálculo de la solución frente a cualquier cambio en el grafo. Esto puede implicar un consumo considerable de recursos y tiempo, especialmente en grafos de gran tamaño o que cambian con frecuencia. A diferencia de trabajos previos, se busca lograr aprender estos *embeddings* espectrales mediante el uso de GNNs que permitan transferabilidad entre grafos con distribuciones similares. La dificultad principal reside en desarrollar una arquitectura que no dependa exclusivamente de componentes convolucionales, como las GCN, da-

do que estas enfrentan obstáculos para realizar la descomposición espectral en ciertas configuraciones de grafos, tales como un SBM simétrico. A su vez, se busca lograr una arquitectura robusta y eficiente en términos computacionales que permita su aplicación sobre grafos de gran porte. La motivación del presente trabajo puede resumirse entonces en los siguientes dos puntos:

1. El uso de técnicas de aprendizaje automático sobre grafos mediante GNNs para aprender *embeddings* espectrales similares a los ASE. En particular, se plantea aplicación de la técnica de *Algorithm Unrolling* al algoritmo GD para la obtención de *embeddings* de grafos RDPG. Esto nos permite ganar generalidad, así como mejorar la eficiencia computacional, reduciendo la dependencia cuadrática de GD a una lineal y así poder atacar problemas que involucren grafos de mayor porte.
2. El aprendizaje de *embeddings* espectrales que puedan ser usados como PE. En particular, es de interés la creación de una arquitectura modular que permita integrarse a otras arquitecturas más específicas, facilitando el aprendizaje de estos *embeddings* como parte de un flujo global *end-to-end*.

La arquitectura propuesta para la obtención de estos *embeddings* se denominó *Learned* ASE (LASE). La misma consiste de una red neuronal basada en componentes clásicos de GNNs: una componente convolucional de grafos (GCN) y una componente de *attention* de grafos (GAT). La misma se entrena de forma *offline* y no supervisada, utilizando muestras de grafos provenientes de una misma distribución. Durante la etapa de experimentación, se utilizaron tanto conjuntos de datos sintéticos generados mediante modelos *Stochastic Block Model* (SBM), como conjuntos datos reales comúnmente utilizados en la literatura. A su vez, durante este trabajo, se plantearon ciertas modificaciones de la arquitectura que buscaron potenciar su performance. Dentro de estas, se encuentra la incorporación de mecanismos de *sparse attention* que permiten optimizar los tiempos de inferencia y la escalabilidad del modelo; así como la introducción de una versión generalizada, *Generalized* LASE (GLASE), que expande su aplicación a escenarios con grafos heterofilios. También se exploran configuraciones que permiten adaptar los modelos a escenarios con información faltante o desconocida. Los resultados obtenidos sugieren que los modelos desarrollados permiten ampliar significativamente la aplicación del algoritmo GD a grafos de gran tamaño, los cuales anteriormente no podían ser procesados eficazmente. Además, se ha demostrado que estos modelos tienen la capacidad de entrenarse en conjuntos de datos relativamente pequeños y generalizar exitosamente a grafos de mayor porte. Los *embeddings* obtenidos tanto con LASE como GLASE demostraron capturar de manera eficiente la información estructural del grafo, lo cual permite enriquecer los atributos de los nodos de un grafo dado. La incorporación de los mismos como *Positional Encodings* (PE) mostró mejorar la performance en ciertas tareas específicas como ser la clasificación de nodos en grafos homofilios y la predicción de enlaces en grafos heterofilios.

El resto de este trabajo se organiza de la siguiente manera. En el Capítulo 2, se brinda una introducción a los fundamentos de los grafos y su relevancia en el aprendizaje automático para para entender y procesar estructuras de datos complejas y altamente interconectadas. Se expone una definición formal de estas estructuras matemáticas y su representación mediante matrices de adyacencia. Además, se discuten diversos modelos estadísticos para la generación de grafos, destacando los modelos de grafos clásicos, como el modelo Erdös-Rényi y Watts-Strogatz, así como modelos de posiciones latentes, incluyendo el modelo SBM y el RDPG. Se describen técnicas para la estimación de *embeddings*, como ASE y se mencionan extensiones del modelo RDPG para adaptarlo a diferentes tipos de estructuras de datos. En el Capítulo 3, se desarrolla el concepto de GNNs presentando dos formas diferentes de modelar las mismas: mediante el método de NMP y desde la perspectiva del *Graph Signal Proccessing* (GSP)

introduciendo las convoluciones en grafos. A su vez, se presentan las *Graph Attention Networks* (GAT) y *Graph Transformers* (GT), que extienden el modelo clásico de *Transformers* al ámbito de los grafos. El Capítulo 4 se dedica al área de *Graph Representation Learning* (GRL), que implica el aprendizaje de *embeddings* en grafos para codificar eficazmente su información estructural. Se discuten técnicas basadas en factorización y en métodos de *Random Walk Embeddings*, además de introducen los *Graph Positional Encoders* (PE) y *Graph Structural Encoders* (SE), que codifican la estructura y la posición de los nodos dentro del grafo. El Capítulo 5 aborda el marco de *Learning to Optimize* (L2O), haciendo énfasis en la técnica de *Algorithm Unrolling*, que permite adaptar algoritmos iterativos de optimización a arquitecturas de redes neuronales. En el Capítulo 6, se explora la aplicación de la técnica de *Algorithm Unrolling* al algoritmo GD para obtener *embeddings* similares a los de ASE, dando lugar a el modelo *Learned* ASE (LASE). Se detallan refinamientos en la arquitectura, presentando una versión generalizada (GLASE) que permite ampliar su aplicación a grafos con comportamiento heterofílicos. A su vez, se discute la aplicación de LASE como PE en tareas específicas de aprendizaje en grafos, como ser la clasificación de nodos y la predicción de enlaces. El Capítulo 7 se centra en la validación experimental de los modelos LASE y GLASE mediante pruebas sobre conjuntos de datos sintéticos y reales, evaluando su desempeño en diferentes escenarios de aplicación y estudiando su capacidad para manejar datos faltantes y reducir el costo computacional mediante técnicas de *sparse attention*. Finalmente, en el Capítulo 8 se exponen las conclusiones obtenidas, proporcionando una síntesis de los hallazgos más relevantes y posibles trabajos a futuro.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 2

Introducción a los grafos

Los grafos son una poderosa estructura de datos utilizada en el campo del aprendizaje automático para representar y analizar relaciones complejas entre elementos. Son especialmente útiles cuando se trabaja con datos que tienen una estructura de interconexión, como redes sociales, sistemas de recomendación, biología molecular, análisis de textos entre otros.

Un grafo está compuesto por nodos (también llamados vértices) que representan entidades, y aristas (también llamadas enlaces) que denotan las relaciones entre nodos.

Definición 1 (Grafo) *Llamamos grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$ a un conjunto $\mathcal{V}$ de vértices conectados por un conjunto $\mathcal{E}$ de aristas. Los elementos de $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ son pares de vértices distintos (u, v) con $u, v \in \mathcal{V}$.*

Las relaciones entre los nodos pueden tener una dirección específica (grafos dirigidos) o no (grafos no dirigidos). Los grafos dirigidos se utilizan para modelar relaciones asimétricas, como relaciones de causa y efecto, flujos de información o jerarquías. Mientras que los no dirigidos utilizan para modelar relaciones simétricas, como relaciones de cercanía o interacciones mutuas. A su vez las aristas pueden llevar asociadas propiedades o características en forma de pesos. Los grafos con pesos son útiles cuando se necesita modelar y analizar relaciones con valores cuantitativos, mientras que los grafos sin pesos son más simples y se utilizan para modelar relaciones binarias.

Algebraicamente, un grafo puede ser representado a través de una matriz de conexiones la cual se denomina matriz de adyacencia.

Definición 2 (Matriz de adyacencia) *Dado un grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$, se define la matriz de adyacencia a una matriz binaria y simétrica $\mathbf{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ tal que:*

$$\mathbf{A}_{ij} = \begin{cases} 1, (u, v) \in \mathcal{E} \\ 0, (u, v) \notin \mathcal{E} \end{cases}$$

Los grafos pueden ser clasificados como homogéneos o heterogéneos. Los grafos homogéneos son aquellos en los que todos los nodos y todas las aristas pertenecen al mismo tipo o categoría. Son comunes en problemas donde todos los elementos son del mismo tipo, como redes sociales donde todos los nodos son usuarios. Los grafos heterogéneos son aquellos en los que los nodos y las aristas pueden pertenecer a diferentes tipos o categorías. Se utilizan para modelar relaciones entre entidades de diferentes clases o tipos. Por ejemplo, en una red de películas y actores, los nodos de película y actor serían de tipos diferentes.

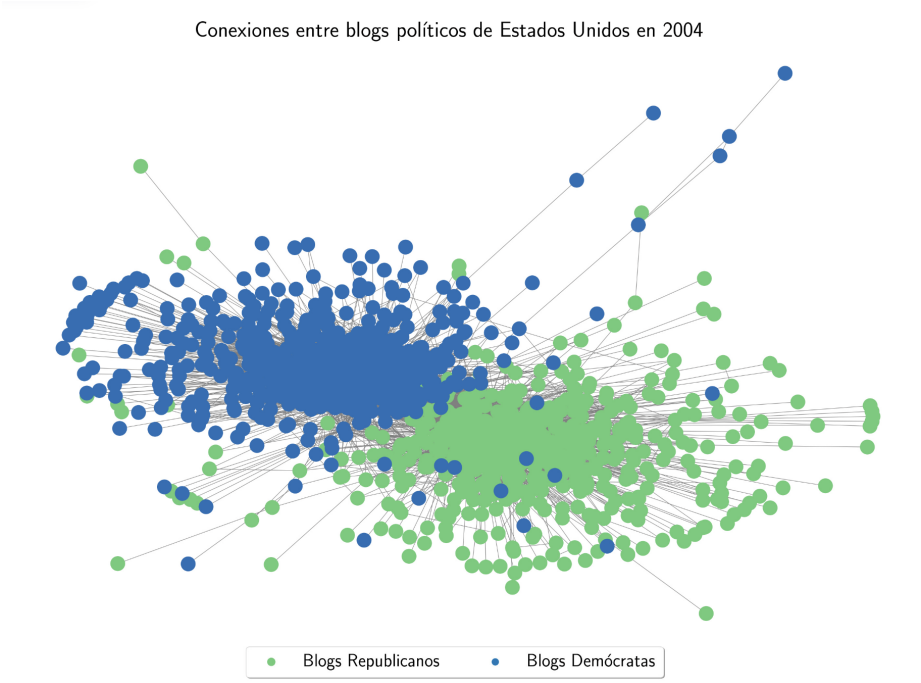


Figura 2.1: Conexiones entre blogs políticos estadounidenses en 2004 etiquetados según su afiliación a uno de los dos grandes partidos de la política estadounidense: Demócratas y Republicanos. Se visualiza como los blogs tienden a conectarse con otros dentro de su mismo partido y no tanto con blogs del partido oposición.

Por su parte, se definen los conceptos de homofilia y heterofilia. La homofilia refiere a la tendencia de que nodos similares se conecten entre sí. En otras palabras, los nodos que comparten características tienden a estar más interconectados que aquellos que difieren en esas características. Un ejemplo de esto, es el conjunto de datos PolBlogs [32]. Se trata de datos de blogs sobre política estadounidense en un momento cercano a la elección del 2004 en ese país. Cada nodo del grafo corresponde a un blog, existiendo una arista del nodo i al nodo j si el blog i tiene un *link* al blog j. Es por lo tanto un grafo dirigido, donde cada blog está etiquetado según su afiliación a uno de los dos grandes partidos de la política estadounidense: Demócratas y Republicanos. En la Figura 2.1 se puede ver claramente como los blogs tienden a conectarse con otros dentro de su mismo partido y no tanto con blogs del partido oposición. Por su parte, la heterofilia refiere el caso opuesto, donde existe una tendencia de que nodos diferentes (o complementarios) se conecten entre sí. Por ejemplo, en la Figura 2.2 se muestra un grafo formado a partir de datos de vuelos entre aeropuertos en Europa [46]. Se trata de un grafo dirigido donde los nodos corresponden a aeropuertos, y una arista entre el aeropuerto i y el j indica la existencia de una ruta comercial que los une. Los aeropuertos están etiquetados según su nivel de actividad, medida como el número total de despegues y aterrizajes entre enero y noviembre del 2016. Se identifican dos categorías: *hubs* y no *hubs*. En dicha figura, se puede visualizar claramente como los aeropuertos que tienen mayor nivel de actividad o *hubs* tienden a conectarse entre sí, mientras que los otros aeropuertos prácticamente solo se conectan a los *hubs*.

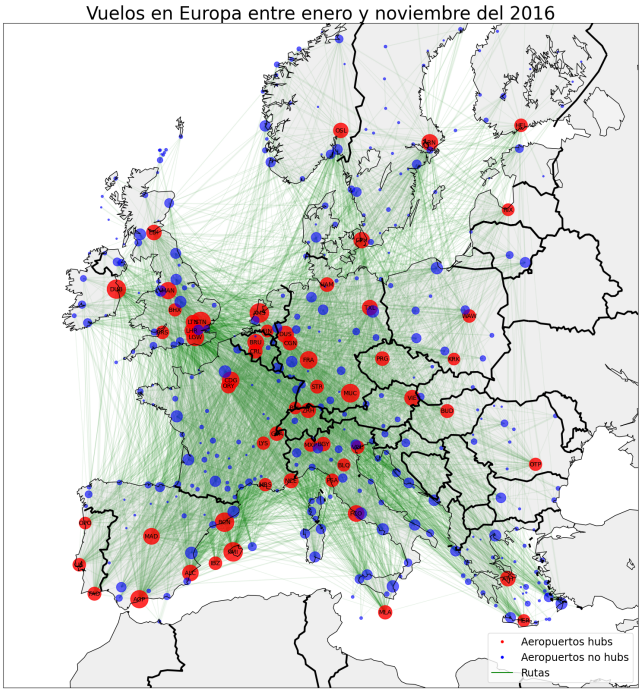


Figura 2.2: Vuelos comerciales entre aeropuertos de Europa de enero a noviembre del 2016. Los aeropuertos se clasifican en dos categorías, *hubs* y no *hubs* de acuerdo a su nivel de actividad. Se visualiza como los aeropuertos que tienen mayor nivel de actividad (*hubs*) tienden a conectarse entre sí, mientras que los otros aeropuertos prácticamente solo se conectan a los *hubs*.

2.1. Modelos de grafos aleatorios

Los modelos de grafos aleatorios son modelos probabilísticos que se utilizan para generar grafos de manera aleatoria según cierta distribución de probabilidad. Estos modelos son útiles para estudiar propiedades estadísticas y topológicas de grafos en situaciones en las que no se tiene acceso a datos reales, o para comprender mejor cómo se comportan las redes en situaciones de incertidumbre [30].

Definición 3 (Modelo de grafos aleatorios) *Un modelo de grafos aleatorios es una colección* $\{\mathbf{P}_\theta(G), G \in \mathcal{G} : \theta \in \Theta\}$ *donde*

- $\mathcal{G}$ *es el conjunto de grafos posibles bajo el modelo*
- $\mathbf{P}_\theta(.)$ *es la distribución de probabilidad sobre* $\mathcal{G}$
- θ *es el vector de parámetros del modelo*

2.1.1. Modelos clásicos

La variante más común es el modelo Erdös-Rényi (ER) el cual denotaremos como $ER(n, p)$. Este modelo genera un grafo no dirigido con n nodos donde existirá una arista (u, v) con probabilidad p independiente del resto.

Definición 4 (Modelo Erdös-Rényi) *Sea* $A = [A_{ij} \in \mathbb{R}^{n\times n}]$ *la matriz de adyacencia de un grafo generado por el modelo ER, entonces:*

$$A_{ij} \sim Bernoulli(p),\ p \in (0, 1),\ A_{ij} = A_{ji},\ A_{ii} = 0$$

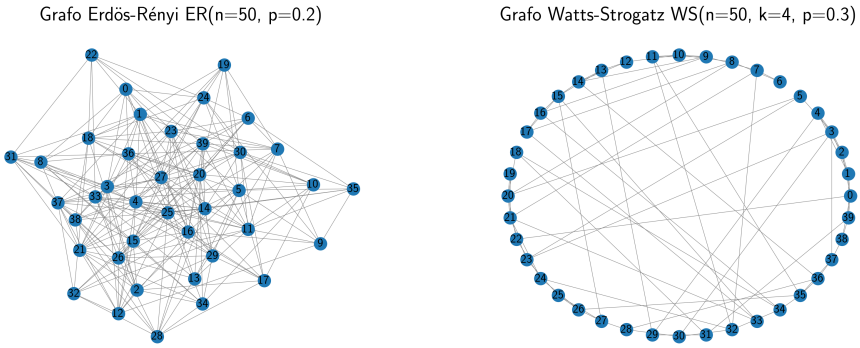


Figura 2.3: Ejemplos de grafos generados con los modelos Erdös-Rényi (izquierda) y Watts-Strogatz (derecha).

Otra variante comúnmente utilizada es el modelo Watts-Strogatz (WS). El mismo se caracteriza por tener una estructura denominada *small-world*, donde los nodos están fuertemente conectados localmente pero aún permiten distancias cortas entre nodos distantes en la red. Inicialmente, en este modelo se tiene una estructura de anillo donde cada nodo está conectado a sus k vecinos más cercanos. Luego se realiza una reconexión aleatoria, donde cada nodo se reconecta a otro con probabilidad p independiente del resto. Esto introduce aleatoriedad en el grafo y crea atajos que permiten que los caminos globales sean efectivamente más cortos. La Figura 2.3 brinda un ejemplo de grafos generados por los dos modelos mencionados.

2.1.2. Modelo SBM

El modelo *Stochastic Block Model* (SBM) es un modelo de clases latentes, donde la pertenencia a cierta clase (no observada) rige la tendencia de los nodos a conectarse entre sí. Este modelo es utilizado para generar grafos que exhiben estructuras de comunidad. En el modelo SBM, los nodos de un grafo se dividen en Q comunidades o bloques distintos, y las conexiones entre nodos se generan de acuerdo a ciertas probabilidades predefinidas [41].

Definición 5 (Stochastic Block Model) *Dadas Q comunidades $\mathcal{C}_1, \ldots, \mathcal{C}_Q$ y tasas de conexión inter-comunidades π_{qr}:*

- *Cada nodo $i \in \mathcal{V}$ pertenece de manera independiente a $\mathcal{C}_q$ con probabilidad α_q tal que*
$$\alpha = [\alpha_1, \ldots, \alpha_Q]^T, \ \mathbf{1}^T\alpha = 1.$$
- *Para cada par $i, j \in \mathcal{V}$ con $i \in \mathcal{C}_q$ y $j \in \mathcal{C}_r \Rightarrow (i, j) \in \mathcal{E}$ con probabilidad π_{qr}.*

Entonces, la matriz de adyacencia de un grafo generado por el modelo SBM $\mathbf{A} = [A_{ij} \in \mathbb{R}^{n\times n}]$ estará dada por:

$$A_{ij}|(i \in \mathcal{C}_q, j \in \mathcal{C}_r) \sim \text{Bernoulli}(\pi_{qr}), \tag{2.1}$$

con $p \in (0, 1)$, $A_{ij} = A_{ji}$ y $A_{ii} = 0$. Una observación interesante es que el modelo ER visto anteriormente es un caso particular de un SBM con $Q = 1$. Por ende, es posible reinterpretar el modelo SBM como una mezcla de modelos ER para diferentes valores de Q, como se detalla en la Figura 2.4.

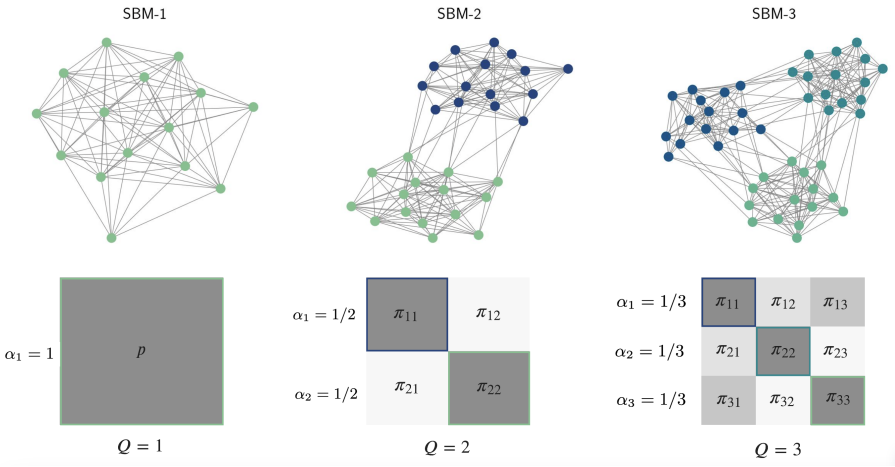


Figura 2.4: Modelo SBM visto como una mezcla de modelos ER para diferentes valores de Q.

2.1.3. Modelo RDPG

El modelo *Random Dot Product Graph* (RDPG) es un modelo generativo que asume posiciones latentes (*embeddings*) para cada nodo en un espacio $\mathbb{R}^d$, en el cual la probabilidad de que exista una arista entre un par de vértices es determinada por el producto interno entre los *embeddings* asociados [6]. En otras palabras, dado $\mathbf{X} \in \mathbb{R}^{n \times d}$ donde n es el número de nodos y d la dimensión de cada vector, la matriz de probabilidad de conexiones $\mathbf{P}$ está dada por $\mathbf{P} = \mathbf{X}\mathbf{X}^T$. Este modelo tiene la particularidad de permitir una interpretación valiosa para los valores propios de la matriz de adyacencia, permitiendo una comprensión geométrica de los mismos como veremos más en detalle más adelante.

Definición 6 (Modelo Random Dot Product Graph) *Considerando el espacio latente $\chi_d \subset \mathbb{R}^d$ tal que para todo $\mathbf{x}, \mathbf{y} \in \chi_d \Rightarrow \mathbf{x}^T\mathbf{y} \in [0,1]$ y una distribución de producto interno $\mathcal{F} : \chi_d \to [0,1]$, el modelo RDPG se define como:*

$$\mathbf{x}_1, \cdots \mathbf{x}_N \overset{\text{iid}}{\sim} \mathcal{F},$$

$$\mathbf{A}_{ij} | \mathbf{x}_i, \mathbf{x}_j \sim Ber(\mathbf{x}_i^T \mathbf{x}_j), 1 \leq i, j \leq n$$

donde $\mathbf{A}_{ij} = \mathbf{A}_{ji}$ y $\mathbf{A}_{ii} \equiv 0$.

El modelo RDPG es un caso especial de modelos de posiciones latentes [40], donde

$$\mathbf{A}_{ij} | \mathbf{x}_i, \mathbf{x}_j \sim Ber(\kappa(\mathbf{x}_i^T \mathbf{x}_j)), \tag{2.2}$$

pudiéndose aproximar cualquier $\kappa(.)$ con un d lo suficientemente grande [34]. Tanto el modelo ER, como el modelo SBM pueden ser formulados como un RDPG. Tomando $\mathbf{X}_i = \sqrt{p} \ \ \forall \ i$ se obtiene un grafo ER con probabilidad de conexión p. Mientras que, un SBM con Q comunidades puede interpretarse considerando que χ_d es un conjunto de Q vectores $x_1, \ldots, x_Q$ tales que $\pi_{qr} = \mathbf{X}_q^T \mathbf{X}_r$ y y la función de distribución $\mathcal{F}$ es tal que $\mathcal{F}(x_q) = \alpha_q$.

En general, los *embeddings* del modelo RDPG son fáciles de interpretar: su módulo representa la conectividad, mientras que el ángulo indica la afinidad entre nodos. En la Figura 2.5 se da un claro ejemplo de este comportamiento considerando el conocido

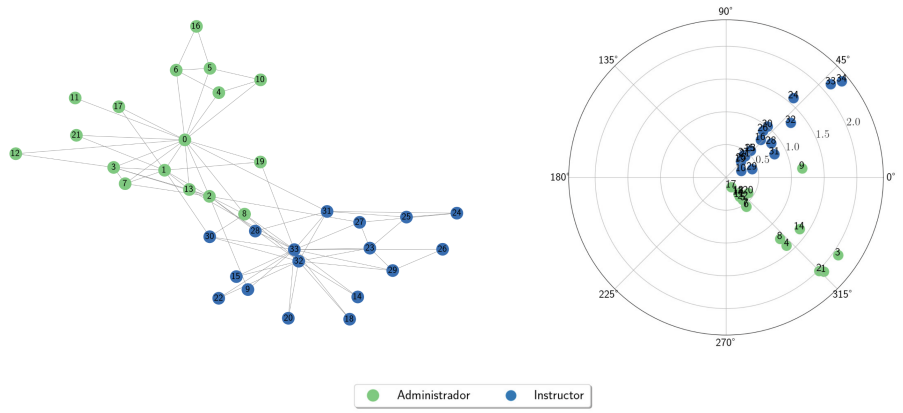


Figura 2.5: A la izquierda se muestra el grafo correspondiente a "Zachary's Karate Club" [18] identificando sus dos comunidades. A la derecha se presentan los embeddings resultantes al modelarlo como un RDPG. Se observa una clara ortogonalidad entre los dos grupos existentes, en especial si se observa los *embeddings* correspondientes administrador del club $\mathbf{x}_0$ y al instructor $\mathbf{x}_{33}$.

grafo de "Zachary's Karate Club" [18]. El mismo modela las relaciones entre los 34 miembros de un club de karate en una universidad estadounidense, donde un conflicto entre el administrador y el instructor produjo una división del club en dos grupos. Una mitad de los miembros estableció un nuevo club mientras que la otra mitad se quedó en el club original. Esta división es claramente visible en el gráfico de la derecha de la Figura 2.5, donde se observa la ortogonalidad entre los embeddings correspondientes al administrador del club $\mathbf{x}_0$ y al instructor $\mathbf{x}_{33}$.

Extensiones del modelo RDPG

Por definición, el modelo RDPG asume que la matriz de probabilidad de interconexiones $\mathbf{P} = \mathbf{X}\mathbf{X}^T$ es una matriz semi-definida positiva, y por ende presenta una limitación no menor de no poder aplicarse en otro caso. Por esta razón, se propone una generalización del mismo que resuelve dicha limitante permitiendo modelar los casos donde $\mathbf{P}$ tenga valores propios negativos. En dichos casos, se manifiestan comportamientos heterofilios, donde los nodos son más propensos a conectarse con nodos de una clase diferente a la propia.

El modelo *Generalized Random Dot Product Graphs* (GRDPG) [47] propone considerar un producto interno *indefinido* de los *embeddings* $\mathbf{X}_i$ y $\mathbf{X}_j$ definido como $\mathbf{X}_i^T \mathbf{I}_{p,q} \mathbf{X}_j$ donde $\mathbf{I}_{p,q}$ es una matriz diagonal compuesta por p valores $+1$ seguidos por q valores -1 en su diagonal, siendo $p \geq 1$ y $q \geq 0$ dos enteros que satisfacen $p+q = d$. La matriz de probabilidades de conexión estará dada entonces por $\mathbf{P} = \mathbf{X}_i^T \mathbf{I}_{p,q} \mathbf{X}_j$.

Definición 7 (Generalized Random Dot Product Graphs) *Sea $\mathcal{X} \subset \mathbb{R}^d$ tal que $\mathbf{x}^T \mathbf{I}_{p,q} \mathbf{y} \in [0,1]$ para todo $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ y $\mathcal{F}$ una distribución conjunta en $\mathcal{X}^n$.*

Decimos que el grafo corresponde a un modelo GRDPG si, para una matriz de posiciones latentes $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_j] \in \mathbb{R}^{n \times d}$, la matriz de adyacencia $\mathbf{A}$ cumple que:

$$\mathbf{A}_{ij} | \mathbf{X} \sim Ber(\mathbf{x}_i^T \mathbf{I}_{p,q} \mathbf{x}_j),$$

para todo $i < j$.

Otra limitante del modelo RDPG es que solamente puede representar grafos no dirigidos, dado que el producto $\mathbf{X}\mathbf{X}^T$ no puede generar matrices $\mathbf{P}$ asimétricas con

probabilidades distintas de un lado y del otro. En grafos dirigidos, las aristas se definen como pares ordenados (i, j) con $i, j \in \mathcal{V}$. Como las aristas (i, j) y (j, i) son diferentes también lo pueden ser las probabilidades $\mathbf{P}_{ij}$ y $\mathbf{P}_{ji}$. Para poder extender el modelo RDPG a grafos dirigidos, surge el modelo *Directed Random Dot Product Graphs* (DRDPG) [36]. En este, cada nodo i de un grafo dirigido tendrá dos vectores asociados, denotados por $\mathbf{x}_i^l$ y $\mathbf{x}_i^r$, permitiendo que la matriz de probabilidades de conexión se convierta en una matriz asimétrica tal que:

$$\mathbf{P} = \mathbf{X}^l(\mathbf{X}^r)^T. \tag{2.3}$$

Estimación de las posiciones latentes

Dada una realización de un grafo, se busca encontrar la matriz de *embeddings* $\mathbf{X} \in \mathbb{R}^{x \times d}$ que mejor aproxime la matriz de adyacencia $\mathbf{A}$. Si bien hace sentido recurrir a un estimador de máxima verosimilitud (MLE) para determinar los mismos, este método no escala para un número de nodos n grande [6].

$$\hat{\mathbf{X}}_{ML} = \text{argmax}_{\mathbf{X}} \prod_{i<j} (\mathbf{x}_i^T \mathbf{x}_j)^{\mathbf{A}_{ij}} (1 - x_i^T \mathbf{x}_j)^{1-\mathbf{A}_{ij}}. \tag{2.4}$$

Como $\mathbf{E}[\mathbf{A}|\mathbf{X}] = \mathbf{P}$, una alternativa viable es hacer una regresión por mínimos cuadrados (LS, por sus siglas en inglés) de modo que:

$$\hat{\mathbf{X}}_{LS} \in \text{argmin}_{\mathbf{X} \in \mathbb{R}^{N \times d}} \|\mathbf{A} - \mathbf{X}\mathbf{X}^T\|_F^2. \tag{2.5}$$

Los *Adjacency Spectral Embeddings* (ASE) son un método para estimar los embeddings del modelo RDPG mediante a la descomposición espectral de la matriz de adyacencia.

Definición 8 (Adjacency Spectral Embeddings) *Sea la descomposición espectral de la matriz de adyacencia* $\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$*, donde* $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]$ *es la matriz ortogonal de vectores propios de* $\mathbf{A}$ *y* $\mathbf{\Lambda} = diag(\lambda_1, \ldots, \lambda_n)$ *la matriz diagonal que contiene los valores propios correspondientes tales que* $\lambda_1 \geq \cdots \geq \lambda_n$*.*

Dado un entero positivo $d \geq 1$*, el ASE de* $\mathbf{A}$ *en* $\mathbb{R}^d$ *está dado por* $\hat{\mathbf{X}}_{LS} = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}^{1/2}$ *siendo* $\hat{\mathbf{\Lambda}} = diag(\lambda_1, \ldots, \lambda_d)$ *la matriz diagonal con los* d *valores propios más grandes en magnitud (considerando su signo) y* $\hat{\mathbf{U}} = [\mathbf{u}_1, \ldots, \mathbf{u}_d]$ *una matriz cuyas columnas son los vectores propios correspondientes.*

La mejor aproximación de rango $d \geq 1$ semi-definida positiva de $\mathbf{A}$ está dada por:

$$\mathbf{A} \approx \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}\hat{\mathbf{U}}^T = \hat{\mathbf{U}}\hat{\mathbf{\Lambda}}^{1/2}\hat{\mathbf{\Lambda}}^{1/2}\hat{\mathbf{U}}^T = \hat{\mathbf{X}}_{LS}\hat{\mathbf{X}}_{LS}^T. \tag{2.6}$$

Dado que el producto interno es invariante a rotaciones, no se tiene una única solución al problema, siendo el *embedding* de un RDPG identificable modulo rotaciones.

Por su parte, ASE no tienen en cuenta la diagonal de ceros de la matriz de adyacencia $\mathbf{A}$. Por lo que, la formulación correcta del problema a optimización a resolver está dada por la siguiente expresión:

$$\hat{\mathbf{X}} \in \text{argmin}_{\mathbf{X} \in \mathbb{R}^{N \times d}} \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}\mathbf{X}^T)\|_F^2, \tag{2.7}$$

donde $\mathbf{M}$ es una matriz máscara binaria con ceros en su diagonal y unos en sus otras entradas: $\mathbf{M} = \mathbf{1}\mathbf{1}^T - \mathbf{I}$. Esta formulación puede resolverse utilizando métodos iterativos como se cubrirá más adelante en la Sección 4.5.2.

La formulación anterior, a su vez puede extenderse al modelo GRDPG de la siguiente forma:

$$\hat{\mathbf{X}} \in \text{argmin}_{\mathbf{X} \in \mathbb{R}^{N \times d}} \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}\mathbf{I}_{p,q}\mathbf{X}^T)\|_F^2, \tag{2.8}$$

donde $\mathbf{I}_{p,q}$ es una matriz diagonal compuesta por p valores $+1$ seguidos por q valores -1 en su diagonal, tales que $p+q=d$ con d la dimensión del *embedding*. La incorporación del producto interno *indefinido* $\mathbf{X}\mathbf{I}_{p,q}\mathbf{X}^T$ hace que puedan producirse varias soluciones de estos *embeddings* que deriven en la misma solución (además de las previamente rotaciones). De hecho, de la propia Definición 7, se desprende que la distribución condicional de $\mathbf{A}$ dado $\mathbf{X}=[\mathbf{x}_1,\ldots,\mathbf{x}_n]$ permanece inalterada si $\mathbf{X}$ fuese reemplazado por $\mathbf{X}'=[\mathbf{Q}\mathbf{x}_1,\ldots,\mathbf{Q}\mathbf{x}_n]$, para cualquier matriz $\mathbf{Q}\in\{\mathbf{M}\in\mathbb{R}^{d\times d}:\mathbf{M}^T\mathbf{I}_{p,q}\mathbf{M}=\mathbf{I}_{p,q}\}$, conocido como el grupo ortogonal *indefinido*. Por lo tanto, los *embeddings* de un GRDPG son identificables hasta tal transformación [47].

2.2. Resumen del capítulo

En este capítulo se presentó una breve introducción a los grafos, los cuales son elementos fundamentales en el campo del aprendizaje automático para analizar datos donde las relaciones entre pares de elementos es importante. Se proporcionó una definición formal de estas estructuras así como de su representación algebraica a través de la matriz de adyacencia.

Por su parte, también se discutieron modelos estadísticos para la generación de grafos, como los modelos de grafos aleatorios que generan grafos de manera probabilística, destacando el modelo Erdös-Rényi y Watts-Strogatz, así como modelos de posiciones latentes que determinan la conexión entre nodos basada en su cercanía en un espacio latente, incluyendo el modelo *Stochastic Block Model* (SBM) y el *Random Dot Product Graph* (RDPG). Se introdujeron técnicas para estimar posiciones latentes, como el *Adjacency Spectral Embedding* (ASE), que utiliza descomposición espectral para ubicar grafos en un espacio euclidiano. Además, se mencionaron extensiones del modelo RDPG para incluir conectividad heterofílica y grafos dirigidos, adaptando el modelo para representar mejor la diversidad de estructuras de red reales.

Capítulo 3

Graph Neural Networks

Los métodos tradicionales para procesar datos y entrenar modelos de aprendizaje profundo (DL, por sus siglas en inglés) no están diseñados para trabajar con grafos, lo que hace necesaria la creación de nuevas arquitecturas que permitan lidiar con estos escenarios. En este contexto, surgen las redes neuronales de grafos o Graph Neural Networks (GNNs), una arquitectura de redes neuronales profundas utilizadas para datos estructurados en forma de grafos. La idea principal detrás de las GNNs es generar representaciones de nodos que consideren tanto la estructura del grafo como cualquier información de características disponible. Las características o atributos a nivel de nodo suelen representarse como una matriz $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$, donde se asume que el ordenamiento de los nodos es consistente con el ordenamiento de los mismos en la matriz de adyacencia $\mathbf{A}$. Las filas de $\mathbf{X}$ suelen también interpretarse como señales en el grafo.

Las GNNs pueden ser modeladas de diversas formas, en particular en este capítulo se hará foco en dos de ellas: el modelo de *Neural Message Passing* (NMP) y la interpretación de GNNs como convoluciones sobre grafos.

3.1. Neural Message Passing

El modelo NMP postula que las GNNs emplean una técnica de pasaje de mensajes mediante la cual los nodos intercambian mensajes expresados como vectores y que se actualizan utilizando redes neuronales [23]. En cada iteración de "pasaje de mensajes"de una GNN, se actualiza el vector de *embedding* oculto $\mathbf{h}_u^{(k)}$ correspondiente al nodo $u \in \mathcal{V}$ de acuerdo a la información agregada de los vecinos de u, los cuales se denotan como $\mathcal{N}(u)$. Este proceso puede expresarse de la siguiente forma:

$$\mathbf{h}_u^{(k+1)} = \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \text{AGGREGATE}^{(k)}(\{\mathbf{h}_v^{(k)}, \forall v \in \mathcal{N}(u)\})\right) \tag{3.1}$$

$$= \text{UPDATE}^{(k)}\left(\mathbf{h}_u^{(k)}, \mathbf{m}_{\mathcal{N}(u)}^{(k)}\right), \tag{3.2}$$

donde UPDATE y AGGREGATE son funciones diferenciables arbitrarias (i.e. redes neuronales) y $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$) es el mensaje que es agregado a partir del vecindario de u, $\mathcal{N}(u)$. La Figura 3.1 ejemplifica este proceso, mostrando como la información del nodo A se combina con la información agregada de sus vecinos (B, C y D), que a su vez, se basa en combinar la información de cada uno de estos con la de sus respectivos vecinos.

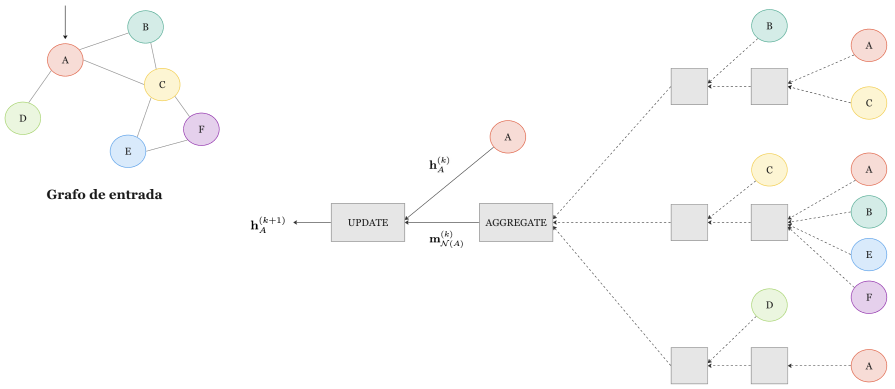


Figura 3.1: Ejemplo del funcionamiento de un modelo de pasaje de mensajes compuesto por dos capas. El modelo recopila mensajes de los nodos vecinos al nodo A (B, C y D), y a su vez, los mensajes provenientes de estos vecinos se basan en la información recopilada de sus respectivos vecindarios, y así sucesivamente [23].

En cada iteración k de la GNN, la función AGGREGATE toma como entrada un conjunto de *embeddings* ocultos del vecindario de u y genera un mensaje $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ resultante de la agregación de la información de estos. Luego, la función UPDATE combina el mensaje $\mathbf{m}_{\mathcal{N}(u)}^{(k)}$ con el *embedding* de la iteración previa $\mathbf{h}_u^{(k)}$ del nodo u, para así generar el nuevo *embedding* $\mathbf{h}_u^{(k+1)}$. El *embedding* inicial de u para la iteración $k = 0$, se define como la señal inicial, siendo $\mathbf{h}_u^{(0)} = \mathbf{x}_u \ , \ \forall \ u \in \mathcal{V}$. Luego de K iteraciones (o capas) de la GNN, la salida de la capa final puede ser utilizada para definir los *embeddings* finales de cada nodo:

$$\mathbf{z}_u = \mathbf{h}_u^{(K)}, \forall \ u \in \mathcal{V}. \tag{3.3}$$

En cada iteración, cada nodo agrega información de sus vecinos locales. En la medida que estas iteraciones progresan, los *embeddings* contienen más y más información. En la primer iteración ($k = 1$) cada *embedding* contiene información sobre los nodos vecinos que se encuentran a 1 salto de distancia. En la segunda iteración ($k = 2$), se agrega información sobre los nodos vecinos a 2 saltos de distancia y así sucesivamente. En general, luego de k iteraciones cada *embedding* contiene información sobre los nodos vecinos a k saltos de distancia. Por lo tanto, estos *embeddings* codifican dos tipos de información: información estructural sobre el grafo e información de atributos (o *features*) de los propios nodos.

La expresión (3.2) utiliza funciones abstractas de actualización y agregación. Una definición más concreta de las mismas estará dada por la siguiente expresión:

$$\mathbf{h}_u^{(k+1)} = \sigma\left(\mathbf{W}_0^{(k+1)}\mathbf{h}_u^{(k)} + \mathbf{W}_1^{(k+1)} \sum_{j \in \mathcal{N}(u)} h_j^{(k)} + \mathbf{b}^{(k+1)}\right), \tag{3.4}$$

donde $\mathbf{W}_0^{(k+1)}, \mathbf{W}_1^{(k+1)} \in \mathbb{R}^{d^{(k+1)} \times d^{(k)}}$ son matrices de parámetros entrenables, σ es una función no lineal punto a punto (tanh, ReLU, etc) y $\mathbf{b}^{(k+1)} \in \mathbb{R}^{d^{(k+1)}}$ el sesgo. La expresión (3.4) es el equivalente en grafos de un *Multi Layer Perceptron* (MLP), puesto a que involucra una serie de operaciones lineales seguidas por una no linealidad. Esta definición a nivel de nodo, también puede expresarse a nivel de grafo de la siguiente

manera:

$$\mathbf{H}^{(k+1)} = \sigma\left(\mathbf{H}^{(k)}\mathbf{W}_0^{(k+1)} + \mathbf{A}\mathbf{H}^{(k)}\mathbf{W}_1^{(k+1)}\right), \tag{3.5}$$

donde $\mathbf{H}^{(k+1)} \in \mathbb{R}^{n\times d}$ es la matriz de *embeddings* correspondientes a la capa $k+1$ y $\mathbf{A}$ es la matriz de adyacencia del grafo.

3.2. Convoluciones en grafos

Desde la perspectiva del procesamiento de señales, los grafos se utilizan como una descripción matemática de las topologías de redes, mientras que los datos pueden interpretarse como señales transmitidas sobre estos. El área de procesamiento de señales en grafos (GSP por su siglas en inglés) tiene como objetivo el procesamiento de estos datos teniendo en cuenta la estructura de la red subyacente. En GSP se extienden los conceptos de Transformada de Fourier, convoluciones y filtros para el procesamiento de señales en grafos de modo de tener en cuenta la topología subyacente.

Las *Graph Convolutional Neural Networks* (GCNNs) son un tipo particular de GNN que se basan en realizar convoluciones sobre un grafo para incorporar eficientemente la estructura del mismo en el proceso de aprendizaje. Las mismas consisten en una concatenación de capas, en donde se aplica una convolución seguida por una no linealidad. Las GCNNs exhiben propiedades interesantes, como ser la equivarianza a permutaciones, la cual permite explotar simetrías topológicas en el grafo; y la estabilidad ante permutaciones, que garantiza cierta robustez en la salida frente a pequeños cambios en la estructura del grafo. Ambos resultados permiten que las GCNNs escalen a grafos grandes, así como transferir el conocimiento adquirido a otros escenarios similares [16].

Sea el grafo $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$, se define la matriz simétrica $\mathbf{S} \in \mathbb{R}^{N\times N}$ conocida como *Graph Shift Operator* (GSO), tal que:

$$[\mathbf{S}]_{ij} = s_{ij} = 0 \text{ si } (j,i) \notin \mathcal{E} \ , \ j \neq i. \tag{3.6}$$

Los operadores GSO más habituales suelen ser la matriz de adyacencia, el Laplaciano del grafo y matrices de Markov. Donde, se denomina Laplaciano de un grafo a la matriz simétrica $\mathbf{L} \in \mathcal{M}_{n\times n}$ tal que:

$$\begin{cases} d_i & \text{si } i = j, \\ -1 & \text{si } (i,j) \in \mathcal{E}, \\ 0 & \text{en otro caso.} \end{cases} \quad , \tag{3.7}$$

siendo d_i el grado del nodo i. Mientras que, las matrices de Markov, también conocidas como matrices estocásticas, son aquellas matrices $n\times n$ cuyas entradas son no negativas y la suma de los elementos de sus columnas es igual a 1.

Los datos forman una señal en el grafo $\mathbf{x} \in \mathbb{R}^N$, donde la i-ésima entrada $[\mathbf{x}]_i = x_i$ es el dato correspondiente al nodo i. La señal puede ser desplazada sobre los nodos del grafo usando el operador $\mathbf{S}$ tal que la i-ésima entrada $[\mathbf{S}\mathbf{x}]_i$ está dada por:

$$[\mathbf{S}\mathbf{x}]_i = \sum_{j=1}^{N} [\mathbf{S}]_{ij}\mathbf{x}_j = \sum_{j\in\mathcal{N}(i)} s_{ij}x_j. \tag{3.8}$$

Esta operación $\mathbf{S}\mathbf{x}$ combina linealmente la información contenida en diferentes vecindarios. Aplicando $\mathbf{S}$ nuevamente se combina la información de los vecinos a dos saltos de distancia, y así sucesivamente. Teniendo en cuenta esto, la convolución en grafos se define como una combinación lineal de las versiones desplazadas de la señal mediante $\mathbf{S}$. Dado un conjunto de parámetros $\mathbf{h} = [\mathbf{h}_0, \cdots, \mathbf{h}_k]$, la convolución del grafo está dada por

$$\mathbf{y} = \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} h_k \mathbf{S}^k \mathbf{x}. \tag{3.9}$$

Esta definición a nivel de nodo, también puede expresarse a nivel de grafo de la siguiente manera:

$$\mathbf{Y} = \sum_{k=0}^{K} \mathbf{S}^k \mathbf{X} \mathbf{H}_k. \tag{3.10}$$

La señal contiene un resumen de la información de vecindarios a k saltos, donde h_k pondera dicho resumen. Esta operación es local, dado que $\mathbf{S}^k\mathbf{x} = \mathbf{S}(\mathbf{S}^{k-1}\mathbf{x})$ que implica k intercambios de información con vecindarios a un salto. La convolución en el grafo puede interpretarse entonces como el filtrado de la señal $\mathbf{x}$ con un filtro FIR, $\mathbf{H}(\mathbf{S})$, siendo los parámetros h_k los coeficientes del filtro.

3.2.1. Transformada de Fourier en grafos

Para analizar las convoluciones en grafos en el dominio de la frecuencia, se considera la descomposición espectral del operador GSO, $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^H$, donde $\mathbf{V} \in \mathbb{R}^{n\times n}$ es la matriz ortogonal de vectores propios de $\mathbf{S}$ y $\mathbf{\Lambda} \in \mathbb{R}^{n\times n}$ es la matriz diagonal valores propios correspondientes ordenados de modo que $\lambda_1 \leq \lambda_2 \leq \lambda_N$.

Asumiendo que $\mathbf{S}$ es de rango completo, los vectores propios $\mathbf{V}_i$ conforman una base en frecuencia del grafo y pueden ser interpretados como modos de oscilación del mismo. Mientras que los valores propios λ_i pueden ser interpretados como las frecuencias del grafo. Cualquier señal $\mathbf{x}$ puede ser expresada en términos estos modos de oscilación:

$$\tilde{\mathbf{x}} = \mathbf{V}^H \mathbf{x}. \tag{3.11}$$

La operación (3.11) se la conoce como Transformada de Fourier en Grafos (GFT). Cada entrada $\tilde{\mathbf{x}}_i$ denota un coeficiente de Fourier asociado a una frecuencia λ_i y cuantifica la contribución del modo $\mathbf{V}_i$ a la señal $\mathbf{x}$. La GFT $\tilde{\mathbf{x}}$ entonces, es una representación de la señal $\mathbf{x}$ en el dominio espectral (frecuencia) del grafo, y puede interpretarse como la proyección de $\mathbf{x}$ en el espacio de vectores propios. En particular, la GFT de la señal de salida $\mathbf{y}$ de la ecuación (3.9) está dada por:

$$\tilde{\mathbf{y}} = \mathbf{V}^H \mathbf{y} = \mathbf{V}^H \mathbf{H}(\mathbf{S})\mathbf{x} = \sum_{k=0}^{K} h_k \mathbf{V}^H \mathbf{S}^k \mathbf{x}. \tag{3.12}$$

Sustituyendo $\mathbf{S}$ por su descomposición espectral, se obtiene que

$$\tilde{\mathbf{y}} = \sum_{k=0}^{K} h_k \mathbf{V}^H \mathbf{V} \mathbf{\Lambda}^k \mathbf{V}^H \mathbf{x} = \sum_{k=0}^{K} h_k \mathbf{\Lambda}^k \tilde{\mathbf{x}} = \mathbf{H}(\mathbf{\Lambda})\tilde{\mathbf{x}}, \tag{3.13}$$

donde $\mathbf{H}(\mathbf{\Lambda})$ es una matriz diagonal cuyos elementos $h(\lambda_i)$ están dados por la función $h: \mathbb{R} \to \mathbb{R}$, siendo

$$h(\lambda) = \sum_{k=0}^{K} h_k \lambda^k. \tag{3.14}$$

La función dada por (3.14) es la respuesta en frecuencia del filtro $\mathbf{H}$ del grafo, la cual está determinada únicamente por los coeficientes h del mismo.

A modo de ejemplo, se considera un ciclo de una señal periódica $\mathbf{x}$, para el cual se toman n muestras x_n como se muestra en la Figura 3.2. Esto puede reinterpretarse como considerar un grafo circular dirigido de n nodos, donde cada nodo corresponde

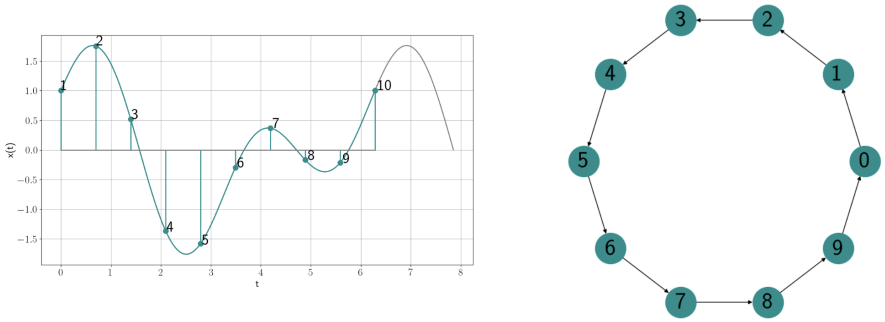


Figura 3.2: Reinterpretación de la DFT de una señal $\mathbf{x}$ como la GFT de $\mathbf{x}$ considerando un grafo circular.

a una muestra de la señal y el instante de tiempo n le sigue al $n-1$. La transformada de Fourier discreta (DFT) de esta señal equivale a hacer una descomposición espectral sobre la matriz de adyacencia del grafo circular y proyectar la señal sobre el espacio de vectores propios obtenidos, es decir la GFT de $\mathbf{x}$ para dicho grafo.

3.2.2. Graph Convolutional Neural Networks

El aprendizaje de datos en grafos implica el cálculo de un mapa de representaciones $\mathbf{\Phi}(\cdot)$ entre los datos $\mathbf{x}$ y una representación objetivo $\mathbf{y} = \mathbf{\Phi}(\mathbf{x}, \mathbf{S})$. La imagen de $\mathbf{\Phi}$ se denomina espacio de representaciones y determina el espacio de posibles representaciones $\mathbf{y}$ para un $\mathbf{S}$ y $\mathbf{x}$ dados. Un ejemplo de mapa de representaciones es la convolución en el grafo $\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \mathbf{H}(\mathbf{S})\mathbf{x}$ donde el conjunto $\mathcal{H} = \{\mathbf{h}\}$ contiene los coeficientes del filtro que caracterizan espacio de representaciones.

Para aprender este mapa de representaciones es necesario considerar una función de costo $J(.)$ y un conjunto de datos de entrenamiento $\mathcal{T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{T}|}\}$. El mapa aprendido estará dado por $\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}^*)$ con

$$\mathcal{H}^* = \text{argmin}_{\mathcal{H}} \frac{1}{\mathcal{T}} \sum_{\mathbf{x} \in \mathcal{T}} J(\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H})). \tag{3.15}$$

Para la función de costo $J(.)$, típicamente se utiliza el error cuadrático medio (MSE) cuando se tratan problemas de regresión y la función *Cross Entropy Loss* en problemas de clasificación. El problema consiste en encontrar los $K+1$ coeficientes del filtro que mejor se ajusten a los datos de entrenamiento, siendo K un hiper-parámetro a definir.

Utilizar simplemente las convoluciones en grafos presenta como limitante que esta involucra únicamente mapeos lineales. Es posible incrementar la expresividad del modelo agregando una no-linealidad, dando lugar al concepto de *Graph Perceptron*. El mismo consiste en agregar una no linealidad $\sigma(.)$ a la salida de la convolución del grafo:

$$\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}) = \sigma(\mathbf{H}(\mathbf{S})\mathbf{x}) \tag{3.16}$$

donde $\mathcal{H} = \{\mathbf{h}\}$ contiene los coeficientes del filtro. La función no lineal $\sigma(.)$ suele ser una ReLU o alguna de sus variantes. En la Figura 3.3 se muestra una GNN de L capas. La GNN estará dada entonces por la concatenación de un numero finito L de capas, compuestas por una convolución (operación lineal) combinada con una operación no lineal. La salida de la capa l se calcula como:

$$\mathbf{x}_l = \sigma(\mathbf{H}_l(\mathbf{S})\mathbf{x}_{l-1}) \ , \ l = 1, \ldots, L. \tag{3.17}$$

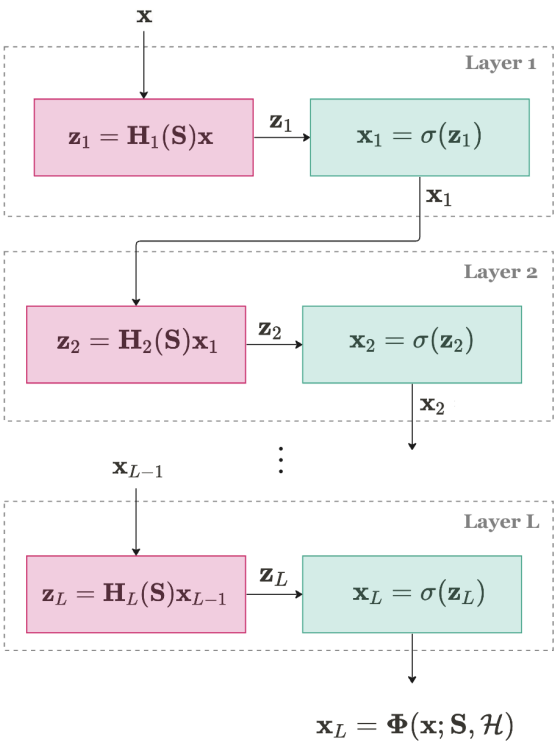


Figura 3.3: Ejemplo de una GNN de L capas. Cada bloque rojo representa un filtro de grafo lineal y cada bloque verde representa una no linealidad. La concatenación de un filtro de grafo convolucional y una no linealidad forma un Graph Perceptron.

La interpretación de las GNNs como convoluciones, permite probar algunas propiedades interesantes de las mismas, como ser la equivarianza a permutaciones, así como la estabilidad frente a perturbaciones, las cuales se cubren con mayor detalle a continuación.

Equivarianza a permutaciones

Las GNNs no se ven afectadas por el orden en el etiquetado de los nodos. Estas propiedad se la conoce como equivarianza ante permutaciones.

Teorema 1 (Equivarianza a permutaciones) *Sea* $\mathbf{P}$ *una matriz de permutaciones y la permutación de la GSO* $\hat{\mathbf{S}} = \mathbf{P}^T\mathbf{S}\mathbf{P}$ *y del vector de entrada* $\hat{\mathbf{x}} = \mathbf{P}^T\mathbf{x}$*. Toda GNN* $\mathbf{\Phi}(.)$ *cumple que:*

$$\mathbf{\Phi}(\hat{\mathbf{x}}; \hat{\mathbf{S}}, \mathcal{S}) = \mathbf{P}^T\mathbf{\Phi}(\mathbf{x}; \mathbf{S}, \mathcal{H}).$$

Este teorema implica que cualquier reordenamiento de los nodos resulta en un correspondiente reordenamiento en la salida de la GNN, lo que significa que las GNN son independientes del etiquetado de los nodos [15]. A su vez, esto sugiere que las convoluciones en grafos aprovechan simetrías inherentes presentes en la estructura de los grafos. Si el grafo exhibe múltiples nodos con topologías idénticas en sus vecindarios (simetrías), entonces el aprendizaje realizado en tales nodos puede ser transferido a

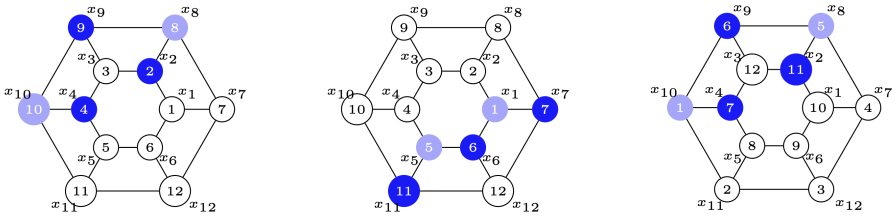


Figura 3.4: Ejemplo de como la salida de una GCNN es equivariante a las permutaciones del grafo. Esto significa independencia del etiquetado y demuestra que las GCNN explotan simetrías internas de la señal [16].

cualquier otro nodo con una topología similar en su vecindario. Esto posibilita que las GNNs puedan aprender con una menor cantidad de muestras y generalizar de manera más efectiva a señales localizadas en topologías de vecindarios similares. La Figura 3.4 ilustra este comportamiento.

Estabilidad frente a perturbaciones

Las GNNs son estables frente a perturbaciones, propiedad que expresa la robustez frente a pequeños cambios en la estructura del grafo. Una variación en la salida debido a modificaciones en el grafo subyacente se encuentra restringida por la magnitud de la perturbación [15]. En particular, es posible probar que las GNNs son estables tanto a perturbaciones absolutas como relativas. Implicando que la diferencia en la salida está acotada linealmente por la distancia entre los GSOs. Esta cota depende tanto del filtro como del modelo de perturbación y es universal para todos los grafos con la misma cantidad de nodos. A continuación se presentan los siguientes resultados ilustrativos:

Teorema 2 (Estabilidad frente a perturbaciones absolutas) *Sean* $\mathbf{S}$ *y* $\hat{\mathbf{S}}$ *GSOs tal que su distancia de perturbaciones absolutas es* $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \epsilon$*, entonces los filtros Lipschitz con constante* C *cumplen*

$$\|\mathbf{H}(\mathbf{S}) - \mathbf{H}(\hat{\mathbf{S}})\|_{\mathcal{P}} \leq C(1 + \delta\sqrt{N}) + \epsilon$$

donde $\delta = (\|\mathbf{U} - \mathbf{V}\|_2 + 1)^2 - 1 \leq 8$ *es el eigenvector misalignment constant entre los vectores propios* $\mathbf{V}$ *del grafo* $\mathbf{S}$ *y los vectores propios* $\mathbf{U}$ *de la matriz de error* $\mathbf{E}$*.*

Teorema 3 (Estabilidad frente a perturbaciones relativas) *Sean* $\mathbf{S}$ *y* $\hat{\mathbf{S}}$ *GSOs con distancia relativa* $d(\mathbf{S}, \hat{\mathbf{S}}) \leq \epsilon$*. Sea* $\mathbf{\Phi}(.; \mathbf{S}, \mathbf{H})$ *una GNN con* L *capas con todos los filtros* $\mathbf{H}$ *son integral Lipschitz con constante* C*. Entonces:*

$$\|\mathbf{\Phi}(.; \mathbf{S}, \mathbf{H}) - \mathbf{\Phi}(.; \hat{\mathbf{S}}, \mathbf{H})\| \leq 2C(1 + \delta\sqrt{N})L\|\mathbf{x}\|\epsilon + \mathcal{O}(\epsilon^2)$$

donde $\delta = (\|\mathbf{U} - \mathbf{V}\| + 1)^2 - 1$ *es el eigenvector misalignment constant entre los vectores propios* $\mathbf{V}$ *del grafo* $\mathbf{S}$ *y los vectores propios* $\mathbf{U}$ *de la matriz de error* $\mathbf{E}$*.*

Dichos resultados sugieren que las GNN funcionan eficazmente en situaciones donde el GSO del grafo original $\mathbf{S}$ es desconocido y en su lugar utilizamos una estimación de este $\hat{\mathbf{S}}$. A su vez implica que también pueden ser útiles cuando la GSO del grafo de entrenamiento $\mathbf{S}$ difiere ligeramente de la del grafo de prueba $\hat{\mathbf{S}}$, situación conocida como *transfer learning*.

3.3. Graph Attention

Una limitación no menor de las GCNNs es que su capa de agregación otorga igual importancia a todos los nodos vecinos, sin tomar en cuenta que algunos podrían ser más informativos que otros. Una variante para mejorar esta capa de agregación consiste aplicar un mecanismo de *graph attention* [7]. La idea implica asignar pesos a los nodos vecinos según su importancia, los cuales se emplean posteriormente para realizar una agregación ponderada.

La primer GNN en aplicar este mecanismo fueron las *Graph Attention Networks* (GAT) [52], las cuales utilizan coeficientes de *attention* para definir una suma ponderada de vecinos:

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v, \tag{3.18}$$

donde $\alpha_{u,v}$ denota el coeficiente de *attention* en el vecino $v \in \mathcal{N}(u)$ cuando se agrega la información en el nodo u. En la implementación original los coeficientes se definen como:

$$\alpha_{u,v} = \frac{\exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in \mathcal{N}(u)} \exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])}, \tag{3.19}$$

donde $\mathbf{a}$ es un vector de pesos de *attention* entrenable, $\mathbf{W}$ es una matriz de pesos entrenables, y $\oplus$ denota la concatenación de vectores. La Figura 3.5 detalla el mecanismo de attention empleado en el modelo GAT.

A su vez es posible adicionar múltiples coeficientes de *attention* lo que se denominan *attention heads*, dando lugar a una arquitectura de *multi-headed attention*. En dicha configuración, por cada nodo se computan K coeficientes de *attention* $\alpha_{u,v,k}$, utilizando capas de independientes. Los mensajes agregados mediante los diferentes coeficientes son luego transformados y combinados en un paso de agregación. Usualmente eso consiste en un proyección lineal seguida por una operación de concatenación.

$$\mathbf{m}_{\mathcal{N}(u)} = [\mathbf{a}_1 \oplus \cdots \oplus \mathbf{a}_K], \tag{3.20}$$

$$\mathbf{a}_k = \mathbf{W}_k \sum_{v \in \mathcal{N}(u)} \alpha_{u,v} \mathbf{h}_v. \tag{3.21}$$

3.3.1. Graph Transformers

Las GNN con *multi-headed attention* se asimilan a la arquitectura de los modelos de *Transformers* [51] utilizados en el ámbito del Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés). De hecho, la arquitectura del *Transformer* clásico es equivalente a una GNN que recibe un grafo completamente conectado como entrada. La conexión entre las GNN y los *Transformers* ha sido explotada en numerosos trabajos en los últimos años. La primera versión de *Graph Transformers* (GT) aplicada a grafos homogéneos se basa en dos conceptos claves: la *sparcity* y la incorporación de *Positional Encodings* (PE) [14].

El primero refiere al uso de un mecanismo de *attention* más *sparse*, en lugar del completamente conectado. En el contexto de NLP, cada oración es considerada como un grafo completamente conectado, esto tiene dos motivos. Primero, es difícil encontrar conexiones significativas entre las palabras en una oración. La dependencia de una palabra en una oración respecto a otra palabra puede variar según el contexto, la perspectiva del usuario y la aplicación en cuestión. Por lo tanto, tiene sentido conectar cada palabra con todas las demás para aplicar el mecanismo de *attention*. Segundo, en el ámbito de NLP los grafos basados en oraciones o documentos suelen contener entre

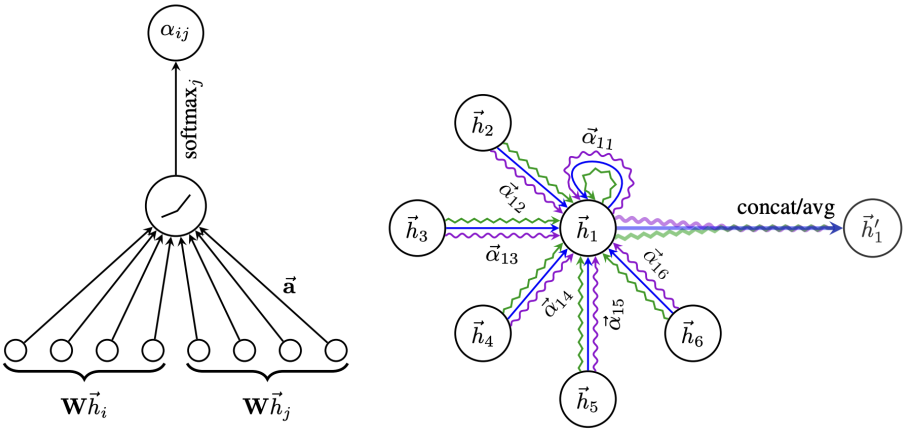


Figura 3.5: A la izquierda se detalla el mecanismo de attention empleado en el modelo GAT. Mientras que a la derecha se ilustra el mecanismo de multi-headed attention con K=3 sobre el nodo 1 [52].

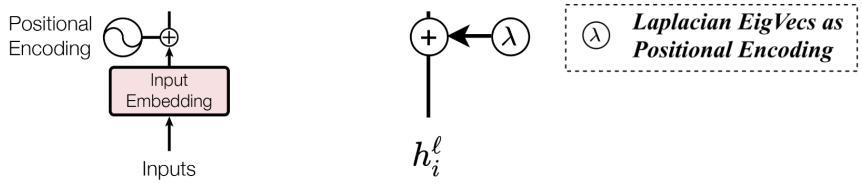


Figura 3.6: Comparación de los Positional Encodings (PE) del Transformer original y los propuestos por *Graph Transformer*.

decenas y centenas de nodos, haciendo que el mecanismo de *attention* sea computacionalmente factible. En cambio, en el caso de grafos genéricos, su estructura varía según cada aplicación, pudiendo tener tamaños del orden de los varios millones de nodos. Esto hace inviable considerar un mecanismo de *attention* completamente conectado. Para solventar esto, una estrategia de implementación consiste en aplicar una máscara binaria en la capa de attention utilizando la matriz de adyacencia del grafo, asegurando así que la información se agregue de forma local, únicamente entre nodos conectados en el grafo. Aún así, los GT introducen una dependencia cuadrática en el cálculo, lo que aumenta la complejidad computacional y los tiempos de ejecución en comparación con las redes GCNNs clásicas.

En cuanto al segundo concepto, el mismo refiere a la extensión del concepto de Positional Encodings (PEs) utilizados en la arquitectura de *Transformers* clásica para el contexto de grafos. En NLP, los modelos de *Transformers* son provistos de PE para cada palabra, los cuales permiten codificar información secuencial entre las mismas. En el contexto de grafos, se propone utilizar representaciones latentes de los nodos (*embeddings*) que codifiquen información de la posición de cada nodos dentro el grafo. En particular, en el diseño original de GT se utilizan *Laplacian Positional Encodings* (LapPE) pre-computados, los cuales cubriremos en mayor detalle en el siguiente capítulo. Estos PE, se concatenan a la entrada de la red junto a los vectores de atributos de los nodos, de forma análoga al *Transformer* clásico. Ver Figura 3.6.

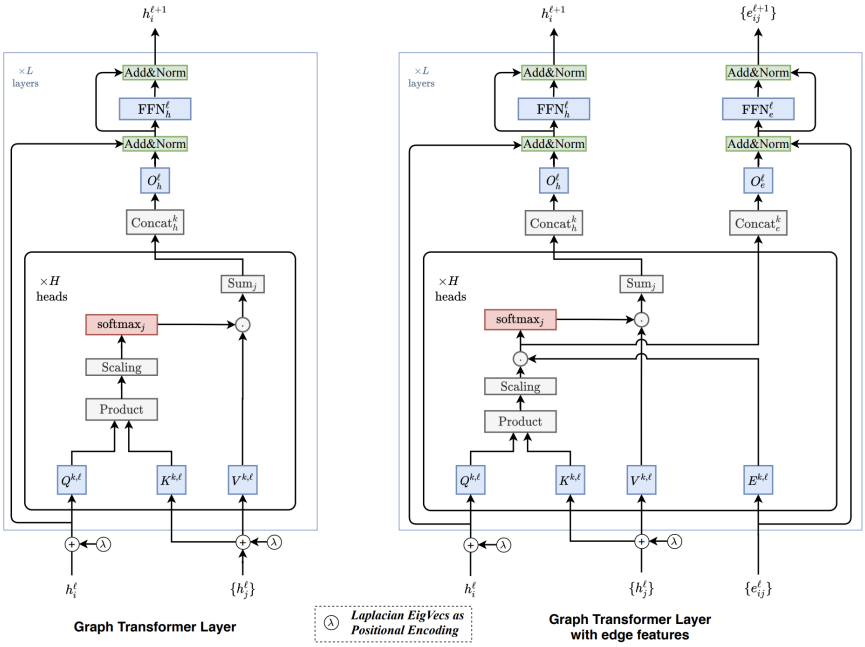


Figura 3.7: Diagrama de Bloques del modelo *Graph Transformer* [14]. Los LapPE se agregan a los *embeddings* de nodos de entrada previo a pasarlos por la primera capa.

La Figura 3.7 detalla la arquitectura propuesta en el modelo *Graph Transformer*, resaltando sus componentes principales: la matriz de LapPE a la entrada, el mecanismo de *multi-head attention* restringido a la matriz de adyacencia, la normalización por *batch* y el módulo *Feed Forward Network* (FFN). El mecanismo de *multi-head attention* se basa en la siguiente expresión:

$$\hat{h}_i^{l+1} = \mathbf{O}_h^l \Big\|_{k=1}^{H} \left(\sum w_{ij}^{k,l} \mathbf{V}^{k,l} h_j^l\right), \tag{3.22}$$

donde

$$w_{ij}^{k,l} = \text{softmax}_i \left(\frac{\mathbf{Q}^{k,l} h_i^l \cdot \mathbf{K}^{k,l} h_j^l}{\sqrt{d_k}} \right) \tag{3.23}$$

y $\mathbf{Q}^{k,l}$, $\mathbf{K}^{k,l}$ $\mathbf{V}^{k,l} \in \mathbb{R}^{d_k \times d}$, $\mathbf{O}_h^l \in \mathbb{R}^{d \times d}$, H es el número de *attention heads* y $\|$ denota la operación de concatenación. La salida de la capa de *attention* $\hat{h}_i^{l+1}$ es luego pasada por una FFN, seguido por conexiones residuales y capas de normalización por *batch*.

3.4. Resumen del capítulo

En este capítulo se brindó una introducción detallada a las *Graph Neural Networks* (GNNs). En particular, se presentaron dos formas diferentes de modelar las mismas: *Neural Message Passing* (NMP) y *Graph Convolutional Neural Networks* (GCNNs). El NMP permite definir el intercambio de mensajes entre nodos expresados en forma de vectores los cuales se actualizan mediante redes neuronales. Por otro lado, las GCNNs consisten de convoluciones sobre un grafo que permiten incorporar eficientemente la

estructura del mismo en el proceso de aprendizaje. Las mismas exhiben propiedades interesantes, como ser la equivarianza y estabilidad ante permutaciones.

A su vez, se introdujeron las *Graph Attention Networks* (GAT), que mejoran la agregación de información al ponderar la importancia de los nodos vecinos y los *Graph Transformers* (GT) como una generalización del clásico modelo de *Transformers* utilizado en el ámbito del Procesamiento del Lenguaje Natural (NLP).

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Graph Representation Learning

El área de *Graph Representation Learning* (GRL) busca aprender *embeddings* de baja dimensión que codifiquen información estructural del grafo. El objetivo es optimizar este mapeo para que las relaciones geométricas en el espacio de *embeddings* reflejen la estructura del grafo original. Las representaciones aprendidas pueden utilizarse como entrada para otras tareas de aprendizaje automático posteriores.

A diferencia de los métodos clásicos vistos en el Capítulo 2, donde se abordaba el problema como un paso de pre-procesamiento mediante el uso de estadísticas diseñadas para extraer información estructural, el GRL considera este problema como una tarea de aprendizaje automático en sí misma, empleando un enfoque basado en datos para aprender *embeddings* que codifican la estructura del grafo [24].

En la mayoría de los casos, este mapeo se realiza de manera no supervisada, haciendo uso únicamente de la información en $\mathbf{A}$ y $\mathbf{X}$, sin conocimiento de una tarea particular de aprendizaje posterior. Sin embargo, también existen enfoques para el aprendizaje de representaciones supervisadas, donde los modelos utilizan etiquetas de clasificación o regresión para optimizar los *embeddings*.

4.1. Perspectiva Encoder-Decoder

En el marco de GRL, se plantea el uso de dos operaciones: *encoder* y *decoder*. El *encoder* se encarga de codificar cada nodo del grafo en un *embedding* de baja dimensión. El *decoder* mapea toma la representación generada por el *encoder* y la utiliza para reconstruir la información de cada nodo y sus vecinos en el grafo original.

4.1.1. Encoder

Formalmente, el *encoder* es una función que mapea nodos $v \in \mathcal{V}$ a *embeddings* $\mathbf{z}_v \in \mathbb{R}^d$. La Figura 4.1 ilustra este proceso. En el caso más simple, el *encoder* realiza la siguiente asignación:

$$\text{ENC} : \mathcal{V} \to \mathbb{R}^d, \tag{4.1}$$

que implica tomar los identificadores de nodo como entrada y generar un *embedding*. Un ejemplo sencillo es el enfoque de *shallow embeddings*, donde la función de mapeo simplemente hace una búsqueda del identificador de nodo en la matriz de *embeddings*:

$$\text{ENC}(v) = \mathbf{Z}[v], \tag{4.2}$$

donde $\mathbf{Z} \in \mathbb{R}^{|\mathcal{V}| \times d}$ es la matriz que contiene los *embeddings* de todos los nodos y $\mathbf{Z}[v]$ indica la fila de $\mathbf{Z}$ correspondiente al nodo v.

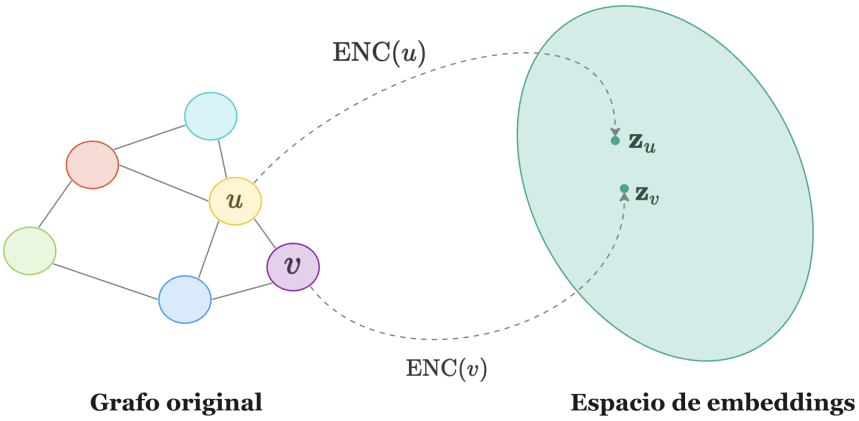


Figura 4.1: Ilustración del problema de *embeddings* de nodos. El objetivo es aprender un *encoder* (ENC), que mapea nodos a un espacio de *embeddings* de baja dimensión. Estos *embeddings* se optimizan para que las distancias en el espacio de *embeddings* reflejen las posiciones relativas de los nodos en el grafo original.

Otro ejemplo bastante intuitivo, es considerar el modelo RDPG cubierto en la Sección 2.1.3. En ese caso, la codificación de un nodo v estará dada por la correspondiente fila de la matriz $\mathbf{X}$ que determina la matriz de probabilidad de conexiones $\mathbf{P}$. Es decir,

$$\text{ENC}(v) = \mathbf{X}[v], \tag{4.3}$$

donde $\mathbf{P} = \mathbf{X}\mathbf{X}^T$ y $\mathbf{X}[v]$ corresponde a la fila de $\mathbf{X}$ correspondiente al nodo v.

4.1.2. Decoder

El rol del *decoder* es reconstruir ciertas estadísticas del grafo original a partir de los *embeddings* de nodos generados por el *encoder*. La práctica estándar es usar *decoders* que buscan predecir la relación entre pares de nodos (*pairwise decoders*):

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+. \tag{4.4}$$

El objetivo es optimizar el *encoder* y el *decoder* para minimizar la pérdida de reconstrucción tal que:

$$\text{DEC}(\text{ENC}(u), \text{ENC}(v)) = \text{DEC}(\mathbf{z}_u, \mathbf{z}_v) \approx \mathbf{S}[u, v], \tag{4.5}$$

donde $\mathbf{S}[u, v]$ es un medida de similitud entre nodos. Por ejemplo, si buscamos predecir si dos nodos son vecinos entonces $\mathbf{S}[u, v] \triangleq \mathbf{A}[u, v]$. La Figura 4.2 resume este proceso.

Volviendo al ejemplo del modelo RDPG, el *decoder* corresponderá a la operación de producto interno entre los *embeddings* $\mathbf{X}$, la cual permite reconstruir la matriz de probabilidad de interconexiones $\mathbf{P}$ y por ende obtener una aproximación de la matriz de adyacencia del grafo.

Una práctica usual para lograr la reconstrucción planteada por la ecuación (4.5) consiste en minimizar una función de costo $\mathcal{L}$ en base a un conjunto de muestras de entrenamiento $\mathcal{D}$:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \ell(\text{DEC}(\mathbf{z}_u, \mathbf{z}_v), \mathbf{S}[u, v]), \tag{4.6}$$

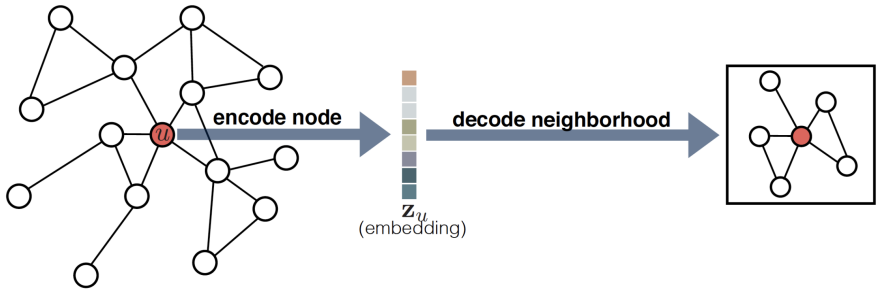


Figura 4.2: Resumen del framework *encoder-decoder*. El *encoder* mapea el nodo u a un *embedding* de baja dimensión z_u. Luego, el *decoder* utiliza z_u para reconstruir la información del vecindario local de u [23].

donde $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ es una función de pérdida que mide la discrepancia entre la estimación resultante del *decoder* y los valores reales de $\mathbf{S}[u, v]$. La misma dependerá de la definición del *decoder* y de la medida de similitud $\mathbf{S}$.

4.2. Métodos basados en factorización de matrices

Dentro de la perspectiva *encoder-decoder*, existen algunos métodos inspirados en reducción de dimensionalidad vía factorización de matrices de bajo rango (*low-rank matrix factorization*), para aprender una aproximación de baja dimensión de la matriz de similitud de nodos $\mathbf{S}$.

4.2.1. Método Laplacian eigenmaps

Laplacian eigenmaps (LE) es una técnica inspirada en *clustering* espectral, donde se define el *decoder* basado en la distancia L2 entre los *embeddings* de nodos:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \|\mathbf{z}_u - \mathbf{z}_v\|_2^2. \tag{4.7}$$

La función de costo luego pondera los pares de nodos acorde a su similitud en el grafo:

$$\mathcal{L} = \sum_{(u,v)\in\mathcal{D}} \mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v], \tag{4.8}$$

siendo $\mathcal{D}$ el conjunto de muestras de entrenamiento. La intuición detrás de este método es penalizar al modelo cuando nodos muy similares tienen *embeddings* lejanos entre sí.

La matriz $\mathbf{S}$ se construye de modo que satisface las propiedades de una matriz Laplaciana ($\mathbf{L}$), haciendo que la solución óptima que minimiza a la ecuación (4.8) esté dada por los vectores propios correspondientes a los d valores propios mas pequeños de la matriz $\mathbf{L}$ (excluyendo al valor propio cero). Los mismos se obtienen mediante técnicas de descomposición espectral donde:

$$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2} = \mathbf{U}^T\mathbf{\Delta}\mathbf{U}, \tag{4.9}$$

donde $\mathbf{L}$ es la matriz Laplaciana del grafo, $\mathbf{A}$ es la matriz de adyacencia, $\mathbf{D}$ es la matriz de grados, y $\mathbf{\Delta}$, $\mathbf{U}$ son a los valores y vectores propios correspondientes.

Método	Decoder	Medida de similitud	Función de pérdida ℓ
Lap. Eigenmaps	$\|\mathbf{z}_u - \mathbf{z}_v\|_2^2$	general	$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) \cdot \mathbf{S}[u, v]$
Graph Fact.	$\mathbf{z}_u^T \mathbf{z}_v$	$\mathbf{A}[u, v]$	$\|\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
GraRep	$\mathbf{z}_u^T \mathbf{z}_v$	$\mathbf{A}[u, v], \cdots, \mathbf{A}^k[u, v]$	$\|\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$
HOPE	$\mathbf{z}_u^T \mathbf{z}_v$	general	$\|\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2$

Tabla 4.1: Resumen de algunos algoritmos de *embeddings* basados en métodos de factorización de matrices [24].

4.2.2. Método producto interno

Los métodos basados en producto interno definen el *decoder* como el producto interno de los *embeddings*, tal como plantea el modelo RDPG cubierto en la Sección 2.1.3:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \mathbf{z}_u^T \mathbf{z}_v. \tag{4.10}$$

Estos métodos asumen que la similitud entre nodos es proporcional al producto interno de sus *embeddings*. Para la función de costo se considera el error cuadrático medio:

$$\mathcal{L} = \sum_{(u,v) \in \mathcal{D}} \|\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) - \mathbf{S}[u, v]\|_2^2. \tag{4.11}$$

Existen diferentes variaciones de este método según cómo se defina $\mathbf{S}[u, v]$. Por ejemplo, el método *Graph Factorization* (GF) [5] directamente utiliza la matriz de adyacencia, $\mathbf{S} \triangleq \mathbf{A}$. Mientras que el método GraRep [10] define $\mathbf{S}$ basado en potencias de la matriz de adyacencia. En todos los casos, las funciones de perdida pueden ser minimizadas utilizando algoritmos de factorización, como ser la descomposición en valores singulares (SVD por sus siglas en inglés).

4.3. Random Walk Embeddings

Los métodos de *Random Walk* se inspiran en los métodos de producto interno con la salvedad de que en lugar de utilizar medidas determinísticas de similitud de nodos, estos utilizan medidas estocásticas.

En particular, en este enfoque se muestrean un número considerable de caminatas aleatorias de longitud fija comenzando desde cada nodo u. Luego, los *embeddings* correspondientes a cada nodo se optimizan de modo que el producto punto o ángulo entre estos, sea proporcional a la probabilidad de pasar por v en la caminata aleatoria saliendo desde u. En otras palabras, dos nodos tendrán *embeddings* similares si tienden a co-ocurrir en caminatas aleatorias (*Random Walks*) sobre el grafo.

El *decoder* estará dado según la siguiente expresión:

$$\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v) = \frac{e^{\mathbf{z}_u^T \mathbf{z}_v}}{\sum_{k \in \mathcal{V}} e^{\mathbf{z}_u^T \mathbf{z}_k}} \approx p_{\mathcal{G},T}(v|u), \tag{4.12}$$

donde $p_{\mathcal{G},T}(v|u)$ representa la probabilidad de visitar el nodo v en una caminata aleatoria de largo T partiendo desde el nodo u. Para la función de costo se considera la *Cross-Entropy Loss*:

$$\mathcal{L} = \sum_{u,v \in \mathcal{D}} -\log\left(\mathrm{DEC}(\mathbf{z}_u, \mathbf{z}_v)\right), \tag{4.13}$$

donde $\mathcal{D}$ es el conjunto de entrenamiento generado tras muestrear caminatas aleatorias empezando desde cada nodo. Dos ejemplos clásicos de este tipo de método son: *DeepWalk* [42] y *node2vec* [22].

4.4. Graph Positional and Structural Encoders

Los Positional (PE) y Structural Encoders (SE) buscan generar *embeddings* de nodos que codifiquen ciertas características estructurales del grafo. Los PE tienen el propósito de dar una noción de la posición de cada nodo dentro del grafo. De modo que, dos nodos que cercanos en el grafo tendrán PE similares.

La técnica más utilizada para obtener los PE son los *Laplacian eigenmaps* (LE) vistos anteriormente, también referidos como *Laplacian Positional Encodings* (LapPE):

$$p_i^{LapPE} = [\mathbf{U}_{i1}, \mathbf{U}_{i2}, \cdots, \mathbf{U}_{ik}] \in \mathbb{R}^k, \tag{4.14}$$

donde $\mathbf{U}_{ij}$ son los vectores propios correspondientes a los k valores propios mas pequeños de la matriz $\mathbf{L}$ (excluyendo al valor propio cero).

Por su parte, los SE tienen el objetivo de codificar estructuras dentro del grafo, de manera que dos nodos que se encuentren en una misma estructura tendrán SE similares. Un ejemplo clásico de SE son los *Random Walk* SE (RWSE). Este método define la matriz de *Random Walk* como:

$$\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}, \tag{4.15}$$

donde $\mathbf{A}$ es la matriz de adyacencia y $\mathbf{D}$ es la matriz de grados. Cada elemento p_{ij} corresponde a la probabilidad de transicionar del nodo i al nodo j. Los RWSE se definen como la probabilidad de volver al mismo nodo luego k pasos de la caminata aleatoria:

$$p_i^{RWSE} = [\mathbf{P}_{ii}, \mathbf{P}_{ii}^2, \cdots, \mathbf{P}_{ii}^k] \in \mathbb{R}^k. \tag{4.16}$$

4.5. Estado del arte

En la siguiente sección se cubre brevemente algunos de los métodos del estado del arte en lo que respecta a GRL.

4.5.1. PowerEmbed

El método *PowerEmbed* [27], inspirado en *clustering* espectral, aplica técnicas de normalización (de capa) para potenciar las clásicas redes basadas en NMP cubiertas en el capítulo anterior. La motivación detrás de este método se basa en que las redes NMP clásicas tienden a perder información de señales de largo alcance (muchos saltos) y suelen no funcionar bien en escenarios con grafos heterofilios. A su vez, las NMPs profundas (muchas capas) suelen sufrir los efectos de *over-smoothing* o *over-squashing*, los cuales se definen en mayor detalle a continuación.

En las GNNs, las features de los nodos tienden a volverse más similares con el aumento de la profundidad de la red. Este efecto se conoce como *over-smoothing*, y se define como la tendencia de los *embeddings* de nodo en un modelo GCNN profundo, de converger al vector propio asociado al valor propio más grande. Este sólo codifica información sobre componentes conectados y grados, pero tiende a perder la estructura de comunidad presente en los vectores propios subsiguientes.

Algoritmo 1 PowerEmbed

Require: Operador de grafo $\mathbf{S} \in \mathbb{R}^{n \times n}$, señal $\mathbf{X} \in \mathbb{R}^{n \times k}$, lista $P = [X]$
1: Inicializar $\mathbf{U}[t] = \mathbf{X}$
2: **for** $t = 0 : L - 1$ **do**
3: $\quad \tilde{\mathbf{U}}[t+1] = \mathbf{S}\mathbf{U}[t]$ [Message Passing]
4: $\quad \mathbf{U}[t+1] = [\tilde{\mathbf{U}}[t+1]^T \tilde{\mathbf{U}}[t+1]]^{-1}$ [Normalización]
5: $\quad$ Concatenar $\frac{\mathbf{U}[t+1]}{\mathbf{U}[t+1][:,k]}$ a P
6: **end for**
7: **return** P

Por su parte, el *over-squashing* refiere a un fenómeno donde las representaciones de los nodos se vuelven demasiado similares entre sí durante el entrenamiento del modelo, lo que puede resultar en una pérdida de información útil y en una disminución del rendimiento del modelo.

Los autores de *PowerEmbed* muestran que su método puede recuperar los k vectores propios dominantes de la matriz de adyacencia del grafo, lo cual previene el *over-smoothing* y es agnóstico a la topología del grafo. A su vez, la combinación de features locales y globales en *PowerEmbed* previene el *over-squashing*.

Algoritmo PowerEmbed

El mecanismo utilizado por el método *PowerEmbed* es detallado en el Algoritmo 1. Este algoritmo está motivado por el método de las potencias, el cual tras converger devuelve los k vectores propios dominantes correspondiente a la matriz simétrica $\mathbf{S}$, asumiendo que la señal $\mathbf{X}$ es de rango completo y que $\mathbf{S}$ tiene un *eigen-gap*, i.e. $\lambda_k \neq \lambda_{k+1}$, donde λ_k denota al k-ésimo valor propio de $\mathbf{S}$ (ordenado por magnitud). El paso de normalización asegura la ortogonalidad de la salida en cada iteración. La tasa de convergencia dependerá de $\frac{\lambda_{k+1}}{\lambda_k}$, la cual es rápida siempre y cuando el *eigen-gap* sea grande, i.e. $|\lambda_k| \gg |\lambda_{k+1}|$.

PowerEmbed para clasificación de nodos

Los autores muestran cómo usar *PowerEmbed* para una tarea supervisada de clasificación de nodos. En primer instancia, *PowerEmbed* extrae atributos locales en las primeras iteraciones e información global en las últimas iteraciones. La información extraída en cada iteración es luego pasada por un conjunto de capas densas, lo que denominan *Inception Network*. Finalmente, se concatena la salida de cada *Inception Network* y se utiliza el vector resultante para entrenar un clasificador g_θ. La Figura 4.3 presenta un diagrama de la arquitectura propuesta.

4.5.2. Efficient ASE

El método *Efficient ASE* propone utilizar el algoritmo de descenso por gradiente (GD) de primer orden para la estimación de *embeddings* en el modelo RDPG [17]. Recordando la ecuación (2.7), la obtención de las posiciones latentes en el modelo RDPG puede formularse como un problema de optimización no convexo, donde se busca resolver:

$$\hat{\mathbf{X}} \in \operatorname{argmin}_{\mathbf{X} \in \mathbb{R}^{N \times d}} \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}\mathbf{X}^T)\|_F^2, \tag{4.17}$$

donde $\mathbf{M}$ es una matriz máscara binaria con ceros en su diagonal y unos en sus otras entradas: $\mathbf{M} = \mathbf{1}\mathbf{1}^T - \mathbf{I}$.

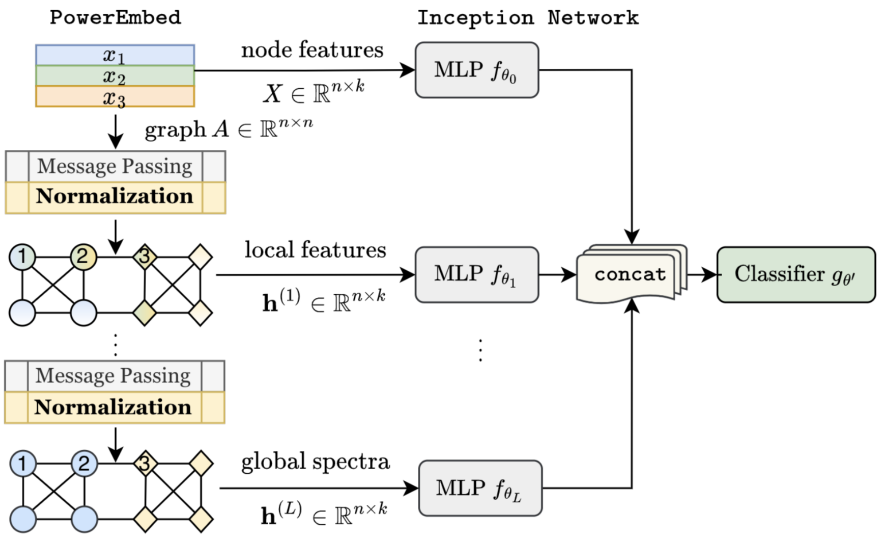


Figura 4.3: Arquitectura de *PowerEmbed* utilizado para clasificación de nodos. El algoritmo produce una lista de *embeddings* que contienen desde información de atributos locales hasta información espectral global, la cual es luego aprendida de manera conjunta utilizando las *Inception Networks* [27].

Si bien esta formulación no es convexa con respecto a $\mathbf{X}$, la función objetivo sí lo es con respecto a $\mathbf{Z} = \mathbf{X}\mathbf{X}^T$. Por lo que, para la estimación de *embeddings* del modelo RDPG, si la condición inicial es cercana a la solución, entonces el algoritmo converge linealmente a la solución $\hat{\mathbf{X}}$ [9] [12].

Estimación con Descenso por Gradiente

Dada la función objetivo $f : \mathbb{R}^{n\times d} \to \mathbb{R}$, $f(\mathbf{X}) = \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}\mathbf{X}^T)\|_F^2$. La iteración del algoritmo descenso por gradiente de primer orden consiste en:

$$\mathbf{X}_{t+1} = \mathbf{X}_t - \alpha \nabla f(\mathbf{X}_t) \ , \ t = 0, 1, \ldots, \tag{4.18}$$

donde $\alpha > 0$ es el tamaño del paso y $\nabla f(\mathbf{X}) = 4[\mathbf{M} \circ (\mathbf{X}\mathbf{X}^T - \mathbf{A})]\mathbf{X}$.

Los autores de *Efficient ASE* muestran que en la medida que aumenta el número de nodos N, el método de GD propuesto escala mejor que el SVD de ASE tradicional. A su vez, la incorporación de la máscara binaria $\mathbf{M}$ permite codificar información faltante, indicando las entradas de $\mathbf{A}$ que fueron observadas y aquellas entradas que no son conocidas.

Generalizaciones

La formulación puede adaptarse para incluir extensiones más generales del modelo RDPG, como ser el modelo *Generalized RDPG* (GRDPG) así como el *Directed RDPG* (DRDPG). Para el modelo GRDPG se considera la siguiente formulación:

$$\hat{\mathbf{X}} \in \operatorname{argmin}_{\mathbf{X}\in\mathbb{R}^{N\times d}} \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}\mathbf{I}_{p,q}\mathbf{X}^T)\|_F^2, \tag{4.19}$$

donde $\mathbf{I}_{p,q}$ es una matriz diagonal compuesta por p valores $+1$ seguidos por q valores -1 en su diagonal, tales que $p + q = d$ con d la dimensión del *embedding*.

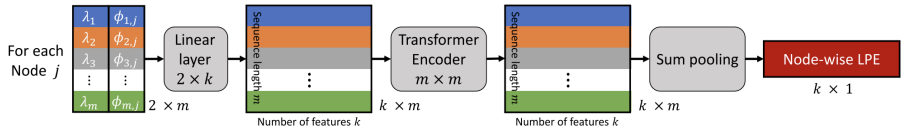


Figura 4.4: Arquitectura propuesta por SAN para la obtención de los *Learned Positional Encodings* (LPE) a partir de *eigenfunctions* basadas en el Laplaciano del grafo [31].

Mientras que para el modelo DRDPG, la iteración de GD comprende dos pasos, uno con respecto al *embedding* izquierdo $\mathbf{X}^l$ y otro con respecto al *embedding* derecho $\mathbf{X}^r$. La formulación del mismo estará dada por la siguiente expresión:

$$\{\hat{\mathbf{X}}^l, \hat{\mathbf{X}}^r\} \in \text{argmin}_{\mathbf{X}^l, \mathbf{X}^r \in \mathbb{R}^{N \times d}} \|\mathbf{M} \circ (\mathbf{A} - \mathbf{X}^l \mathbf{X}^{r\,T})\|_F^2. \tag{4.20}$$

4.5.3. Spectral Attention Network (SAN)

Los autores de *Spectral Attention Network* (SAN) [31] proponen un método para aprender Positional Encodings (PE) utilizando técnicas espectrales. Los PE aprendidos, denominados *Learned Positional Encodings* (LPE), son utilizados luego en la entrada de un modelo *Graph Transformer* (GT).

En particular, este método se basa en el uso de *eigenfunctions* del Laplaciano del grafo para aprender las posiciones absolutas y relativas dentro del mismo, permitiendo medir interacciones entre los nodos y "escuchar" ciertas sub-estructuras. En la arquitectura del *Transformer* clásico de NLP, se emplean las funciones seno y coseno para representar la posición de una palabra en una secuencia. Sin embargo, en el caso de los grafos, al no tener una estructura lineal, no se puede definir una noción de posición a lo largo de un eje. Los vectores propios de la matriz Laplaciana del grafo actúan como un equivalente a las funciones seno y coseno, permitiendo representar posiciones o estructuras en grafos. A su vez, los valores propios del Laplaciano permiten diferenciar diferentes estructuras dentro del grafo, dado que pueden ser interpretados como frecuencia de resonancia del grafo.

Arquitectura propuesta

La Figura 4.4 presenta la arquitectura propuesta por SAN para obtener los LPEs. Primero, para cada nodo j, se construye una matriz de *embeddings* de tamaño $2 \times m$, mediante la concatenación del valor propios más pequeño con su correspondiente vector propio. Luego se aplica una capa lineal en la dimensión de tamaño 2 para generar nuevos *embeddings* de tamaño k. Estos *embeddings* alimentan un *Transformer Encoder* de tamaño $m \times m$ que computa *self-attention*. Finalmente se aplica *sum-pooling* obteniendo un *embeddings* de dimensión fija k.

En paralelo a los pasos de LPE, el grafo se convierte a uno completamente conectado. Tanto los atributos de los nodos como de las aristas, se procesan a través de bloques MLP. La salida de los nodos se concatena a los LPE obtenidos, y finalmente se ingresa todo al *Transformer* principal. La Figura 4.5 ilustra el proceso completo. Para el *Transformer* principal se utiliza la arquitectura de GT propuesta en [14] y cuyo cálculo de *multi-head attention* está dado por la ecuación (3.22) vista en el capítulo anterior.

Como mejora, se considera un sistema doble *attention* que considera pesos diferentes de acuerdo a si los nodos se encuentran realmente conectados en el grafo original

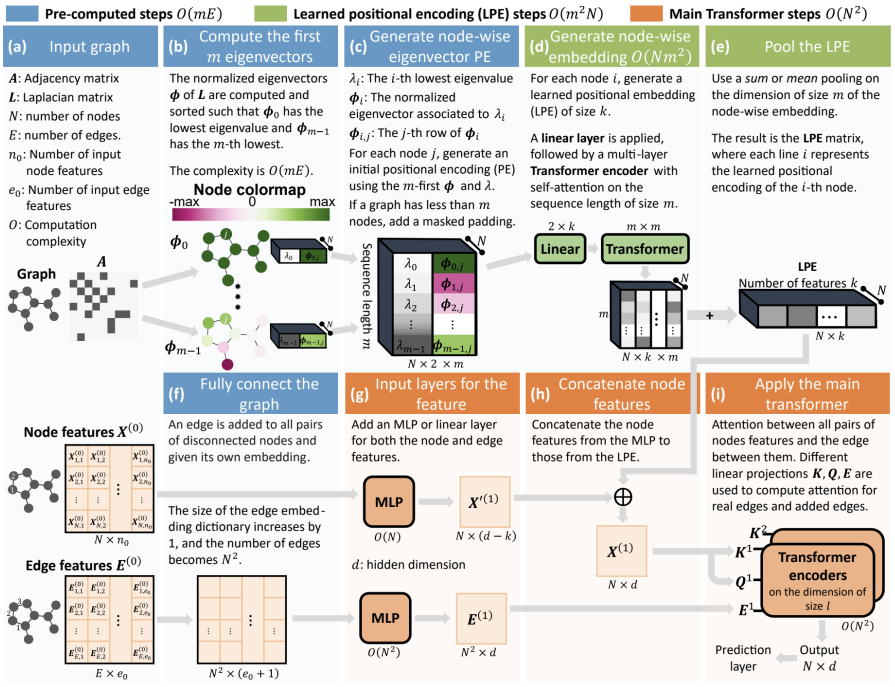


Figura 4.5: Arquitectura propuesta para el modelo SAN junto con los LPEs. [31]

$\mathcal{G}_0$, o si sólo se conectan en el grafo *fully-connected* $\mathcal{G}_1$. Los coeficientes de *attention* $w_{ij}^{k,l}$ de la capa l y la *head* k se modifican según la siguiente expresión:

$$w_{ij}^{k,l} = \begin{cases} \frac{1}{1+\gamma} \cdot \text{softmax}\left(\sum_{d_k} \hat{\mathbf{w}}_{ij}^{k,l}\right) & \text{si } i,j \in \mathcal{G}_0 \\ \frac{\gamma}{1+\gamma} \cdot \text{softmax}\left(\sum_{d_k} \hat{\mathbf{w}}_{ij}^{k,l}\right) & \text{si } i,j \in \mathcal{G}_1 \end{cases}, \tag{4.21}$$

siendo

$$\hat{\mathbf{w}}_{ij}^{k,l} = \begin{cases} \frac{\mathbf{Q}^{1,k,l}\mathbf{h}_i^l \circ \mathbf{K}^{1,k,l}\mathbf{h}_j^l \circ \mathbf{E}^{1,k,l}\mathbf{e}_{ij}}{\sqrt{d_k}} & \text{si } i,j \in \mathcal{G}_0 \\ \frac{\mathbf{Q}^{2,k,l}\mathbf{h}_i^l \circ \mathbf{K}^{2,k,l}\mathbf{h}_j^l \circ \mathbf{E}^{2,k,l}\mathbf{e}_{ij}}{\sqrt{d_k}} & \text{si } i,j \in \mathcal{G}_1 \end{cases}, \tag{4.22}$$

donde $\circ$ denota la multiplicación punto a punto, $\mathbf{Q}^{1,k,l}, \mathbf{Q}^{2,k,l}, \mathbf{K}^{1,k,l}, \mathbf{K}^{2,k,l}, \mathbf{E}^{1,k,l}, \mathbf{E}^{2,k,l} \in \mathbb{R}^{d_k \times d}$ y $\gamma \in \mathbb{R}^+$ es un hiperparámetro que permite controlar el sesgo hacia *full attention*.

Aunque el método propuesto en SAN es interesante y novedoso, la afirmación de los autores sobre aprender los PE no es completamente precisa. Los *embeddings* propuestos como PE implican un pre-cálculo de los valores y vectores propios del Laplaciano del grafo, y por ende no son aprendidos intrínsecamente dentro del modelo.

4.5.4. GraphGPS

El modelo *GraphGPS* [45] consiste en una variante del model *Graph Transformer* (GT) que se caracteriza por utilizar un mecanismo de *attention* lineal el cual permite extender su uso a grafos de mayor porte, así como así como la utilización de *Positional*

(PE) y *Structural Encodings* (SE) que dan noción de distancia y de similitud estructural. Los autores de *GraphGPS* proponen clasificar los PE y SE en tres categorías: locales, globales y relativos.

- **PE Locales**: Permiten que un nodo conozca su posición y rol dentro de un cluster local de nodos. Un ejemplo de esto podría ser considerar la distancia entre un nodo y el centroide de un cluster que lo contiene.
- **PE Globales**: Permiten que un nodo conozca su posición global dentro de un grafo. Un ejemplo de esto son los LapPE vistos anteriormente en la Sección 4.4.
- **PE Relativos**: Permite que dos nodos entiendan su relación en distancia o dirección. Por ejemplo, tomando el gradiente de los vectores propios de la matriz de adyacencia o Laplaciana del grafo.
- **SE Locales**: Permiten que un nodo conozca a la sub-estructura que forma parte. Un ejemplo de esto son los RWSE vistos anteriormente en la Sección 4.4.
- **SE Globales**: Provee a cada nodo información de la estructura global del grafo. Por ejemplo, podrían considerarse los valores propios de la matriz de adyacencia o Laplaciana del grafo.
- **SE Relativos**: Permite que dos nodos entiendan cuan diferentes son sus estructuras. Por ejemplo, tomando el gradiente de un local SE.

Arquitectura propuesta

El modelo *GraphGPS* consta de tres componentes principales:

1. Un módulo encargado de pre-computar los PE y SE.
2. Un módulo de NMP local
3. Un mecanismo de *attention* global de orden lineal $\mathcal{O}(N+E)$

En la Figura 4.6 se presenta un diagrama que resume dicha arquitectura. Se utilizan *embeddings* pre-computados para los PE y SE, los cuales se pasan luego por bloques MLP junto con los atributos de nodos y aristas. Las diferentes salidas de los MLP se concatenan y se pasan por un MLP final, dando lugar a las matrices X^l y E^l, que alimentan la capa GPS final. La misma consiste de un híbrido entre una NMP y un GT, cuya expresión está dada por las ecuaciones presentadas a continuación:

$$\mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1} = \text{GPS}(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}), \tag{4.23}$$

$$\mathbf{X}_M^{\ell+1}, \mathbf{E}^{\ell+1} = \text{NMP}_e^{\ell}(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}), \tag{4.24}$$

$$\mathbf{X}_T^{\ell+1} = \text{GlobalAttn}^{\ell}(\mathbf{X}^{\ell}), \tag{4.25}$$

$$\mathbf{X}^{\ell+1} = \text{MLP}^{\ell}(\mathbf{X}_M^{\ell+1} + \mathbf{X}_T^{\ell+1}), \tag{4.26}$$

donde $A \in \mathbb{R}^{N\times N}$ es la matriz de adyacencia del grafo con N nodos y E aristas, $\mathbf{X}^{\ell} \in \mathbb{R}^{N\times d_\ell}$ y $\mathbf{E}^{\ell} \in \mathbb{R}^{E\times d_\ell}$ son los vectores de atributos de los nodos y las aristas respectivamente con dimensión d_ℓ, NMP_e^{ℓ} es una instancia de NMP con atributos de aristas, GlobalAttn^{ℓ} es el mecanismo de *attention* global en la capa ℓ y MLP^{ℓ} cosiste de un bloque de 2 capas.

Una vez más, se nota que aunque *GraphGPS* presenta un enfoque interesante al incorporar un mecanismo de atención lineal que permite la extensión a grafos de mayor tamaño, los *embeddings* utilizados como PE requieren el cálculo explícito de los valores y vectores propios, ya sea de la matriz de adyacencia o del Laplaciano del grafo.

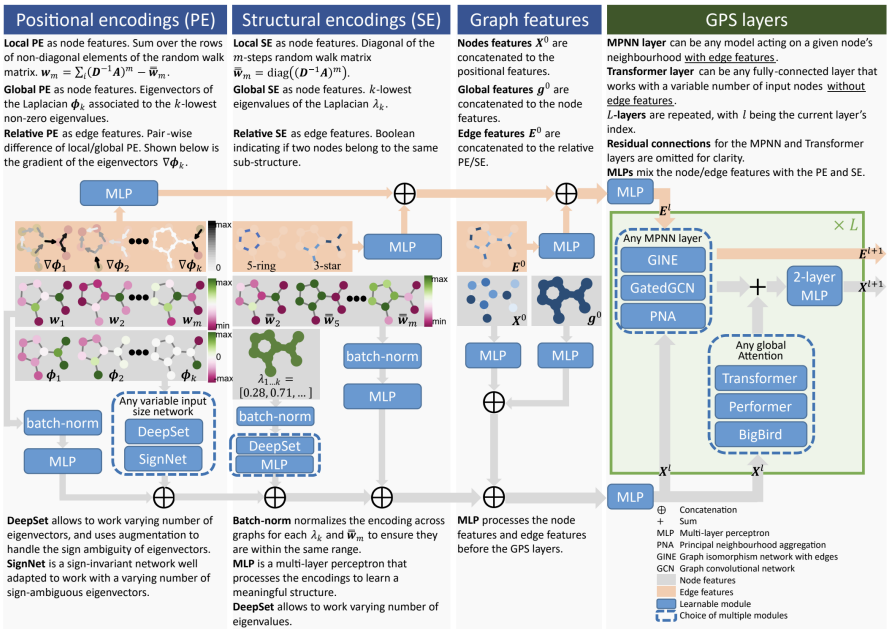


Figura 4.6: Arquitectura propuesta para el modelo GraphGPS [45].

4.6. Resumen del capítulo

En el presente capítulo se introdujo el área de *Graph Representation Learning* (GRL) la cual comprende el aprendizaje de *embeddings* de grafos, con el fin de codificar de manera efectiva la información estructural de estos en un espacio vectorial de baja dimensión. Estos *embeddings* pueden ser utilizados posteriormente en diversas tareas de aprendizaje automático. En particular, se introdujo el uso de dos operaciones básicas: *encoder* y *decoder*. El primero se encarga de codificar cada nodo del grafo en un *embedding*, mientras que el segundo intenta reconstruir la información de cada nodo y su vecindad a partir los mismos. Se describieron métodos basados en factorización, como *Laplacian Eigenmaps* y técnicas de producto interno, que aprenden a replicar la matriz de similitud de los nodos. También, se exploraron los métodos de *Random Walk Embeddings*, que usan caminatas aleatorias para medir la co-ocurrencia de nodos y optimizar los *embeddings* de acuerdo con esta información estocástica.

A su vez, el capítulo introdujo el concepto de *Graph Positional Encoders* (PE) y *Graph Structural Encoders* (SE), los cuales implican la generación de *embeddings* que codifican la estructura y la posición de los nodos dentro dentro del grafo. En particular, se destacó la importancia del uso de *embeddings* espectrales para este fin. Finalmente, se realizó un breve repaso de algunos métodos del estado del arte en lo que respecta a GRL, incluidos *PowerEmbed*, *Efficient-ASE*, *Spectral Attention Network* (SAN) y *GraphGPS*. En particular, los últimos dos métodos se basan en el uso de PE como parte fundamental de su arquitectura, utilizando *embeddings* espectrales que implican un calculo explícito de los valores y vectores propios, ya sea de la matriz de adyacencia o del Laplaciano del grafo. *PowerEmbed* es el único que propone el aprendizaje de este tipo de representaciones de forma integral en su arquitectura.

Esta página ha sido intencionalmente dejada en blanco.

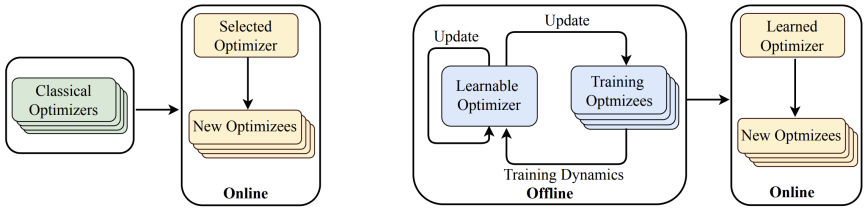
Capítulo 5

Learning to Optimize

Los métodos de optimización clásicos típicamente requieren de ajustes manuales de sus hiperparámetros en base a la teoría y en la experiencia de los ejecutores. El marco *Learning to Optimize* (L2O) surge como una alternativa a los métodos convencionales. El mismo utiliza redes neuronales y técnicas de aprendizaje profundo con el objetivo de reducir los ajustes finos en hiperparámetros de los métodos de optimización, acelerando y mejorando su implementación [11].

Mientras que los métodos de optimización clásicos suelen construirse usando componentes básicos, en pocas líneas de código y respaldados por la teoría, L2O desarrolla un método de optimización vía entrenamiento, aprendiendo en base a su performance en problemas similares. Si bien puede carecer de un respaldo teórico sólido, su performance mejora en el entrenamiento, logrando resultados comparables en calidad con los obtenidos tras implementar el algoritmo iterativo a convergencia. El proceso de entrenamiento suele conllevar bastante tiempo y por lo general es ejecutado *offline*. Una vez finalizado el entrenamiento, la aplicación *online* es rápida permitiendo ahorros significativos de tiempo. La Figura 5.1 presenta un diagrama comparativo entre el proceso típico de los optimizadores clásicos y los aprendidos en el marco de L2O. Estos últimos, han mostrado ser extremadamente útiles y alcanzar resultados incluso de mejor calidad que los métodos tradicionales, en situaciones donde las soluciones objetivo son difíciles de obtener, como ser el caso de problemas de optimización no convexos y aplicaciones *inverse-problem*.

Los modelos L2O pueden clasificarse en dos familias: *model-free* y *model-based*. Los



(a) Classic Optimizer (b) Learned Optimizer by L2O

Figura 5.1: Comparación de los optimizadores clásicos con respecto a los aprendidos en el marco L2O [11].

Algorithm 1

Require: Inicializar $\mathbf{X} \leftarrow \mathbf{X}_0$
  for $l = 1 : L$ **do**
    $\mathbf{X}^{l+1} \leftarrow h(\mathbf{X}^l, \theta^l)$
  end for
  return $\mathbf{X}^L$

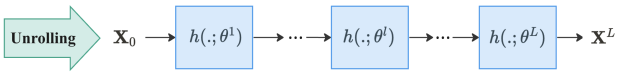


Figura 5.2: Ejemplo del funcionamiento de *Algorithm Unrolling*: dado un algoritmo iterativo, cada iteración se mapea en una capa de red neuronal. Luego, las capas se concatenan para formar una red neuronal profunda.

L2O *model-free* parten de una red neuronal, típicamente una RNN, la cual entrenan con el fin de aprender una regla de optimización sin necesidad de partir de un resultado analítico. En cambio los L2O *model-based*, parten un algoritmo de optimización y relajan ciertas condiciones para entrenar. Un ejemplo de esto es la técnica de *Algorithm Unrolling* sobre la cual se profundiza a continuación.

5.1. Algorithm Unrolling

La técnica de *Algorithm Unrolling* consiste en adaptar algoritmos iterativos de optimización de modo de poder interpretarlos como una arquitectura de redes neuronales [38]. Cada iteración del algoritmo es representada como una capa de la red, las cuales se concatenan para formar una red neuronal densa. El *forward pass* emula la ejecución del algoritmo iterativo un número finito de iteraciones.

Los parámetros del algoritmo son mapeados como parámetros de la red y son aprendidos, usando *back-propagation*, a través de entrenamiento sobre datos reales. De este modo, la red entrenada puede interpretarse naturalmente como un algoritmo con parámetros optimizados, venciendo la falta de interpretabilidad que suele tener la mayoría de las redes neuronales convencionales. La Figura 5.2 ilustra este proceso.

La red neuronal utilizada a su vez permite ciertas modificaciones que potencian la performance de este método. Por ejemplo, es posible relajar las condiciones sobre los parámetros de la red, de modo que estos no tengan que ser compartidos por las diferentes capas. Utilizar parámetros independientes en cada capa permite disminuir el número de iteraciones necesarias para alcanzar la convergencia, ergo el número de capas necesarias en la red, lo cual conlleva a una eficiencia computacional mucho más alta en comparación con el algoritmo de optimización original. Por ejemplo, la aplicación de *Algorithm Unrolling* al algoritmo *Iterative Shrinkage and Thresholding Algorithm* (ISTA) mejora notoriamente su desempeño, haciéndolo 20 veces más rápido que el algoritmo original [20].

5.1.1. Learned ISTA

La técnica de *Algorithm Unrolling* fue introducida por primera vez en 2010 con la motivación de mejorar la eficiencia computacional en algoritmos de *sparse coding* basándose en redes neuronales y en su entrenamiento *end-to-end*. En particular, su primera versión se enfocó en cómo mejorar la eficiencia de uno de los algoritmos más populares de *sparse coding*, el ISTA.

Dado un vector $\mathbf{y} \in \mathbb{R}^n$ y una matriz $\mathbf{W} \in \mathbb{R}^{n \times m}$ con $m > n$, el *sparse coding* busca encontrar una representación *sparse* (muchas entradas nulas) de $\mathbf{y}$ usando $\mathbf{W}$. En otras palabras, se busca encontrar un vector $\mathbf{x} \in \mathbb{R}^m$ tal que $\mathbf{y} \approx \mathbf{W}\mathbf{x}$, procurando que $\mathbf{x}$

tenga la mayor cantidad de entradas nulas posibles. Un enfoque clásico para determinar $\mathbf{x}$ consiste en resolver el siguiente problema de optimización convexa, conocido como Lasso:

$$\mathbf{x}^{lasso} \leftarrow \min_{\mathbf{x}} \frac{1}{2}\|\mathbf{W}\mathbf{x} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_1, \tag{5.1}$$

donde $\lambda > 0$ es un parámetro de regularización que controla la *sparcity* de la solución.

El algoritmo ISTA puede ser usado para resolver este problema. Cada iteración del algoritmo ISTA comprende una operación lineal seguida por una operación no-lineal $\eta_{\lambda(.)}$, similar a una activación ReLU, como se desarrolla a continuación:

$$\mathbf{x}^{k+1} \leftarrow \eta_\lambda(\mathbf{x}^k - \alpha\mathbf{W}^T(\mathbf{W}\mathbf{x}^k - \mathbf{y})), \tag{5.2}$$

donde $\eta_\lambda(\mathbf{x}) = \text{sign}(\mathbf{x}) \cdot \max\{|\mathbf{x}| - \lambda, 0\}$. Tomando $\mathbf{x}^k$ como factor común, la expresión (5.2) puede reescribirse de modo que:

$$\mathbf{x}^{k+1} \leftarrow \eta_\lambda((\mathbf{I} - \alpha\mathbf{W}^T\mathbf{W})\mathbf{x}^k + \alpha\mathbf{W}^T\mathbf{y}). \tag{5.3}$$

Luego, definiendo $\mathbf{W}_t = \mathbf{I} - \alpha\mathbf{W}^T\mathbf{W}$ y $\mathbf{W}_e = \alpha\mathbf{W}^T$ se obtiene que:

$$\mathbf{x}^{k+1} \leftarrow \eta_\lambda(\mathbf{W}_t\mathbf{x}^k + \mathbf{W}_e\mathbf{y}). \tag{5.4}$$

Como puede comprobarse la expresión (5.4) se asemeja a una capa red neuronal simple. La concatenación de estas capas equivale a múltiples ejecuciones del algoritmo ISTA. Dado que los parámetros son compartidos entre las diferentes capas, la red neuronal resultante se asemeja a la arquitectura de una RNN.

La red es entrenada sobre conjuntos de datos reales para optimizar los parámetros $\mathbf{W}_t$, $\mathbf{W}_e$ y λ, usando técnicas de aprendizaje basadas en descenso por gradiente estocástico. La aplicación de *Algorithm Unrolling* al algoritmo de ISTA, se denomina *Learned* ISTA (LISTA). La misma puede alcanzar una mayor eficiencia, requiriendo una cantidad de capas que suelen ser órdenes de magnitud menores que el número de iteraciones requerido por el algoritmo original para lograr la convergencia. A su vez es útil cuando $\mathbf{W}$ no es conocido.

5.2. Resumen del capítulo

El capítulo se introdujo el marco de *Learning to Optimize* (L2O), una alternativa a los métodos convencionales de optimización que hace uso de técnicas de aprendizaje automático para optimizar los hiperparámetros tradicionalmente ajustados de forma manual. A diferencia de los métodos de optimización clásicos, que se basan en componentes teóricos bien establecidos, L2O aprende de datos reales mediante un entrenamiento *offline*.

En particular se hizo énfasis en la técnica de *Algorithm Unrolling*, que consiste en adaptar algoritmos iterativos de optimización de modo de poder interpretarlos como una arquitectura de redes neuronales. Cada iteración del algoritmo se representa como una capa de la red, las cuales se concatenan para formar una red neuronal densa. La red resultante se entrena utilizando datos reales para optimizar los parámetros de estas capas. Esto permite que el algoritmo ajuste sus parámetros automáticamente, mejorando la eficiencia y reduciendo el número de iteraciones necesarias para la convergencia.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Algorithm Unrolling aplicado a RDPG ASE

Tal como fue cubierto en la Sección 4.5.2, los *embeddings* de RDPG se pueden calcular de manera eficiente y más general mediante un algoritmo iterativo, como ser el descenso por gradiente (GD). En base a esto, surge la idea de aplicar técnicas del marco L2O como las que se discutieron en el capítulo anterior. Este capítulo entonces, se basa en la aplicación de la técnica de *Algorithm Unrolling* al algoritmo de GD aplicado a los ASE del modelo RDPG, y es motivado por los resultados logrados por *Efficient* ASE [17]. El objetivo es poder reinterpretar el algoritmo como la arquitectura de una red neuronal, en particular una GNN, la cual denominaremos *Learned* ASE (LASE).

En línea a esto, se comienza por considerar la función objetivo $f : \mathbb{R}^{n \times d} \to \mathbb{R}$ de ASE vista en la Sección 2.1.3 en su versión más simple:

$$f(\mathbf{X}) = \|\mathbf{A} - \mathbf{X}\mathbf{X}^T\|_F^2. \tag{6.1}$$

El algoritmo GD consiste de la siguiente iteración:

$$\mathbf{X}^{k+1} \leftarrow \mathbf{X}^k - 4\alpha(\mathbf{X}^k{\mathbf{X}^k}^T - \mathbf{A})\mathbf{X}^k. \tag{6.2}$$

Tomando $\mathbf{X}^k$ como factor común y reordenando, la expresión anterior se convierte en:

$$\mathbf{X}^{k+1} \leftarrow (\mathbf{I} + 4\alpha\mathbf{A})\mathbf{X}^k - 4\alpha\mathbf{X}^k{\mathbf{X}^k}^T\mathbf{X}^k. \tag{6.3}$$

El primer término de la expresión (6.3) corresponde a la operación de convolución en grafos vista en la sección 3.2. Considerando $\mathbf{S} = \mathbf{A}$ y un filtro de orden $K = 1$ con matrices de coeficientes $\mathbf{H}_0 = \mathbf{I}$ y $\mathbf{H}_1 = 4\alpha\mathbf{I}$ la expresión (3.10) se convierte en:

$$\mathbf{Y} = \sum_{k=0}^{K-1} \mathbf{S}^k\mathbf{X}\mathbf{H}_k = \mathbf{H}_0\mathbf{X} + \mathbf{A}\mathbf{X}\mathbf{H}_1 = (\mathbf{I} + 4\alpha\mathbf{A})\mathbf{X}, \tag{6.4}$$

coincidiendo efectivamente con el primer término de la expresión (6.3).

Mientras que el segundo término de la expresión (6.3), puede relacionarse con un tipo de operación de *Graph Transformer* conocida como *Transformer Convolution* (TransformerConv) [49], con algunas modificaciones leves. La operación TransformerConv se define a partir de la siguiente expresión:

$$\mathbf{Y} = \mathbf{H}_0\mathbf{X} + \mathbf{B}\mathbf{H}_1\mathbf{X}, \tag{6.5}$$

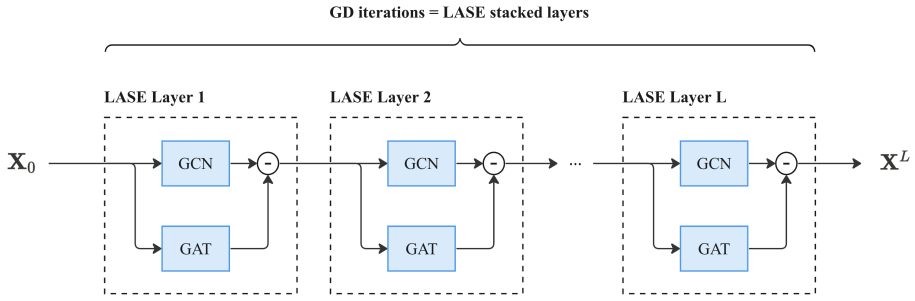


Figura 6.1: Ilustración de la arquitectura de LASE. Cada capa consiste en combinar una operación GCN con una operación GAT. Las capas se concatenan de modo de lograr una red neuronal profunda la cual es equivalente a L iteraciones del algoritmo GD.

donde $\mathbf{B} = [\beta_{ij}]$ es la matriz de coeficientes de *attention*. Los mismos se calculan como *multi-head dot product attention* según la siguiente expresión:

$$\beta_{ij} = \text{softmax}\left(\frac{(\mathbf{H}_2\mathbf{X}_i)(\mathbf{H}_3\mathbf{X}_j)^T}{\sqrt{d}}\right). \tag{6.6}$$

Admitiendo una modificación en la operación de *attention*, de modo de no considerar la función no lineal softmax, la matriz de *attention* estará dada por:

$$\beta_{ij} = \frac{\mathbf{H}_2\mathbf{X}_i(\mathbf{H}_3\mathbf{X}_j)^T}{\sqrt{d}} \Leftrightarrow \mathbf{B} = \frac{\mathbf{H}_2\mathbf{X}\mathbf{X}^T\mathbf{H}_3^T}{\sqrt{d}}. \tag{6.7}$$

Por otro lado, considerado una matriz de adyacencia de un grafo completamente conectado $\mathbf{A} = \mathbf{1}\mathbf{1}^T$ y tomando las matrices de coeficientes $\mathbf{H}_0 = \mathbf{0}$, $\mathbf{H}_1 = \mathbf{H}_3 = \mathbf{I}$ y $\mathbf{H}_2 = (4\alpha\sqrt{d})\mathbf{I}$ se obtiene que

$$\mathbf{Y} = \mathbf{H}_0\mathbf{X} + \frac{\mathbf{H}_2\mathbf{X}\mathbf{X}^T\mathbf{H}_3^T}{\sqrt{d}}\mathbf{H_1}\mathbf{X} = \frac{(4\alpha\sqrt{d})\mathbf{X}\mathbf{X}^T}{\sqrt{d}}\mathbf{X} = 4\alpha\mathbf{X}\mathbf{X}^T\mathbf{X}, \tag{6.8}$$

coincidiendo efectivamente con el segundo término de la expresión (6.3).

En resumen, es posible reinterpretar cada paso del descenso por gradiente como una capa de una red neuronal de grafos (GNN) compuesta por dos componentes, una componente convolucional de grafos (GCN) y una componente de *attention* de grafos (GAT) correspondiente a la operación TransformerConv, como se muestra en las expresiones a continuación:

$$\mathbf{X}_{out} \leftarrow \text{GCN}(K = 1, \mathbf{A}_1, \mathbf{X}_{in}) - \text{TransformerConv}(\mathbf{A}_2, \mathbf{X}_{in}), \tag{6.9}$$

$$\mathbf{X}_{out} \leftarrow \mathbf{X}_{in} + \mathbf{A}\mathbf{X}_{in}\mathbf{W}_1 - \mathbf{X}_{in}\mathbf{X}_{in}^T\mathbf{X}_{in}\mathbf{W}_2, \tag{6.10}$$

donde $\mathbf{A}_1$ es la matriz de adyacencia del grafo original, $\mathbf{A}_2 = \mathbf{1}\mathbf{1}^T$ es la matriz de adyacencia de un grafo completamente conectado, y $\mathbf{W_1}$ y $\mathbf{W_2}$ las matrices de pesos a aprender. La concatenación de estas capas equivale a múltiples ejecuciones del algoritmo GD. La aplicación de *Algorithm Unrolling* al algoritmo de GD para la obtención de *embeddings* ASE en el modelo RDPG, la denominaremos *Learned* ASE (LASE). Este proceso es ilustrado en detalle por el diagrama de la Figura 6.1.

La red LASE se somete a un entrenamiento no supervisado, donde se utilizan realizaciones de una distribución de grafos específica con el fin de optimizar los parámetros $\mathbf{W_1}$ y $\mathbf{W_2}$ de modo de minimizar la función objetivo (6.1) mediante técnicas de

aprendizaje basadas en descenso por gradiente estocástico. A través de experimentos detallados en el Capítulo 7, se muestra que el número de capas necesario en la red LASE entrenada es de más de un orden de magnitud menor que el número de iteraciones requeridas por el algoritmo GD para lograr la convergencia.

6.1. Refinamientos de la arquitectura

Con el objetivo de mejorar la robustez y performance del modelo LASE propuesto, se aplicaron ciertos refinamientos sobre la arquitectura del mismo, los cuales se detallan a continuación.

Normalización por nodo

Para mejorar la estabilidad durante el entrenamiento, se incorporó al modelo LASE una normalización por nodo. La misma consiste básicamente en dividir tanto el término de la capa GCN como el de la capa GAT por el número total de nodos n. Esta acción ayuda a mantener los valores en una escala equilibrada, mitigando cualquier inestabilidad que pueda surgir debido a valores grandes, i.e. *exploding gradient*.

El número de nodos se define como un parámetro de la red, lo que permite cambiarlo fácilmente durante la etapa de inferencia, de modo de no perder poder de generalización. La arquitectura de cada capa LASE presentada en (6.9) se actualiza entonces según la siguiente expresión:

$$\mathbf{X}^{out} \leftarrow \mathbf{X}^{in} + \mathbf{A}\mathbf{X}^{in}\frac{\mathbf{W_1}}{n} - \mathbf{X}^{in}(\mathbf{X}^{in})^T\mathbf{X}^{in}\frac{\mathbf{W_2}}{n}. \tag{6.11}$$

Incorporación de máscara binaria

Por su parte, también se incorpora al modelo LASE una matriz de máscara binaria $\mathbf{M}$ adicional como entrada del mismo. Esta incorporación no solo permite que el modelo se adhiera a la formulación de ASE original (2.7) en lugar de su versión simplificada (2.5), sino que también permite que el modelo acomode sin problemas escenarios donde se tengan datos faltantes o no observados. De esta forma se actualiza la expresión (6.11) por la siguiente:

$$\mathbf{X}^{out} \leftarrow \mathbf{X}^{in} + (\mathbf{M} \circ \mathbf{A})\mathbf{X}^{in}\frac{\mathbf{W_1}}{np} - \mathbf{M} \circ (\mathbf{X}^{in}(\mathbf{X}^{in})^T)\mathbf{X}^{in}\frac{\mathbf{W_2}}{np}, \tag{6.12}$$

donde p es la proporción de aristas respecto al número de nodos en la matriz de máscara $\mathbf{M}$.

Pesos independientes entre capas

Inspirados la técnica de *Algorithm Unrolling* desarrollada en la sección 5.1, se habilita la posibilidad de que el modelo tenga pesos variables $\mathbf{W_1^l}$ y $\mathbf{W_2^l}$, a lo largo de sus L capas, con el objetivo de lograr una mayor expresividad y mejorar la performance del modelo.

Otros mecanismos de *attention*

El mecanismo de *attention* utilizado en el término TransformerConv hace uso de un grafo completamente conectado, emulando un mecanismo de *fully-connected attention* con una dependencia cuadrática dada por el término $\mathbf{X}\mathbf{X}^T$. Para obtener una mayor eficiencia computacional, se exploran algunos mecanismos de *sparse attention*

los cuales reducen la dependencia cuadrática a una lineal. Estos mecanismos consisten en utilizar otro tipos de grafos en el término TransformerConv.

En particular, se consideraron: grafos generados con un modelo Erdös–Rényi con diferentes niveles de *sparcity*, grafos generados con un modelo Watts–Strogatz y la integración del mecanismo de *attention BigBird* [35] comúnmente utilizado en *Transformers* de lenguaje. Con estas consideraciones, el modelo LASE se actualiza de la siguiente manera:

$$\mathbf{X}^{out} \leftarrow \mathbf{X}^{in} + (\mathbf{M} \circ \mathbf{A})\mathbf{X}^{in}\frac{\mathbf{W_1}}{np_1} - \mathbf{M} \circ (\mathbf{M}^{Att} \circ (\mathbf{X}^{in}(\mathbf{X}^{in})^T))\mathbf{X}^{in}\frac{\mathbf{W_2}}{np_1p_2}, \quad (6.13)$$

donde $\mathbf{M}$ corresponde a la matriz de máscara previamente incorporada, $\mathbf{M}^{Att}$ corresponde a la matriz de máscara de *attention* del mecanismo elegido, p_1 es la proporción de aristas respecto al número de nodos en la matriz de máscara $\mathbf{M}$, y p_2 es la proporción de aristas respecto al número de nodos en la matriz de máscara de *attention* $\mathbf{M}^{Att}$.

6.2. Generalized RDPG ASE

Con el objetivo de obtener una versión más generalizada del modelo que permita su aplicación en escenarios que involucren grafos heterofilios, se procede a aplicar de *Algorithm Unrolling* al cálculo de *embeddings* utilizando ASE para el modelo *Generalized RDPG*. Como se detalla en la sección 2.1.3, el mismo permite capturar patrones de conectividad heterofilios, adoptando el producto interno *indefinido* entre las posiciones latentes, mediante a la incorporación de una matriz diagonal $\mathbf{I}_{pq}$ compuesta por p valores $+1$ seguidos por q valores -1 en su diagonal.

Recordando la función objetivo a minimizar para el modelo el GRDPG,

$$\hat{\mathbf{X}} \in \text{argmin}_{\mathbf{X}\in\mathbb{R}^{N\times d}} \|\mathbf{A} - \mathbf{X}\mathbf{I}_{p,q}\mathbf{X}^T\|_F^2, \quad (6.14)$$

el algoritmo GD consistirá de la siguiente iteración:

$$\mathbf{X}^{k+1} \leftarrow \mathbf{X}^k - \alpha\nabla f(\mathbf{X}^k), \qquad k = 1, 2, ..., n, \quad (6.15)$$

siendo α el tamaño del paso. Desarrollando la expresión anterior se obtiene:

$$\mathbf{X}^{k+1} \leftarrow \mathbf{X}^k - 4\alpha(\mathbf{X}^k\mathbf{I}_{p,q}{\mathbf{X}^k}^T - \mathbf{A})\mathbf{X}^k\mathbf{I}_{p,q}, \quad (6.16)$$

$$\mathbf{X}^{k+1} \leftarrow \mathbf{X}^k + 4\alpha\mathbf{A}\mathbf{X}^k\mathbf{I}_{p,q} - 4\alpha(\mathbf{X}^k\mathbf{I}_{p,q}{\mathbf{X}^k}^T)\mathbf{X}^k\mathbf{I}_{p,q}. \quad (6.17)$$

Similar al caso LASE, la expresión anterior puede reinterpretarse como una capa de una red neuronal GNN compuesta por dos componentes. El primero término corresponde a una GCN considerando $\mathbf{S} = \mathbf{A}$ y un filtro de orden $K = 1$ con matrices de coeficientes $\mathbf{H}_0 = \mathbf{I}$ y $\mathbf{H}_1 = 4\alpha\mathbf{I}_{p,q}$. Mientras que el segundo, corresponde a un TransformerConv, considerado una matriz de adyacencia de un grafo completamente conectado $\mathbf{A} = \mathbf{1}\mathbf{1}^T$, tomando las matrices de coeficientes $\mathbf{H}_0 = \mathbf{0}$, $\mathbf{H}_1 = \mathbf{H}_3 = \mathbf{I}$ y $\mathbf{H}_2 = (4\alpha\sqrt{d})\mathbf{I}_{p,q}$ y eliminando la no linealidad softmax.

La expresión final estará dada por:

$$\mathbf{X}_{out} \leftarrow \text{GCN}(K = 1, \mathbf{A}_1, \mathbf{X}_{in}) - \text{TransformerConv}(\mathbf{A}_2, \mathbf{X}_{in}), \quad (6.18)$$

$$\mathbf{X}_{out} \leftarrow \mathbf{X}_{in} + \mathbf{A}\mathbf{X}_{in}\mathbf{W}_1\mathbf{Q} - \mathbf{X}_{in}\mathbf{Q}{\mathbf{X}_{in}}^T\mathbf{X}_{in}\mathbf{W}_2\mathbf{Q}, \quad (6.19)$$

donde $\mathbf{A}_1$ es la matriz de adyacencia del grafo original, $\mathbf{A}_2 = \mathbf{1}\mathbf{1}^T$ es la matriz de adyacencia de un grafo completamente conectado, $\mathbf{Q} = \mathbf{I}_{p,q}$ y y $\mathbf{W}_1$ y $\mathbf{W}_2$ las matrices de pesos a aprender. La aplicación de *Algorithm Unrolling* al algoritmo de GD para la obtención de *embeddings* ASE en el modelo GRDPG, la denominamos *Generalized* LASE (GLASE).

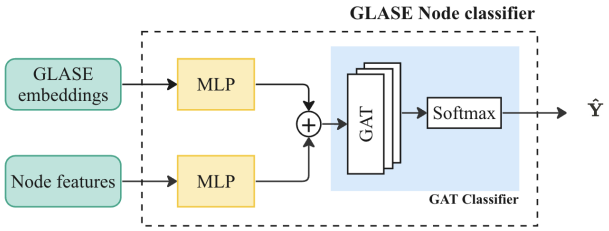


Figura 6.2: Diagrama de la arquitectura del modelo de clasificación de nodos que utiliza GLASE-PE pre-entrenados.

6.3. Uso de GLASE como Positional Encodings (PE)

Se propone el uso de los *embeddings* obtenidos con el modelo GLASE (o LASE) como Positional Encodings (GLASE-PE) que alimentan ya sea un bloque de Graph Attention (GAT) o un Graph Transformer (GT). Los GLASE-PE pueden ser pre-computados realizando un entrenamiento sobre un conjunto de grafos específico, para luego ser inyectados como PE en un modelo final utilizado para entrenar tareas más específicas. Alternativamente, los GLASE-PE pueden ser incorporados directamente como parte del modelo final y aprendidos durante el proceso de entrenamiento para la tarea específica, obteniendo una arquitectura *end-to-end*.

A continuación se presentan en detalle las arquitecturas propuestas para la incorporación de los GLASE-PE en tareas específicas de clasificación de nodos y predicción de enlace.

6.3.1. Clasificación de nodos

En lo que refiere a tareas de clasificación de nodos se consideran dos posibles arquitecturas. En la primera, los GLASE-PE pre-entrenados y los vectores de atributos de los nodos pasan por bloques MLP, para luego ser concatenados. El resultado alimenta un clasificador compuesto por una serie de capas GAT combinadas con *BatchNorm* y ReLU, para luego pasar por una softmax. Como función de pérdida se considera la *Cross Entropy Loss*.

La segunda consiste en incorporar el modelo GLASE como parte de la arquitectura. Se utiliza un vector inicial aleatorio a la entrada el cual alimenta al modelo GLASE. Luego, tanto la salida de GLASE como los vectores de atributos de nodos, pasan por bloques MLP, para luego ser concatenados. El resto de la arquitectura es análoga a la anterior. En cuanto a la función de pérdida, se considera una combinación lineal entre la *Cross Entropy Loss* y la función objetivo de GLASE dada por la ecuación (6.14), ponderadas por un factor μ:

$$\mathcal{L} = \mu \text{CrossEntropy}(\mathbf{Y}, \hat{\mathbf{Y}}) + (1-\mu)\mathbf{M} \circ \|\mathbf{A} - \hat{\mathbf{X}}\mathbf{Q}\hat{\mathbf{X}}^T\|_F^2. \tag{6.20}$$

Las figuras (6.2) y (6.3) muestran el detalle de cada arquitectura respectivamente.

6.3.2. Predicción de enlaces

En cuanto a las tareas de predicción de enlace, se considera una arquitectura basada en el modelo propuesto por *Variational Graph Auto-Encoders* (VGAE) [29]. La misma esta compuesta de las siguientes partes:

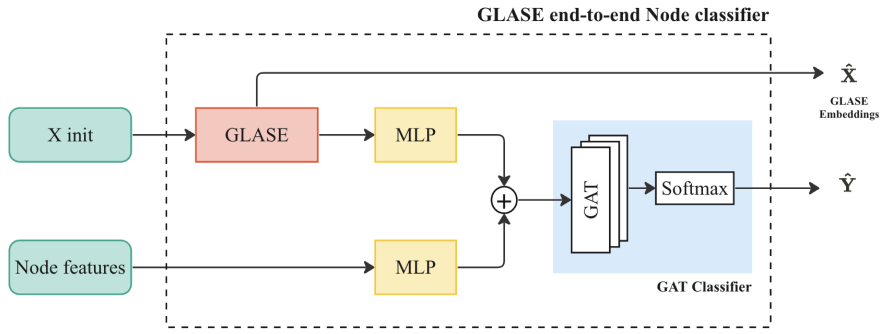


Figura 6.3: Diagrama del modelo de clasificación compuesto por un bloque GLASE como parte de la arquitectura end-to-end.

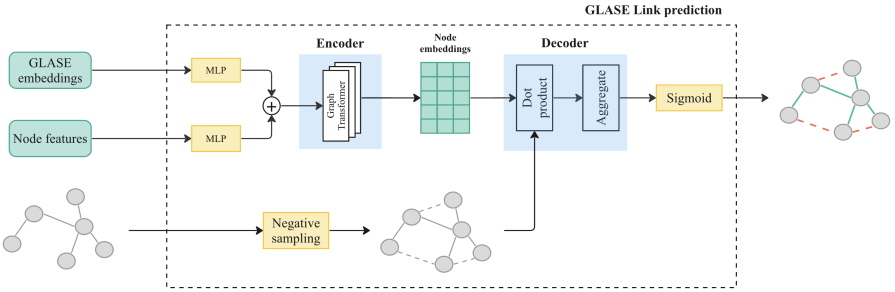


Figura 6.4: Diagrama de la arquitectura del modelo de predicción de enlaces basado en VGAE que utiliza GLASE-PE pre-entrenados.

1. Un módulo *encoder*, el cual se encarga de generar los *embeddings* de nodo. Este suele estar compuesto por un par de capas convolucionales GCN.
2. Un módulo *negative sampling*, el cual adiciona aristas negativas al grafo, permitiendo reinterpretar la tarea como una clasificación binaria entre aristas positivas y negativas.
3. Un módulo *decoder*, el cual ejecuta las predicciones de enlace computando el producto interno entre los *embeddings* de los nodos de cada arista (positivas y negativas), obteniendo así una probabilidad de existencia de aristas.

La incorporación de los GLASE-PE a esta arquitectura puede darse de dos maneras. La primera, consiste en pasar los *embeddings* de GLASE pre-entrenados y los vectores de atributos de los nodos por bloques MLP y luego concatenar sus salidas. El resultado es usado como entrada al módulo *encoder*. A diferencia del VGAE original que utiliza componentes GCN, este módulo se construye utilizando una serie de capas *Graph Transformer* combinadas con *BatchNorm* y ReLU. El resto de los módulos son análogos a los descriptos para VGAE, y se utiliza la *Cross Entropy Loss* como función de pérdida.

La segunda consiste en incorporar el modelo GLASE como un bloque previo en la arquitectura y realizar un entrenamiento *end-to-end*. De forma análoga al caso de clasificación de nodos, se utiliza un vector inicial aleatorio a la entrada para alimentar el modelo GLASE. La salida de este y los vectores de atributos de los nodos son pasados

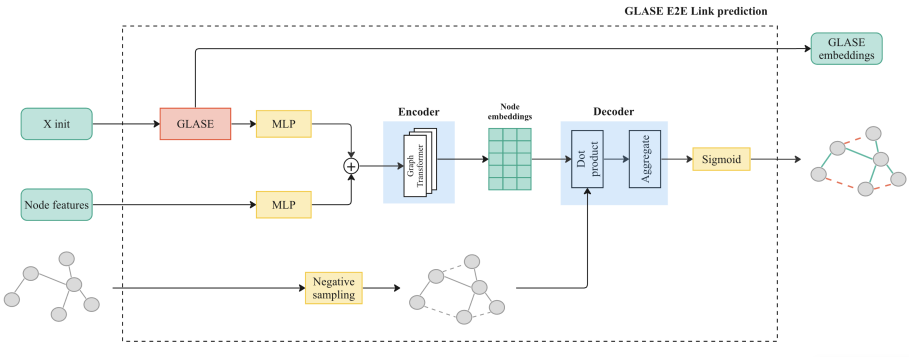


Figura 6.5: Diagrama de la arquitectura del modelo de predicción de enlaces basado en VGAE compuesto por un bloque GLASE como parte de la arquitectura *end-to-end*.

por bloques MLP y luego son concatenados. El resultado se pasa como entrada al módulo *decoder*, el cuál es análogo al del primer modelo. En cuanto a la función de pérdida se utiliza la misma que en el modelo de clasificación *end-to-end* dada por la ecuación (6.20).

6.4. Resumen del capítulo

En este capítulo se detalló la aplicación de la técnica de *Algorithm Unrolling* al algoritmo de descenso por gradiente (GD) aplicado a la obtención de los ASE del modelo RDPG. La red neuronal resultante basada en *Graph Neural Network* (GNN) se le denominó *Learned* ASE (LASE).

Se describió en detalle el proceso de *unrolling* de la iteración del GD y su mapeo a una GNN compuesta de dos componentes: una componente convolucional de grafos (GCN) y una componente de *attention* de grafos (GAT). Por su parte, se abordaron mejoras arquitectónicas sobre el modelo propuesto incluyendo normalización por nodo y la incorporación de una matriz de máscara binaria para manejar datos faltantes o no observados. Además, se exploró el uso de pesos independientes entre capas para mejorar la expresividad del modelo y se investigaron mecanismos de *attention* más eficientes computacionalmente.

Finalmente, se discutió la aplicación de LASE como *Positional Encodings* (PE) que alimentan ya sea un bloque GAT o un Graph Transformer (GT), para luego ser utilizados en tareas de aprendizaje específicas como ser la clasificación de nodos o la predicción de enlaces. Se proporcionaron ejemplos de arquitecturas que integran GLASE-PE en configuraciones tanto pre-entrenadas como entrenadas de forma *end-to-end*.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 7

Resultados experimentales con LASE

El presente capítulo se dedica a detallar los experimentos realizados y los resultados alcanzados empleando los modelos LASE y GLASE introducidos en el capítulo anterior. Para ello, se llevaron a cabo pruebas tanto en conjuntos de datos sintéticos como en conjuntos reales, buscando validar la efectividad y adaptabilidad de estos modelos en una variedad de escenarios.

El código correspondiente a la totalidad de experimentos realizados en este trabajo se encuentra disponible en un repositorio de Github de acceso libre [2].

7.1. Conjuntos de datos sintéticos

Se comenzó utilizando conjuntos de datos sintéticos de modo de tener un ambiente de pruebas controlado. Para esto se generaron grafos a partir de diferentes realizaciones de un modelo SBM, con un cierto número de comunidades y distribución de grados. Se generó un conjunto de 1000 grafos, de los cuales se destinaron 800 para entrenamiento y 200 para validación. Se generaron diferentes conjuntos de datos, variando el número de comunidades, el número de nodos y su distribución de grados.

7.1.1. LASE

Se implementó un modelo LASE como el descrito por la expresión (6.12) haciendo uso de la biblioteca *PyTorch Geometric* (PyG) [3]. El mismo está compuesto por 5 capas que utilizan pesos distribuidos y normalización por nodo. El modelo fue entrenado emulando la arquitectura de *autoencoder* [8], con el objetivo de minimizar la función objetivo del modelo RDPG presentada en (2.7). Salvo que se aclare lo contrario, todos los modelos fueron entrenados usando el optimizador ADAM [28] provisto por PyTorch, considerando los siguientes hiperparámetros: 300 épocas y una tasa de aprendizaje de 0,001. A su vez se utilizó un criterio de parada anticipada si la función de pérdida no disminuye durante 10 iteraciones consecutivas.

Como entrada $\mathbf{X}_{in}$ se utilizaron matrices aleatorias, cuyos elementos son muestras extraídas de una distribución uniforme entre 0 y 1. Durante la etapa de inferencia, se evaluó el modelo sobre grafos con distribuciones idénticas a las utilizadas en el conjunto de entrenamiento y entradas $\mathbf{X}_{in}$ aleatorias. Se buscó poder recuperar con precisión la matriz $\mathbf{P} = \mathbf{X}\mathbf{X}^T$. A su vez se comparó los valores obtenidos para la función objetivo en

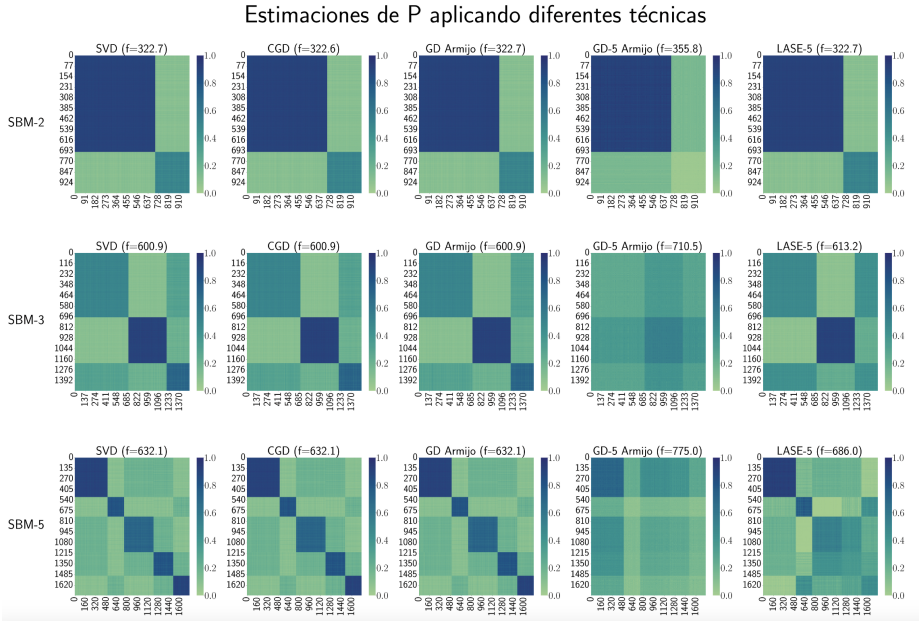


Figura 7.1: Se ilustra las diferentes estimaciones de las matrices $\mathbf{P} = \mathbf{X}\mathbf{X}^T$ obtenidas para los diferentes métodos evaluados (SVD, CGD y GD Armijo) en comparación con el modelo LASE de 5 capas, considerando grafos de $n = 1000$ nodos, con cantidad de comunidades $c \in [2, 3, 5]$ y dimensión de *embedding* $d = c$. Junto a cada título se especifica el valor obtenido para la función objetivo.

relación a otros métodos tradicionales como ser la descomposición SVD y los algoritmo Coordinate GD (CGD) y GD clásico con regla de Armijo (GD Armijo) a convergencia con un criterio de exactitud de 0,001 y un máximo de iteraciones de 10000. También se contrastaron los resultados con los obtenidos tras aplicar GD Armijo para 5 iteraciones (GD-5 Armijo) equivalente a la cantidad de capas utilizadas en LASE.

En los caso analizados se observó que efectivamente el modelo logra reconstruir la matriz $\mathbf{P} = \mathbf{X}\mathbf{X}^T$ de forma exitosa, logrando alcanzar valores de la función objetivo muy similares a los métodos clásicos. En la medida que se aumenta el número de comunidades, y por ende la dimensión de los *embeddings*, la aproximación de la matriz $\mathbf{P}$ comienza a deteriorarse un poco. Aún así, se logran recuperar exitosamente las comunidades del modelo SBM utilizado. La Figura 7.1 muestra en detalle las matrices $\mathbf{P}$ resultantes en comparación otros algoritmos.

Inferencia sobre grafos grandes

Se evaluó aumentar el número de nodos en aproximadamente un orden de magnitud en relación al número de nodos utilizados en las muestras de entrenamiento, manteniendo el mismo modelo aleatorio. Para todos los casos evaluados se obtuvieron resultados exitosos, confirmándose así la capacidad de generalización del modelo. El mismo puede ser entrenado sobre muestras pequeñas agilizando su entrenamiento sin perder calidad en inferencia.

En la Tabla 7.1 se compara la pérdida obtenida con los *embeddings* LASE contra otros algoritmos tradicionales como ser SVD, CGD y GD Armijo, considerando diferentes números de comunidades c y tamaños de grafo n. Los resultados obtenidos con

c	d	n	SVD	CGD (*)	GD (*)	GD iter	GD-5	LASE-5
2	2	1000	322.65	322.65	322.65	44	355.85	322.69
3	3	1500	448.73	448.73	448.74	38	518.86	451.45
5	5	1750	632.08	632.08	632.12	22	774.96	686.02
10	10	2750	1014.20	1014.21	1020.49	24	1156.46	1104.09

Tabla 7.1: Se detallan los valores resultantes tras evaluar la función objetivo $\|\mathbf{M} \circ (\mathbf{X}\mathbf{X}^T - \mathbf{A})\|_2^2$ para diferentes métodos (SVD, Coord GD y GD) en comparación con el modelo LASE de 5 capas (LASE-5), considerando grafos de diferente número de nodos n, cantidad de comunidades c y dimensión de *embedding* d. El (*) implica que los resultados son convergencia. La columna 'GD iter' indica la cantidad de iteraciones requeridas por GD para lograr la convergencia. La columna 'GD-5' presenta los resultados obtenidos considerando el algoritmo GD acotado a 5 iteraciones.

LASE mostraron ser competitivos respecto a los demás algoritmos tradicionales. A su vez, se observa que el número de iteraciones requeridas por el algoritmo GD tradicional para llegar a convergencia es hasta un orden de magnitud más que las capas necesarias por LASE para llegar a resultados similares. Esto confirma que el modelo LASE es significativamente más eficiente en términos computacionales.

Entrenamiento sobre subgrafos

Con el objetivo de soportar un escenario de entrenamiento más realista, donde posiblemente solo se tenga acceso a un único grafo de gran porte del cual se quieran obtener representaciones latentes, se propone realizar ciertos ajustes al conjunto de entrenamiento.

Para representar dicho escenario, se parte por generar un grafo con un conteo de nodos considerable, del orden de los 10 mil nodos. En particular, se considera una única realización de un SBM de 3 comunidades (SBM-3) con $n = 12000$ nodos distribuidos de forma asimétrica entre comunidades según el vector $\mathbf{n} = [6000, 4000, 2000]$ y cuya probabilidad de conexión inter-comunidad está dada por la siguiente matriz:

$$\mathbf{B} = \begin{bmatrix} 0{,}9 & 0{,}2 & 0{,}1 \\ 0{,}2 & 0{,}6 & 0{,}2 \\ 0{,}1 & 0{,}2 & 0{,}7 \end{bmatrix}. \tag{7.1}$$

Dado que el entrenamiento sobre el grafo completo no es viable computacionalmente, se propone particionar el mismo en subgrafos tomados aleatoriamente del mismo. Los subgrafos aleatorios, mantienen la distribución del grafo original, simplemente poseen una menor cantidad de nodos, haciendo de este un escenario análogo al primer método de entrenamiento, donde se entrena con grafos de menor porte para luego inferir en grafos considerablemente mayores.

Se generaron 1000 muestras de subgrafos con un tamaño del orden de los 600 nodos, lo que representa un 5 % del tamaño original. Se tomaron 800 subgrafos para entrenamiento y 200 para validación. Una vez finalizado el entrenamiento se realizó la inferencia sobre el grafo original, confirmando la capacidad de LASE de obtener *embeddings* que permiten aproximar de forma precisa la matriz $\mathbf{P}$ como se muestra en la Figura 7.2.

Esta forma de entrenamiento mediante el uso de subgrafos, posibilita la aplicación de LASE en problemas que involucren grafos de dimensiones grandes, los cuales no suelen ser viables de computar usando algoritmos tradicionales, como ser el GD, debido a su alto costo computacional consecuencia del término de segundo orden $\mathbf{X}\mathbf{X}^T$.

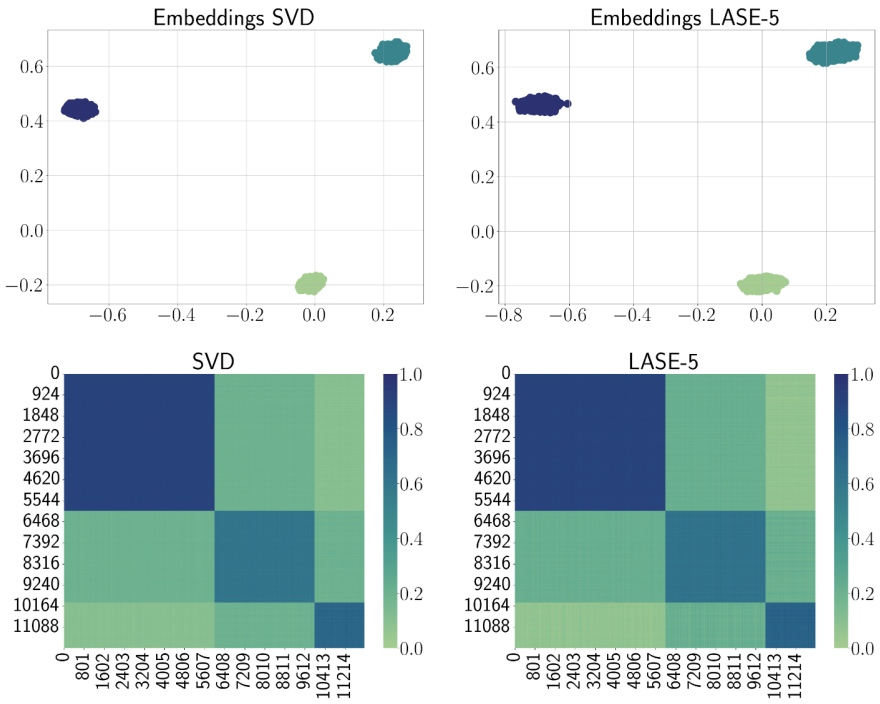


Figura 7.2: Matriz $\mathbf{P}$ resultante tras aplicar el modelo LASE de 5 capas (LASE-5) a un grafo SBM-3 de 12000 nodos en comparación con la obtenida tras aplicar la descomposición SVD tradicional. El modelo LASE-5 fue entrenado con subgrafos tomados de forma aleatoria del grafo original.

Comportamiento frente a grafos simétricos

Con el propósito de demostrar la eficiencia de LASE sobre otras arquitecturas GNN basadas únicamente en componentes convolucionales (GCN), se examinó el escenario de un grafo SBM simétrico, el cual comparte la misma cantidad de nodos en cada comunidad y posee una distribución idéntica de grados. Las GCNs suelen presentar dificultades para representar estos casos, dado que solamente logran capturar estructuras locales, y no poseen la habilidad de reflejar propiedades globales del grafo [43].

Para ejemplificar este comportamiento, y compararlo con los resultados de LASE, se consideraron dos familias de grafos SBM: una simétrica y otra asimétrica. Cada familia SBM posee 3 comunidades, una distribución de nodos dada por los vectores $\mathbf{n}_1 = [700, 500, 300]$ y $\mathbf{n}_2 = [300, 300, 300]$ respectivamente y probabilidad de conexión inter-comunidad dadas por:

$$\mathbf{B}_1 = \begin{bmatrix} 0{,}5 & 0{,}1 & 0{,}3 \\ 0{,}1 & 0{,}9 & 0{,}2 \\ 0{,}3 & 0{,}2 & 0{,}7 \end{bmatrix}, \ \mathbf{B}_2 = \begin{bmatrix} 0{,}7 & 0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}7 & 0{,}1 \\ 0{,}1 & 0{,}1 & 0{,}7 \end{bmatrix}. \tag{7.2}$$

Para obtener los *embeddings* de nodos se utilizaron dos métodos diferentes. El primero consistió de entrenar una GNN compuesta de un par de capas convolucionales TAGConv y una no linealidad tanh. Mientras que para el segundo se utilizó el modelo LASE con una configuración de 5 capas, con pesos distribuidos y normalización por nodo.

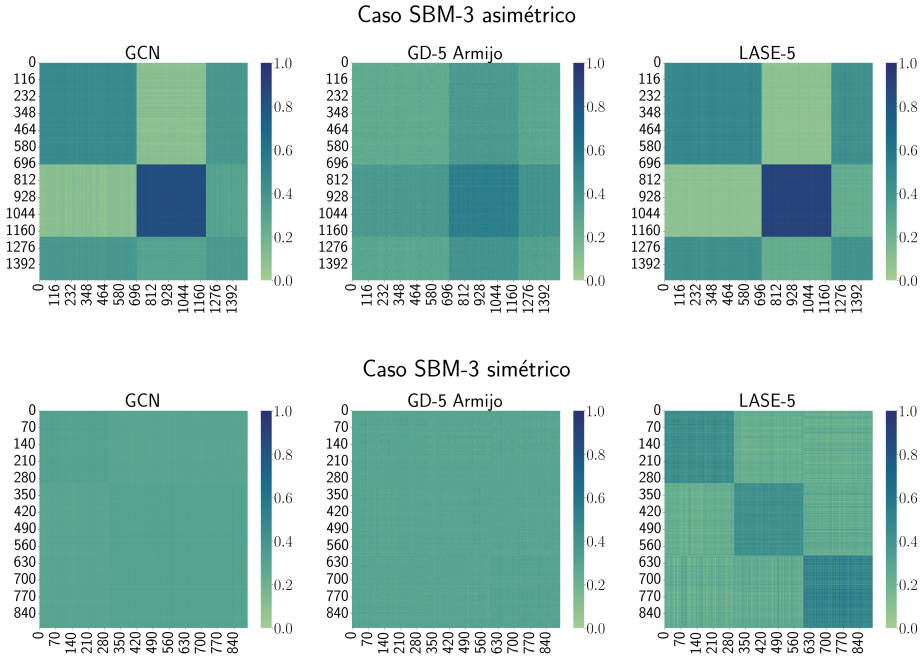


Figura 7.3: Comparación de los resultados obtenidos con un modelo GCN y con LASE-5, considerando los casos SBM asimétrico y simétrico. Se observa como el modelo GCN falla en recuperar la matriz $\mathbf{P}$ para el caso simétrico, frente al modelo LASE que sí la logra recuperar completamente.

En la primer familia de grafos SBM, asimétrica, se observa que los *embeddings* generados por ambos métodos permiten recuperar correctamente la matriz de probabilidades de interconexión $\mathbf{P}$. En cambio, cuando se considera la segunda familia de grafos SBM con distribución simétrica, solamente LASE es capaz de reconstruir la misma. Se atribuye este comportamiento al carácter local de los componentes convolucionales GCN. Particularmente en los casos simétricos, la red no logra diferenciar entre las comunidades vecinas dado que estas presentan exactamente la misma distribución. El modelo LASE en cambio, además de la componente convolucional GCN, posee una componente de *attention* GAT. Esta componente es clave en el proceso de capturar propiedades globales y así poder diferenciar efectivamente las comunidades en el caso simétrico. En la Figura 7.3 se presentan las matrices $\mathbf{P}$ resultantes para cada caso.

La incapacidad de las GCN de funcionar correctamente en casos simétricos puede justificarse analizando el problema en el dominio espectral (frecuencia). Para lo cual, se analizan los vectores propios asociados a los tres valores propios dominantes. La Figura 7.4 ilustra los vectores propios obtenidos para ambas configuraciones, SBM simétrico y asimétrico. En particular, se puede visualizar que para el caso simétrico, el primer vector propio resulta en una constante que no permite diferenciar entre las tres comunidades del grafo, mientras que el segundo y tercer vector propio sí aportan suficiente información para separar las mismas.

En el dominio espectral, el equivalente de pasar una señal $\mathbf{x}$ por una capa GCN es considerar la Graph Fourier Transform (GFT) de la señal. Tal cual se detalló la Sección 3.2.1, la GFT se define como la proyección de una señal $\mathbf{x}$ en el espacio de vectores propios del operador GSO $\mathbf{S}$. Es decir, dado $\mathbf{S} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$:

$$\text{GFT}(\mathbf{x}) = \tilde{\mathbf{x}} = \mathbf{V}^T\mathbf{x}. \tag{7.3}$$

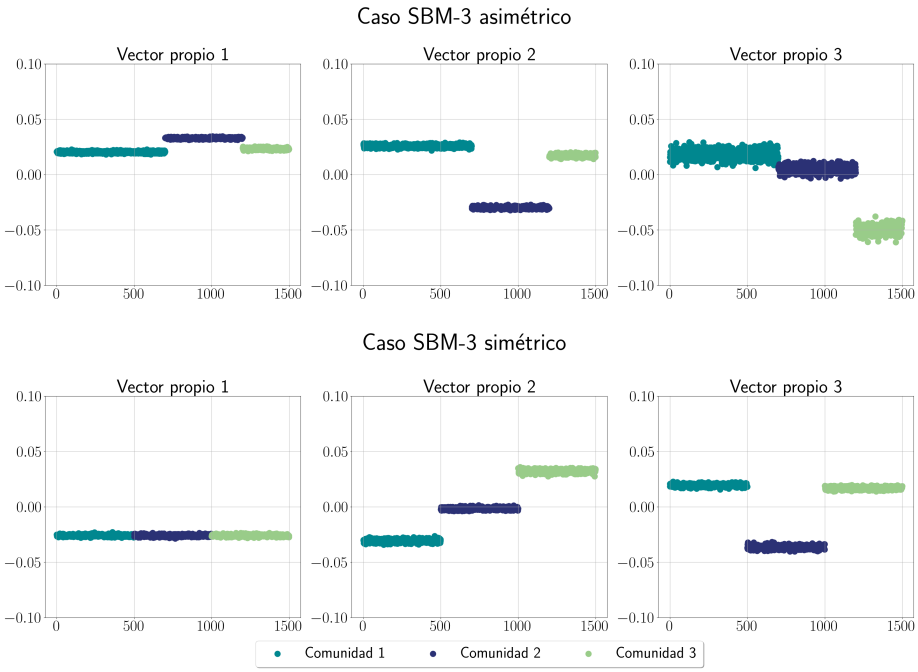


Figura 7.4: Comparación de los vectores propios dominantes asociados a realizaciones de un modelo SBM de 3 comunidades con distribución simétrica y asimétrica.

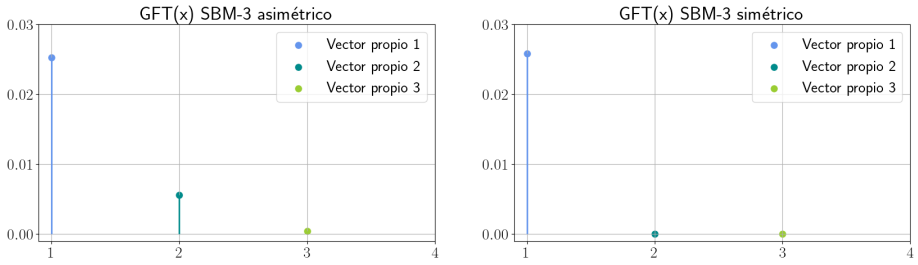


Figura 7.5: Comparación de las Graph Fourier Transform (GFT) obtenidas en los tres vectores propios dominantes para los casos SBM-3 simétrico y asimétrico considerando una señal anónima $\mathbf{x}$.

Entonces, si se considera una señal anónima $\mathbf{x}$, como ser un vector constante cuyas entradas son todas unos, y tomando $\mathbf{S} = \mathbf{A}$ como operador GSO en ambos escenarios, se puede observar que en particular para el caso simétrico, las GFT de los vectores propios 2 y 3 se van a cero (ver Figura 7.5). Como estos son los únicos que aportan información suficiente para distinguir las comunidades, queda demostrada la incapacidad de las GCN frente a este tipo de escenario.

7.1.2. GLASE

Dentro de los experimentos realizados, también se evaluó la performance de la versión generalizada del modelo (GLASE), la cual nos habilita a modelar grafos hete-

rofilios. Como se detalla en la Sección 6.2, la misma incorpora una matriz diagonal $\mathbf{Q}$ la cual se utiliza para definir el producto interno indefinido $\mathbf{XQX}^T$. Para obtener esta matriz, es necesario primero calcular los d valores propios dominantes (en magnitud) de la matriz de adyacencia del grafo $\mathbf{A}$, donde d representa la dimensión elegida para el embedding $\mathbf{X}$. La matriz $\mathbf{Q}$ tendrá en su diagonal los primeros p elementos seteados en $+1$, siendo p el numero de valores propios dominantes positivos de $\mathbf{A}$, y los siguientes $q = d - p$ elementos seteados en -1, con q correspondiendo al número de valores propios dominantes negativos.

Obtención de $\mathbf{Q}$

Dentro de esta fase experimental, uno de los objetivos planteados fue evaluar la viabilidad de aprender $\mathbf{Q}$, considerándola como parte de los parámetros aprendibles del modelo GLASE. Para esto, se estudiaron dos posibles caminos que se describen brevemente a continuación.

El primero y más directo, consiste simplemente en considerar a $\mathbf{Q}$ como una salida del modelo, la cual se desea aprender. Se ajusto entonces la arquitectura del mismo de modo de dar dos salidas: $\mathbf{X}$ y $\mathbf{Q}$. En lugar de considerar la matriz $\mathbf{Q}$ completa, se consideró solamente su diagonal diag($\mathbf{Q}$), fijando el resto de sus elementos en cero. Esto permite disminuir la complejidad del problema, al disminuir la cantidad de parámetros del modelo y por ende tener un menor grado de libertad. A su vez, como los elementos de diag($\mathbf{Q}$) deben converger en valor absoluto a 1, se agregó un término de regularización L1 a la función de pérdida, la cual está dada por la siguiente ecuación:

$$f(\mathbf{X}, \mathbf{Q}) = \lambda \|\mathbf{A} - \mathbf{XQX}^T\|_F^2 + (1 - \lambda) f_{reg}(\mathbf{Q}), \tag{7.4}$$

siendo $f_{reg}(\mathbf{Q}) = \sum_{v \in \text{diag}(\mathbf{Q})} |v - 1| + \sum_{v \in \text{diag}(\mathbf{Q})} |v + 1|$.

En el segundo camino, se consideró separar el entrenamiento en dos etapas. La primera dedicada en su totalidad a aprender $\mathbf{Q}$. Para esto se fijaron los pesos $\mathbf{W}_i$ con una inicialización sub-óptima, considerándolos como matrices identidad multiplicadas por un factor $\alpha = 0{,}001$. En la segunda etapa se fija el valor de $\mathbf{Q}$ según el valor obtenido en la etapa anterior, y se retoma el entrenamiento para aprender las matrices $\mathbf{W}_i$ óptimas.

Lamentablemente ambos intentos no fueron exitosos y dieron lugar a resultados poco concluyentes. Se considera continuar esta línea de investigación como un trabajo futuro.

Como alternativa, se optó por determinar $\mathbf{Q}$ directamente calculando los valores propios de la matriz $\mathbf{A}$. Para los casos donde se tiene un grafo con un número elevado de nodos (i.e. $n \geq 1000$), suele ser deseable evitar el cálculo de los valores propios sobre la matriz de adyacencia completa. Como alternativa se propone generar subgrafos del grafo original y computar para cada uno de estos los valores propios dominantes. La matriz final $\mathbf{Q}$ queda determinada por el resultado más frecuente para el total de subgrafos generados.

Entrenamiento GLASE

Una vez definida la matriz $\mathbf{Q}$, se procedió a entrenar un modelo GLASE con 5 capas, pesos distribuidos y normalización de nodo, con el objetivo de minimizar la función objetivo del modelo GRDPG detallada en la ecuación (6.14).

Se utilizaron diferentes conjuntos de datos de distribuciones SBM variando el número de comunidades, el número de nodos y su distribución de grados. Durante la etapa de entrenamiento, se observó una mejor performance tras inicializar las entradas de la matriz $\mathbf{X}$ como parte de una esfera de dimensión d y norma 1. Por ejemplo,

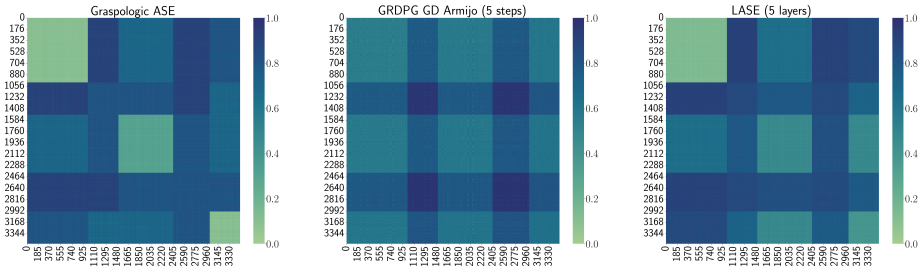


Figura 7.6: Matriz $\mathbf{P}$ resultante tras aplicar el modelo GLASE a un grafo SBM-5 de 3500 nodos en comparación con las obtenidas tras aplicar la función ASE provista por la biblioteca Graspologic [1] y el algoritmo GD.

para el caso $d = 2$, las entradas de $\mathbf{X}$ corresponden a un vector de norma 1 y cuyas coordenadas varían a lo largo de la semi-circunferencia formada entre 0 y $\pi/2$.

La Figura 7.6 ilustra como el modelo GLASE es capaz de aproximar eficazmente la matriz $\mathbf{P}$ para casos con valores propios negativos. Para esto se consideró una distribución SBM-5 de 3500 nodos distribuidos según el vector $\mathbf{n} = [1000, 500, 900, 600, 500]$ y cuya probabilidad de conexión inter-comunidad está dada por:

$$\mathbf{B} = \begin{bmatrix} 0{,}1 & 0{,}9 & 0{,}7 & 0{,}9 & 0{,}8 \\ 0{,}9 & 0{,}8 & 0{,}8 & 0{,}9 & 0{,}7 \\ 0{,}7 & 0{,}8 & 0{,}3 & 0{,}8 & 0{,}7 \\ 0{,}9 & 0{,}9 & 0{,}8 & 0{,}8 & 0{,}8 \\ 0{,}8 & 0{,}7 & 0{,}7 & 0{,}8 & 0{,}1 \end{bmatrix}. \tag{7.5}$$

7.1.3. Comportamiento frente a información faltante

Otro punto de interés dentro de este marco experimental fue poder validar el comportamiento de LASE (y GLASE) frente a situaciones con información faltante. Como se cubre en la Sección 6.1, la familia de modelos LASE heredan esta capacidad de generalización al estar basados en el algoritmo utilizado en *Efficient ASE* y por ende permitir la incorporación de una máscara binaria dada por la matriz $\mathbf{M}$ como parte de su arquitectura.

Se plantea entonces el escenario donde se tiene un grafo $\mathcal{G}$ del cual se desconocen ciertas aristas de ciertos nodos. Las aristas faltantes son codificadas en la matriz máscara $\mathbf{M}$. En particular, se consideran grafos con una distribución SBM-2 de 2000 nodos, distribuidos según el vector $\mathbf{n} = [1400, 600]$ y con una matriz de probabilidad de conexión inter-comunidad dada por:

$$\mathbf{B}_1 = \begin{bmatrix} 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}5 \end{bmatrix}. \tag{7.6}$$

Por su parte, se generan varias máscaras $\mathbf{M}_i$ codificando diferentes niveles de aristas faltantes. Para la construcción de las mismas, se eligen aleatoriamente 150 nodos, para los cuales se eliminan aristas con cierta probabilidad $p_i \in [0{,}3, 0{,}5, 0{,}7, 0{,}9]$.

Se entrena cada una de estas configuraciones utilizando un modelo LASE con 10 capas, con pesos distribuidos, normalización por nodo y dimensión de *embeddings* $d = 2$. A su vez, se computan los *embeddings* ASE mediante otros métodos tradicionales como ser: la descomposición SVD sobre las matrices de adyacencia enmascaradas, así como utilizando el GD propuesto en *Efficient ASE* a convergencia y acotado a 10

iteraciones (equivalentes al número de capas de LASE). La Figura 7.7 muestra los *embeddings* obtenidos en cada caso.

Se puede verificar que a diferencia de ASE basado en SVD, el modelo LASE logra codificar correctamente los nodos con aristas faltantes. Incluso en escenarios con un 90 % de aristas desconocidas, LASE con solo 10 capas, logra separar exitosamente las dos comunidades, superando ampliamente al algoritmo GD acotado a 10 iteraciones.

Caso de uso: Votos de senadores en un parlamento

A continuación se presenta un ejemplo de un escenario más realista que contempla información faltante, el cual considera un parlamento con senadores pertenecientes a dos partidos políticos y ciertas leyes que se deben votar. Para simular este escenario, se considera un grafo bipartito donde los nodos corresponden a los senadores y a la leyes a votar, y las aristas (i, j) corresponden a los votos afirmativos de un senador i a una ley j.

Las leyes podrán ser propuestas por ambos partidos, siendo más probable que un senador vote afirmativamente las leyes propuestas por su partido en lugar de las propuestas por la oposición. También existen ciertas leyes que son trasversales a los partidos, y que por ende tendrán votos afirmativos similares entre partidos.

A su vez, se tiene un subconjunto de senadores de ambos partidos los cuales pueden estar ausenten en la votación de las leyes. La matriz $\mathbf{M}$ codificará si estos senadores están presentes al momento de votar.

Se simulo el grafo con 100 senadores en cada partido y un total de 460 leyes, 200 propuestas por el partido 1, otras 200 propuestas por el partido 2 y 60 leyes transversales a ambos partidos. La probabilidad de que un senador i vote una ley j se distribuye de la siguiente manera:

- La probabilidad que un senador vote una ley propuesta por su partido es $p = 0{,}8$
- La probabilidad que un senador vote una ley propuesta por el partido opositor es $p = 0{,}01$
- La probabilidad que un senador vote una ley bi-partidaria es $p = 0{,}2$

Se compararon los *embeddings* ASE generados utilizando SVD tradicional, GD Armijo y entrenando un modelo GLASE de 10 capas. En la Figura 7.8 se visualizan los resultados obtenidos. En los casos de SVD y GD se destaca una clara alineación de los *embeddings* de leyes y senadores del mismo partido, así como ortogonalidad respecto a los *embeddings* de leyes y senadores de la oposición. Mientras que para GLASE dicha alineación se pierde, solo se mantiene la ortogonalidad entre *embeddings* de leyes oposición y entre *embeddings* de senadores oposición. Se atribuye este comportamiento al producto interno *indefinido* $\mathbf{XI}_{p,q}\mathbf{X}^T$ del modelo GRDPG. Como vimos en la Sección 2.1.3, pueden producirse otras soluciones de *embeddings* GRDPG a la optimización (2.8) que son producto de transformaciones dentro del grupo ortogonal *indefinido*. La Figura 7.8 también confirma que tanto GD como GLASE, logran asignar los senadores ausentes con su correspondiente partido, mientras que el SVD no logra mapearlos correctamente.

Por su parte, se plantea un problema de predicción de enlaces, donde se busca determinar cómo hubieran resultado los votos si los senadores hubiesen estado presentes en las diferentes sesiones. Para esto se consideraron las dos arquitecturas propuestas en la Sección 6.3.2. Para realizar la tarea de predicción de enlaces, se particionan las aristas del grafo en conjuntos de entrenamiento, validación y testeo. Para esto, se hace uso del módulo *Random Link Split* de la biblioteca de PyG. El modelo se entrena con el objetivo de minimizar la función de pérdida *Binary Cross Entropy* (BCE). La Tabla 7.2 resume los resultados obtenidos tras comparar la performance de VGAE

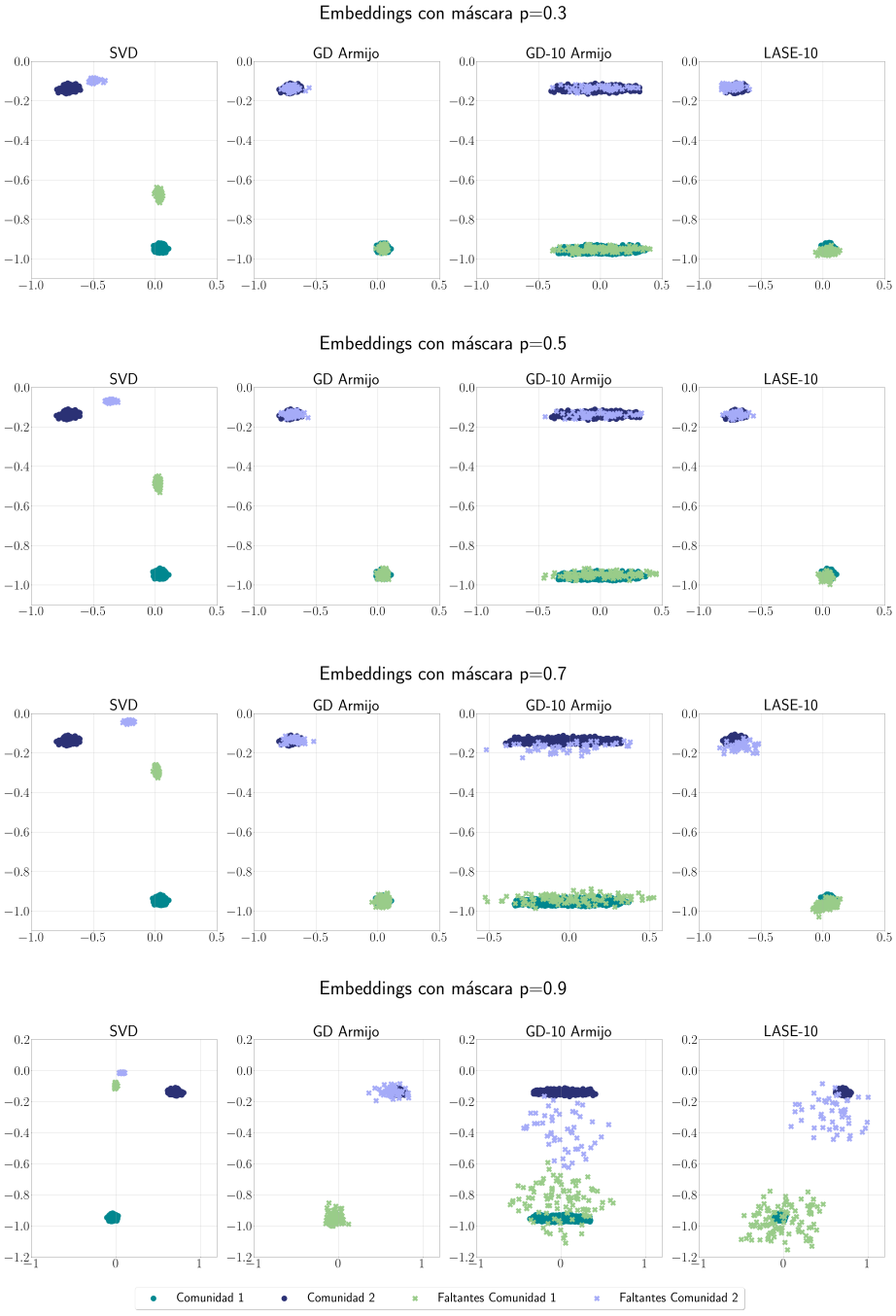


Figura 7.7: Embeddings ASE obtenidos para SVD y GD comparados con los obtenidos con LASE considerando máscaras $\mathbf{M}_i$ que codifican diferentes grados de información faltante.

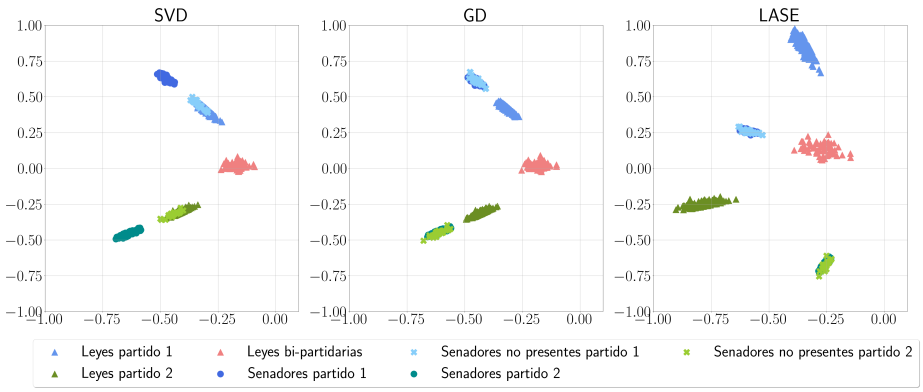


Figura 7.8: La gráfica ilustra como tanto GD y LASE logran mapear los senadores ausentes con sus respectivos partidos.

missing values	**GCN**	**ASE PE**	**GLASE PE**	**GLASE PE e2e**
30 %	0.446 ± 0.014	0.854 ± 0.004	**0.856 ± 0.006**	0.855 ± 0.002
50 %	0.450 ± 0.014	0.851 ± 0.005	**0.855 ± 0.006**	0.852 ± 0.008
70 %	0.511 ± 0.010	0.848 ± 0.006	**0.855 ± 0.005**	0.852 ± 0.005
90 %	0.528 ± 0.004	0.823 ± 0.041	0.830 ± 0.025	**0.835 ± 0.016**

Tabla 7.2: Se presentan los resultados obtenidos en la predicción de votos de los senadores considerando diferentes probabilidades de estar ausentes durante la sesión. Se evalúan el accuracy considerando diferentes modelos y Positional Encodings.

tradicional basado en GCN y su adaptación considerando GLASE para diferentes grados de aristas desconocidas. Se comprueba que la incorporación de GLASE mejora notablemente el *accuracy* de la predicción. Como comentario adicional, se destaca que a simple vista los resultados de *accuracy* del modelo GCN parecen tener un efecto de "mejora" a mayor grado de información faltante. Sin embargo, si se analiza en detalle, este efecto se debe a que cuánto más *sparse* es A, el modelo GCN tiende a predecir más ceros que unos, y como de por sí la proporción de entradas cero y uno de la matriz A no es balanceada (hay más ceros que unos) causa este falso efecto de "mejora".

De forma análoga al análisis realizado en la Sección 7.1.1, se procede a estudiar en detalle la performance de la GCN considerando el dominio espectral. Para esto se calculan los vectores propios correspondientes a los cuatro valores propios dominantes de la matriz $\mathbf{A}$. En la Figura 7.9 se detallan los vectores propios resultantes. Se observa que los vectores correspondientes a las dimensiones 1 y 3 solo permiten distinguir los diferentes tipos de nodos: senadores, leyes partidarias y leyes bi-partidarias; mientras que los vectores 2 y 4 son los que permiten discriminar entre las agrupaciones internas de estos, en este caso el partido político al cual pertenece cada senador y el partido político del cual se origina cada ley partidaria.

Se procede a calcular la GFT de dos señales anónimas: $\mathbf{x}_1$, que toma valores constantes en las posiciones correspondientes a los senadores y cero en el resto; y $\mathbf{x}_2$ que toma valores constantes en las posiciones correspondientes a las leyes y cero en el resto. En la Figura 7.10 se puede comprobar que en ambos casos la GFT resultante para los vectores propios 2 y 4 tiende a cero. Esto implica que la GCN a lo sumo podrá distinguir entre grupos de senadores y de leyes, y entre leyes partidarias y bi-partidarias;

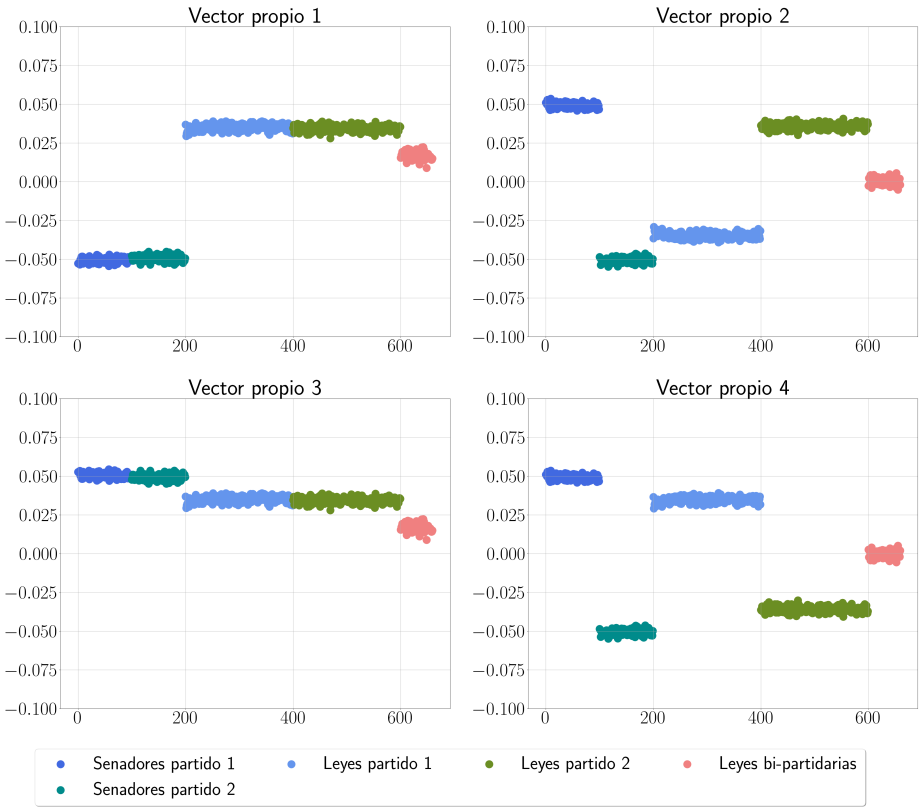


Figura 7.9: Comparación de los vectores propios dominantes asociados al grafo de senadores y leyes. Las dimensiones 1 y 3 permiten diferenciar únicamente los diferentes tipos de nodos: senadores, leyes partidarias y leyes bi-partidarias. Las dimensiones 2 y 4 permiten diferenciar agrupaciones internas dentro de cada tipo de nodo, i.e. partido político al cual pertenece cada senadores y partido político del cual origina cada ley.

pero no es capaz de distinguir los sub-grupos existentes dentro de senadores y leyes partidarias.

7.1.4. Mecanismos de *sparse attention*

La siguiente sección describe los experimentos realizados para diferentes configuraciones de *attention* en el modelo LASE. El objetivo de los mismos fue comprobar formas de minimizar el costo computacional del modelo, y por ende reducir los tiempos de inferencia del mismo, sin tener pérdidas significativas en la calidad en los *embeddings* resultantes.

Como se describe en la Sección 6.1, se propone utilizar mecanismos de *sparse attention* los cuales permiten reducir la dependencia cuadrática $\mathcal{O}(n^2)$ del término $\mathbf{X}\mathbf{X}^T$ de LASE por una dependencia lineal $\mathcal{O}(n)$. Para lograr esto se incorpora una matriz *attention mask* a la arquitectura LASE según la ecuación (6.13).

En particular, se experimentó con tres tipos de matrices de *attention*:

- Matrices generadas por un modelo Erdös–Rényi (ER)
- Matrices generadas por un modelo Watts-Strogatz (WS)

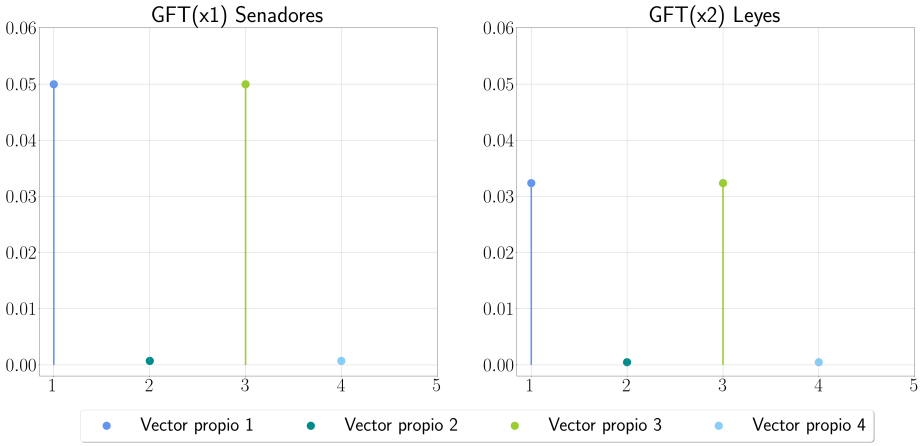


Figura 7.10: Comparación de las Graph Fourier Transform (GFT) obtenidas en los cuatro vectores propios dominantes considerando una señal anónima $\mathbf{x}_1$ asociada únicamente a los senadores y una señal anónima $\mathbf{x}_2$ asociada únicamente a las leyes partidarias.

- Matrices generadas con el mecanismo BigBird (BB)

Se utilizó un modelo LASE de 5 capas, pesos distribuidos, normalización por nodo y dimensión de *embedding* $d = 3$; el cual se entrenó a partir de muestras de grafos SBM-3 de 240 nodos, con distribución $\mathbf{n} = [120, 80, 40]$ y probabilidad inter-comunidad dada por:

$$\mathbf{B} = \begin{bmatrix} 0{,}9 & 0{,}2 & 0{,}1 \\ 0{,}2 & 0{,}6 & 0{,}2 \\ 0{,}1 & 0{,}2 & 0{,}7 \end{bmatrix}. \tag{7.7}$$

Tras el entrenamiento, se evaluó la posibilidad de recuperación de la matriz $\mathbf{P}$ para los diferentes mecanismos, así como los resultados obtenidos de la función objetivo a minimizar. Para la matriz *attention* ER se consideraron tres configuraciones con probabilidad de aristas: $p = 0{,}5$, $p = 0{,}3$ y $p = 0{,}1$ respectivamente. Como se puede apreciar en la Figura 7.11, se observan buenos resultados para el primer caso, pero no así para los últimos dos.

Por su parte, el modelo WS permite variar dos parámetros: k la cantidad de vecinos con los que se conecta cada nodo, lo cual determina el grado, y p la probabilidad de reconectar un nodo con otro cualquiera, lo cual determina el nivel de aleatoriedad. En este caso, se consideraron configuraciones con grados análogos a los elegidos en el caso ER. Se observó que la incorporación de *attention* WS permite incrementar el nivel de "esparcidad" manteniendo resultados consistentes. En particular, se observan mejores resultados en escenarios con bajo nivel de aleatoriedad. Esto último da la intuición de que el modelo se ve favorecido cuando se retiene la estructura de anillo en la matriz de *attention*. La Figura 7.12 ilustra los resultados obtenidos para cada configuración de WS.

Por último, para el modelo BB se consideran dos parámetros: w el tamaño de la ventana de desplazamiento (*sliding window*) donde se atiende a los vecinos de cada nodo que se encuentran a $w/2$ nodos a la derecha y a la izquierda, y r la cantidad de entradas aleatorias. La matriz de *attention* BB también llevó a resultados de buena calidad que permiten incrementar el grado de esparcidad en comparación con el caso ER. Similar al caso WS, se observan mejores resultados en escenarios con bajo nivel

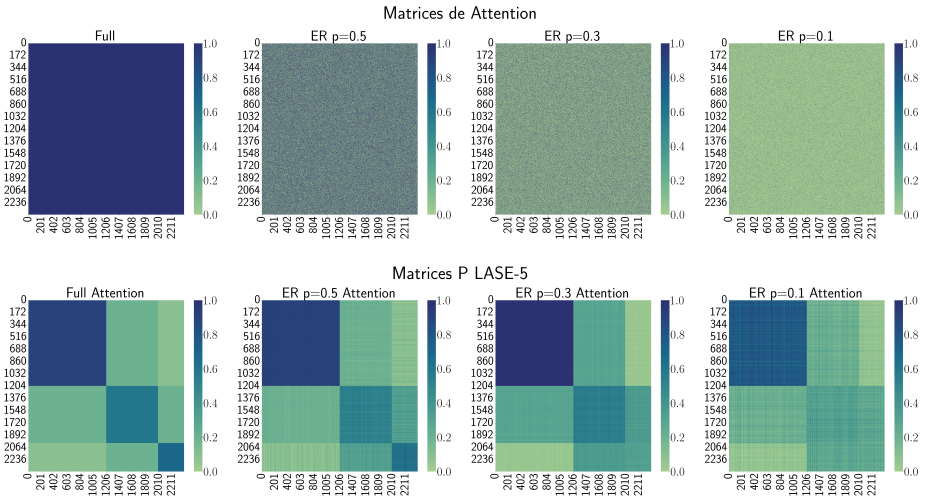


Figura 7.11: Matrices $\mathbf{P}$ resultante tras aplicar el modelo LASE con diferentes attention masks basadas en modelos Erdös Rényi.

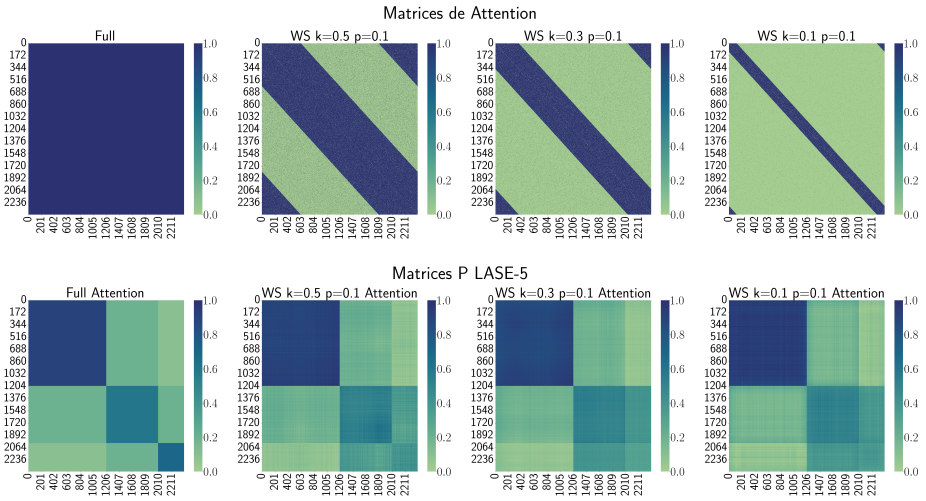


Figura 7.12: Matrices $\mathbf{P}$ resultante tras aplicar el modelo LASE con diferentes attention masks basadas en modelos Watts Strogatz.

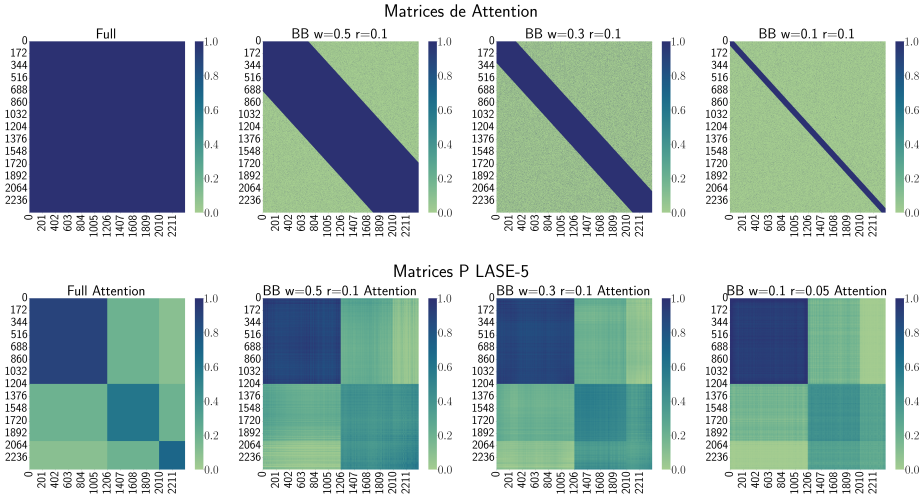


Figura 7.13: Matrices $\mathbf{P}$ resultante tras aplicar el modelo LASE con diferentes attention masks basadas en modelos BigBird.

Attention p_i	**ER**	**WS (p = 0.1)**	**BB (r = 0.1)**
0.5	912.99	**912.52**	947.99
0.3	945.63	932.41	**930.47**
0.1	984.46	**930.67**	948.97

Tabla 7.3: Se detallan los valores resultantes tras evaluar la función objetivo $\|\mathbf{M} \circ (\mathbf{X}\mathbf{X}^T - \mathbf{A})\|_2^2$ considerando las diferentes máscaras de attention ER, WS y BB, variando su densidad de probabilidad de aristas p_i.

de aleatoriedad. Los resultados obtenidos para cada configuración de BB se detallan en la Figura 7.13.

La Tabla 7.3 compara los resultados de la función de pérdida (6.1) para las diferentes configuraciones de *attention*, reafirmando el comportamiento observado en los gráficos de la matriz $\mathbf{P}$.

Tiempos de inferencia

Por su parte, se analizan los tiempos de inferencia para las diferentes configuraciones de *attention*. Para esto se considera un grafo SBM de 10 comunidades, con distribución de nodos dada por el vector $\mathbf{n} = [80, 50, 80, 40, 50, 60, 50, 50, 40, 50]$ y cuya probabilidad de conexión inter-comunidad está dada por:

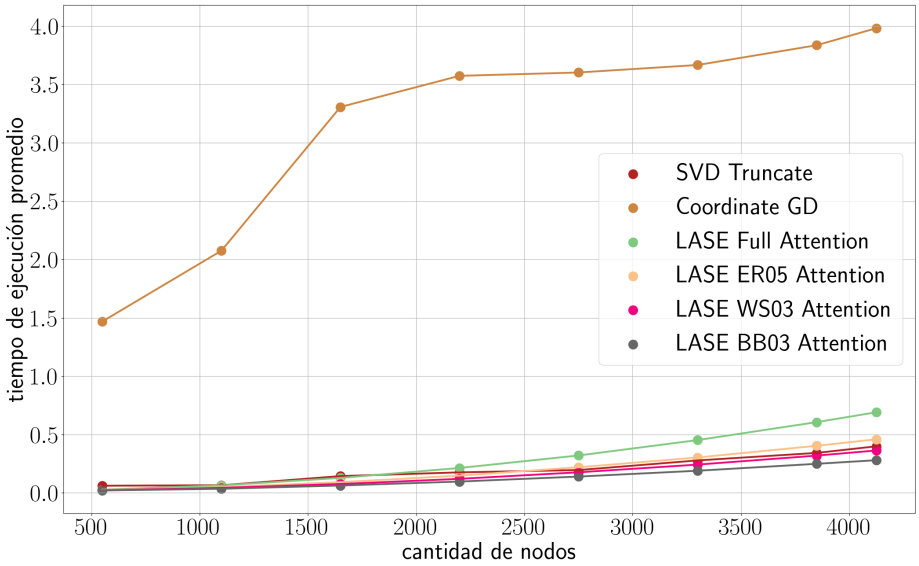


Figura 7.14: Comparación de los tiempos de inferencia promedio del modelo LASE considerando diferentes attention masks contra los tiempos de ejecución de algoritmos tradicionales como ser SVD y CGD, en función de la cantidad de nodos del grafo.

$$\mathbf{B} = \begin{bmatrix} 0{,}9 & 0{,}1 & 0{,}1 & 0{,}3 & 0{,}1 & 0{,}1 & 0{,}2 & 0{,}1 & 0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}6 & 0{,}3 & 0{,}3 & 0{,}2 & 0{,}2 & 0{,}1 & 0{,}2 & 0{,}3 & 0{,}2 \\ 0{,}1 & 0{,}3 & 0{,}8 & 0{,}2 & 0{,}1 & 0{,}3 & 0{,}1 & 0{,}1 & 0{,}2 & 0{,}2 \\ 0{,}3 & 0{,}3 & 0{,}2 & 0{,}7 & 0{,}3 & 0{,}1 & 0{,}3 & 0{,}1 & 0{,}3 & 0{,}3 \\ 0{,}1 & 0{,}2 & 0{,}1 & 0{,}3 & 0{,}9 & 0{,}1 & 0{,}2 & 0{,}1 & 0{,}1 & 0{,}2 \\ 0{,}1 & 0{,}2 & 0{,}3 & 0{,}1 & 0{,}1 & 0{,}5 & 0{,}2 & 0{,}1 & 0{,}1 & 0{,}3 \\ 0{,}2 & 0{,}1 & 0{,}1 & 0{,}3 & 0{,}2 & 0{,}2 & 0{,}8 & 0{,}2 & 0{,}3 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0{,}1 & 0{,}1 & 0{,}1 & 0{,}1 & 0{,}2 & 0{,}6 & 0{,}1 & 0{,}1 \\ 0{,}1 & 0{,}3 & 0{,}2 & 0{,}3 & 0{,}1 & 0{,}1 & 0{,}3 & 0{,}1 & 0{,}9 & 0{,}1 \\ 0{,}1 & 0{,}2 & 0{,}2 & 0{,}3 & 0{,}2 & 0{,}3 & 0{,}1 & 0{,}1 & 0{,}1 & 0{,}7 \end{bmatrix}. \tag{7.8}$$

En la Figura 7.14, se puede apreciar la variación en los tiempos de inferencia en la medida que se aumenta el número de nodos del grafo, para cada una de las configuraciones de *attention* estudiadas, así como para métodos tradicionales como CGD y descomposición SVD. Para estos experimentos se utilizó un servidor con GPU *NVIDIA GeForce RTX 3060* de 12GB de RAM. Se observa una reducción de hasta un 60 % usando mecanismos de *sparse attention* en comparación con el escenario original de *full attention*.

En la Tabla 7.4 se da un detalle de los tiempos de ejecución observados en inferencia para cada caso. El mecanismo de *attention* BB con $w = 0{,}3$ y $r = 0{,}1$, lo que equivale a una probabilidad de arista de $p = 0{,}1$, logra los mejores tiempos de inferencia. Siendo hasta 14.2 veces más rápido que el algoritmo CGD y 1.4 veces más rápido que el SVD clásico.

nodos	SVD	GD	LASE full	LASE ER 0.5	LASE WS 0.3	LASE BB 0.3
550	0.06 ± 0.01	1.47 ± 0.04	0.03 ± 0.01	0.02 ± 0.01	0.02 ± 0.01	$\mathbf{0.02 \pm 0.01}$
1650	0.14 ± 0.01	3.31 ± 0.08	0.13 ± 0.01	0.09 ± 0.01	0.08 ± 0.01	$\mathbf{0.06 \pm 0.01}$
2750	0.20 ± 0.01	3.44 ± 0.07	0.32 ± 0.01	0.22 ± 0.01	0.18 ± 0.01	$\mathbf{0.14 \pm 0.01}$
3850	0.34 ± 0.01	3.84 ± 0.08	0.61 ± 0.01	0.40 ± 0.01	0.32 ± 0.01	$\mathbf{0.25 \pm 0.01}$
4125	0.40 ± 0.01	3.98 ± 0.09	0.69 ± 0.01	0.46 ± 0.01	0.36 ± 0.01	$\mathbf{0.28 \pm 0.01}$

Tabla 7.4: Se reportan los tiempos de inferencia promedio $\pm$ su desviación estándar considerando 10 iteraciones de cada algoritmo.

7.2. Datasets reales

En esta sección se analizan los resultados de los experimentos realizados considerando ciertos conjuntos de datos basados en información real. En particular, se estudian dos tipos de escenarios de grafos: homofilios y heterofilios. El objetivo de estos experimentos fue probar las arquitecturas propuestas en la Sección 6.3 en escenarios reales, usando los *embeddings* GLASE como *Positional Encodings* (PE).

7.2.1. Grafos Homofilios

Para el caso de grafos homofilios, se examino la performance del modelo GLASE como parte de una tarea de clasificación de nodos considerando dos conjuntos de datos comúnmente utilizados en la literatura: CORA [37] y Amazon Photo [39].

El conjunto de datos CORA consiste de una colección de 2708 publicaciones científicas las cuales se clasifican en 7 clases. Las conexiones entre las publicaciones son basadas en las citas de unas a las otras, comprendiendo un total de 5429 aristas. Cada nodo posee un vector de atributos el cual consiste en un *embedding* de dimensión 1433 generado con la técnica de *bag of words*, sus entradas toman valores de 0 ó 1 indicando la presencia o ausencia de una palabra en la publicación.

El conjunto de datos Amazon Photo es una colección de 7650 productos de 8 categorías diferentes. Las conexiones entre estos significan que son productos que frecuentemente son comprados juntos. Cada nodo tiene un vector de atributos asociado el cual consiste de un *embedding* de dimensión 745 también generado con un *bag of words*, sus entradas toman 0 ó 1 indicando la presencia o ausencia de una palabra en la correspondiente evaluación del producto.

Para la obtención de *embeddings*, en ambos conjuntos de datos se consideró un modelo GLASE con 5 capas, pesos distribuidos, normalización por nodo y dimensión de *embedding* $d = 6$ para CORA y $d = 8$ para Amazon. Para el caso de Amazon a su vez se consideró una matriz de *attention* WS con $r = 0{,}1$ y $p = 0{,}1$, dado que el tamaño del grafo y el hardware disponible no permitieron que fuese viable entrenar usando *full attention*.

Para la tarea de clasificación, se estudiaron las dos arquitecturas de clasificación propuestas en la Sección 6.3.1. En la primera, se comienza entrenando un modelo GLASE sobre el grafo de forma independiente (*offline*) para obtener los *embeddings* GLASE-PE. Luego, se concatenan los mismos con los vectores de atributos de nodos, para finalmente alimentar un clasificador basado en una GAT de 3 capas. En la segunda, se propone incorporar el modelo GLASE como parte de la arquitectura de clasificación basada en GAT y realizar un entrenamiento *end-to-end* (e2e). En particular, se buscó evaluar dos conceptos: la mejora provista al incorporar PE en el modelo de clasificación, así como la calidad de los *embeddings* GLASE como PE. Por

δ	**sin PE**	**ASE PE**	**GLASE PE**	**GLASE PE e2e**	**PowerEmbed**
0	86.8 ± 0.4	86.5 ± 0.6	86.4 ± 0.3	$\mathbf{86.9 \pm 0.3}$	85.0 ± 0.4 (*)
0.2	84.6 ± 0.3	85.0 ± 0.5	85.8 ± 0.3	$\mathbf{85.8 \pm 0.6}$	-
0.4	83.1 ± 0.3	83.2 ± 0.5	84.3 ± 0.4	$\mathbf{85.4 \pm 0.5}$	-
0.6	79.3 ± 0.5	79.0 ± 0.3	83.2 ± 0.6	$\mathbf{85.5 \pm 0.6}$	-
0.8	73.8 ± 0.8	73.0 ± 0.6	79.6 ± 0.4	$\mathbf{84.6 \pm 0.5}$	-

Tabla 7.5: Se reporta el accuracy promedio $\pm$ su desviación estándar considerando 10 particiones aleatorias del grafo CORA y diferentes proporciones de aristas desconocidas δ. El resultado en (*) corresponde al método PowerEmbed(Lap)-10 que usa la matriz Laplaciana y 10 iteraciones, el cual fue obtenido de [27] ("-" indica que los resultados no están disponibles).

δ	**sin PE**	**ASE PE**	**GLASE PE**	**GLASE PE e2e**	**PowerEmbed**
0	94.3 ± 0.6	93.9 ± 0.6	94.2 ± 0.6 (*)	94.4 ± 0.4 (*)	$\mathbf{94.6 \pm 0.2}$ (**)
0.2	94.2 ± 0.4	94.0 ± 0.5	93.7 ± 0.8 (*)	$\mathbf{94.3 \pm 0.6}$ **(*)**	-
0.4	93.9 ± 0.7	93.6 ± 1.5	93.8 ± 0.6 (*)	$\mathbf{94.0 \pm 0.3}$ **(*)**	-
0.6	93.5 ± 0.6	93.5 ± 0.5	93.1 ± 0.9 (*)	$\mathbf{94.0 \pm 0.6}$ **(*)**	-
0.8	92.9 ± 0.7	92.9 ± 0.6	92.6 ± 0.7 (*)	$\mathbf{93.3 \pm 0.6}$ **(*)**	-

Tabla 7.6: Se reporta el accuracy promedio $\pm$ su desviación estándar considerando 10 particiones aleatorias del grafo Amazon y diferentes proporciones de aristas desconocidas δ. Los resultados resaltados en (*) refieren a que el modelo fue entrenado utilizando una matriz de attention WS con $r = 0{,}1$ y $p = 0{,}1$ en lugar del *full attention*. El resultado en (**) corresponde al método PowerEmbed(RW)-2 que usa la matriz Random Walk y 2 iteraciones, el cual fue obtenido de [27] ("-" indica que los resultados no están disponibles).

esta razón, se contrastaron los resultados obtenidos con los dos modelos de GLASE, pre-entrenado y e2e, con:

1. Los resultados de entrenar el clasificador considerando únicamente los vectores de atributos de los nodos (sin PEs).
2. Los resultados de utilizar los *embeddings* de ASE como PEs (ASE-PE), los cuales son obtenidos mediante la librería *Graspologic* basada en descomposición SVD.
3. Los resultados obtenidos por el método *PowerEmbed* extraídos de [27].

Para el entrenamiento *offline* de los *embeddings* GLASE-PE, se particionaron los grafos originales en subgrafos tomados aleatoriamente de los mismos. Se generaron 1000 muestras de subgrafos de cada conjunto de datos, considerando un 5 % del total de nodos. Se tomaron 800 subgrafos para entrenamiento y 200 para validación. A su vez estos subgrafos se utilizaron para estimar los signos de los valores propios dominantes y poder determinar las respectivas matrices $\mathbf{Q}$ en cada caso. A su vez se incorporaron escenarios de información faltante, borrando algunas aristas al azar en los conjuntos de entrenamiento. En las Tablas 7.5 y 7.6 se pueden observar los resultados obtenidos en cada caso. Los *embeddings* obtenidos con GLASE muestran una performance superior en ambos conjuntos de datos incluso usando mecanismos de *sparse attention*, con una mejora más significativa en los escenarios que presentan un mayor grado de información faltante.

7.2.2. Grafos Heterofilios

Para el caso de grafos heterofilios, se consideró un escenario similar al caso de los senadores descrito en la Sección 7.1.3 pero con datos reales. Se utilizó un conjunto de

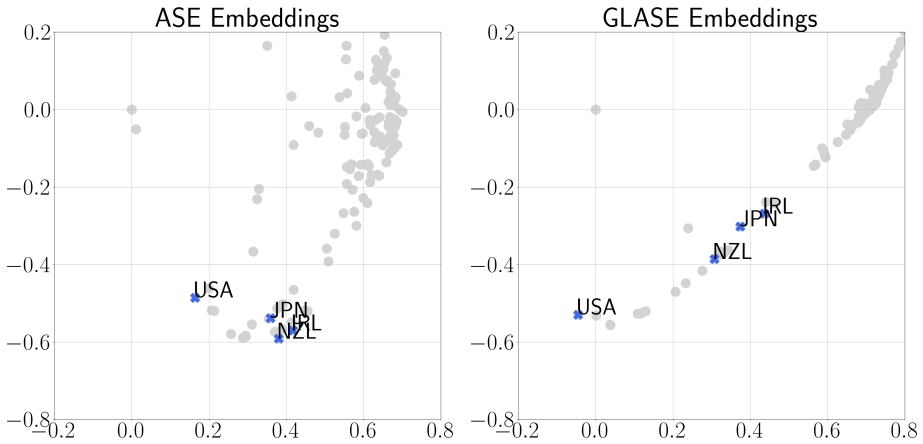


Figura 7.15: Datos de votación de la Asamblea General de la ONU para 1980. ASE *embeddings* obtenidos mediante la librería *Graspologic* (izquierda) y GLASE *embeddings* con matriz de máscara codificando votantes presentes y ausentes o abstenciones (derecha).

datos que contiene resultados de votaciones de países a propuestas presentadas en la Asamblea General de las Naciones Unidas (ONU) [53] durante el período de 1947 a 2018. Para cada votación y país, el conjunto de datos incluye si el país estuvo presente y, en caso afirmativo, el voto correspondiente (ya sea "Sí", "No" o "Abstención") para cada propuesta. Por su parte, las propuestas se encuentran etiquetadas con diferentes categorías:

- Propuestas relacionadas al conflicto de Palestina.
- Propuestas relacionadas a las armas y material nuclear.
- Propuestas relacionadas al control de armas y al desarmamiento.
- Propuestas relacionadas al Colonialismo.
- Propuestas relacionadas a los Derechos Humanos.
- Propuestas relacionadas al desarrollo económico.

Para cada año se construye un grafo bipartito compuesto de dos tipos de nodos, países y las propuestas, y donde la existencia de una arista entre estos está dada en el caso de un voto afirmativo por parte del país. Existen votaciones para los cuales no se conoce el resultado del voto debido a que el representante del país estuvo ausente durante la sesión. A su vez existen casos donde el representante directamente se abstuvo a votar. En lo que respecta a estos experimentos, se optó por modelar ambos casos como información faltante, etiquetando la arista correspondiente como desconocida y mapeando la misma en una matriz máscara binaria tal que $\mathbf{M}_{ij} = \mathbf{M}_{ij} = 0$.

La Figura 7.15 muestra los *embeddings* $d = 4$ de nodos para el año 1980. Se comparara los *embeddings* obtenidos utilizando *Graspologic* ASE, los cuales no pueden codificar datos desconocidos; con los obtenidos tras entrenar un modelo GLASE de 10 capas con pesos distribuidos, normalización y matriz de máscara binaria $\mathbf{M}$ que que codifica los votos desconocidos. En particular se destacan 4 países: EE.UU., Japón, Nueva Zelanda e Irlanda. Al examinar los *embeddings* generados por ASE, es posible notar una alineación entre los cuatro países, sugiriendo un grado alto de afinidad entre ellos. Sin embargo, si se observan los *embeddings* obtenidos con el modelo GLASE se identifica una divergencia clara entre EE.UU y los otros tres países. De hecho, el *embedding* correspondiente a Irlanda presenta una desviación de aproximadamente unos

45 grados con respecto al de EE.UU. El problema de ASE proviene de interpretar una ausencia o abstención igual a un voto negativo. En cambio GLASE permite codificar mejor el comportamiento de los votos desconocidos al utilizar la máscara binaria $\mathbf{M}$.

Por su parte, se propone el uso de GLASE como PE para atacar un problema de predicción de enlaces, cuyo objetivo es inferir cómo habrían resultado los votos desconocidos ya sea por abstinencia o ausencia. Se consideró usar tanto los *embeddings* GLASE pre-entrenados como el escenario *end-to-end* descrito en la Sección 6.3.2. Para ambos casos se utilizó un modelo GLASE de 10 capas con pesos distribuidos, normalización de nodo y tamaño de *embeddings* $d = 4$. Similar al experimento de clasificación de nodos realizado en los grafos homofilios, se pretende evaluar, por un lado, el beneficio de incorporar PE al modelo en comparación con un enfoque *naive* basado en VGAE y GCNs. Por otro lado, se busca comparar la calidad de los PE generados por GLASE en sí mismo, comparándolo con usar PE obtenidos con ASE.

Para evaluar la performance de los modelos propuestos, se crea un conjunto de prueba donde para ciertos países específicos se eligen aleatoriamente un cierto número de aristas equivalente a un 30 % del total y se las etiqueta como desconocidas. Dichas aristas son codificadas en la máscara $\mathbf{M}$ que usa GLASE. Los atributos de los nodos se definen como un vector de dimensión 12 que concatena atributos de países y propuestas, donde primeras 6 entradas corresponden a una codificación *one-hot encoding* del continente al cual pertenece un país y las siguientes 6 entradas corresponden a una codificación *one-hot encoding* de las categorías que pertenece la propuesta.

La Figura 7.16 resume los resultados obtenidos considerando votaciones de diferentes años. Los resultados obtenidos confirman que la incorporación de PE al modelo logra una mejora considerable en el accuracy de la predicción. A su vez, el uso de GLASE como PE logra resultados semejantes a ASE, incluso superándolo en algunos años. A su vez, es posible observar que el comportamiento de la GCN presenta variaciones significativas entre los años analizados. Se procede realizar un en el plano espectral con el fin de determinar los escenarios que favorecen y desfavorecen el funcionamiento de la GCN. Para esto, se calculan los vectores propios correspondientes a los cuatro valores propios dominantes de la matriz $\mathbf{A}$. Se observa que los vectores correspondientes a las dimensiones 1 y 2 solo permiten diferenciar entre tipo de nodo: países o propuestas pero no así entre grupos de países o tipos de propuestas con votaciones similares; mientras que los vectores 3 y 4 sí codifican la suficiente información para eventualmente poder realizar una discriminación coherente entre posibles subgrupos internos presentes en los dos tipos de nodos. A modo de ejemplo, en la Figura 7.17 se detallan los vectores propios 1 y 3 resultantes para el año 1980.

Para cada año, se calcula la GFT de dos señales anónimas: $\mathbf{x}_1$, que toma valores constantes en las posiciones correspondientes a los países y cero en el resto; y $\mathbf{x}_2$ que toma valores constantes en las posiciones correspondientes a las resoluciones y cero en el resto. En la Figura 7.18 se puede comprobar que tanto para países como para resoluciones, las GFT resultantes para los valores propios 3 y 4 tienden a cero. A su vez, se destaca la presencia una correlación fuerte entre el accuracy obtenido utilizando la GCN y los resultados la GFT de los países. Si se compara las primeras dos gráficas de la Figura 7.18 se puede observar claramente que en la medida que la GFT de los vectores 3 y 4 para la señal $\mathbf{x}_1$ se aleja de cero, el resultando obtenido con la GCN mejora. Al ser estos los vectores propios que permiten diferenciar posibles agrupaciones internas entre países, se concluye la ineficiencia de la GCN para realizar este tipo de clasificación.

Respecto a las diferentes categorías de las propuestas, se destaca que las etiquetas disponibles en el conjunto de datos original no son del todo informativas, puesto las mismas pueden englobar propuestas con contenidos opuestos sobre una misma temática. Este punto imposibilita realizar una segmentación efectiva de las mismas. Aún así,

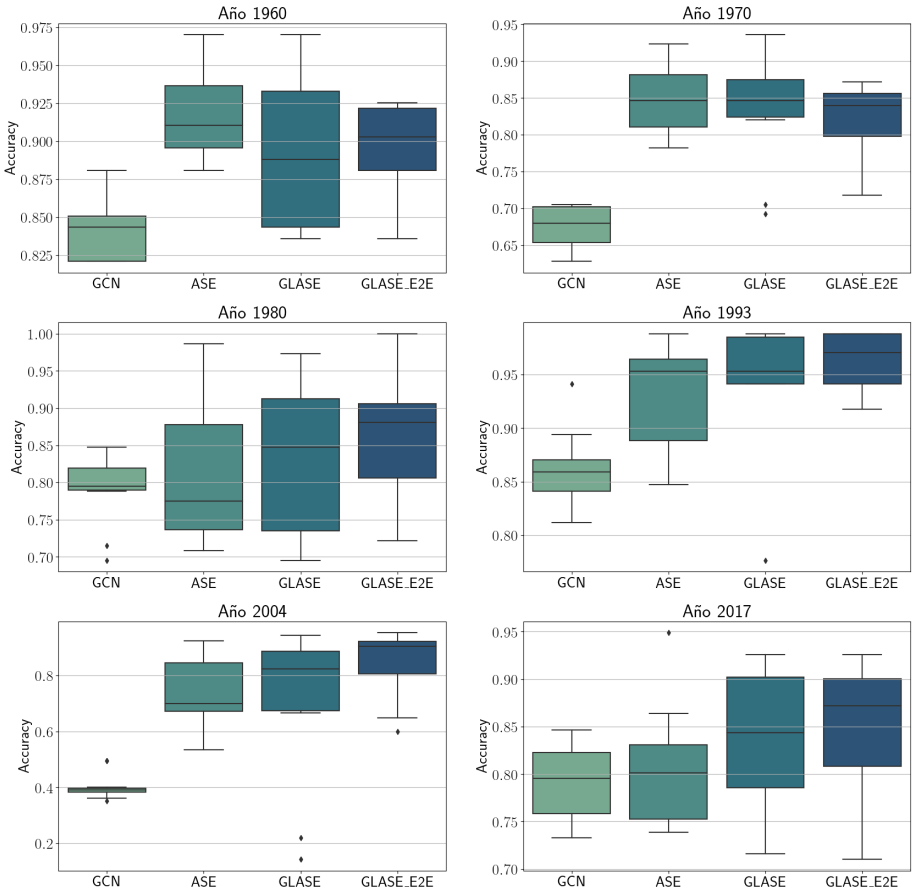


Figura 7.16: Resultados de accuracy para la predicción de aristas desconocidas utilizando *embeddings* de GLASE pre-entrando y e2e como PE, en comparación con ASE y con VGAE tradicional basado en GCN.

dicha información estaría contenida en los vectores propios 3 y 4, cuya GFT se anula ante el pasaje de una señal anónima $\mathbf{x}_2$, con lo cual la GCN fallaría en la clasificación. Como posible mejora, eventualmente se podría considerar realizar una mejor categorización de las propuestas empleando técnicas clustering para NLP, como ser mediante el uso de *BERT Topic* [21], los cuales permiten realizar una clasificación de tópicos considerando información semántica contenida en el texto de la propuesta en sí misma.

7.3. Resumen del capítulo

El capítulo se centró en validar experimentalmente la performance de los modelos LASE y GLASE mediante pruebas sobre conjuntos de datos sintéticos y reales que abordan diferentes escenarios de aplicación.

Para los datos sintéticos se utilizaron grafos generados por el modelo Stochastic Block Model (SBM) con diferentes configuraciones de comunidades y nodos. En particular se evaluó el desempeño de LASE en grafos de gran tamaño, para lo cual se realiza un entrenamiento utilizando muestras pequeñas tomadas aleatoriamente de los

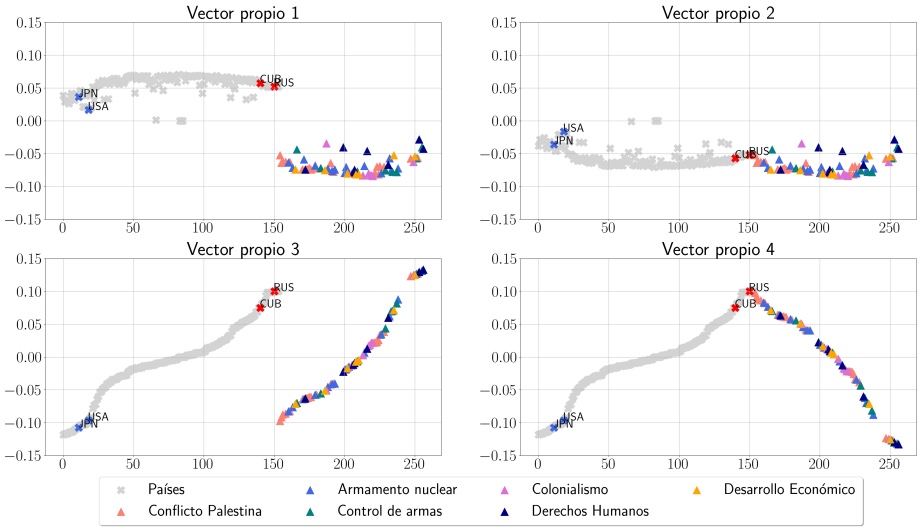


Figura 7.17: Comparación de los vectores propios dominantes asociados al grafo de ONU correspondiente al año 1980. La dimensión 1 permite diferenciar únicamente los diferentes tipos de nodos: países o propuestas. La dimensión 3 permiten diferenciar agrupaciones internas dentro de los países, i.e. países con comportamiento similar en lo que refiere a los votos emitidos.

mismos para luego inferir utilizando el grafo original. A su vez se estudió el comportamiento sobre grafos simétricos donde las GCN fallan en capturar propiedades globales del grafo debido a su enfoque en estructuras locales.

Por su parte, se aplicaron diversas configuraciones de *sparse attention* buscando reducir el costo computacional y los tiempos de inferencia. Las matrices de *attention* utilizadas fueron generadas mediante diferentes modelos como ser el Erdös–Rényi (ER), Watts-Strogatz (WS) y BigBird (BB) que permiten reducir la dependencia cuadrática de la arquitectura LASE original a una lineal.

Otra parte esencial de estos experimentos fue evaluar cómo los modelos LASE y GLASE manejan datos faltantes. Gracias a su diseño basado en el algoritmo de descenso por gradiente propuesto en *Efficient ASE*, estos modelos pueden incorporar una máscara binaria $\mathbf{M}$ en su arquitectura para adaptarse a la falta de información.

En cuanto a los datos reales se estudiaron dos tipos de grafos: homofilios y heterofilios, con datos de CORA, Amazon Photo y la Asamblea General de las Naciones Unidas (ONU). El objetivo de estos experimentos fue probar el uso de los *embeddings* GLASE como *Positional Encodings* (PE) utilizando las arquitecturas definidas en la Sección 6.3.

Las conclusiones derivadas de estos análisis serán sintetizadas en el siguiente capítulo.

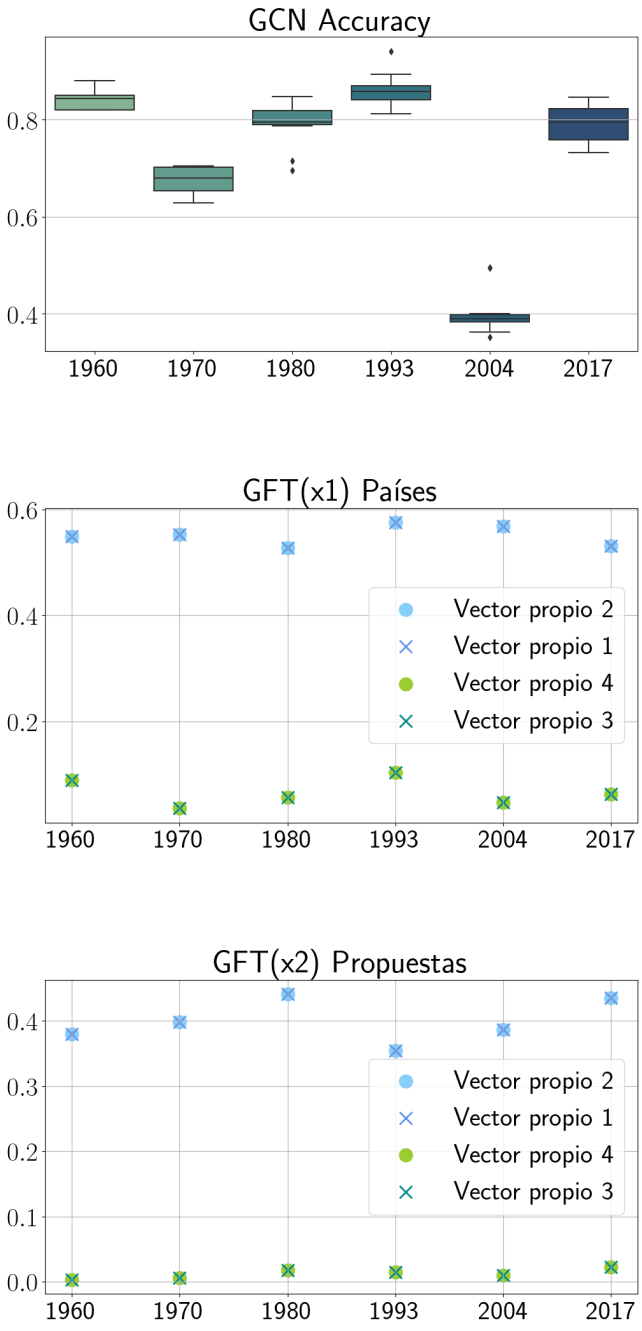


Figura 7.18: Comparación de la performance obtenida con GCN para cada año y los resultados obtenidos de las Graph Fourier Transform (GFT) en los cuatro vectores propios dominantes considerando una señal anónima $\mathbf{x}_1$ asociada únicamente a los países y una señal anónima $\mathbf{x}_2$ asociada únicamente a las propuestas. Se destaca la existencia de una correlación fuerte entre estas, en particular contrastando la primer y segunda figura.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Conclusiones y Trabajo Futuro

El presente trabajo abordó el problema de aprender *embeddings* espectrales similares a los ASE mediante el uso de GNNs. Con este fin, se propuso aplicar la técnica de *Algorithm Unrolling* al algoritmo de GD aplicado a la obtención de *embeddings* de grafos RDPG. Esto permitió modelar las iteraciones de este algoritmo de optimización como capas de una red neuronal basada en componentes clásicos de GNNs como ser las convoluciones y los mecanismos de *attention* en grafos. Los parámetros del algoritmo son aprendidos por la red mediante entrenamiento sobre muestras de grafos reales el cual se realiza offline.

La arquitectura resultante, la cual se denominó *Learned* ASE (LASE) admite además ciertas modificaciones que potencian su performance. De hecho, se demostró que la cantidad de capas requeridas por LASE es de hasta un orden de magnitud menor que la cantidad de iteraciones requeridas por el algoritmo GD convencional para alcanzar el mismo nivel de resultados. Esto hace que el proceso de inferencia, el cual es *online*, sea mucho más eficiente computacionalmente y requiera mucho menos tiempo. Dentro de los experimentos realizados, se demostró que LASE habilita extender la aplicación de GD a grafos de mucho mayor porte, que no serían viables de otra forma. De hecho, se comprueba que el modelo puede entrenarse en muestras pequeñas y generalizar a grafos más grandes con éxito. Los resultados obtenidos con LASE a su vez probaron ser competitivos con respecto a otros métodos tradicionales como ser el SVD.

A su vez, durante este trabajo se mostró como la arquitectura propuesta admite variaciones en el tipo de *attention* utilizado, permitiendo incorporar mecanismos de *sparse attention* basados en modelos generativos de grafos como ser el modelo Erdös Rényi, el modelo Watts Strogatz o la arquitectura BigBird (BB) usada para NLP. La incorporación de *sparse attention* permite reducir complejidad, sustituyendo la dependencia cuadrática del término de *attention* por una lineal, sin una pérdida significativa de calidad en los resultados obtenidos. Los tiempos de inferencia tras realizar este tipo de modificaciones posicionan a LASE en un orden de 8 veces más rápido que SVD tradicional y 14 veces más rápido que GD.

Por su parte, inspirada en *Efficient ASE* [17], la arquitectura propuesta para LASE permite la incorporación de una máscara binaria la cual puede ser utilizada para codificar información faltante. Se muestra que los *embeddings* obtenidos con LASE para este tipo de escenarios son más efectivos que aquellos obtenidos mediante la descomposición SVD, dado que se codifica mejor la información faltante. Esto es crucial en aplicaciones reales donde se tienen datos incompletos, ofreciendo una ventaja significativa sobre otros métodos tradicionales que requieren de información completa para

lograr modelar correctamente.

Otro punto de sumo interés estudiado a lo largo de este trabajo fue la extensión de LASE a escenarios de grafos heterofilios. Se introdujo una versión generalizada de LASE, denominada *Generalized* LASE (GLASE). La misma deriva de aplicar el mecanismo de *Algorithm Unrolling* al modelo *Generalized RDPG*. El resultado de esto es una arquitectura muy similar a la de LASE, la cual pasa a incorporar una matriz $\mathbf{Q}$ que define el producto interno indefinido $\mathbf{XQX}^T$ introducido en [47]. En particular, se demuestra que el modelo GLASE logra resultados igual de competitivos que LASE y que pueden entonces usarse indistintamente tanto en grafos homofilios como heterofilios. Los *embeddings* obtenidos tanto con LASE como GLASE mostraron captar eficazmente información estructural del grafo, la cual puede ser beneficiosa como forma de enriquecer los atributos de los nodos de un grafo dado. Por este motivo, se planteó a su vez, la incorporación de los mismos como *Positional Encodings* (PE) para alimentar por ejemplo otros modelos basado en *Graph Attention* o *Graph Transformer*.

La modularidad de la arquitectura, posibilita su fácil incorporación como un módulo adicional de otras arquitecturas más específicas. De esta forma, se logró tener un único modelo *end-to-end* para resolver tareas de aprendizaje puntuales como ser clasificación de nodos y predicción de enlaces. A diferencia de otros trabajos, los *embeddings* LASE son completamente aprendidos como parte de esta arquitectura global y no requieren de cálculos previos asociados a valores y vectores propios de la matriz de adyacencia ni del Laplaciano del grafo. A través de los experimentos realizados, se verificó cómo los *embeddings* GLASE utilizados como PE (GLASE-PE), mejoran la clasificación de nodos en grafos homofilios y facilitan la predicción de enlaces en grafos heterofilios. Los experimentos demuestran que GLASE supera o iguala el rendimiento de modelos tradicionales y otros métodos de *embeddings* en presencia de información faltante, destacando su utilidad en aplicaciones reales y complejas. A su vez, se comprueba que este método alcanza resultados comparables a los obtenidos con *PowerEmbed* para los conjuntos de grafos heterofílicos evaluados, e incluso los supera en el caso de CORA.

En resumen, la implementación de LASE y GLASE ha representado un avance significativo en el campo del aprendizaje automático sobre grafos, demostrando ser una arquitectura robusta capaz de capturar y aprovechar la estructura inherente de los grafos en diversas aplicaciones de aprendizaje. La solución propuesta logra generalizar y adaptarse a escenarios con información faltante y manejar tanto grafos homofílicos como heterofílicos. A su vez, mejora la eficiencia computacional del algoritmo GD original, habilitando su aplicación a grafos de gran porte.

8.1. Trabajo Futuro

En lo que respecta al trabajo futuro, se identifican cuatro posibles líneas que continúan y complementan la investigación realizada a lo largo de este trabajo. En primer lugar, se identifica mejorar la implementación del modelo GLASE de modo de independizarse del cálculo previo de la matriz $\mathbf{Q}$ asociada al producto interno *indefinido* del modelo *Generalized* RDPG. Hoy por hoy, el modelo propuesto requiere conocer de antemano los signos de los valores propios dominantes del grafo a utilizar lo cual lleva a tener que realizar la descomposición espectral del mismo de antemano agregando un costo computacional importante. Se plantea incorporar a la arquitectura un modelo que permita aprender estos signos como parte del proceso de aprendizaje de los *embeddings*.

Por otra parte, se plantea estudiar la extensión de LASE a grafos dirigidos, aplicando *Algorithm Unrolling* al modelo *Directed RDPG* propuesto en [36] para obtener una arquitectura basada en GNNs análoga.

A su vez, también resulta de interés extender esta metodología a métodos iterativos acelerados basados en gradiente, por ejemplo considerando la inversa de la matriz Hessiana (o una aproximación) como parte del cálculo [19]. De esta forma se podría plantear ampliar la aplicación de *Algorithm Unrolling* de modo de poder mapear la inversa de matriz Hessiana como otro componente basado en GNNs y así aprender la misma mediante entrenamiento.

Finalmente, se plantea estudiar la estabilidad de la arquitectura propuesta, en particular para la componente basada en *Graph Attention* desde la perspectiva de los *Edge-Variant Graph Filters* [13] [4]. De esta manera, posibilitar el uso del modelo GLASE en grafos dinámicos, entrenando sobre el grafo en un cierto tiempo $t = t_0$ y utilizando los *embeddings* resultantes para inicializar (*warm-start*) el entrenamiento para la obtención de embeddings del grafo en un tiempo posterior $t = t_0 + \delta$ con $\delta > 0$.

Esta página ha sido intencionalmente dejada en blanco.

Referencias

[1] Graspologic. `https://microsoft.github.io/graspologic/latest/`.

[2] Learned Adjacency Spectral Embeddings. `https://github.com/sofiperez91/LASE`.

[3] Pytorch Geometric. `https://pytorch-geometric.readthedocs.io/en/latest/`.

[4] Asynchronous Distributed Edge-Variant Graph Filters. In *2019 IEEE Data Science Workshop (DSW)*, pages 115–119, 2019.

[5] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J. Smola. Distributed large-scale natural graph factorization. *Proceedings of the 22nd International Conference on World Wide Web*, page 37–48, 2013.

[6] A. Athreya, D. E. Fishkind, M. Tang, C. E. Priebe, Y. Park, J. T. Vogelstein, K. Levin, V. Lyzinski, and Y. Qin. Statistical Inference on Random Dot Product Graphs: A Survey. *J. Mach. Learn. Res.*, 18(1):8393–8484, January 2017.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *Proc. Int. Conf. Learn. Representations*, 2016.

[8] Pierre Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. In *Proc. Int. Conf. Mach. Learn.*, volume 27, pages 37–49, 2012.

[9] S. Bhojanapalli, A. Kyrillidis, and S. Sanghavi. Dropping convexity for faster semi-definite optimization. In *Proc. Conf. Learn. Theory*, pages 530–582, 2016.

[10] Shaosheng Cao, Wei Lu, and Qiongkai Xu. GraRep: Learning Graph Representations with Global Structural Information. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, page 891–900, 2015.

[11] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to Optimize: A Primer and A Benchmark. *J. Mach. Learn. Res.*, 23(189):1–59, 2021.

[12] Y. Chi, Y. Lu, and Y. Chen. Nonconvex Optimization Meets Low-Rank Matrix Factorization: An Overview. *IEEE Trans. Signal Process.*, 67(20):5239–5269, 2019.

[13] Mario Coutino, Elvin Isufi, and Geert Leus. Advances in Distributed Graph Filtering. *IEEE Transactions on Signal Processing*, 2019.

[14] Vijay Prakash Dwivedi and Xavier Bresson. A Generalization of Transformer Networks to Graphs. *ArXiv*, abs/2012.09699, 2020.

[15] Alejandro Ribeiro Fernando Gama, Joan Bruna. Stability Properties of Graph Neural Networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020.

[16] Geert Leus Fernando Gama, Elvin Isufi and Alejandro Ribeiro. Graphs, Convolutions, and Neural Networks From Graph Filters to Graph Neural Networks. *IEEE Signal Processing Magazine*, 37(6):128–138, 2020.

[17] M. Fiori, B. Marenco, F. Larroca, P. Bermolen, and G. Mateos. Algorithmic Advances for the Adjacency Spectral Embedding. In *Proc. of European Signal Process. Conf.*, August 2022.

[18] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, June 2002.

[19] Dong Gong, Zhen Zhang, Qinfeng Shi, Anton van den Hengel, Chunhua Shen, and Yanning Zhang. Learning Deep Gradient Descent Optimization for Image Deconvolution. *IEEE Transactions on Neural Networks and Learning Systems*, 31:5468–5482, 2018.

[20] Karol Gregor and Yann LeCun. Learning Fast Approximations of Sparse Coding. *International Conference on International Conference on Machine Learning*, 2010.

[21] Maarten Grootendorst. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv*, 2022.

[22] Aditya Grover and Jure Leskovec. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864. ACM, 2016.

[23] William L. Hamilton. Graph Representation Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 14:1–159, 2020.

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2018.

[25] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 639–648. ACM, 2020.

[26] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *Proc. Adv. Neural. Inf. Process. Syst.*, 2021.

[27] Ningyuan Huang, Soledad Villar, Carey E. Priebe, Da Zheng, Chengyue Huang, Lin Yang, and Vladimir Braverman. From Local to Global: Spectral-Inspired Graph Neural Networks. *Proc. Adv. Neural. Inf. Process. Syst.*, 2022.

[28] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *Proc. Int. Conf. Learn. Representations*, 2015.

[29] Thomas N. Kipf and Max Welling. Variational Graph Auto-Encoders. *CoRR*, abs/1611.07308, 2016.

[30] Eric D. Kolaczyk. Statistical Analysis of Network Data. *Springer Series In Statistics*, 2009.

[31] Devin Kreuzer, Dominique Beaini, William L. Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking Graph Transformers with Spectral Attention. *Proc. Adv. Neural. Inf. Process. Syst.*, 2021.

[32] Natalie Glance Lada A. Adamic. The Political Blogosphere and the 2004 U.S. Election: Divided They Blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[33] Derek Lim, Joshua David Robinson, Lingxiao Zhao, Tess Smidt, Suvrit Sra, Haggai Maron, and Stefanie Jegelka. Sign and Basis Invariant Networks for Spectral Graph Representation Learning. In *The Eleventh International Conference on Learning Representations*, 2023.

[34] D. L. Sussman M. Tang and C. E. Priebe. Universally consistent vertex classification for latent position graphs. *The Annals of Statistics*, 41(3), 2013.

[35] Avinava Dubey et. al Manzil Zaheer, Guru Guruganesh. Big Bird: Transformers for Longer Sequences. *arXiv preprint arXiv:2007.14062v2*, 2021.

[36] Bernardo Marenco, Paola Bermolen, Marcelo Fiori, Federico Larroca, and Gonzalo Mateos. Online Change Point Detection for Weighted and Directed Random Dot Product Graphs. *IEEE Trans. Signal Inf. Process. Netw.*, 8:144–159, 2022.

[37] Andrew K. McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the Construction of Internet Portals with Machine Learning. *Inf. Retr.*, 3(2):127–163, 2000.

[38] Vishal Monga, Yuelong Li, and Yonina C. Eldar. Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing. *IEEE Signal Process. Mag.*, 38(2):18–44, 2021.

[39] Aleksandar Bojchevski Stephan Günnemann Oleksandr Shchur, Maximilian Mumme. Pitfalls of Graph Neural Network Evaluation. *ArXiv*, abs/1811.05868, 2019.

[40] A. E. Raftery P. D. Hoff and M. S. Handcock. Latent Space Approaches to Social Network Analysis. *Journal of the American Statistical Association*, 97(460):1090–1098, 2002.

[41] K. Laskey P. W. Holland and S. Leinhardt. Stochastic blockmodels: First steps. *Social Networks*, 5:109–137, 1983.

[42] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '14. ACM, August 2014.

[43] Carey E. Priebe, Cencheng Shen, Ningyuan Huang, and Tianyi Chen. A Simple Spectral Failure Mode for Graph Convolutional Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2021.

[44] Yujie Qian, Enrico Santus, Zhijing Jin, Jiang Guo, and Regina Barzilay. GraphIE: A Graph-Based Framework for Information Extraction. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 751–761, 2019.

[45] Ladislav Rampášek, Mikhail Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a General, Powerful, Scalable Graph Transformer. *Proc. Adv. Neural. Inf. Process. Syst.*, 2023.

[46] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. struc2vec: Learning Node Representations from Structural Identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17. ACM, August 2017.

[47] P. Rubin-Delanchy, J. Cape, M. Tang, and C. E. Priebe. A Statistical Interpretation of Spectral Embedding: The Generalised Random Dot Product Graph. *CoRR*, abs/1709.05506, 2017.

[48] Michael Sejr Schlichtkrull, Nicola De Cao, and Ivan Titov. Interpreting Graph Neural Networks for NLP with Differentiable Edge Masking. *Proc. Int. Conf. Learn. Representations*, 2021.

[49] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification. In *IJCAI*, pages 1548–1554. ijcai.org, 2021.

[50] Balasubramaniam Srinivasan and Bruno Ribeiro. On the Equivalence between Positional Node Embeddings and Structural Graph Representations. *Proc. Int. Conf. Learn. Representations*, 2020.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. *Proc. Adv. Neural. Inf. Process. Syst.*, 30, 2017.

[52] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *Proc. Int. Conf. Learn. Representations*, 2018.

[53] Erik Voeten, Anton Strezhnev, and Michael Bailey. United Nations General Assembly Voting Data. *Harvard Dataverse*, 2009.

[54] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural Graph Collaborative Filtering. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '19. ACM, July 2019.

[55] Boris Weisfeiler and AA Lehman. A Reduction of a Graph to a Canonical Form and an Algebra Arising during This Reduction. *Journal of Applied Mathematics and Physics*, 1968.

[56] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful are Graph Neural Networks? In *Proc. Int. Conf. Learn. Representations*, 2018.

[57] Ningyu Zhang, Shumin Deng, Zhanlin Sun, Guanying Wang, Xi Chen, Wei Zhang, and Huajun Chen. Long-tail Relation Extraction via Knowledge Graph Embeddings and Graph Convolution Networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019.

Índice de tablas

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

Esta es la última página.
Compilado el viernes 3 mayo, 2024.