TOWARDS EFFECTIVE GRAPH REPRESENTATIONS BY LEVERAGING GEOMETRIC CONCEPTS

LEE SEE HIAN

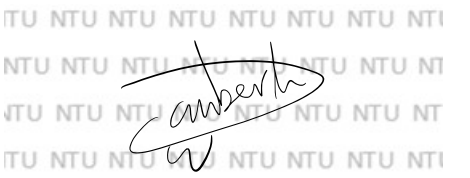
School of Electrical & Electronic Engineering

A thesis submitted to Nanyang Technological University in partial fulfillment of the requirement for the degree of Doctor of Philosophy

2024

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.



18 April 2024

. . . . . .

Date

. . . . . .

Lee See Hian

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

18 April 2024

.....

Date

.....

Tay Wee Peng

Authorship Attribution Statement

This thesis contains material from one paper published in the following peer-reviewed journal(s) / from papers accepted at conferences in which I am listed as an author.

To improve the readability, parts of this thesis have been grammatically revised using ChatGPT [1].

Parts of Chapter 2, Chapter 4 and Appendix A are published as S. H. Lee, F. Ji, and W. P. Tay, "SGAT: Simplicial Graph Attention Network" in Proc. International Joint Conference on Artificial Intelligence, Jul. 2022.

The same work is also accessible as "SGAT: Simplicial Graph Attention Network" on ArXiv, under the identifier abs/2207.11761, 2022.

The contributions of the co-authors are as follows:

- Prof. Tay Wee Peng provided the initial project direction for using simplicial complexes in graph neural networks.
- I prepared the manuscript drafts, and the manuscript was revised and edited by Dr. Ji Feng and Prof. Tay Wee Peng.
- I designed and discussed the methods with Dr. Ji Feng and Prof. Tay Wee Peng.
- I made essential modifications to existing code from open-source libraries and also wrote the code for the additional logic to implement our idea in the experiments.
- I analyzed the experimental results. Dr. Ji Feng and Prof. Tay Wee Peng assisted in interpreting the results.

Parts of Chapter 2 and Chapter 3 contains S. H. Lee, F. Ji, and W. P. Tay, "Node-Specific Space Selection via Localized Geometric Hyperbolicity in Graph Neural Networks". ArXiv, abs/2303.01724, 2023.

- Prof. Tay Wee Peng provided the initial direction of this work.
- I prepared the manuscript drafts, and the manuscript was revised and edited by Dr. Ji Feng and Prof. Tay Wee Peng.
- I designed and discussed the methods with Dr. Ji Feng and Prof. Tay Wee Peng.
- I made essential modifications to existing code from open-source libraries and also wrote the code for the additional logic to implement our idea in the experiments.

- I analyzed the experimental results. Dr. Ji Feng and Prof. Tay Wee Peng assisted in interpreting the results.

Parts of Chapter 2, Chapter 5 and Appendix B contains S. H. Lee, F. Ji, K. Xia, and W. P. Tay, "Graph Neural Networks with a Distribution of Parametrized Graphs". ArXiv, abs/2310.16401, 2023.

- I prepared the manuscript drafts, and the manuscript was revised and edited by Dr. Ji Feng, Prof. Tay Wee Peng and Prof. Xia Kelin.
- I made essential modifications to existing code from open-source libraries and also wrote the code for the additional logic to implement our idea in the experiments.
- I discussed the experimental setup and results with Dr. Ji Feng.
- Dr. Ji Feng conducted the necessary mathematical proof.

18 April 2024 ..... Date

..... Lee See Hian

Acknowledgements

I would like to express my utmost gratitude to the wonderful individuals who have shared this PhD journey with me and played pivotal roles towards the completion.

First and foremost, I am immensely grateful to my supervisor, Prof. Tay Wee Peng, who has consistently provided guidance to his students. I am truly amazed by his dedication to each of us and have greatly benefited from our fruitful discussions. These discussions were the major driving force for the continuous progress of my research. While he might be occasionally strict, his constructive criticisms are rooted in good intentions and have allowed me to learn from his thought process, fostering my growth in the research domain. Moreover, his generosity in sharing his expertise, insights and ideas has always awed and inspired me. Pursuing my PhD journey under his guidance has allowed me to hone crucial skills and has been one of the most rewarding decisions thus far.

I gratefully acknowledge Dr. Ji Feng whom I have had the privilege to work closely, throughout my entire PhD program on various research works. His ideas and suggestions have propelled my research in fascinating directions and also helped enrich my research proposals. Additionally, I am thankful for his patience in clarifying my doubts especially when my mathematical skills did not align with his expertise. His mentorship has been instrumental in transforming me from a student who was once intimidated by mathematics to someone who now feels considerably more at ease with it.

I would also like to thank my friends who similarly pursued graduate studies at various universities: Kenneth Ooi Wen Rui, Low Jian Liang, Xiao Fei and Benjamin Ho See Cheng. Your encouragement, diverse perspectives, discussions and shared experiences have been helpful in countless ways.

Special thanks to my mother, Lui Ming Chu and my boyfriend, Nasakol Pongkorpsakol. Both of you always provide much-needed reassurance and unwavering love whenever I have doubts. I also thank Nasakol for his role as my "rubber duck", patiently listening and accompanying me as I worked through problems and bugs.

This thesis would not have been possible without all of your support, intellectually and emotionally. Thank you from the bottom of my heart.

Contents

Summary

The field of graph representation learning (GRL) is dedicated to the task of encoding graph-structured data into low-dimensional vectors, often referred to as embeddings. Obtaining effective representations for various graph-related tasks, such as node classification and link prediction, hinges on effectively leveraging both the attributes and structural aspects of the graph data. A prominent approach for acquiring these graph representations involves the utilization of Graph Neural Networks (GNNs), a specialized class of neural networks designed for learning from graph data. Existing GNNs have limitations that can be mitigated by tailoring refinements based on the graph data's characteristics, enabling a more effective capture of graph intricacies. These improvements stand to benefit various applications, such as recommendation systems and social networks, as the graph structure often unveils valuable latent information.

This thesis investigates the application of geometric concepts in GNNs and proposes techniques inspired by these concepts to improve the embeddings learned. Firstly, we propose an approach that leverages multiple geometric spaces to embed nodes, guided by a hyperbolicity measure. This accounts for the diverse underlying geometry in different regions of a graph, minimizing distortion and yielding refined representations by selecting the more appropriate space. Secondly, we present a method that integrates geometric structures, such as triangles and tetrahedrons, into GNNs by incorporating the concept of simplicial complexes. This integration enriches the expressive power of GNNs, enabling them to capture complex interactions that extend beyond pairwise connections. Lastly, we present a method that leverages multiple graphs with different topologies and geometries during the learning process. These additional graphs are generated by introducing latent variables into the framework. Learning the distribution of these graphs reveals useful topologies and geometries, providing additional information for both training and inference.

Besides novel designs, we have conducted empirical assessments on graph-related tasks using benchmark datasets and compared our approaches to relevant baselines, confirming their effectiveness in improving graph representations. In summary, this thesis contributes to the progress in GRL by introducing three enhanced GNNs based on geometric concepts.

List of Figures

List of Tables

List of Abbreviations and Acronyms

ACM	Adaptive Channel Mixing
AUC	Area under curve
CGNN	Curvature Graph Neural Network
EM	Expectation-Maximization
EMGNN	Expectation-Maximization for Graph Neural Network
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GIL	Geometric Interaction Learning
GIN	Graph Isomorphism Network
GNN	Graph Neural Network
GSP	Graph Signal Processing
GTN	Graph Transform Network
GRL	Graph Representation Learning
HAN	Heterogeneous graph Attention Network
JSGNN	Joint Space Graph Neural Network
MCMC	Markov Chain Monte Carlo
MLE	Maximum Likelihood Estimation
MLP	Multilayer perceptron
MPSN	Message Passing Simplicial Network
NTU	Nanyang Technological University (Singapore)
PAC	Probably Approximately Correct
RMSE	Root mean squared error
ROC	Receiver Operating Characteristic
SGAT	Simplicial Graph Attention Network
SHAN	Simplicial Hyperbolic Attention Network
TDA	Topological Data Analysis
WL	Weisfeilar-Lehman

Chapter 1

Introduction

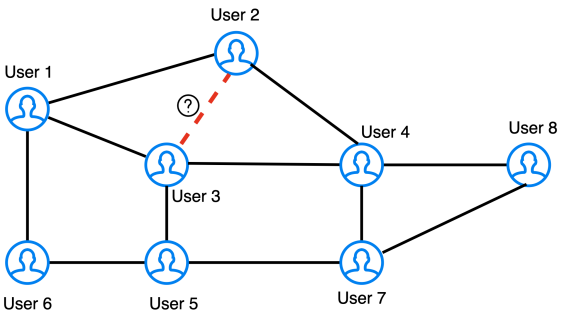
This thesis explores three approaches to enhance Graph Neural Networks (GNNs), a class of neural networks that operate on graph-structured data and learn representations, by leveraging geometric concepts such as hyperbolicity, non-Euclidean geometry, simplicial complex, and topology. Graphs, ubiquitous in real-world data, are data structures capturing relations between entities. Meanwhile, geometry, which concerns properties such as distances, directions, positions, shapes, and more, plays a crucial role in accurately capturing the relationships and properties within these graphs. Our approaches aim to improve learned embeddings for downstream machine-learning tasks, such as node classification and link prediction, by addressing the limitations of current GNNs when dealing with complex graph characteristics. Challenges include capturing high-order interactions involving multiple nodes, adapting to varying underlying geometries, and handling uncertain graphs. The GNN designs in this thesis draw inspiration from geometric concepts to explicitly or implicitly incorporate geometric information into the learning process to tackle these challenges.

In the upcoming sections, we formalize the concept of a graph and provide some background to representation learning for graphs. Following that, we outline our research aim and objectives, define the scope of our study, introduce the research questions we intend to address, and underscore the contributions we bring to the field. Subsequently, we describe the organization of the following chapters of the thesis.

1.1 Graph-structured Data

Complex relationships between entities are widespread across various applications such as social networks [2, 3], chemistry [4–6] and recommendation systems [7, 8]. In these applications, data is often represented as graphs. A graph is a data structure where nodes represent the entities (such as users, atoms or products), and edges represent the relationships or interdependencies between the entities (such as friendship, covalent bonds or clicks). The interactions among entities hold latent information that is not available when considering individual entities alone. Consequently, by extracting this latent information for downstream machine-learning tasks, we can improve the task performance. These downstream tasks include recommending friends or products (link prediction), predicting movie genres, or identifying fraudsters (node classification).

To further illustrate, consider a small subset of a social network depicted as a graph with eight users (refer to Fig. 1.1). The objective in this case is to predict whether a connection is likely between User 2 and User 3 (a link prediction task). Without using the graph information, we can utilize user attributes (if available), such as age, hobbies, and school affiliation, to assess whether recommending User 2 and User 3 to befriend each other is appropriate. However, if we stop here, we would not be using all the information we have, as examining the graph reveals that both users share two common friends - User 1 and User 4 - indicating they likely know each other or would have a good chance of getting along. This showcases how the graph structure holds valuable information that can potentially assist in tasks and further refine predictions.



$$\begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & ? & 1 & 0 & 0 & 0 & 0 \\ 1 & ? & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 1.1: An example social network graph and its symmetric adjacency matrix

To utilize the structural information and relationships for any task, the graph has to be represented in a numerical form. The most common way is to use an adjacency matrix $A \in \mathbb{R}^{N \times N}$, where N is the number of nodes. The presence of an edge between node i and j would result in a nonzero entry at A_{ij}, else $A_{ij} = 0$. An illustration of the adjacency matrix is provided in Fig. 1.1. Note that the graph depicted is undirected, resulting in a symmetric adjacency matrix.

However, adjacency matrices are often sparse and high-dimensional ($N \times N$) taking up considerable storage space. Additionally, graphs may have node or edge features. Hence, there is a need to encode the high-dimensional graph-structured data, along with its attributes, into *low-dimensional* vectors called embeddings. Specifically, for each node in the graph, a node embedding $h \in \mathbb{R}^d$ is learned, where d represents the embedding dimension, which is usually smaller than N [9]. After which, edge and graph embeddings are often obtained from these node embeddings. For instance, edge embeddings can be derived using binary operations like mean or concatenation, and graph embeddings can be acquired by pooling functions on all the nodes in the graph [10]. These embeddings should ideally preserve the topology and attribute information within the embedding space [9].

1.1.1 Types of Graphs

Graphs can be broadly classified into two types: homogeneous and heterogeneous. Homogeneous graphs are graphs of a single node type and a single edge type. A homogeneous graph is formally denoted as $G = (V, E)$, where V is the set of nodes, and E is the set of edges. In contrast, heterogeneous graphs are more intricate, involving multiple node types and edge types, making them semantically richer and more complex. Mathematically, a heterogeneous graph $G = (V, E)$ is further associated with a node-type mapping function $\phi : V \rightarrow \mathcal{S}$ and an edge-type mapping function $\varphi : E \rightarrow \mathcal{T}$ where $\mathcal{S}$ and $\mathcal{T}$ represents the sets of node and edge types, respectively. Moreover, the condition $| \mathcal{S} | + | \mathcal{T} | > 2$ holds. If $| \mathcal{S} | = 1$ and $| \mathcal{T} | = 1$, the graph reduces to a homogeneous graph.

1.2 Graph Representation Learning

The field of Graph Representation Learning (GRL) is dedicated to learning embedded representations of graphs. The representations are to appropriately preserve graph structural information and encapsulate graph properties. Ideally, they capture the most relevant information from graph data while reducing dimensionality [11]. Typically, "similar" nodes are positioned in close proximity in the embedding space while "dissimilar" nodes are placed far apart [12]. The notion of "similarity" depends on factors like the specific graph and chosen embedding method, which can be based on local graph structure, node characteristics, or a designated similarity measure.

These learned embeddings serve as valuable resources for downstream tasks. Various methods can be employed to acquire graph representations, typically falling into two categories: traditional graph embedding methods and GNN-based methods [10]. The former encompasses techniques such as matrix factorization [13, 14] as well as random walks approaches like DeepWalk [12] and node2vec [15] which typically takes the graph structure as input and ignores node attributes associated with each node [16]. This implies that the full information available is not being leveraged, leading to suboptimal representations [16]. In this thesis, GNN-based methods, a specialized type of neural network designed for graph data, will be of central importance. In Chapter 2, we will provide a comprehensive overview of GNNs.

Despite the excellent performance that GNNs have exhibited across a wide range of tasks and applications, they are not without their limitations. These limitations encompass but are not limited to, over-smoothing and limited expressive power (see Remark 2, and Section 2.3, respectively). It is also important to note that graphs exhibit diverse characteristics and complexities. Consequently, the inductive biases such as homophily bias and structural bias (refer to Section 2.4.5 and Section 2.4.8, respectively) embedded in various GNNs may not universally accommodate all graph types and characteristics.

As such, this thesis embarks on the exploration of leveraging geometric concepts to enhance GNNs, addressing specific limitations and empowering them to handle graph characteristics that were previously challenging to handle. Geometric concepts refer to a

wide array of ideas related to geometry and topology. Specifically, our research includes two explicit uses of geometry concepts: simplicial complexes and hyperbolic geometry, and a more nuanced idea of employing multiple graphs, which can be thought of as geometric objects. Each of these graphs can exhibit different topologies and geometrical properties. By selectively choosing the most relevant graphs and integrating them into a model, we infuse the learning process with a wealth of geometric information. This then enables us to achieve a more profound understanding of the underlying dynamics that a single graph may fail to capture.

1.3 Aim and Objectives

Our primary aim is to enhance GNNs, mitigating some of their limitations in capturing the structural characteristics inherent in graph data by using geometric concepts, leading to enhanced graph representations and, consequently, improved task performances.

The objectives of our research are as follows:

1. To capitalize on the strengths of various geometric spaces and embed nodes according to their local structure, to reduce embedding distortion, considering that graphs often exhibit varying geometries across different regions.

2. To capture high-order interactions involving multiple nodes by incorporating geometric shapes like triangles through simplicial complexes into the learning process.

3. To generate and utilize multiple graphs with varying topologies and geometric characteristics (such as edge weights) for learning, given that the provided graph is a single instance from an unknown distribution thus, insufficient to model the underlying dynamics.

1.4 Scope and Research Questions

This thesis introduces three GNNs designed for learning on *undirected, small to medium-sized graphs*. These GNNs employ various geometric ideas to include geometric information

for learning, to distinguish different neighborhoods within a graph and to differentiate various graphs. This capability proves especially valuable when dealing with graph neighborhoods or graphs that share similar topological characteristics. Briefly, we first present an approach tailored for homogeneous graphs and another for heterogeneous graphs within a non-probabilistic, deep-learning framework. Subsequently, we introduce a data-driven approach rooted in the probabilistic framework, suitable for learning on both heterogeneous and homogeneous graphs as well as molecular graphs (chemistry datasets). An overview of the research gap, geometric idea, and motivation for each of our works can be found in Table 1.1.

The contributions of this thesis are driven by the following research questions:

Research Question 1: *Can we enable GNNs to leverage multiple geometric spaces for learning, capitalizing on each space's unique strengths, given the diverse underlying geometries within graphs?*

We introduce the Joint Space Graph Neural Network (JSGNN) in Chapter 3 to address this question. JSGNN utilizes multiple geometric spaces for learning to avoid structural biases and distortions that can arise from relying solely on one space. It allocates nodes to suitable spaces using an attention mechanism with non-uniform constraints. Moreover, the attention scores are guided by a hyperbolicity measure that reflects the local geometry of each node. Our approach outperforms single-spaced GNNs and other mixed-space GNNs in tasks like node classification and link prediction. This work [17] is currently under review at the Pattern Recognition journal (Elsevier).

Research Question 2: *Can we develop a GNN for heterogeneous graphs that can capture high-order relations without relying on metapath or metagraph?*

Our main contribution in addressing this question is a novel GNN that we call Simplicial Graph Attention Network (SGAT). This method is introduced in Chapter 4. Unlike conventional approaches for handling heterogeneous graphs, which rely on metapaths or

metagraphs to encode multi-hop relations and capture semantics, SGAT employs simplicial complexes to capture high-order relations. This is because metagraphs, representing a generalization of metapaths, are predefined composite relations between pairs of nodes and have inherent limitations. Our approach surpasses previous methods in terms of node classification accuracy. Notably, our work has been published in the 31st International Joint Conference on Artificial Intelligence (IJCAI-ECAI), 2022 [18].

Research Question 3: *How can we generate and use multiple graphs with diverse topologies and geometric characteristics for learning, considering the potential inaccuracies in the provided graph, to include missing information and reduce noise?*

To address this question, we introduce an Expectation-Maximization (EM) framework for graph learning and a generic model, Expectation-Maximization for Graph Neural Network (EMGNN), in Chapter 5. In EMGNN, we introduce latent variables to generate multiple graph instances to learn from. The EM algorithm iteratively learns the distribution of these latent variables, which, in turn, defines the distribution of parameterized graphs and optimizes model parameters to maximize the joint likelihood of the observed data and latent variables. The task-specific loss is used to estimate the likelihood (following PAC-Bayesian principles), enabling the model to discern useful graph instances from noisy ones. EMGNN shows improved performance in tasks like node classification and graph regression. This work [19] is under review at the International Conference on Machine Learning (ICML).

1.5 Organisation of the Thesis

This thesis is structured as follows. In Chapter 2, we introduce related works and concepts that serve as the foundation for the material presented in the subsequent chapters. We then dedicate individual chapters for our three proposed methods. Within each of these method-focused chapters, we thoroughly describe the model, showcase the conducted experiments and their outcomes, and engage in in-depth discussions regarding our findings. In Chapter 6,

Table 1.1: Overview: Research gap, geometric idea and motivation of each work.

Model	**Research Gap**	**Geometric Idea**	**Motivation**
JSGNN	Many GNNs rely on a single geometric space, which may lead to structural bias and distortions in embeddings. Standard GNNs in Euclidean geometry struggle with embedding tree-like hierarchical structures.	Utilize hyperbolicity to evaluate the local structure of each node, determining whether to embed it in Euclidean or Hyperbolic space accordingly. Certain structures are better embedded in one space over another.	Utilizing a single space is suboptimal for graphs with varied geometries across regions. By leveraging the strengths of different spaces, the model can effectively capture underlying geometries and reduce distortions.
SGAT	Heterogeneous GNNs often rely on metapaths or metagraphs which have innate limitations, namely sensitivity to metapath choice, disregarding non-target nodes' features, and limited expressiveness in capturing complex relations.	Using simplicial complex to capture high-order interactions involving multiple nodes that cannot be reduced to pairwise interactions. It also enhances GNN model expressivity beyond Weisfeiler-Lehman test.	Metapaths or metagraphs are structures representing connections between two nodes. Heterogeneous graphs, which are semantically complex, would benefit from harnessing more intricate structures.
EMGNN	Many GNNs utilize solely the given observed graph for learning which may be inaccurate or insufficient to describe the underlying dynamics.	Using multiple graphs from a learned distribution for learning. Each graph may contain distinct measurements (such as edge weights) or topologies, providing valuable geometric information.	Each topology within a collection of graphs can offer unique insights and contribute to the problem more comprehensively.

we revisit the research questions (see Section 1.4), provide a more thorough response to these questions and propose potential avenues for future research and exploration.

Chapter 2

Background

In this chapter, we provide essential concepts needed to comprehend the subsequent chapters. We begin by examining alternative methodologies for handling graph data. Subsequently, we provide a review of GNN models designed for homogeneous and heterogeneous graphs, respectively. We then delve into the specific tasks and datasets that our work utilized.

2.1 Neural Networks on Relational Data

Our work focuses on GNNs, which are a specialized class of neural networks designed for graph data. The term "neural networks" generally refers to machines learning from data. Mathematically, it involves learning a hypothetical function to yield the desired output [20]. In general, the learning process involves a model adjusting its parameters to fit the data and the task at hand. Moreover, the primary intent is to train a model capable of making accurate predictions on the training samples and generalizing to new, unseen data [20]. Neural networks are commonly associated with deep learning, however, it is important to note that the latter entails architectures with many layers, whereas neural networks can vary in depth, ranging from shallow to deep architectures.

Deep Neural Networks (DNNs) have achieved remarkable success in domains like computer vision and natural language processing, exemplified by Convolutional Neural

Networks (CNNs) [21] and Bidirectional Encoder Representations from Transformers (BERT) [22]. However, their direct application to graph-structured data is limited. Graphs exhibit inherent complexity and irregularity, characterized by the varying number of edges and unordered nodes [23]. Often the nodes also have different numbers of neighbors. This contrasts with the regular grid-like structure of images and the sequential linearity found in text data. Essentially, images and text can be viewed as special cases of graphs, with images resembling grid-like structures and text exhibiting linear sequences. Nonetheless, just as models designed for text cannot seamlessly adapt to image data due to structural disparities, models optimized for grid-like or linear data struggle to accommodate the broader and more intricate structures inherent in graphs.

Workaround 1. One naive approach is to rely on carefully engineered graph-related features to handle relational data [24]. To mitigate the need for handcrafted features, studies [16, 24] have employed random walk-based networks to learn graph-related features. These features, containing some local structural information, are then passed into deep networks to perform the required task[1]. While these features derived from random-walk-based methods may capture certain structural aspects, they are usually not as task-specific or effective as those learned through GNNs. This is because they typically overlook the interaction between the given node attributes and graph structures [16]. In contrast, GNNs can learn structural information from the input graph(s) in an end-to-end manner, as well as task-specific structural information (since learning is guided by labels), without requiring extensive domain knowledge or manual effort.

Workaround 2. Another approach is to entirely disregard the structural information and solely utilize the given node features for learning. Nevertheless, this approach does not fully leverage the information present in graph data and can lead to suboptimal results. Previous studies have shown that methods that do not use the graph structure such as multilayer perceptron (MLP) and transductive support vector machine (TSVM) perform significantly worse than methods that are designed to handle graph data such as Graph

[1]Random-walk algorithms are typically unsupervised.

Convolutional Network (GCN) and Graph Attention Network (GAT) [25, 26]. This is not surprising, considering that the node features provided in benchmark datasets typically originate from a bag-of-words model, which describes the occurrence of words within a document. Consequently, these features lack richness in graph structural information. However, despite this deficiency, they may still hold valuable task-relevant data.

Remark 1. *There are also efforts directed towards making MLPs graph structure-aware. These approaches involve techniques to implicitly integrate graph information into the learning process, such as knowledge distillation [27] or incorporating the adjacency matrix into newly designed loss terms like Laplacian regularization [28, 29] and neighboring contrastive loss [30]. This enriches a model's capacity to utilize graph structures alongside node features.*

Large Language Models (LLMs). Recently, there also has been an increase in interest regarding the application of LLMs to relational data. Similar to the above-mentioned BERT, LLMs are primarily built on transformer architectures, which are inherently designed for natural language encoding. Consequently, inputting graph structures into these models is challenging [31]. To address this, various strategies have been explored to enable LLMs to effectively utilize relational data. These include converting graphs into sequences, accomplished through either rule-based methods or GNN-based methods [31]. The ultimate goal is to develop an LLM capable of processing both text and relational data simultaneously, and GNNs facilitate LLMs' handling of relational data. Interested readers can refer to [31] for more information.

2.2 Message Passing Graph Neural Networks

GNNs [32–36], which are neural networks designed to operate on graphs, typically employ the message-passing framework. Within this framework, the features of each node are integrated with those of its neighboring nodes to update its feature representation.

More specifically, suppose that we have a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. Moreover, each node $v \in V$ is associated with (initial)

node features represented by x_v^0. The node features can then be updated in the k-th layer as follows:

$$m_v^k = \text{AGGREGATE}^k(x_v^{k-1}, \{x_u^{k-1}, \forall\, u \in \mathcal{N}(v)\}) \tag{2.1}$$

$$x_v^k = \text{UPDATE}^k(x_v^{k-1}, m_v^k) \tag{2.2}$$

where $\mathcal{N}(v)$ is the set of neighbors of v in G and m_v^k is the "message" that is obtain from aggregating v's neighborhood at the k-th layer. $\text{AGGREGATE}^k(\cdot)$ represents a message aggregation function at the k-th layer, while $\text{UPDATE}^k(\cdot)$ signifies a node update function at the k-th layer. These arbitrary functions can contain learnable parameters and various logics. The choice of $\text{AGGREGATE}^k(\cdot)$ and $\text{UPDATE}^k(\cdot)$ defines various variants of GNNs [37, 38]. In a GNN with K layers, the last layer outputs node embeddings $\{x_1^K, \ldots, x_{|V|}^K\}$. These embeddings can be subsequently utilized for various tasks, including node classification, link prediction and graph regression as discussed in Section 2.6 as edge and graph embeddings are often derived from node embeddings [10].

Remark 2. *The receptive field of nodes in a GNN is influenced by the number of layers, where K layers correspond to considering K-hop neighbors. Typically, GNNs consist of two to three layers due to concerns related to oversmoothing and oversquashing issues. Consequently, adding more layers can lead to deteriorated results unless additional techniques are incorporated, such as skip connections [32], edge dropout [39], and graph rewiring [40]. Oversmoothing occurs when node features become more similar after being aggregated over multiple layers, causing important distinctions to be lost. Conversely, oversquashing refers to a situation where the dimensionality of node features is insufficient to capture information from an excessive number of neighbors [41].*

2.3 Expressivity of GNNs

Studies [38, 42] have theoretically shown that message-passing GNNs have, at most, the same expressive power as the Weisfeiler-Lehman (WL) graph isomorphism test [43]. The graph isomorphism problem involves determining whether two graphs are isomorphic

or topologically identical. Mathematically, this entails determining the existence of an edge-preserving bijection between them [44].

WL test. The algorithm is based on the idea of color refinement [43], it first assigns labels to the vertices. Subsequently, in each iteration, it (1) aggregates the labels from neighboring nodes for each node (2) hashes the aggregated labels to serve as new labels. The hash function has to be injective. This process continues until convergence or when the fixed number of iterations is reached. If, at any iterations, the node labelings between the two graphs differ, the graphs are deemed non-isomorphic.

Remark 3. *The WL test has limited capability in differentiating certain graph structures like triangles and cliques, preventing it from conclusively determining isomorphism [37]. As a result, when two graphs are non-isomorphic, the WL test may not always correctly identify them as such. Furthermore, the WL test, as well as GNNs, struggle to differentiate between topologically similar graphs that possess varying geometrical attributes such as edge lengths and angles [44]. This limitation is particularly evident in chemistry, where molecules can exhibit identical topology while having different conformations, such as cis-trans isomers.*

2.4 Review of GNN Models for Homogeneous Graphs

A multitude of GNNs have been introduced for homogeneous graphs. Here, we explore a selection of established baselines and models that are related to our research. For an overview of the plethora of GNN models, we recommend survey papers [23] and [45].

2.4.1 Classic Homogeneous Models

Classic models for homogeneous graphs include GCN [46], GAT [25], and GraphSage [47]. GCN is a spectral-based approach motivated by spectral graph convolution. In GCN, all neighboring nodes are considered equally important during aggregation. In contrast, GAT is a spatial-based approach with attention-based neighborhood aggregation while

GraphSage introduced node neighborhood sampling to enhance runtime efficiency. In GAT, the attention mechanism learns edge weights, assigning higher weights to important neighbors and lower weights to insignificant ones. This also serves as an attempt to capture implicit structural information to differentiate between neighbors, potentially reflecting some specific underlying geometric relationships among them [48].

Building on the success of GCN and GAT, extensions such as GCNII [32] and GATv2 [34] have been introduced. GCNII enhanced its layers by incorporating residual connections and identity mappings to address oversmoothing issues (refer to Remark 2). On the other hand, GATv2 improved the attention mechanism in GAT by reordering internal operations, effectively preventing the collapse of two linear layers.

2.4.2 Models Addressing the WL Test

As mentioned in Section 2.3, message-passing GNNs have, at most, the same expressive power as the WL test. In [38], GCN and GraphSage were shown to be less discriminative than the WL test. To achieve equivalent expressivity as the WL test, the Graph Isomorphism Network (GIN) [38] utilized injective functions to transform various node neighborhoods into distinct representations. Furthermore, [49] presented a model with similar expressive power as the k-WL test. The k-WL test is an extension of the WL algorithm (1-WL), where k-WL is strictly more powerful compared to $(k-1)$-WL for $k \geq 3$. On the other hand, in [42], the Simplicial Weisfeiler-Lehman (SWL) test was introduced and established to be strictly more powerful than the WL test. Subsequently, the authors introduced Message Passing Simplicial Networks (MPSNs), which utilized simplicial complexes and four different types of adjacencies in their message-passing framework, to elevate expressiveness beyond the WL test.

2.4.3 Models Altering Graph Structure

Approaches that modify the observed graph to improve results include DropEdge [39], where edges in the given graph were randomly dropped at a manually selected rate in each epoch. This process generated different copies of the original graph, diversifying

the input data used to train the model. Consequently, DropEdge was deemed capable of overcoming overfitting and oversmoothing. Meanwhile, for RSGNN [50], the model trained a link predictor to rewire the observed graph, eliminating or down-weighting noisy edges (connecting nodes with dissimilar features) and adding edges to densify the graphs. This was done to ensure the model is robust to noise and to address the issue of label sparsity by connecting more unlabeled nodes to labeled ones. Other than that, in [51], a two-stage scheme is proposed to enhance the performance of existing GNN models, where the second stage involves edge drop along estimated class boundaries.

2.4.4 Models Leveraging Additional Structural Information

Research has shown that GNNs have limited capabilities in determining the position of a given node relative to all other nodes within a graph [52]. Suppose that we temporarily disregard node features, if two nodes are topologically the same in their local neighborhood structure yet in actuality they are far apart in the graph, the GNN would embed them onto the same point in an embedding space [52, 53]. Therefore, there is a need to incorporate additional structural information, such as positional information, into GNNs to distinguish between topologically similar but distant nodes.

Various works [52–54] have proposed using anchor nodes as markers for determining the global positioning of nodes within the graph. An arbitrary positional function, like the shortest path, can then be used to evaluate a node's position relative to the anchor nodes. The positional information and node attributes are then combined through concatenation or product operations, leading to more powerful representations. Additionally, positional encoding be it absolute or relative, such as Laplacian eigenvectors, can be injected into the input layer of GNNs, similar to Transformers as seen in [55, 56].

A closely related idea that can achieve a similar effect of increased expressiveness is the notion of virtual nodes [57]. Unlike anchor nodes, which consist of nodes originally present in the graph, virtual nodes are artificial nodes that are added to the graph and connected to all other nodes. This approach serves the dual purpose of linking nodes to distant neighbors, which can be beneficial considering that GNNs are often shallow

and may struggle to effectively reach far-away nodes. This concept has been explored in various works, including [57–61].

2.4.5 Models for Heterophilic Graphs

Many GNNs designed for homogeneous graphs are based on the homophily assumption [62]. This assumption is grounded in the belief that nodes with similar attributes or structural characteristics tend to be connected. However, this assumption may not always hold, especially in the case of heterophilic graphs. Heterophilic graphs, which are a subtype of homogeneous graphs containing one node type and one edge type, exhibit the characteristic of being more likely to link nodes with dissimilar features and class labels [63]. Consequently, directly applying these GNNs to such graphs may lead to poor performance, to the extent that disregarding the graph structure might yield better results [62]. To tackle this challenge, it is essential to implement appropriate adaptations.

These adaptations involve techniques such as preventing the mixing of a node's feature with its aggregated neighborhood feature, typically achieved through operations like concatenation. Additionally, explicitly aggregating information from higher-order neighborhoods, which may contain more relevant neighbors, and combining intermediate representations have shown to be effective strategies. For instance, H2GCN [62] combined all of the abovementioned techniques to create a versatile GNN suitable for both homophilic and heterophilic homogeneous graphs. Likewise, UGCN [63] also considered higher-order neighborhoods by employing 1-hop, 2-hop, and κ-nearest neighbor (κNN) neighborhoods simultaneously and an attention mechanism to fuse the three specific embeddings.

Another approach involves utilizing multiple filter channels, such as low-pass, high-pass, and identity filters as exemplified by Adaptive Channel Mixing (ACM) [64]. This approach leverages high-frequency graph signals and captures relevant neighborhood information from different filter channels. This is deemed beneficial because high-pass filters focus on the differences between nodes, rather than their commonality as in low-pass filters [65].

2.4.6 Models using Advanced Topological Information

Some works employ advanced topological information, particularly curvature to enhance GNNs. Curvature is a fundamental concept in geometry. Two notable examples that utilize the notion of curvature are Curvature Graph Neural Network (CGNN) [66] and CurvGN [67], which use Ollivier's Ricci curvature to reweigh messages propagated between nodes, in Euclidean space. Edges with low curvature are assumed to be less crucial, as they often coincide with class boundaries where the flow of information should be limited. Additionally, there is Stochastic Discrete Ricci Flow (SDRF) [40] that makes use of curvature to rewire the graph. This approach is founded on the premise that edges with significant negative curvature indicate bottlenecks in the graph. Consequently, additional edges can be added at these bottlenecks to alleviate the bottleneck situation.

2.4.7 Models using Hyperbolic Geometry

Research has demonstrated that hyperbolic spaces, non-Euclidean spaces with constant negative curvature, are more effective for embedding tree-like hierarchical structures than Euclidean geometry, primarily attributed to their curvature and distinctive geometric properties [68, 69]. As a result, GNNs operating solely in the Euclidean space are constrained by its representation capabilities and may not achieve optimal performance when faced with data exhibiting non-Euclidean characteristics, such as scale-free, tree-like, or hierarchical structures [70]. Consequently, researchers have turned to hyperbolic geometry to generate more effective representations. In essence, the node feature aggregation step of GNN is extended into hyperbolic space. This led to the development of models like HGCN [71], HGNN [72], HGAT [73], and Lorentzian Graph Convolutional Network (LGCN) [74]. HGCN, HGNN and LGCN serve as hyperbolic counterparts to GCN, while HGAT represents the hyperbolic adaptation of GAT.

In HGCN and HGNN, basic operations are conducted in the tangent space, employing the exponential and logarithmic maps. Moreover, HGCN proposed to learn the curvature of the hyperbolic space. Meanwhile, LGCN formulates its neighborhood aggregation logic based on the centroid of Lorentzian distance, and it modified how the mappings to tangent

spaces are used to ensure that the basic operations strictly adhere to hyperbolic geometry. On the other hand, there is HyboNet [75], a fully hyperbolic framework based on the Lorentz model, that bypasses the use of the tangent space altogether as they perceive such an approach as limiting the model's modeling capacity.

2.4.8 Models using Multiple Geometric Spaces

Most GNNs typically embed all the nodes in a graph into a single geometric space. However, this approach can introduce undesired structural biases and distortions, potentially limiting the model's ability to accurately capture the underlying geometries within the graph data [70]. Prominent models that leverage multiple spaces for embeddings are GIL [76] and κ-GCN [68]. κ-GCN utilizes the (Cartesian) product space to model data in different spaces and employs the κ-stereographic model in each of the spaces. The Cartesian product enables a combinatorial construction of the mixed curvature space, thus the representations are first learned independently in the respective spaces and then concatenated [77].

On the other hand, GIL proposes a feature interaction scheme to leverage different spaces and a probability assembling module to combine the classification probabilities for obtaining the final prediction. The feature interaction scheme is where the node features in the respective two spaces are enhanced based on the distance similarity of the two sets of spatial embeddings. The larger the distance between the different spatial embeddings, the more significant the portion of features from the other space is summed to itself.

2.5 Review of GNN Models for Heterogeneous Graphs

Heterogeneous graphs are semantically richer and more complex than homogeneous graphs given the presence of different node types and different edge types. Consequently, techniques designed for homogeneous graphs cannot be directly applied to heterogeneous graphs. One common adaptation involves the introduction of feature-specific transformation matrices, as each node type may have different feature dimensions. These learnable matrices enable the projection of node features into a common space, facilitating comparisons and analysis

across different node types. Models designed for heterogeneous graphs can be broadly classified into two categories: metapath-based models and metapath-free models.

2.5.1 Metapath-based models

Learning on heterogeneous graphs mainly utilizes predefined metapaths or automatically learning the optimal metapaths during the end-to-end training process [78]. A metapath is a predefined sequence of node types describing a multi-hop relationship between nodes [79]. For instance, in a citation network with author, paper and venue node types, the metapath Author-Paper-Author (APA) represents the co-author relationship. Moreover, Author-Paper-Venue-Paper-Author (APVPA) connects two authors who published papers at the same venue. This implies that metapath can represent high-level semantic relations and thus, contribute both semantic and structural information to the learning process [80]. Works along this line include Heterogeneous graph Attention Network (HAN) [81], Metapath Aggregated Graph Neural Network (MAGNN) [82], SeHGNN [80], Graph Transformer Network (GTN) [83] and REGATHER [35].

HAN employed predefined metapaths to transform heterogeneous graphs into metapath-based homogeneous graphs, selectively retaining features of target nodes while discarding intermediate nodes. Dual-level attention was then applied to the resulting metapath-based homogeneous graphs. In constrast, MAGNN introduced intra-metapath aggregation to integrate intermediate node features rather than discarding them. Meanwhile, SeHGNN proposed to precompute neighbor aggregation and use a single-layer structure with long metapaths to expand the receptive field. Additionally, a transformer-based module is then utilized to fuse metapath-specific features. These approaches relied on predefined metapaths, and suboptimal path selection led to suboptimal results [78].

Implicitly learning meta-path-like structures. GTN proposed to learn metapath-based graph structures by softly selecting candidate adjacencies and multiplying them together. The resulting weighted adjacency matrices represent metapath-based graph structures. Meanwhile, REGATHER generated a set of multi-hop relation-type subgraphs,

which indirectly include metapath-based graph structures by multiplying adjacency matrices of first-order relation types during data preprocessing. The attention mechanism was then utilized to assign different weights to the subgraphs.

To generalize further, the concept of a metagraph was introduced [84]. A metagraph is a nonlinear combination of metapaths or more specifically, a subgraph of node types defined on the network schema, describing a more elaborated pairwise relation between two nodes through auxiliary nodes. We can therefore view metapaths and metagraphs as multi-hop relations between two nodes. An example of an approach that utilized metagraphs is Meta-GNN [85]. Meta-GNN introduced a metagraph convolution layer, employing metagraphs to define the receptive field of nodes. The attention mechanism was then employed to combine node features from different metagraphs.

2.5.2 Metapath-free models

HetGNN [86], Simple-HGN [87], HetSANN [88], Simplicial Neural Network (SNN) [89] and SHAN [90] are examples of approaches that do not make use of metapaths or metagraphs. HetGNN employed a random walk with restart (RWR) strategy [91] to create neighbors for nodes, implicitly enabling the acquisition of higher-order relations. Subsequently, it utilized Bi-LSTM for node feature aggregation. In contrast, Simple-HGN [87] enhanced GAT for heterogeneous graphs by introducing three additional techniques: learnable type-specific embedding, residual connection, and L_2 normalization. Meanwhile, HetSANN proposed a multi-layer GAT architecture incorporating type-specific scoring functions to achieve type-aware attention for varying edge types.

Simplicial Complex-related Models. On the other hand, SNN employed simplicial complexes on heterogeneous *bipartite* graphs, transforming such graphs into a homogeneous complex with co-chains' data. Co-chain features were then enhanced using *spectral* simplicial convolution with Hodge Laplacians [92] of the simplicial complex. SNN was specifically designed for bipartite graphs and primarily served the purpose of data imputation. On a different note, SHAN leverages the original graph structure along with extracted

simplicial complexes and hyperbolic graph attention to learn high-order relations. The simplicial complexes were generated based on the notion of neighbor overlap where if a set of p target nodes has a neighbor overlap greater than a predefined threshold, a p-order relation or simplex is formed.

2.6 Graph-based Tasks

Graph embeddings learned can be used for graph-based tasks that vary across different graph levels. In our research, we explore node classification at the node-level, link prediction at the edge-level, and graph regression at the graph-level, encompassing the entire graph.

Node classification involves predicting the class of individual nodes in the test set, using ground truth labels from the training set for learning. For example, in the Citeseer dataset, where nodes represent scientific publications, the goal is to predict the category of each publication. Typically, this is achieved in an end-to-end manner by applying a softmax operation to the learned embeddings to calculate class probabilities for each node and using cross-entropy loss. The predicted class is the one with the highest probability.

Link prediction is the task of predicting the existence of an edge between two nodes. It is typically formulated as a binary classification problem, where an edge label 0 represents the absence of an edge, and a label 1 indicates the presence of an edge. To elaborate, the edges existing in the given graph are considered positive samples, while non-existent edges are randomly chosen as negative samples. These positive and negative samples are then divided into training, validation, and test sets, each containing a mix of positive and negative samples. The model is trained using the binary cross-entropy loss.

Graph regression predicts floating-point numbers representing properties of the entire graph, such as estimating the hydration-free energy of molecules in datasets like FreeSolv. To perform graph regression, the learned node embeddings are first aggregated using a readout function, such as mean or sum pooling, to create a graph-level representation.

The representation is then used as input to a MLP for the final prediction, and the model is optimized using the smooth L1 loss[2].

2.7 Graph Datasets

Here we describe the datasets we utilized for our experiments.

2.7.1 Datasets for Homogeneous Node Classification

Ten benchmark datasets are employed for this task. Specifically, three citation datasets: Cora, Citeseer, Pubmed; a flight network: Airport; a disease propagation tree: Disease; an Amazon co-purchase graph dataset: Photo; and a coauthor dataset: CS. Additionally, three heterophilic datasets are incorporated: Texas, Wisconsin, and Cornell, which feature connections between dissimilar nodes. The statistics of the datasets are as shown in Table 2.1.

Table 2.1: Statistics of Homogeneous Graph Datasets.

Dataset	Nodes	Edges	Classes	Features
Cora	2708	5429	7	1433
Citeseer	3327	4732	6	3703
Pubmed	19717	44338	3	500
Aiport	3188	18631	4	4
Disease	1044	1043	2	1000
Photo	7650	119081	8	745
CS	18333	81894	15	6805
Texas	183	295	5	1703
Cornell	183	280	5	1703
Wisconsin	251	466	5	1703

2.7.2 Datasets for Link Prediction

The same datasets as discussed in Section 2.7.1 are used for the link prediction task.

[2]https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html

2.7.3 Datasets for Heterogeneous Node Classification

The heterogeneous graph datasets used are publicly available and are the same as those employed in [83]. These datasets include two citation networks named DBLP[3] and ACM[4], as well as a movie dataset named IMDB[5]. Within the DBLP dataset, there are three distinct node types (Paper (P), Author (A), Conference (C)) and four edge types (PA, AP, PC, CP). The ACM dataset also comprises three node types (Paper (P), Author (A), and Subject (S)) and four edge types (PA, AP, PS, SP). Similarly, the IMDB dataset has three node types (Movie (M), Actor (A), Director (D)) along with four edge types (MD, DM, MA, AM). The characteristics of the three heterogeneous benchmark datasets are as summarized in Table 2.2. In the DBLP dataset, the research areas of the authors are to be predicted. Meanwhile, in the ACM and IMDB datasets, the categories of papers and genres of movies are to be determined, respectively.

Table 2.2: Heterogeneous datasets. The number of A-B edges is equal to the number of B-A edges thus omitted.

	Edge types (A-B)	# of A-B	# Edges	# Edge types	# Features
DBLP	Paper - Author Paper - Conference	19645 14328	67946	4	334
ACM	Paper - Author Paper - Subject	9936 3025	25922	4	1902
IMDB	Movie - Director Movie - Actor	4661 13983	37288	4	1256

2.7.4 Datasets for Graph Regression

The datasets, FreeSolv, ESOL, and Lipophilicity, are part of MoleculeNet[6], a popular benchmark for molecular machine learning. In FreeSolv, the task involves estimating solvation energy. In ESOL, the objective is to estimate solubility. Lastly, Lipophilicity is for the estimation of lipophilicity. These are important molecular properties in the realm of physical chemistry and provide insights into how molecules interact with solvents.

[3]https://dblp.uni-trier.de/
[4]http://dl.acm.org/
[5]https://www.imdb.com/interfaces/
[6]https://moleculenet.org/datasets-1

- FreeSolv. The dataset contains experimental and calculated hydration-free energy of 642 small neural molecules in water.

- ESOL (for *E*stimated *SOL*ubility). ESOL consists of water solubility data for 1128 compounds.

- Lipophilicity. This dataset contains experimental results of the octanol/water distribution coefficient of 4200 compounds, namely logP at pH 7.4, where P refers to the partition coefficient. Lipophilicity refers to the ability of a compound to dissolve in fats, oils and lipids.

Chapter 3

Node-Specific Space Selection via Localized Geometric Hyperbolicity

In this chapter, we introduce an approach for improving representations learned by harnessing multiple geometric spaces - hyperbolic and Euclidean - instead of relying on a single space. Most works explore learning graph representations in a single space. Employing just one geometric space might lead to suboptimal embeddings as graphs are often complex, containing a mixture of underlying geometries. As such, learning graph representations in either Euclidean or hyperbolic space, with all nodes' representations embedded in a single space, is not ideal as it inevitably leads to undesired structural inductive biases and distortions [70]. This concept of leveraging multiple spaces was initially introduced by a prior method named GIL [76], establishing it as one of the pioneering works in this area. Subsequently, another approach κ-GCN [68] also explores leveraging multiple spaces by employing the (Cartesian) product space. However, empirical results indicate that κ-GCN and GIL do not consistently outperform their single-space components, highlighting the need for better integration of multiple spaces during learning.

Our approach distinguishes itself from prior research by learning non-uniform selection weights guided by the notion of "local geometric hyperbolicity", to determine the most suitable embedding space for each node. The concept of "local geometric hyperbolicity" assesses the tree-like nature of a node's neighborhood, and the selection weights reflect

model-based hyperbolicity. The two hyperbolicities' distributions are aligned using the Wasserstein metric such that the calculated geometric hyperbolicity guides the choice of the learned model hyperbolicity. This approach has produced promising results in node classification and link prediction tasks, surpassing baseline models.

3.1 Introduction

Hyperbolic spaces have gained traction in research as they have been proven to better embed tree-like, hierarchical structures compared to the Euclidean geometry [68, 69]. Intuitively, encoding non-Euclidean structures such as trees in the Euclidean space would result in more considerable distortion since the number of nodes in a tree increases exponentially with the depth of the tree while the Euclidean space only grows polynomially [76]. In such cases, the hyperbolic geometry serves as an alternative to learning those structures with comparably smaller distortion as the hyperbolic space has the exponential growth property [70]. As such, hyperbolic versions of GNNs such as HGCN [71], HGNN [72], HGAT [73] and LGCN [74] have been proposed.

Nevertheless, real-world graphs are often complex. They are neither solely made up of Euclidean nor non-Euclidean structures alone but a mixture of geometrical structures. Our proposed method in this chapter considers a localized version of geometric hyperbolicity, a concept from geometry group theory measuring how tree-like the underlying space is for each node in the graph (refer to Section 3.4.1 for more details). We observe a mixture of local geometric hyperbolicity values in most of the benchmark datasets we employ for our experiments as seen in Fig. 3.2. This implies that the graphs contain a mixture of geometries and thus, it is not ideal to embed the graphs into a single geometry space, regardless of Euclidean or hyperbolic [70].

Taking a graph containing both lattice-like and tree-like structures as an example, Fig. 3.1c and Fig. 3.1f shows that 15 of the blue-colored nodes in the tree structure are calculated to have 2-hop local geometric hyperbolicity value of zero, while 12 of the purple nodes have a value of one and the other 3 purple nodes (at the center of the lattice) have

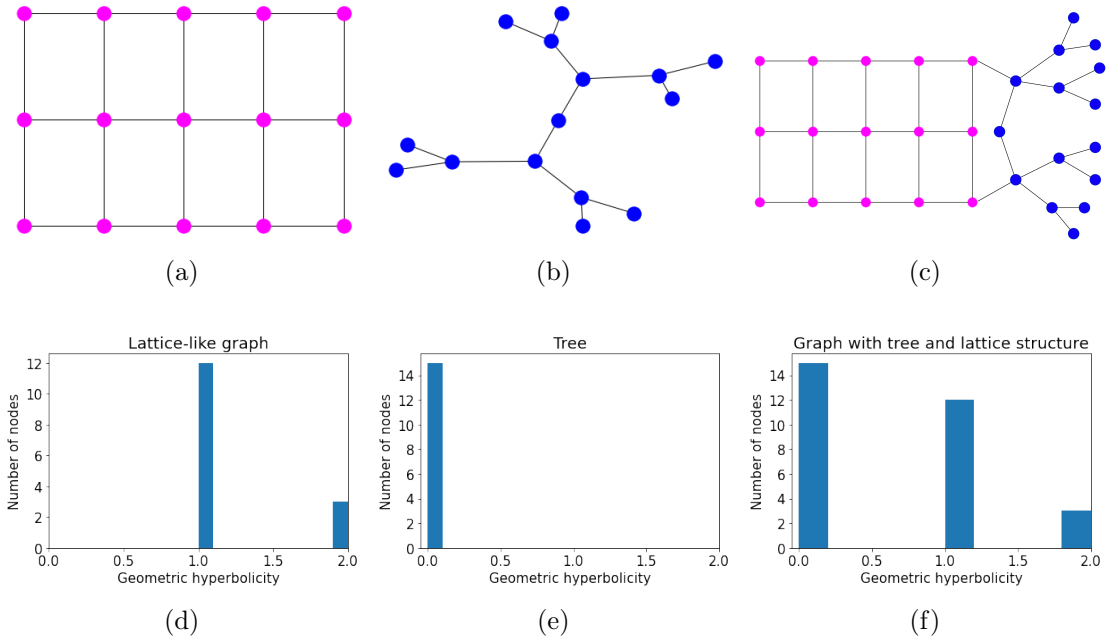


Figure 3.1: Example graphs. (a) Lattice-like graph. (b) A tree. (c) A combined graph containing both lattice and tree structure. (d-f) The histograms reflect the geometric hyperbolicity in the respective graphs.

a value of two (the smaller the hyperbolicity value, the more hyperbolic). This localized metric can therefore serve as an indication during learning on which of the two spaces is more suitable to embed the respective nodes.

Here we address this mixture of geometry in a graph and propose JSGNN that performs learning on a joint space consisting of both Euclidean and hyperbolic geometries. To achieve this, we first update all the node features in both Euclidean and hyperbolic spaces independently, giving rise to two sets of updated node features. Then, we employ exponential and logarithmic maps to bridge the two spaces and an attention mechanism is used as a form of model hyperbolicity, taking into account the underlying structure around each node and the corresponding node features. The learned model hyperbolicity is guided by geometric hyperbolicity and is used to "softly decide" the most suitable embedding space for each node and to reduce the two sets of updated features into only one set. Ideally, a node should be either hyperbolic or Euclidean and not both simultaneously, thus, we also introduce an additional loss term to achieve this non-uniform characteristic.

The subsequent sections of this chapter are structured as follows: We elaborate on how the related works, which serve as baselines, differ from our approach in Section 3.2.

The necessary preliminaries to comprehend the proposed method are then presented in Section 3.3. Following that, we delve into the model's specifics. Experimental setups, the results and discussions are shown in Section 3.5. Finally, we conclude this chapter in Section 3.6.

3.2 Related works

To the best of our knowledge, the closest works to our research are GIL [76] and κ-GCN [68]. We have discussed these works in detail in Section 2.4.8. In this section, we highlight the distinctions between our work and these existing models.

In comparison to κ-GCN, our framework shares similarities in its implementation. However, instead of concatenating the representations, we introduce a space selection mechanism guided by hyperbolicity to fuse the representations. This approach sets our work apart from κ-GCN.

On the other hand, our approach differs from GIL [76] in some key aspects. Firstly, we leverage the distribution of geometric hyperbolicity to guide our model to learn to decide if each node is better embedded in an Euclidean or hyperbolic space instead of performing feature interaction learning. This is done by aligning the distribution of the learned model hyperbolicity and geometric hyperbolicity using the Wasserstein distance. Our motivation is that if a node can be best embedded in one of the two spaces, encoding it in another space other than the optimal one would result in comparably larger distortions. Minimal information would be present in the sub-optimal space to help "enhance" the representation in the better space. Hence, promoting feature interaction could introduce more noise to the corresponding spaces. The ideal situation is then to learn normalized selection weights that are non-uniform for each node so that we select for each node a single, comparably better space's output embedding. To achieve this, we introduce an additional loss term that promotes non-uniformity. Lastly, we do not require probability assembling since we only have one set of output features at the end of the selection process.

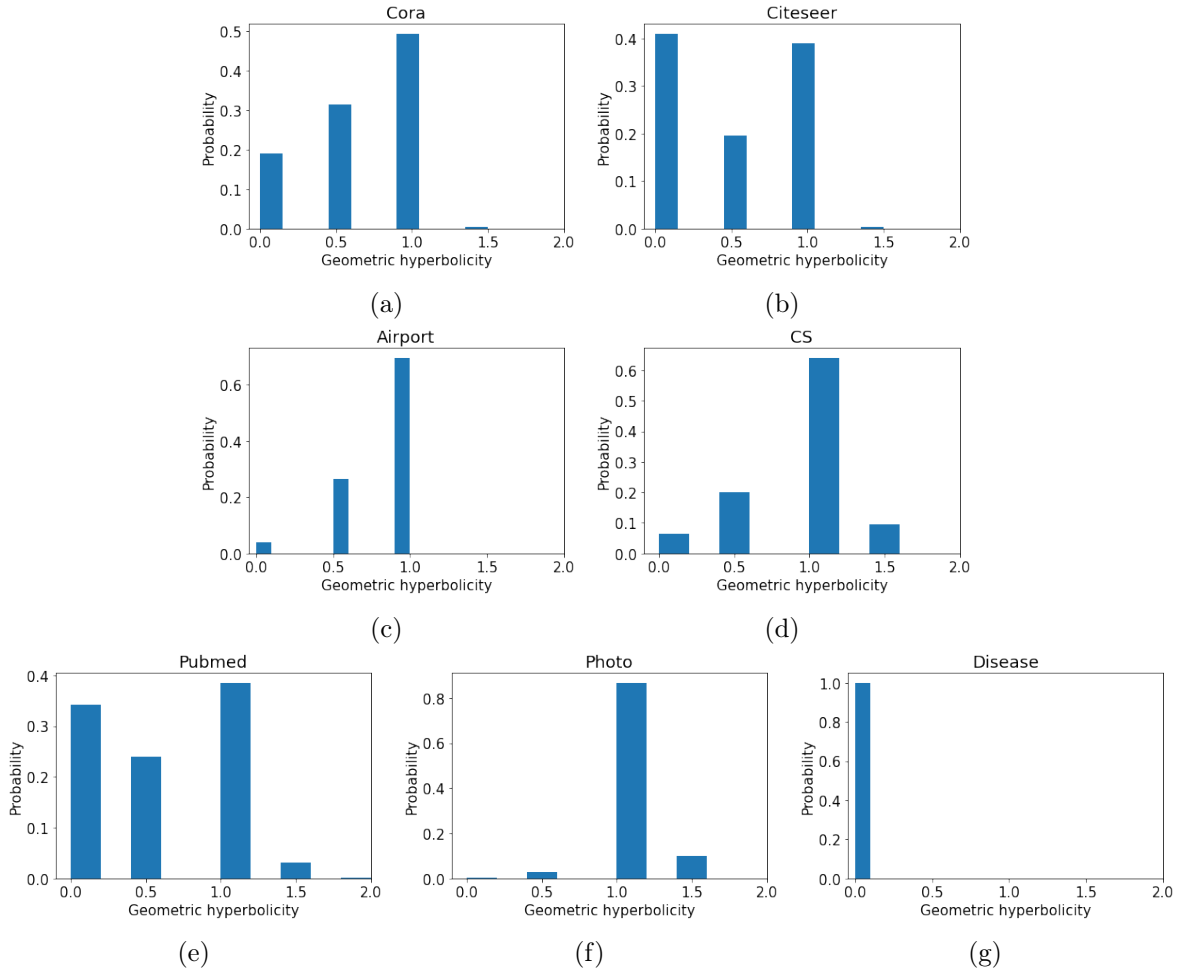


Figure 3.2: Distributions of geometric hyperbolicity for all datasets, obtained by computing $\delta_{G_v,\infty}$ on each of the nodes' 2-hop subgraph.

3.3 Preliminaries

In this section, we give a brief overview of hyperbolic geometry that will be used in this chapter. Readers are referred to [93] for further details. Moreover, we review GAT and its hyperbolic version.

3.3.1 Hyperbolic geometry

A hyperbolic space is a non-Euclidean space with constant negative curvature. There are different but equivalent models to describe the same hyperbolic geometry. In this work, we work with the Poincaré ball model, in which all points are inside a ball. The hyperbolic space with constant negative curvature c is denoted by $(\mathbb{D}_c^n, g_{\mathbf{x}}^c)$. It consists of the n-dimensional hyperbolic manifold $\mathbb{D}_c^n = \{\mathbf{x} \in \mathbb{R}^n : c\|\mathbf{x}\| < 1\}$ with the Riemannian

metric $g^c_{\mathbf{x}} = (\lambda^c_{\mathbf{x}})^2 g^E$, where $\lambda^c_{\mathbf{x}} = 2/(1 - c\|\mathbf{x}\|^2)$ and $g^E = \mathbf{I}_n$ is the Euclidean metric.

At each $\mathbf{x} \in \mathbb{D}^n_c$, there is a tangent space $\mathcal{T}_{\mathbf{x}}\mathbb{D}^n_c$, which can be viewed as the first-order approximation of the hyperbolic manifold at $\mathbf{x}$ [68]. The tangent space is then useful to perform Euclidean operations that we are familiar with but are undefined in hyperbolic spaces. A hyperbolic space and the tangent space at a point are connected through the exponential map $\exp^c_{\mathbf{x}} : \mathcal{T}_{\mathbf{x}}\mathbb{D}^n_c \to \mathbb{D}^n_c$ and logarithmic map $\log^c_{\mathbf{x}} : \mathbb{D}^n_c \to \mathcal{T}_{\mathbf{x}}\mathbb{D}^n_c$, specifically defined as follows:

$$\exp^c_{\mathbf{x}}(\mathbf{v}) = \mathbf{x} \oplus_c \Big(\tanh\Big(\sqrt{c} \frac{\lambda^c_{\mathbf{x}} \|\mathbf{v}\|}{2} \Big) \frac{\mathbf{v}}{\sqrt{c}\|\mathbf{v}\|} \Big), \tag{3.1}$$

$$\log^c_{\mathbf{x}}(\mathbf{y}) = \frac{2}{\sqrt{c}\lambda^c_{\mathbf{x}}} \tanh^{-1}(\sqrt{c}\| - \mathbf{x} \oplus_c \mathbf{y}\|) \frac{-\mathbf{x} \oplus_c \mathbf{y}}{\| - \mathbf{x} \oplus_c \mathbf{y}\|}, \tag{3.2}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{D}^n_c$, $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{D}^n_c$ and $\oplus_c$ is the Möbius addition. For convenience, we write $\mathbb{D}$ for $\mathbb{D}^n_c$ if no confusion arises.

A salient feature of hyperbolic geometry is that it is "thinner" than Euclidean geometry. Visually, more points can be squeezed in a hyperbolic subspace having the same shape as its Euclidean counterpart, due to the different metrics in the two spaces. We discuss the graph version in Section 3.4.1 below.

3.3.2 Graph attention and message passing

Consider a graph $G = (V, E)$, where V is the set of vertices, E is the set of edges, and each node in V is associated with a node feature h_v. Recall that GAT is a GNN that updates node representations using message passing by updating edge weights concurrently. Specifically, for one layer of GAT [25], the node features are updated as follows:

$$h'_v = \sigma\Big(\sum_{j \in N(v)} \alpha_{vj} \mathbf{W} h_j \Big), \tag{3.3}$$

$$\alpha_{vj} = \frac{\exp(e_{vj})}{\sum_{k \in N(v)} \exp(e_{vk})}, \tag{3.4}$$

$$e_{vj} = \text{LeakyReLU}(\mathbf{a}^\intercal [\mathbf{W} h_v \parallel \mathbf{W} h_j]), \tag{3.5}$$

where $\|$ denotes the concatenation operation, σ denotes an activation function, $\mathbf{a}$ represents the learnable attention vector, $\mathbf{W}$ is the weight matrix for a linear transformation and α denotes the normalized attention scores.

3.3.3 Hyperbolic attention model

To derive a hyperbolic version of GAT, we adopt the following strategy. We perform feature aggregation in the tangent spaces of points in the hyperbolic space. Features are mapped between hyperbolic space and tangent spaces using the pair of exponential and logarithmic functions: $\exp_{\mathbf{x}}^{c}$ and $\log_{\mathbf{x}}^{c}$.

With this, we denote Euclidean features as $h_{\mathbb{R}}$ and hyperbolic features as $h_{\mathbb{D}}$. Then one layer of message propagation in the hyperbolic GAT is as follows [76]:

$$m_v = \sigma\Big(\sum_{j \in \{v\} \cup N(v)} \alpha_{vj} \log_{\mathbf{o}}^{c}(\mathbf{W} \otimes_c h_{j,\mathbb{D}} \oplus_c \mathbf{b}) \Big), \tag{3.6}$$

$$h_{v,\mathbb{D}}^{'} = \exp_{\mathbf{o}}^{c}(\sigma(m_v)), \tag{3.7}$$

where $d_{\mathbb{D}}$ is the normalized hyperbolic distance, while $\otimes_c$ and $\oplus_c$ represent the Möbius matrix multiplication and addition, respectively. Additionally, α_{vj} is obtained as follows:

$$e_{vj} = \text{LeakyReLU}\Big(\mathbf{a}^{\intercal} \Big[\hat{h}_v \parallel \hat{h}_j \Big] \times d_{\mathbb{D}}(h_{v,\mathbb{D}}, h_{j,\mathbb{D}}) \Big), \tag{3.8}$$

$$d_{\mathbb{D}}(h_{v,\mathbb{D}}, h_{j,\mathbb{D}}) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c} \| -h_{v,\mathbb{D}} \oplus_c h_{j,\mathbb{D}} \|), \tag{3.9}$$

$$\alpha_{vj} = \text{softmax}_j(e_{vj}), \tag{3.10}$$

where $\hat{h}_j = \log_{\mathbf{o}}^{c}(\mathbf{W} \otimes_c h_{j,\mathbb{D}})$.

3.4 Proposed Method

In this section, we propose our joint space learning model. The model relies on comparing two different notions of hyperbolicity: geometric hyperbolicity and model hyperbolicity. We start by introducing the former, which also serves as the motivation for the design of

our GNN model.

3.4.1 Local geometry and geometric hyperbolicity

Gromov's δ-hyperbolicity is a mathematical notion from geometry group theory to measure how tree-like a metric space is in terms of metric or distance structure [71, 94]. The precise definition is given as follows.

Definition 1 (Gromov 4-point δ-hyperbolicity [95] p.410)**.** *For a metric space X with metric $d(\cdot,\cdot)$, it is δ-hyperbolic, where $\delta \geq 0$ if the four-point condition holds:*

$$\begin{aligned} &d(x,y) + d(z,t) \leq \\ &\quad \max\{d(x,z) + d(y,t), d(z,y) + d(x,t)\} + 2\delta, \end{aligned} \tag{3.11}$$

for any $x, y, z, t \in X$. X is hyperbolic if it is δ-hyperbolic for some $\delta \geq 0$.

This condition of δ-hyperbolicity is equivalent to the Gromov thin triangle condition. For example, any tree is (0-)hyperbolic, and $\mathbb{R}^n$, where $n \geq 2$ is not hyperbolic. However, if X is a compact metric space, then X is always δ-hyperbolic for some δ large enough such as $\delta = \text{diameter}(X)$. Therefore, it is insufficient to just label X as hyperbolic or not. We want to quantify hyperbolicity such that a space with smaller hyperbolicity resembles more of a tree.

Inspired by the four-point condition, we define the ∞-version and the 1-version of hyperbolicity as follows.

Definition 2. *For a compact metric space X and elements $x, y, z, t \in X$, denote* $\inf_{\delta \geq 0}\{(3.11)$ *holds for $x, y, z, t\}$ by $\tau_X(x, y, z, t)$. Define*

$$\begin{aligned} \delta_{X,\infty} &= \sup_{x,y,z,t \in X} \tau_X(x,y,z,t), \\ \delta_{X,1} &= \mathbb{E}_{x,y,z,t \sim \text{Unif}(X^4)}[\tau_X(x,y,z,t)], \end{aligned}$$

where Unif *represents the uniform distribution.*

In order for these invariants to be useful for graphs, we require them to be almost identical for graphs with similar structures. We shall see that this is indeed the case. Before stating the result, we need a few more concepts.

Let $\mathcal{G}$ be the space of weighted, undirected simple graphs. Though for most experiments, the given graphs are unweighted. However, aggregation mechanisms such as attention essentially generate weights for the edges. Therefore, for both theoretical and practical reasons, it makes sense to expand the graph domain to include weighted graphs.

For each $G = (V, E) \in \mathcal{G}$, it has a canonical path metric d_G, and d_G makes G into a metric space including non-vertex points on the edges. For $\epsilon > 0$, there is the subspace $\mathcal{G}_\epsilon$ of $\mathcal{G}$ consisting of graphs whose edge weights are greater than ϵ.

On the other hand, there is a metric on the space $\mathcal{G}$ and $\mathcal{G}_\epsilon$, called the Gromov-Hausdorff metric ([95] p.72). To define it, we first introduce the Hausdorff distance. Let X and Y be two subsets of a metric space (M, d). Then the Hausdorff distance $d_H(X, Y)$ between X and Y is

$$d_H(X, Y) = \max\{\sup_{x \in X} d(x, Y), \sup_{y \in Y} d(X, y)\},$$

where $d(x, Y) = \inf_{y \in Y} d(x, y)$, $d(X, y) = \inf_{x \in X} d(x, y)$. The Hausdorff distance measures in the worst case, how far away a point in X is away from Y and vice versa.

In general, we want to also compare spaces that do not a priori belong to a common ambient space. For this, if X, Y are two compact metric spaces, then their Gromov-Hausdorff distance $d_{GH}(X, Y)$ is defined as the infimum of all numbers $d_H(f(X), g(Y))$ for all metric spaces M and all isometric embeddings $f : X \to M, g : Y \to M$. Intuitively, the Gromov-Hausdorff distance measures how far X and Y are from being isometric. The following is proved in the Appendix.

Proposition 1. *Suppose $\mathcal{G}$ and its subspaces have the Gromov-Hausdorff metric. Then $\delta_{G,\infty}$ is Lipschitz continuous w.r.t. $G \in \mathcal{G}$ and $\delta_{G,1}$ is continuous w.r.t. $G \in \mathcal{G}_\epsilon$ for any $\epsilon > 0$.*

Consider a graph G. We fix either $\delta_{G,\infty}$ or $\delta_{G,1}$ as a measure of hyperbolicity, and

apply to each local neighborhood of G. To be more precise, it is studied [32, 39] that many popular GNN models have a shallow structure. It is customary to have a 2-layer network possibly due to oversmoothing [96–98] and oversquashing [40] phenomena. In such models, each node only aggregates information in a small neighborhood.

Therefore, if we fix a small k and let G_v be the subgraph of the k-hop neighborhood of $v \in V$, then it is more appropriate to study the hyperbolicity δ_v, either $\delta_{G_v,\infty}$ or $\delta_{G_v,1}$, of G_v. For our experiments, the former is utilized. We call δ_v the *geometric hyperbolicity* at node v. The collection $\Delta_V = \{\delta_v : v \in V\}$ allows us to obtain an empirical distribution μ_G of geometric hyperbolicity on the sample space $\mathbb{R}_{\geq 0}$.

For instance, we can build histograms to acquire the distributions as observed in Fig. 3.2. We see, for example, for Cora, a substantial number of nodes have small (local) hyperbolicity, in contrast with many works that claim Cora to be relatively Euclidean due to its high global hyperbolicity value [71, 99]. On the other hand, Airport is argued to be globally hyperbolic, but a large proportion of nodes has large local hyperbolicity. However, this is not a contradiction as we are considering the local structures of the graph. We call μ_G the *distribution of geometric hyperbolicity*. It depends only on G and k.

3.4.2 Space selection and model hyperbolicity

In this section, we describe the backbone of our model and introduce the notion of model hyperbolicity. Our model consists of two branches, one using Euclidean geometry and the other using hyperbolic geometry. We primarily use GAT (cf. Section 3.3.2) for the Euclidean model and HGAT (cf. Section 3.3.3) for the hyperbolic model. Other pairs of Euclidean or hyperbolic models (e.g., GCN and HGCN) can also be applied to the corresponding branches. In Section 3.5.2, we show the experimental results on two variants of JSGNN.

After the respective message propagation, we would have two sets of updated node embeddings, the Euclidean embedding $Z_{\mathbb{R}}$ and the hyperbolic embedding $Z_{\mathbb{D}}$. The two sets of embeddings are combined into a single embedding $Z = \{z_v, v \in V\}$ through an attention mechanism that serves as a space selection procedure. The attention mechanism

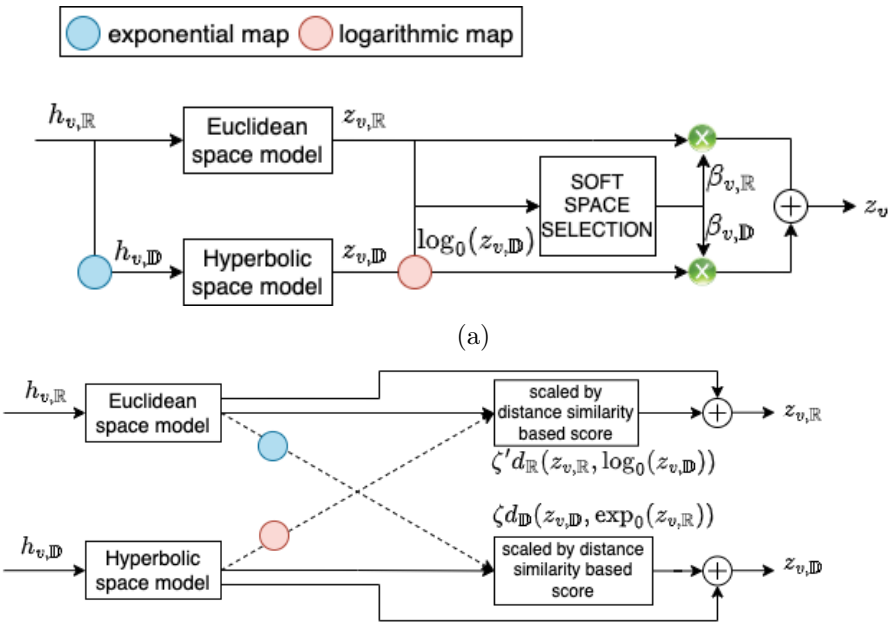


Figure 3.3: Comparison between JSGNN and GIL [76] in leveraging Euclidean and hyperbolic spaces. Both models utilizes GAT in Section 3.3.2 and HGAT in Section 3.3.3 for message passing in Euclidean and hyperbolic space respectively. (a) Soft space selection mechanism of JSGNN where trainable selection weights $\beta_{v,\mathbb{R}}, \beta_{v,\mathbb{D}}$ are non-uniform, effectively selecting the better of the two spaces considered. (b) Feature interaction mechanism of GIL where $\zeta, \zeta' \in \mathbb{R}$ are trainable weights and $d_{\mathbb{D}}, d_{\mathbb{R}}$ are the hyperbolic distance (cf. (3.9)) and Euclidean distance respectively. The node embeddings of both spaces in GIL are adjusted based on distance, potentially introducing more noise to the branches as there is minimal information in the sub-optimal space to "enhance" the representation in the better space.

is performed in a Euclidean space. Thus, the hyperbolic embeddings are first mapped into the tangent space using the logarithmic map. Mathematically, the normalized attention score indicating whether a node should be embedded in the hyperbolic space $\beta_{v,\mathbb{D}}$ or Euclidean space $\beta_{v,\mathbb{R}}$ is as follows:

$$w_{v,\mathbb{R}} = \mathbf{q}^\intercal \tanh(\mathbf{M} z_{v,\mathbb{R}} + \mathbf{b}), \tag{3.12}$$

$$w_{v,\mathbb{D}} = \mathbf{q}^\intercal \tanh(\mathbf{M} \log_{\mathbf{o}}^c(z_{v,\mathbb{D}}) + \mathbf{b}), \tag{3.13}$$

$$\beta_{v,\mathbb{R}} = \frac{\exp(w_{v,\mathbb{R}})}{\exp(w_{v,\mathbb{R}}) + \exp(w_{v,\mathbb{D}})}, \tag{3.14}$$

$$\beta_{v,\mathbb{D}} = \frac{\exp(w_{v,\mathbb{D}})}{\exp(w_{v,\mathbb{R}}) + \exp(w_{v,\mathbb{D}})}, \tag{3.15}$$

where $\mathbf{q}$ refers to the learnable space selection attention vector, $\mathbf{M}$ is a learnable weight

matrix, $\mathbf{b}$ denotes a learnable bias and $\beta_{v,\mathbb{D}} + \beta_{v,\mathbb{R}} = 1$, for all $v \in V$. The weights $\beta_{v,\mathbb{D}}$ and $\beta_{v,\mathbb{R}}$ are conditioned to be non-uniform as illustrated in Section 3.4.4. The two sets of space-specific node embeddings can then be combined via a convex combination using the learned weights as follows:

$$z_v = \beta_{v,\mathbb{R}} z_{v,\mathbb{R}} + \beta_{v,\mathbb{D}} \log_{\mathbf{o}}^{c}(z_{v,\mathbb{D}}), \forall\, v \in V. \tag{3.16}$$

This gives one layer of the model architecture of JSGNN, as illustrated in Fig. 3.3.

The parameter $\beta_{v,\mathbb{R}}, v \in V$ controls whether the combined output, consisting of both hyperbolic and Euclidean components, should rely more on the hyperbolic components or not. We call $\beta_{v,\mathbb{R}}$ the *model hyperbolicity* at the node v. The notion of model hyperbolicity depends on node features as well as the explicit GNN model. Similar to geometric hyperbolicity, the collection $\Gamma_G = \{\beta_{v,\mathbb{R}} : v \in V\}$ gives rise to an empirical distribution ν_G on $[0, 1]$. We call ν_G the *distribution of model hyperbolicity*.

To motivate the next subsection, from (3.16), we notice that the output depends smoothly on $\beta_{v,\mathbb{R}}$. If we wish to have a similar output for nodes with similar neighborhood structures and features, we want their selection weights to have similar values. On the other hand, we have seen (cf. Proposition 1) that geometric hyperbolicities, which can be computed given G, are similar for nodes with similar neighborhoods. It suggests that we may use geometric hyperbolicities to "guide" the choice of model hyperbolicities.

3.4.3 Model hyperbolicity *vs.* geometric hyperbolicity

We have introduced geometric and model hyperbolicities in the previous subsections. In this subsection, we explore the interconnections between these two notions.

Let Θ be the parameters of a proposed GNN model. We assume that the model has the pipeline shown in Fig. 3.4. Given node features $\{h_v, v \in V\}$ and model parameters Θ, the model generates (embedding) features $\{z_v, v \in V\}$ and selection weights or model hyperbolicity $\{\beta_{v,\mathbb{R}}, v \in V\}$ in the intermediate stage. For each $v \in V$, there is a combination function ϕ_v such that the final output $\{\hat{y}_v, v \in V\}$ satisfies $\hat{y}_v = \phi_v(z_v, \beta_v)$.

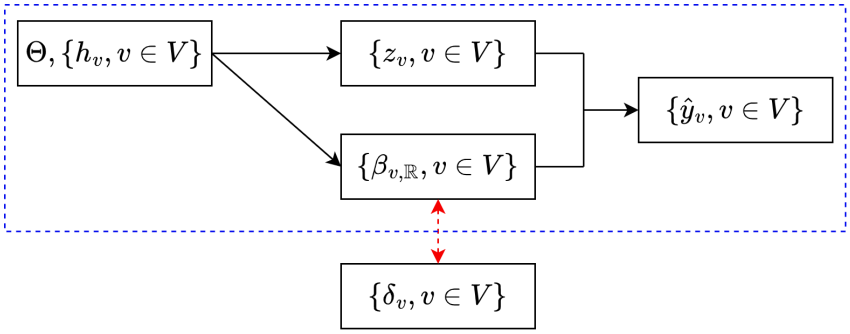


Figure 3.4: The model pipeline is shown in the (blue) dashed box, while the geometric hyperbolicity can be computed independently of the model.

In principle, we want to compare $\{\beta_{v,\mathbb{R}}, v \in V\}$ and $\{\delta_v, v \in V\}$ so that the geometric hyperbolicity guides the choice of model hyperbolicity. However, comparing pairwise β_v and δ_v for each $v \in V$ may lead to overfitting. An alternative is to compare their respective distributions ν_G and μ_G, or even coarser statistics (e.g., mean) of ν_G and μ_G (cf. Fig. 3.5). The latter may lead to underfitting. We perform an ablation study on the different comparison methods in Section 3.5.4.

$$\begin{array}{ccccc} \{\beta_{v,\mathbb{R}}, v \in V\} & \mapsto & \nu_G & \mapsto & \text{Stat. of } \nu_G \\ \updownarrow & & \updownarrow & & \updownarrow \\ \{\delta_v, v \in V\} & \mapsto & \mu_G & \mapsto & \text{Stat. of } \mu_G \end{array}$$

Figure 3.5: Different ways of comparing geometric and model hyperbolicities.

We advocate choosing the middle ground by comparing the distributions μ_G and ν_G. The former can be computed readily as long as the ambient graph G is given, while the latter is a part of the model that plays a crucial role in feature aggregation at each node. Therefore, μ_G can be pre-determined but not ν_G. We propose to use the known μ_G to constrain ν_G and thus the model parameters Θ. A widely used comparison tool is the Wasserstein metric.

Definition 3 (Wasserstein distance)**.** *Given $p \geq 1$, the p-Wasserstein distance metric [100] measures the difference between two different probability distributions [101]. Let $\Pi(\nu_G, \mu_G)$ be the set of all joint distributions for random variables x and y where $x \sim \nu_G$*

and $y \sim \mu_G$*. Then the p-Wasserstein distance between* μ_G *and* ν_G *is as follows:*

$$W_p(\nu_G, \mu_G) = \left\{ \inf_{\gamma \in \Pi(\nu_G, \mu_G)} \mathbb{E}_{(x,y)\sim\gamma} \|x - y\|^p \right\}^{1/p}. \tag{3.17}$$

To compute the Wasserstein distance exactly is costly given that the solution of an optimal transport problem is required [102, 103]. However, for one-dimensional distributions, the p-Wasserstein distance can be computed by ordering the *samples* from the two distributions and then computing the average p-distance between the ordered samples [102, 104].

In ideal circumstances, considering the distributions do not lose much information. We first notice that for both $\beta_{v,\mathbb{R}}$ and δ_v, a smaller value means more hyperbolic in an appropriate sense. Suppose $\beta_{v,\mathbb{R}}$ is increasing w.r.t. δ_v, i.e., $\delta_v \leq \delta_u$ implies that $\beta_{v,\mathbb{R}} \leq \beta_{u,\mathbb{R}}$. Then, $W_2(\mu_G, \nu_G) = \sqrt{\frac{1}{|V|} \sum_{v \in V} |\beta_{v,\mathbb{R}} - \delta_v|^2}$.

3.4.4 Non-uniformity of selection weights

A node is considered to be more suitable to be embedded in the hyperbolic space when $\beta_{v,\mathbb{D}} > \beta_{v,\mathbb{R}}$. Meanwhile when $\beta_{v,\mathbb{D}} \leq \beta_{v,\mathbb{R}}$, the node is considered to be Euclidean. Nevertheless, to align with our motivation that each node can be better embedded in one of the two spaces and the less suitable space would result in distortion in representation, we require JSGNN to learn non-uniform attention weights, meaning that each pair of attention weights $(\beta_{v,\mathbb{D}}, \beta_{v,\mathbb{R}})$ should significantly deviate from the uniform distribution. This is because soft selection without a non-uniformity constraint may result in the assignment of nodes to be partially Euclidean and partially hyperbolic with $\beta_{v,\mathbb{R}} \approx \beta_{v,\mathbb{D}} \approx 0.5$. Hence, we include an additional component to the standard loss function encouraging non-uniform learned weights as follows:

$$L_{\text{nu}} = -\frac{1}{|V|} \sum_{v \in V} \big(\beta_{v,\mathbb{R}}^2 + \beta_{v,\mathbb{D}}^2\big). \tag{3.18}$$

Since $-1 \leq -(\beta_{v,\mathbb{R}}^2 + \beta_{v,\mathbb{D}}^2) \leq -0.5$ and $\beta_{v,\mathbb{R}} + \beta_{v,\mathbb{D}} = 1$, minimizing the term would favor non-uniform attention weights for each node.

In summary, we may combine hyperbolicity matching discussed in Section 3.4.3 and the non-uniformity loss to form the loss function to optimize JSGNN.

$$L_{\text{overall}} = L_{\text{task}} + \omega_{\text{nu}} L_{\text{nu}} + \omega_{\text{was}} W_2(\nu_G, \mu_G), \tag{3.19}$$

where L_{task} is the task-specific loss, while ω_{nu} and ω_{was} are balancing factors. For the node classification task, L_{task} refers to the cross-entropy loss over all labeled nodes while for link prediction, it refers to the cross-entropy loss with negative sampling. This completes the description of the JSGNN model.

We speculate that the non-uniform component L_{nu} should push the model hyperbolicities towards the two extremes 0 and 1. On the other hand, as we have seen in Section 3.4.3, to compute $W_2(\nu_G, \mu_G)$, we need to order $(\delta_v)_{v\in V}$, $(\beta_{v,\mathbb{R}})_{v\in V}$ respectively, and compute their pairwise differences. Therefore, $W_2(\nu_G, \mu_G)$ aligns the shapes of ν_G and μ_G.

3.5 Experiments

In this section, we evaluate JSGNN on node classification and link prediction tasks against twelve baselines. A total of seven benchmark datasets are employed for both tasks. Specifically, Cora, Citeseer, Pubmed, Airport, Disease, Photo and CS. For detailed information about the datasets and their statistics, refer to Section 2.7.1.

3.5.1 Baselines and settings

For JSGNN, we consider two variants: GAT as the Euclidean model with HGAT as the hyperbolic model, and GCN as the Euclidean model with HGCN as the hyperbolic model. We compare against the following models:

- Euclidean methods: GCN [46], GraphSAGE [47] and GAT [25];
- Hyperbolic models: HGCN [71], HGNN [72], HGAT [73], LGCN [74] and κ-GCN ($\mathbb{D}^{16}$) [68], which is a constant (negative) curvature graph neural network based on the κ-stereographic model;

Table 3.1: Node classification result on Cora, Citeseer and Pubmed datasets. Performance score averaged over ten runs. The best performance is boldfaced while the second-best performance is underlined.

Method	Standard split			60/20/20% split		
	Cora	Citeseer	Pubmed	Cora	Citeseer	Pubmed
GCN	81.53 ± 0.84	70.47 ± 0.64	78.30 ± 0.63	91.99 ± 0.79	84.13 ± 0.98	88.79 ± 1.63
GAT	81.68 ± 1.06	70.96 ± 0.96	78.05 ± 0.50	91.63 ± 0.57	83.93 ± 0.85	<u>89.99 ± 1.31</u>
GraphSAGE	76.59 ± 1.06	65.26 ± 2.91	77.90 ± 0.71	91.25 ± 0.22	84.08 ± 0.25	89.62 ± 0.18
CurvGN	81.58 ± 0.51	71.14 ± 0.67	78.17 ± 0.48	91.60 ± 0.25	84.18 ± 0.37	86.65 ± 0.14
CGNN	<u>82.15 ± 0.60</u>	**71.31 ± 1.16**	78.34 ± 0.83	91.96 ± 0.27	84.29 ± 0.34	86.86 ± 0.16
HGNN	79.28 ± 0.77	70.00 ± 0.74	77.45 ± 1.40	89.73 ± 0.84	80.27 ± 0.21	88.27 ± 0.51
HGCN	78.68 ± 0.77	67.25 ± 1.45	76.72 ± 0.92	91.57 ± 0.28	83.68 ± 0.52	86.83 ± 0.31
HGAT	78.81 ± 1.49	68.16 ± 1.34	77.43 ± 1.20	90.27 ± 0.81	81.29 ± 0.79	86.27 ± 0.47
LGCN	78.93 ± 0.79	68.59 ± 0.64	78.08 ± 0.65	92.55 ± 0.57	<u>85.03 ± 0.28</u>	89.59 ± 0.11
κ-GCN ($\mathbb{D}^{16}$)	78.64 ± 0.83	67.17 ± 0.73	78.01 ± 0.67	89.16 ± 0.59	84.57 ± 0.20	88.37 ± 0.37
κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$)	78.71 ± 1.02	66.96 ± 1.13	77.67 ± 0.74	88.85 ± 0.88	84.04 ± 0.81	85.59 ± 0.53
GIL	79.97 ± 1.93	67.54 ± 1.23	76.62 ± 0.81	91.90 ± 0.84	82.39 ± 0.90	87.39 ± 0.21
JSGNN (GCN+HGCN)	81.79 ± 0.80	70.55 ± 1.09	<u>78.38 ± 0.74</u>	**93.26 ± 0.92**	84.95 ± 0.31	89.68 ± 0.60
JSGNN (GAT+HGAT)	**82.94 ± 0.55**	<u>71.26 ± 1.13</u>	**78.57 ± 0.90**	<u>93.10 ± 0.86</u>	**85.10 ± 0.64**	**90.53 ± 0.32**

Table 3.2: Node classification result on CS, Photo, Airport and Disease datasets. OOM corresponds to out-of-memory.

Method	CS	Photo	Airport	Disease
GCN	96.53 ± 0.10	94.05 ± 0.27	79.62 ± 1.28	83.32 ± 1.37
GAT	96.36 ± 0.38	94.45 ± 0.92	83.07 ± 1.52	86.05 ± 1.08
GraphSAGE	96.45 ± 0.91	96.13 ± 1.61	81.73 ± 0.98	83.47 ± 1.77
CurvGN	96.91 ± 0.09	94.21 ± 0.22	87.25 ± 0.88	84.35 ± 3.20
CGNN	96.27 ± 0.08	93.93 ± 0.18	87.21 ± 1.08	85.75 ± 2.08
HGNN	96.72 ± 0.19	94.74 ± 0.66	84.37 ± 1.19	87.40 ± 1.66
HGCN	96.58 ± 0.10	95.27 ± 0.25	89.39 ± 1.52	87.93 ± 1.61
HGAT	96.65 ± 0.15	96.62 ± 0.28	89.31 ± 1.09	90.04 ± 1.50
LGCN	OOM	<u>96.71 ± 0.24</u>	88.53 ± 1.26	<u>91.15 ± 1.02</u>
κ-GCN ($\mathbb{D}^{16}$)	97.04 ± 0.10	95.56 ± 0.54	89.08 ± 1.15	**92.39 ± 0.73**
κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$)	96.97 ± 0.10	94.31 ± 0.41	88.38 ± 0.62	89.47 ± 1.56
GIL	95.83 ± 0.30	94.41 ± 0.57	**90.78 ± 1.74**	90.67 ± 1.98
JSGNN (GCN+HGCN)	<u>97.08 ± 0.04</u>	95.69 ± 0.22	<u>90.59 ± 1.75</u>	90.73 ± 1.44
JSGNN (GAT+HGAT)	**97.40 ± 0.14**	**97.16 ± 0.44**	90.33 ± 1.61	90.88 ± 1.54

- CurvGN [67] and CGNN [66] which utilizes graph curvature information to filter messages differently based upon different local structures;
- We also consider GIL [76] and κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$), which similar to JSGNN, leverages multiple spaces or a mixed curvature space. κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$) employs a two-component product space of negative and zero curvature.

For all models, the hidden units are set to 16. We set the early stopping patience to 100 epochs with a maximum limit of 2000 epochs. The hyperparameter settings for the baselines are the same as [76] if given. The only difference is that the hyperparameter

h-drop for GIL in [76] (which determines the dropout to the weight associated with the hyperbolic space embedding) is set to 0 for all datasets as setting a large value essentially explicitly chooses one single space. Else, the hyperparameters are chosen to yield the best performance. For JSGNN, we perform a grid search on the following search spaces: Learning rate: [0.01, 0.005]; Dropout probability: [0.0, 0.1, 0.5, 0.6]; Number of layers: [1, 2, 3]; ω_{nu} and ω_{was}: [1.0, 0.5, 0.2, 0.1, 0.01, 0.005]; $\mathbf{q}$ (cf. (3.12)): [16, 32, 64]. The Wasserstein-2 distance is employed in all variants of JSGNN.

3.5.2 Node classification with discussion

For the node classification task, each of the nodes in a dataset belongs to one of the C classes in the dataset. With the final set of node representations, we aim to predict the labels of nodes that are in the testing set.

To test the performance of each model under both semi-supervised and fully-supervised settings, two data splits are used in the node classification task for the Cora, Citeseer and Pubmed datasets. In the first split, we followed the standard split for semi-supervised settings used in [25, 46, 47, 76, 97, 105–108]. The train set consists of 20 train examples per class while the validation set and test set consist of 500 samples and 1,000 samples, respectively. Meanwhile, in the second split, all labels are utilized and the percentages of training, validation, and test sets are set as 60/20/20%. For the Photo and CS datasets, the labeled nodes are also split into three sets where 60% of the nodes made up the training set, and the rest of the nodes were divided equally to form the validation and test sets. Airport and Disease datasets were split in similar settings as [76].

In Table 3.1 and Table 3.2, the mean accuracy with standard deviation is reported for node classification, except for the case of Airport and Disease datasets where the mean F1 score is reported. Our empirical results demonstrate that JSGNN frequently outperforms the baselines, especially HGAT and GAT which are the building blocks of JSGNN. Even though the performance of the variant JSGNN (GCN+HGCN) is often slightly lower than JSGNN (GAT+HGAT), we have similarly observed it to consistently outperform its building blocks GCN and HGCN. This is not necessarily observed for other mixed

space models. Thus, this shows the superiority of not only using both Euclidean and hyperbolic spaces but also our method of incorporating the two spaces for graph learning as compared to GIL and κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$).

We also observe that Euclidean models such as GCN, GAT, GraphSAGE, CurvGN and CGNN perform better than hyperbolic models in general on the Cora, Citeseer, and Pubmed datasets for both splits. Meanwhile, hyperbolic models achieve better results on the CS, Photo, Airport, and Disease datasets. This means that Euclidean features are more significant for representing Cora, Citeseer and Pubmed datasets while hyperbolic features are more significant for the others. Nevertheless, JSGNN is able to perform relatively well across all datasets. We note that JSGNN exceeds the performance of single-space baselines on all datasets except for Disease. This can be explained by the fact that Disease consists of a perfect tree and thus, does not exhibit different hyperbolicities in the graph. However, JSGNN still outperforms 3 hyperbolic benchmarks and all the other mixed models.

We also particularly note that the difference in results between single-space models using only the Euclidean embedding space and hyperbolic models is not significant. This means that many of the node labels can be potentially predicted even without the best representation from the right space. This might be the reason why the gain in performance for the node classification task is not exceptional from embedding nodes in the better space. Nevertheless, we still see improvements in predictions for cases where there is a mixture of local hyperbolicities. Moreover, embedding nodes in a more suitable space can benefit other tasks that require more accurate representations such as link prediction.

3.5.3 Link prediction with discussion

We employ the Fermi-Dirac decoder with a distance function to model the probability of an edge based on our final output embedding, similar to [71, 76, 77]. The probability that an edge exists is given by $\mathbb{P}(e_{vj} \in E \mid \Theta) = (e^{(d(x_i,x_j)-r)/t} + 1)^{-1}$ where $r, t > 0$ are hyperparameters and d is the distance function. The edges of the datasets are randomly split into 85/5/10% for training, validation, and testing. The average ROC AUC for link prediction is recorded in Table 3.3. We observe that JSGNN (GAT+HGAT) performs

Table 3.3: Link prediction result averaged over ten runs.

Method	Cora	Citeseer	Pubmed	Airport	Disease
GCN	88.22 ± 1.01	90.60 ± 1.10	87.63 ± 3.25	91.79 ± 1.48	61.60 ± 3.76
GAT	85.47 ± 2.28	85.31 ± 1.89	85.30 ± 1.46	93.70 ± 0.65	61.23 ± 2.75
GraphSAGE	88.94 ± 0.81	91.61 ± 1.00	88.42 ± 1.14	91.63 ± 0.81	68.31 ± 2.94
CurvGN	94.40 ± 2.13	95.38 ± 2.33	94.55 ± 0.89	93.90 ± 0.37	95.47 ± 0.81
CGNN	94.26 ± 1.36	96.54 ± 0.78	94.71 ± 3.14	95.13 ± 0.97	95.35 ± 1.22
HGNN	91.48 ± 0.38	93.63 ± 0.14	92.95 ± 0.35	96.31 ± 0.30	82.98 ± 0.98
HGCN	93.72 ± 0.26	96.72 ± 1.69	96.68 ± 0.04	97.55 ± 0.08	87.14 ± 1.34
HGAT	94.06 ± 0.11	95.60 ± 0.20	95.78 ± 0.05	97.86 ± 0.08	86.61 ± 1.67
LGCN	93.10 ± 0.30	93.40 ± 0.70	95.45 ± 0.08	97.88 ± 0.19	95.99 ± 0.58
κ-GCN ($\mathbb{D}^{16}$)	92.43 ± 0.63	94.38 ± 0.51	94.89 ± 0.07	96.78 ± 0.19	93.58 ± 0.31
κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$)	91.32 ± 0.38	92.87 ± 0.27	93.53 ± 0.06	97.17 ± 0.10	90.15 ± 0.77
GIL	98.04 ± 1.64	99.95 ± 0.09	92.50 ± 0.50	97.20 ± 1.04	**100.00 ± 0.00**
JSGNN (GCN+HGCN)	98.83 ± 0.92	99.97 ± 0.09	**97.67 ± 0.03**	98.94 ± 0.91	**100.00 ± 0.00**
JSGNN (GAT+HGAT)	**99.43 ± 0.21**	**99.98 ± 0.05**	96.95 ± 0.03	**99.26 ± 1.23**	99.97 ± 0.08

better than the baselines in most cases. For the link prediction task, we notice that hyperbolic models consistently outperform Euclidean models by a significant margin. Moreover, Euclidean methods such as CurvGN and CGNN benefit from using topological information during learning. Empirical results also suggest that predicting the existence of edges seems to benefit from dual space models, i.e., GIL and JSGNN, except for the case of κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$).

This finding is similar to that reported in [68, 109] where despite slightly different settings, the *constant (negative) curvature κ-stereographic model frequently outperforms the κ-GCN leveraging on the product of multiple constant curvature spaces*. We hypothesize that the simple concatenation to combine the embeddings of the two different component spaces in κ-GCN ($\mathbb{D}^{16} \times \mathbb{R}^{16}$) might be insufficient and might have resulted in noise from the other space being passed to the negatively curved space which was performing well standalone as seen in κ-GCN ($\mathbb{D}^{16}$). As such, our aim to learn to select the better space for each node is potentially capable of offering better representations with reduced distortions.

Table 3.4: Ablation study of JSGNN (GAT+HGAT) for node classification task. Cora, Citeseer and Pubmed on the standard split.

Method	CS	Photo	Cora	Citeseer	Pubmed	Airport	Disease
JSGNN (GAT+HGAT)	**97.40 ± 0.14**	**97.16 ± 0.44**	**82.94 ± 0.55**	**71.26 ± 1.13**	**78.57 ± 0.90**	**90.33 ± 1.61**	**90.88 ±1.54**
w/o NU & W_2	97.15 ± 0.10	95.95 ± 0.41	81.65 ± 1.08	70.87 ± 1.22	78.14 ± 1.02	89.67 ± 1.26	89.85 ± 1.61
w/o W_2	97.33 ± 0.20	96.51 ± 0.67	82.36 ± 0.78	71.15 ± 1.17	78.50 ± 0.53	90.02 ± 1.63	90.66 ± 2.22
w/o NU	97.38 ± 0.15	96.42 ± 0.37	82.67 ± 0.51	70.86 ± 1.45	78.48 ± 0.47	89.98 ± 1.72	90.37 ± 2.12

3.5.4 Ablation study

We conduct an ablation study on the node classification task by introducing three variants of JSGNN (GAT+HGAT) to validate the effectiveness of the different components:

- Without the non-uniformity constraint (w/o NU): This does not enforce the model to learn non-uniform selection weights.
- Without the Wasserstein metric (w/o W_2): The learning of model hyperbolicity is not guided by geometric hyperbolicity.
- Without the non-uniformity loss and Wasserstein distance (w/o NU & W_2): Only guided by the cross entropy loss, i.e., $\omega_{\text{nu}} = 0, \omega_{\text{was}} = 0$ (cf. (3.19)).

Table 3.4 summarizes the results of our study, from which we observe that all variants of JSGNN (GAT+HGAT) with some components discarded perform worse than the full model. Moreover, JSGNN (GAT+HGAT) without W_2 always achieves better results than JSGNN (GAT+HGAT) without NU and W_2, signifying the importance of selecting the better of the two spaces instead of combining the features with relatively uniform weights. Similarly, JSGNN (GAT+HGAT) without NU performs better than JSGNN (GAT+HGAT) without NU and W_2 in most cases, suggesting that incorporating geometric hyperbolicity through distribution alignment does help to improve the model.

To further analyze our model, we present a study regarding our method of incorporating the guidance of geometric hyperbolicity through distribution alignment. The result is as seen in Table 3.5. We test and analyze empirically different variants of our model based on the different comparisons shown in Fig. 3.5. Pairwise match indicates minimizing the mean squared error between elements of Γ_G and Δ_V (without sorting) while mean match minimizes the squared loss between the means of Γ_G and Δ_V. We observe that comparing the distributions of ν_G and μ_G consistently outperforms comparing their mean, demonstrating the insufficiency of utilising coarse statistics for supervision. Secondly, pairwise matching gave better results than mean matching, though still lower than distribution matching, suggesting the importance of fine-scale information yet, a need to avoid potential overfitting.

Table 3.5: Node classification results of different comparison methods to incorporate geometric hyperbolicity to guide model hyperbolicity.

Dataset	Pairwise match	Distribution	Mean match
Cora	82.35 ± 1.06	**82.94 ± 0.55**	81.36 ± 1.50
Citeseer	70.06 ± 2.05	**71.26 ± 1.13**	69.64 ± 1.18
Pubmed	78.46 ± 0.86	**78.57 ± 0.90**	78.08 ± 0.62
Aiport	90.13 ± 1.53	**90.33 ± 1.61**	89.31 ± 2.22
Disease	90.66 ± 1.91	**90.88 ± 1.54**	87.53 ± 6.24
Photo	96.17 ± 0.23	**97.16 ± 0.44**	95.96 ± 0.59
CS	97.20 ± 0.16	**97.40 ± 0.14**	97.17 ± 0.11

3.5.5 Analysis of hyperbolicities

We have speculated the effects of different components of our proposed model at the end of Section 3.4.4. To verify that our model can learn model hyperbolicity that is non-uniform and similar in distribution as geometric hyperbolicity, we analyze the learned model hyperbolicities $(\beta_{v,\mathbb{R}})_{v\in V}$ of JSGNN (GAT+HGAT) and the model w/o NU & W_2 for the node classification task. Specifically, we extract the learned values from the first two layers of JSGNN and its variant for ten separate runs. The learned values from the first two layers were then averaged before determining $W_2(\nu_G, \text{Unif})$ and $W_2(\nu_G, \mu_G)$.

In Fig. 3.6, it can be inferred that JSGNN's learned model hyperbolicity is always less uniform than that of the model w/o NU & W_2 given JSGNN's larger $W_2(\nu_G, \text{Unif})$ score, demonstrating a divergence from uniform distribution. Meanwhile, for most cases, JSGNN's $W_2(\nu_G, \mu_G)$ is smaller than that of the model w/o NU & W_2, suggesting that the shape between ν_G and μ_G of JSGNN is relatively more similar. At times, JSGNN's $W_2(\nu_G, \mu_G)$ is larger than the model w/o NU & W_2, suggesting a tradeoff between NU and W_2 as we choose the optimal combination for the model's best performance.

3.6 Conclusion

In this chapter, we introduced JSGNN, a GNN that learns in a joint space setting given that different regions of a graph can have different geometrical characteristics. In these situations, it would be beneficial to embed different regions of the graph in different

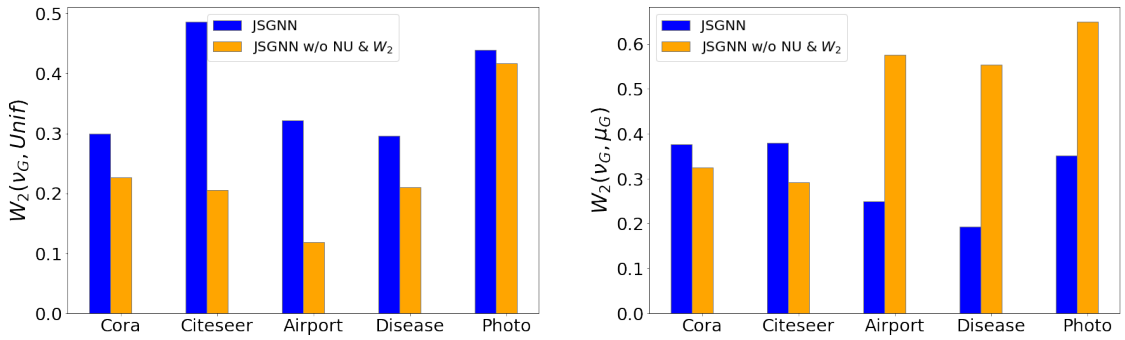


Figure 3.6: Analysis of hyperbolicities on different datasets. (a) $W_2(\nu_G, \text{Unif})$. (b) $W_2(\nu_G, \mu_G)$.

spaces that are better suited for their underlying structures, to reduce the distortions incurred while learning node representations. Our method JSGNN utilizes a soft attention mechanism with non-uniformity constraint and distribution alignment between model and geometric hyperbolicities to select the best space-specific feature for each node. This indirectly finds the space that is best suited for each node. Experimental results of node classification and link prediction demonstrate the effectiveness of JSGNN against various baselines.

From past research, we have observed the evolutionary trajectory of GNNs on homogeneous graphs, which initially centered on Euclidean geometry before branching out into hyperbolic spaces and eventually mixed spaces. While leveraging multiple geometric spaces has proven to be effective in reducing distortions and improving results, there is a need to explore additional avenues for augmenting the model's expressive power beyond merely incorporating diverse geometric spaces. This is particularly relevant for more complex graphs, such as heterogeneous graphs. To address this limitation, in the next chapter, we present an approach to enhance acquired graph embeddings by incorporating simplicial complexes — spaces formed by glueing together simplices. This has the potential to improve the expressivity of models beyond the 1-WL test. Geometrically, simplices can be interpreted as geometric shapes, such as triangles and tetrahedrons. This approach not only enhances the model's expressive power but also captures essential high-order interactions between multiple nodes, a critical consideration for heterogeneous graphs, as indicated by the widespread use of constructs like metapaths and metagraphs.

Chapter 4

Simplicial Graph Attention Network

In this chapter, we make a pivotal shift from analyzing homogeneous graphs to exploring the intricacies of heterogeneous graphs, which possess greater complexity and richer semantics, stemming from the multiple node and edge types present. Models tailored for homogeneous graphs, like the components of JSGNN, cannot be applied directly to heterogeneous graphs. Although heterogeneous graphs also have diverse graph structures across different regions and may benefit from using multiple spaces for embedding learning, the new focus here would be to capture high-order interactions involving multiple nodes that cannot be reduced to pairwise interactions. More specifically, we delve into the concept of simplicial complexes (defined in Section 4.2) to model high-order relations.

In this context, we present SGAT, a GNN model that extends GAT to heterogeneous graphs with simplicial complexes. Handling heterogeneous graphs demands greater intricacy and a higher modelling capacity. Conventional methods rely on metapaths or metagraphs to address heterogeneity. Nonetheless, there are three key limitations in these methods, which we will discuss in detail later (in Section 4.1). To overcome these limitations, we advocate the use of simplicial complexes for modeling higher-order relationships and facilitating information exchange among higher-order simplices. This enhances the model's capacity to extract intricate structural information and learn high-quality embeddings, leading to performance improvements over baseline models in heterogeneous node classification tasks.

4.1 Introduction

While previous methods [81, 82, 85, 110] have achieved commendable results by relying on metapaths and metagraphs, SGAT takes a distinct path by refraining from relying on these constructs. This is because metapath-based methods have at least one of the following limitations:

1. Metapath-based models are sensitive to metapaths with suboptimal paths leading to suboptimal results [78]. Metapath selection requires domain knowledge of the dataset and the task at hand. Likewise, metagraphs are predefined based on domain knowledge. Alternatively, heuristic algorithms can be employed to extract the most common structures to form metagraphs [84]. Yet, the generated metagraphs may not necessarily be useful [111].

2. Metapath-based models do not incorporate features from non-target nodes along the metapaths, thus discarding potentially useful information [82]. Examples include HERec [110] and HAN [81].

3. The model relies on *structures between two nodes* to capture complex pairwise relations and semantics. This, however, is not always sufficient to capture more complex interactions in graphs which simultaneously involve multiple target nodes and cannot be reduced to pairwise relationships, especially for heterogeneous graphs given their richer semantics [112].

Several works have aimed to improve the expressive power of GNNs and generate more accurate representations of heterogeneous graph components without utilizing metapaths or metagraphs. One potential alternative is a simplicial complex. Simplicial complexes are natural extensions of graphs that can represent high-order interactions between nodes. A graph defines pairwise relationships between elements of a vertex set. Meanwhile, a simplicial complex defines higher order relations, e.g., a 3-tuple is a triangle, a 4-tuple is a tetrahedron and so on (for a proper definition, see Section 4.2). For instance, a triangle made up of three author vertices in a citation network indicates co-authorship of the same

paper between the three authors, whereas in a graph, edges between pairs of these three authors only tell us that they are pairwise co-authors with each other on some papers. These structures are already employed in Topological Data Analysis (TDA) to extract information from data and in other applications such as tumor progression analysis [113] and brain network analysis [114] to represent complex interactions between elements. The GNN literature [42, 112] on simplicial complexes thus far have focused on homogeneous graphs. These works cannot be directly applied to heterogeneous graphs.

To apply simplicial complex to heterogeneous graphs, we first propose a procedure to generate k-order homogeneous simplices from a heterogeneous graph since heterogeneous datasets do *not* always possess higher-order simplices, given their innate schemas. Additionally, to avoid discarding potentially useful information when transforming the heterogeneous graph into homogeneous simplices, we populate the k-simplices, for $k \geq 1$, with non-target node features and learn the importance of each of the k-order simplicies through attention mechanisms with upper adjacencies. Since SGAT does not utilize metapaths or metagraphs. Hence, SGAT is not subjected to the three characteristic limitations of metapath-based methods

The remainder of this chapter is structured as follows: We begin by introducing the concept of a simplicial complex, its adjacencies, and relevant notations. Subsequently, we provide a detailed explanation of the construction of simplices and how they are incorporated into our model. Experimental results and model analysis are presented in Section 4.3 and Section 4.4 respectively before concluding in Section 4.5.

4.2 Simplicial Graph Attention Network

Simplicial Complexes. A simplicial complex χ is a collection of finite sets (simplices) closed under subsets. A member of χ is a called a k-simplex and denoted as σ^k if it has cardinality $k+1$. A k-simplex has $k+1$ faces of dimension $k-1$. A face is obtained by omitting one element from the simplex. Specifically, by omitting the j-th element, a face of a k-simplex is a set containing elements of the form $(v_0, \ldots, v_{j-1}, v_{j+1}, \ldots, v_k)$.

Subsequently, it is possible to think of 0-simplices σ^0 as vertices, 1-simplices σ^1 as edges, 2-simplices σ^2 as triangles, 3-simplices σ^3 as tetrahedron and so on. Lastly, the dimension of χ is the largest dimension of any simplex in it and is denoted by K. The set of all simplices in χ of dimension k is denoted by χ^k with cardinality $|\chi^k|$.

Simplicial Adjacencies. In this work, we utilize the notion of upper-adjacency to denote the neighborhood of simplices. Two k-simplices in χ^k are upper-adjacent if they are faces of the same $(k+1)$-simplex. For instance, two edges are upper-adjacent if they are part of a same triangle. Hence, the neighborhood of a k-simplex, $N(\sigma_i^k)$ is the set of k-simplices upper-adjacent to it. We include self-loops to the upper adjacency matrix so that a k-simplex is upper-adjacent to itself. We have

$$N(\sigma_i^k) = \left\{ \sigma_j^k \in \left\{ \sigma_1^k \dots \sigma_{|\chi^k|}^k \right\} \; : \; A^k(i,j) = 1 \right\}, \tag{4.1}$$

where $A^k \in \mathbb{R}^{|\chi^k| \times |\chi^k|}$ denotes the upper adjacency matrix indicating whether pairs of k-simplices are upper-adjacent. The reader is referred to the Appendix A and [115] for further details.

4.2.1 Construction of Simplices

Many real-world heterogeneous graph datasets do not have predefined higher-order k-simplices for $k \geq 2$ due to their unique schemas. We propose a procedure to transform heterogeneous graphs into homogeneous simplices where the 0-simplices are the singleton subsets of the target node type's vertices. The target node type is the node type whose labels we want to predict (node classification). Meanwhile, non-target nodes features reside on higher order simplices.

Consider a heterogeneous graph $G = (V, E)$ with a edge-types and b node types, where $a + b > 2$. The set of nodes is V and the set of edges is given by E. Let $\tilde{V} \subset V$ be the set of target nodes. We assume that each $v \in \tilde{V}$ has a feature vector h_v. To simplify the presentation, we assume that every non-target node $u \in V \backslash \tilde{V}$ is also associated with a feature vector h_u (if a non-target node does not come with a feature vector, we can

associate with it a one-hot encoding of its node type).

For $1 \leq k \leq K$, where K is a hyperparameter, we choose the k-simplices to be the sets of $k+1$ target vertices that share at least ϵ_η^k common non-target neighbors which are exactly η hops away, where $\eta > 0$ is a chosen parameter. This is performed for each $\eta \leq \eta_{\max}$, where $\eta_{\max}$ is a hyperparameter, to generate η number of K dimensional simplicial complexes. Consider a k-simplex σ^k, where $k > 1$, whose vertices share $m \geq \epsilon_\eta^k$ common η-hop non-target neighbors, $s_1, \ldots, s_m$. The k-simplex is assigned the feature $h_{\sigma^k} = \phi(h_{s_1}, \ldots, h_{s_m})$ where ϕ is the average operator.

For 1-simplices, the features of the intermediate node(s) along a path between the 0-simplices are first summed to form the path feature, $\Theta_{\sigma^1} = \sum_{u \in \theta} h_u$ where θ is the set of intermediate nodes along the path connecting the faces of σ^1. The path features between the same pair of 0-simplices are then averaged and assigned as the feature of the 1-simplex. In addition, we place the simplex's own feature on the connecting simplex when a self-loop is present.

Taking the IMDB[1] dataset as an example in Fig. 4.1, we set $\eta = 1$ and $\epsilon_1^1 = 1$. The constructed simplicial complex is illustrated in Fig. 4.1. We note that the set of 1-simplices is different from the original edges in the graph.

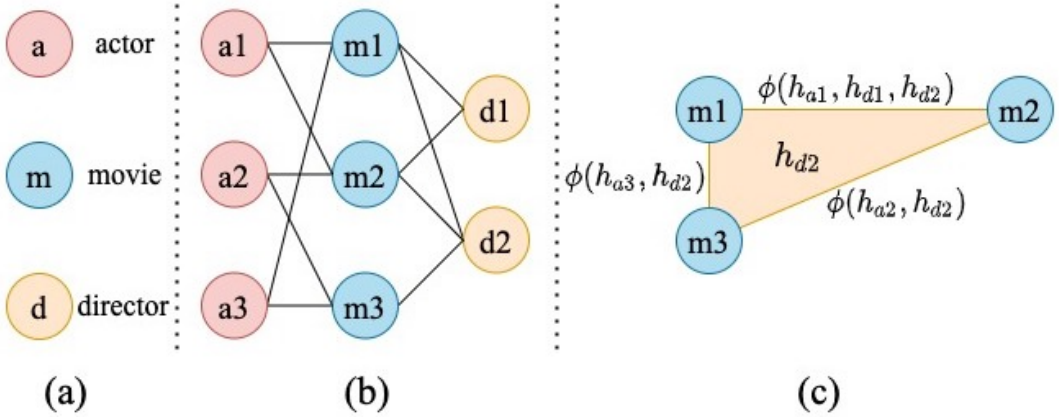


Figure 4.1: Constructing simplicial complexes from the heterogeneous IMDB graph (a)-(c) with $\eta = 1$. (a) IMDB node types. (b) IMDB graph illustration. (c) Resulting movie one-hop sharing complex with $\epsilon_1^1 = 1$.

The technical details to construct all the k-simplices of a single simplicial complex

[1]https://www.imdb.com/interfaces/

within a network are given in Algorithm 1.

4.2.1.1 Pseudocode

For simplicity, we assume that the ϵ_η^k is the same for all k-order simplices and the specified η. The Geometric Understanding in Higher Dimensions library [116] for computational topology is utilised to ensure the constructed simplices form simplicial complex(es) that adhere to the inclusion property.

4.2.1.2 Incorporation of Edge Features

In the case where each edge $(i, j) \in E$ has a feature vector e_{ij}, we concatenate the feature generated as described above for a 1-simplex $h_{\sigma^1} = \phi(\Theta_{\sigma^1,1}, \ldots, \Theta_{\sigma^1,t})$ where t is the number of paths between the faces of σ^1, with the average of the edge features along the original path in the graph that gives rise to σ^1. For σ^1 with end nodes $\tilde{v}$ and v, we have

$$h_{\sigma^1} = \phi(\Theta_{\sigma^1,1}, \ldots, \Theta_{\sigma^1,t}) \parallel \phi(\Phi_1, \ldots, \Phi_m), \tag{4.2}$$

$$\Phi_i = \phi(e_{\tilde{v}s_{i,1}}, \ldots, e_{s_{i,j}s_{i,j+1}}, \ldots, e_{s_{i,\kappa}v}), \tag{4.3}$$

where κ is the number of intermediate nodes between the end nodes and Φ_i is the summarized edge feature along each of the original m paths. We note that many heterogeneous GNNs such as GTN and HAN are not designed to incorporate edge features during their learning process.

When $n > k$ target nodes share more than ϵ_η^k non-target nodes among them, a total of $\binom{n}{k+1}$ k-simplices are constructed, which may cause memory overloading issues for large values of n. To further control the the amount of constructed simplices, a hyperparameter $\lambda > K$ is introduced. We do not construct the simplices (and their corresponding faces) if $n \geq \lambda$. This can also be interpreted as disregarding the k-simplices (and their associated faces) where $k \geq \lambda$.

Algorithm 1 Construction of k-simplices

Input: The adjacency list of heterogeneous graph Adj_list,
Node features X,
Number of shared non-target neighbours ϵ,
Number of hops away η,
Maximal k-order considered, K,
The maximum simplex order to construct λ.
Output: Set of all k-simplices, AllKSimplices.

▷ `DFSFindPaths` is a modified depth first search that breaks at 2η depth. To get paths that starts and ends with target node type. Each path should contain at least one non-target node.
▷ Each path is of form [target_src, node_k(s), target_dst], where node_k is in θ and θ is the set of intermediate nodes along the path.
Initialise PathList as empty list.
for $v \in V$ **do**
    PathList ← `DFSFindPaths`(Adj_list, $v, 2\eta$)
end for
▷ The middle node refers to the non-target node η hops away from the target_src and target_dst.
Initialise MidNodeNeighborDict, D_{mid}
▷ where key is MidNodeID and value is its set of unique Neighbors.
for path ∈ PathList **do**
    D_{mid}[path.MidNodeID].insert(path.src)
    D_{mid}[path.MidNodeID].insert(path.dst)
end for
Initialise KSimplicesDict
▷ where key is the Neighbors and value is the list of MidNodeID
for MidNodeID, Neighbors ∈ D_{mid} **do**
    if $2 \leq$ size(Neighbors) $\leq \lambda$ **then**
        KSimplicesDict[Neighbors].insert(MidNodeID)
    end if
end for
for Neighbors, MidNodeIDList in KSimplicesDict **do**
    if size(MidNodeIDList) $\geq \epsilon$ **then**
        ▷ Simplex tree is a data structure in Gudhi library
        SimplexTree.insert(Neighbors)
    end if
end for
▷ Get up to K simplices from SimplexTree
AllKSimplices ← SimplexTree.get_simplices()
return AllKSimplices

4.2.2 Simplicial Attention Layer

After constructing the k-simplices, $0 \leq k \leq K$, from the original graph, the inputs to the SGAT model are $K+1$ different sets of features, $\left\{H_{\chi^0}, H_{\chi^1}, \ldots, H_{\chi^K}\right\}$, where

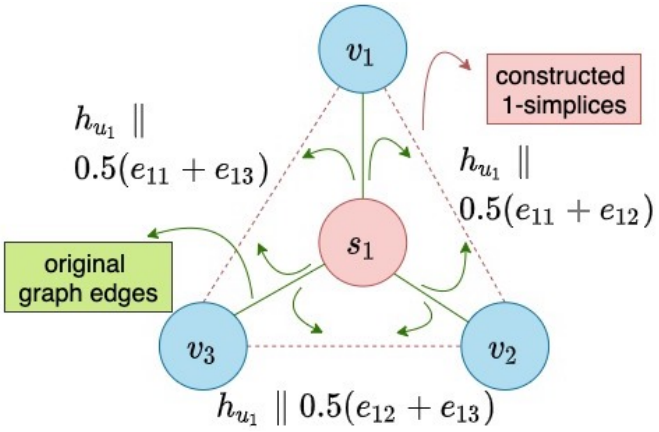


Figure 4.2: Combining edge features with features of 1-simplices to enrich features of 1-simplices for $\eta = 1$.

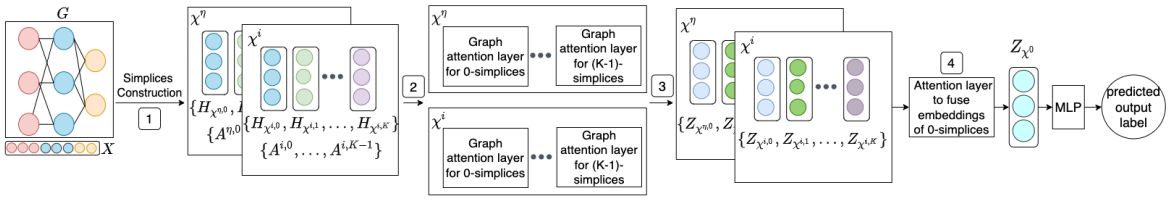


Figure 4.3: The overall architecture of SGAT (self-loops are omitted for clarity). (1) Given the heterogeneous graph G and node features X as inputs, we first construct K simplices for each of the η simplicial complex(es) and the simplices' respective features. For each simplicial complex χ^i, (2) its associated simplices, features and upper adjacency matrices are fed into a simplicial attention layer which is made up of K graph attention layers. Then, (3) η sets of simplicial complex specific embeddings are generated. For node classification task, (4) we fuse the η sets of 0-simplices' embeddings using attention to obtain the final embedding which is then fed into a layer of MLP (the classifier).

$H_{\chi^k} = \left\{h_{\sigma_1^k}, \ldots, h_{\sigma^k_{|\chi^k|}}\right\}$, $h_{\sigma_i^k}$ is the feature vector of the i-th k-simplex, $i = 1, \ldots, |\chi^k|$. Moreover, each of the $K+1$ feature sets is associated with an upper adjacency matrix, A^k.

We employ a simplicial graph attention layer to compute the importance of the neighboring simplices and update the various simplices' features. A simplicial graph attention layer is made up of K graph attention layers, one for each of the k-order simplices, $0 \leq k < K$, excluding the K-order. This captures the interactions between higher-order structures of the same order by passing messages between themselves through upper adjacencies. For each of the K graph attention layers, given a k-simplex pair σ_i^k, σ_j^k, the unnormalized attention score between the two simplicies is defined as

$$\zeta_{ij}^{k} = \text{LeakyReLU}\Big((\mathbf{a}^{k})^{\intercal}\Big[\mathbf{W}^{k}h_{\sigma_i^k} \Big\| \mathbf{W}^{k}h_{\sigma_j^k} \Big\| \mathbf{W}^{k+1}h_{\sigma_{ij}^{k+1}}\Big]\Big), \tag{4.4}$$

where $\|$ represents concatenation, $^{\intercal}$ denotes transposition, $\mathbf{a}^k$ is the learnable attention vector and $\mathbf{W}^k$ is the weight matrix for a linear transformation specific to the k-order and are shared by all the k-simplices. The feature vector $h_{\sigma_{ij}^{k+1}}$ belongs to the $k+1$-simplex that both σ_i^k and σ_j^k are faces of.

The normalized attention score for each of the k-simplices can then be obtained by applying the softmax function as given by:

$$\alpha_{ij}^{k} = \frac{\exp(\zeta_{ij}^{k})}{\sum_{r\in N(\sigma_i^k)}\exp(\zeta_{ir}^{k})}, \tag{4.5}$$

where $N(\sigma_i^k)$ refers to the neighborhood of σ_i^k (cf. (4.1)). The weight coefficients learned can then be applied along with the corresponding neighbors' features to update the feature for each of the k-simplices as given by

$$z_{\sigma_i^k} = \mu\left(\sum_{j\in N(\sigma_i^k)} \alpha_{ij}^{k}\mathbf{W}^{k}h_{\sigma_j^k}\right), \tag{4.6}$$

where $\mu(\cdot)$ denotes an activation function. To stabilise the training, we employ multi-head attention where P independent attention mechanisms are performed. The results are then concatenated to generate the required learned simplex features:

$$z_{\sigma_i^k} = \Big\|_{p=1}^{P} \mu\left(\sum_{j\in N(\sigma_i^k)} \alpha_{ij}^{k,p}\mathbf{W}^{k,p}h_{\sigma_j^k}\right), \tag{4.7}$$

where $\alpha_{ij}^{k,p}$ and $\mathbf{W}^{k,p}$ are the attention and transformation weights corresponding to α_{ij}^{k} and $\mathbf{W}^{k}$, respectively, in (4.6) for the p-th attention head.

The output for the k-order simplices χ^k is a set of simplicial complex specific embeddings, $Z_{\chi^k} = \{z_{\sigma_1^k}, \ldots, z_{\sigma_{|\chi^k|}^k}\}$. Since we have K graph attention layers, there are K sets of simplicial complex specific embeddings $Z_{\chi^0}, \ldots, Z_{\chi^{K-1}}$. Depending on the downstream task, not all of the embeddings are utilized. For instance, in the last softmax layer for

node classification tasks, only the Z_{χ^0} embeddings are employed.

4.2.3 Attention to Fuse Simplicial Complexes

Recall that for $\eta_{\max} > 1$ and for each $\eta = 1, \ldots, \eta_{\max}$, we construct k-simplices for $0 \leq k \leq K$. Applying the simplicial attention layer in Section 4.2.2 to each of the η number of K dimensional simplicial complexes, we obtain the embeddings $Z_{\chi^{0,\eta}}, \ldots, Z_{\chi^{K-1,\eta}}$ for each η-specific simplicial complex. In our framework, a k-simplex σ^k is a unique k-tuple that is independent of η. We arbitrarily order all k-simplices as $\sigma_1^k, \sigma_2^k, \ldots, \sigma_\tau^k$, where $\tau = \max_\eta |\chi^{k,\eta}|$. The collection of embeddings for $\chi^{k,\eta}$ can then be written as $Z_{\chi^{k,\eta}} = \{z^\eta_{\sigma_1^k}, \ldots, z^\eta_{\sigma_\tau^k}\}$ where $z^\eta_{\sigma_i^k}$ is the feature vector of the i-th k-simplex for the η-specific simplicial complex, which is taken to be null if $\sigma_i^k \notin \chi^{k,\eta}$.

In this subsection, we propose an attention layer to account for the importance of different simplicial complexes in describing the respective k-dimensional simplices. An attention layer is required to fuse the η groups of k-simplex embeddings.

The unnormalized attention score to fuse the η-specific embeddings of k-simplices is defined as follows:

$$w^{k,\eta} = \frac{1}{|\chi^{k,\eta}|} \sum_{i=1}^{\tau} (\mathbf{q}^{k,\eta})^\intercal \tanh(\mathbf{F}^{k,\eta} z^\eta_{\sigma_i^k} + \mathbf{b}^{k,\eta}) \mathbf{1}\{\sigma_i^k \in \chi^{k,\eta}\}, \tag{4.8}$$

where $\mathbf{q}^{k,\eta}$ is the learnable attention vector, $\mathbf{F}^{k,\eta}$ is a learnable weight matrix and $\mathbf{b}^{k,\eta}$ is a learnable bias vector. Here, $\mathbf{1}\{A\}$ is the indicator function, which takes value 1 if A is true and 0 otherwise. The normalized weight coefficients can then be attained through the softmax function as

$$\beta^{k,\eta} = \frac{\exp(w^{k,\eta})}{\sum_{j=1}^{\eta_{\max}} \exp(w^{k,j})}. \tag{4.9}$$

We fuse the features $z^\eta_{\sigma_i^k}$ across $\eta = 1, \ldots, \eta_{\max}$ by a linear combination using the

learned weights to generate the final embedding of σ_i^k as

$$z_{\sigma_i^k} = \sum_{\eta=1}^{\eta_{\max}} \beta^{k,\eta} z_{\sigma_i^k}^{\eta} \mathbf{1}\{\sigma_i^k \in \chi^{k,\eta}\}. \tag{4.10}$$

Let $Z_{\chi^k} = \left\{z_{\sigma_1^k}, \ldots, z_{\sigma_\tau^k}\right\}$ be the embedding of $\chi^k = \{\sigma_1^k, \ldots, \sigma_\tau^k\}$.

Lastly, when L layers are utilized, let $Z_{\chi^0}^{(l)}$ be the embedding of χ^0 in the l-th layer, for $l = 1, \ldots, L$. We concatenate the learned 0-simplex features of each layer at the last layer to obtain

$$Z_{\chi^0}^{(L)} = \Big\Vert_{l=1}^{L} Z_{\chi^0}^{(l)}, \tag{4.11}$$

and feed it into a linear layer for our semi-supervised, node classification task along with cross-entropy loss. Nonetheless, we note that the final learned features, $Z_{\chi^0}^{(L)}$ can be used for other downstream tasks similarly optimising the model via backpropagation. The overall architecture of SGAT is as shown in Fig. 4.3.

4.3 Numerical Experiments

In this section, we verify the empirical performance of our proposed method against several state-of-the-art methods on node classification task for heterogenous graphs.

4.3.1 Datasets

The heterogeneous datasets utilized are DBLP, ACM, and IMDB. We have provided details on the datasets in Section 2.7.3. Since these heterogeneous datasets do not consist of edge features, when edge features are required, we form h_e for each $e \in E$ of the respective datasets by concatenating its starting node feature, ending node feature and a one-hot encoding of its edge-type.

4.3.2 Baselines and Settings

We compare SGAT and SGAT-EF with six state-of-the-art GNN models. For all models, the hidden units are set to 64, the Adam optimizer was used and its hyperparameters such as learning rate and weight decay, are respectively chosen to yield best performance. When metapaths are required, the metapaths employed in [81] are utilized. As for metagraphs, the relevant metapaths and 3-node metagraphs as in [85] are used.

For SGAT, we set K, the dimension of the simplicial complexes to be 2, the number of layers to be 2 for all the datasets. Moreover, when $\eta >= 2$, the dimension of the attention vector $\mathbf{q}^{k,\eta}$ (cf. (4.8)) is set to 128. Besides the parameters mentioned above, ϵ_η^k, η and λ are tuned for each dataset. Specifically, for ACM, we choose $\eta = 1$, $\epsilon_1^1 = 1$, and $\lambda = 20$. For DBLP, $\eta = 2$, $\epsilon_1^1 = 3$, $\epsilon_2^1 = 4$, and $\lambda = 10$. For IMDB, $\eta = 1$, $\epsilon_1^1 = 1$, and $\lambda = 10$.

Table 4.1: Node classification result on heterogeneous datasets (Standard split). Averaged over five runs, best performance boldfaced.

Datasets	Metrics	GCN	GAT	Meta-GNN	HAN	REGATHER	GTN	SGAT	SGAT-EF
DBLP	Macro-F1	87.65 ± 0.29	91.69 ± 0.27	92.47 ± 0.92	91.93 ± 0.27	91.79 ± 0.69	93.59 ± 0.40	**93.80 ± 0.20**	93.73 ± 0.23
	Micro-F1	88.71 ± 2.74	92.65 ± 0.25	93.48 ± 0.75	92.51 ± 0.24	92.70 ± 0.67	94.17 ± 0.26	**94.58 ±0.20**	94.51 ± 0.19
ACM	Macro-F1	91.46 ± 0.48	92.16 ± 0.32	88.74 ± 0.99	91.01 ± 0.76	92.38 ± 0.57	92.23 ± 0.60	92.41 ± 0.36	**92.91 ± 1.79**
	Micro-F1	91.33 ± 0.47	92.06 ± 0.33	88.71 ± 1.01	90.93 ± 0.73	92.30 ± 0.56	92.12 ± 0.62	92.35 ± 0.36	**92.86 ± 1.75**
IMDB	Macro-F1	56.72 ± 0.49	57.32 ± 0.88	56.19 ± 0.97	56.56 ± 0.77	56.34 ± 0.54	59.12 ± 1.58	59.97 ± 0.41	**60.36 ± 0.53**
	Micro-F1	58.31 ± 0.51	58.75 ± 0.98	58.68 ± 1.49	57.83 ± 0.93	57.59 ± 0.64	60.58 ± 2.10	62.51 ± 0.64	**62.74 ± 0.77**

4.3.3 Node Classification Performance

Table 4.1 shows the performances of SGAT, SGAT-EF and other node classification baselines. Our empirical results demonstrate that SGAT and SGAT-EF outperform the baselines for node classification task. We observe that SGAT performs better than GTN, HAN and REGATHER. This demonstrates that nonlinear structures encoded by the simplicial complexes are useful for learning more effective node representations and more complicated relationships, which may not be possible using linear structures such as metapaths.

4.4 Model Analysis

4.4.1 Ablation Study

We conduct experiments on different SGAT variants to validate the effectiveness of the components of our model. SGAT considers message passing between higher-order simplices without incorporating edge features while SGAT-EF is the equivalent model utilizing edge features. Moreover, since SGAT generalizes to the GAT model when $K = 1$, the GAT model can also be taken as an ablation study of SGAT without message passing on higher order structures. As observed in Table 4.1, by employing message passing on higher-order simplices, SGAT obtained a significant improvement over GAT. We also observe that SGAT-EF frequently performs better than SGAT. This show that our method incorporating edge features may bring benefits when the task of interest is edge dependent or when the dataset has meaningful edge features.

4.4.2 Parameter Sensitivity (for λ and ϵ_η^k)

Given that SGAT involves several parameters to control the construction of simplices, we further examine how λ (the maximum simplex order to construct, including their faces. The simplex order refers to its dimension) and ϵ_η^k (the minimum number of shared non-target nodes that is k and η-specific) affect the performance of SGAT on the DBLP dataset. We first keep all the constructed edges by setting $\epsilon_1^1 = \epsilon_2^1 = 1$ and measure the ratio of constructed triangles to constructed edges, γ as a function of λ. The Micro-F1 and Macro-F1 scores are then obtained as a function of γ as seen in Fig. 4.4. We observe that an optimal γ where the information included from the amount of simplices considered is most beneficial and does not negatively affect the model's performance. Increasing λ beyond the resulting optimal γ is likely to introduce noise by including unnecessary messages being passed between the simplices.

We also analyse how ϵ_η^k affect SGAT's performance by setting λ to be fixed at 10 and the default values of $\epsilon_1^1 = 3$ and $\epsilon_2^1 = 4$. Besides the parameter being tested, the other parameters assume their default values. When ϵ_1^1 and ϵ_2^1 are increased, the number of

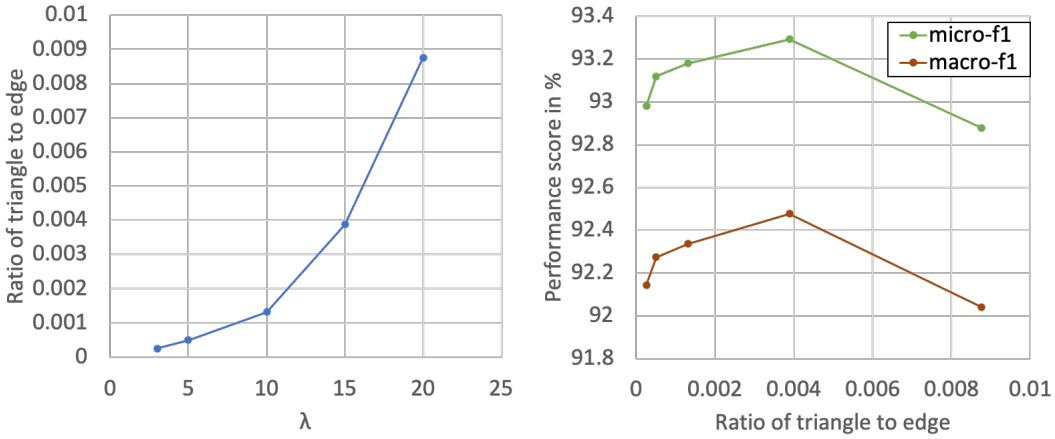


Figure 4.4: λ against γ (left) and γ against performance score (right)

constructed edges decreases. We observe that increasing ϵ_1^1 improves the performance until the default value of $\epsilon_1^1 = 3$ is reached. After which, the performance deteriorates. This phenomenon is similar for ϵ_2^1. Increasing ϵ_η^k reduces noise in the model as it removes weaker edges (those that shared fewer nodes).

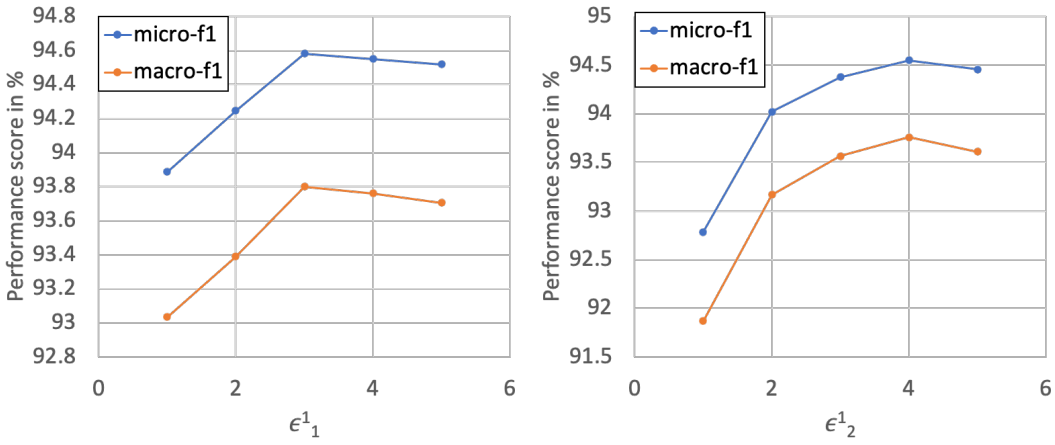


Figure 4.5: ϵ_1^1 against γ (left) and ϵ_2^1 against γ (right)

Lastly, we examine how different combinations of ϵ_1^1 and ϵ_2^1, each chosen in the range of $[1, 5]$, affect SGAT's performance. Interestingly, we found that the performance of SGAT fluctuates minimally across different values of these parameters when the ratio of triangles to edges, γ is similar.

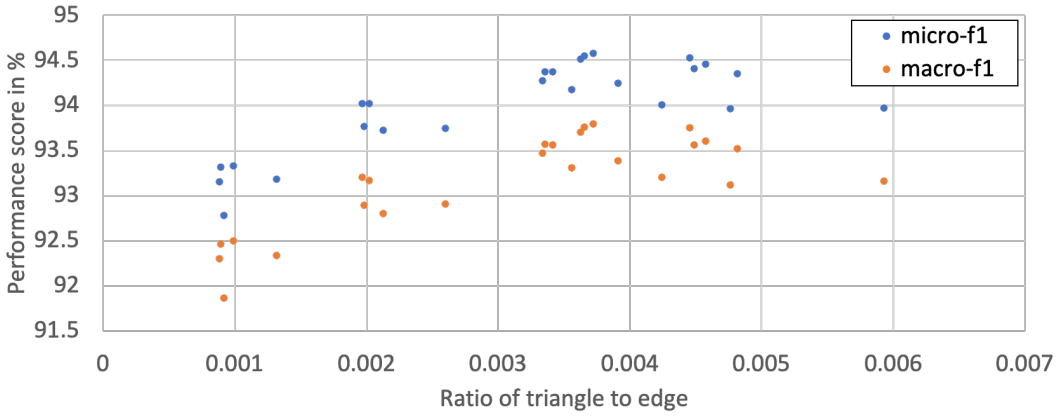


Figure 4.6: γ against performance score

4.4.3 Meta-GNN, SGAT and SGAT-EF

To evaluate the advantage of utilising simplicial complexes over metagraphs, we conduct experiments to compare Meta-GNN, SGAT and SGAT-EF. The results are shown in Fig. 4.7. We can clearly observe that SGAT and SGAT-EF, which performs message passing between simplices, consistently outperforms Meta-GNN. Meta-GNN uses metagraphs, which despite being a subgraph pattern, solely represents relationships between *two* target nodes and not *multiple* target nodes as in the case of simplices. The result demonstrates that metagraphs cannot adequately represent high-order interactions between multiple target nodes.

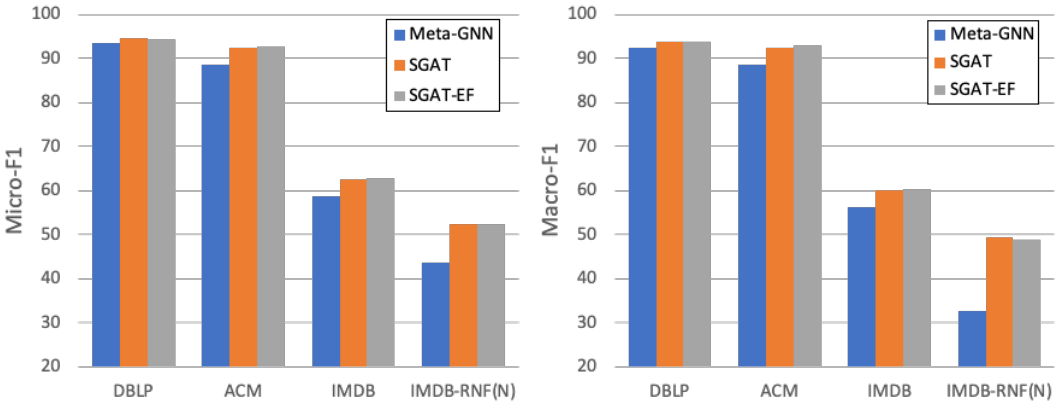


Figure 4.7: Node classification F1 scores (%) for Meta-GNN, SGAT and SGAT-EF

4.4.4 Random Node Features

A survey [117] has shown that node features of benchmark graphs usually contain substantial information, allowing models to achieve close to optimal results. Hence, we prepare an additional dataset to classify nodes solely based upon graph structure. We replace node features of IMDB dataset by random ones sampled from standard normal distribution, naming it IMDB-RNF(N). The random node features are the same size as the replaced node features. This leaves only the graph structure for the classification task and thus, we can utilise the resulting performance to interpret the models' capability in extracting structural information. This approach is also used in [118]. Note that for IMDB-RNF(N) dataset, the models are observed to be highly sensitive, thus we re-tuned all the models to ensure a fair comparison. Table 4.2 depicts the results on such dataset.

Table 4.2: Node classification result on heterogeneous dataset with random node features, best performance boldfaced.

	IMDB-RNF(N)	
Methods	Macro-F1	Micro-F1
GAT	36.45 ± 2.03	37.94 ± 2.07
GCN	40.10 ± 1.09	41.84 ± 1.24
Meta-GNN	32.58 ± 1.62	43.75 ± 5.62
HAN	39.47 ± 1.03	40.97 ± 1.13
REGATHER	28.60 ± 3.45	42.62 ± 8.10
GTN	33.71 ± 0.51	36.52 ± 1.38
SGAT	**49.29 ± 1.34**	52.34 ± 1.81
SGAT-EF	48.90 ± 1.00	**52.37 ± 1.09**

We observe clear advantage of SGAT and SGAT-EF over their comparison partners. For instance, SGAT and SGAT-EF surpass the baselines by up to approximately 10% F1 scores on the IMDB-RNF(N) dataset. In addition, we find that the performance of all models decreased when the bag-of-words node features are replaced with uninformative ones. This shows that the original node features indeed contain useful information for the task, aiding all models to garner almost optimal results. Subsequently, it also implies that our models can effectively extract and utilize structural information from graphs instead of relying on the original node features that may already contain important information.

4.5 Conclusion

In this chapter, we introduced SGAT, a novel extension of GAT to heterogeneous graphs using simplicial complexes and upper adjacencies. SGAT does not utilize metapaths or metagraphs. Hence, SGAT is not subjected to the three characteristic limitations of metapath-based methods, namely (1) performance being sensitive to the choice of metapaths, (2) discarding non-target node features and (3) limited expressiveness when using structures between two nodes to capture complex interactions. Specifically, we utilize simplicial complex to capture high-order relations among multiple target nodes. We avoid discarding non-target node features when transforming the graph into homogeneous simplicies by placing those features on simplices. We also introduced a variant incorporating edge features that can boost embedding performance. Empirically, we demonstrated that SGAT performs favorably on node classification tasks for heterogeneous datasets and even when random node features were employed.

This method constructs simplices and incorporates upper adjacencies to capture high-order interactions within a graph, providing an alternative to the conventional metapath/metagraph construct. However, similar to metapath-based techniques, we deviate from utilizing the original graph structure represented by the given adjacency matrix. This illustrates that the provided graph is often inadequate to model the underlying dynamics fully and therefore fails to effectively provide essential information for the task.

Furthermore, note that for heterogeneous graphs, multiple graphs are often being utilized to train a model. Each of the graphs represents a different metapath-based graph structure or, in the case of SGAT, a different simplicial complex. These graphs are typically simultaneously fed into the model, and attention mechanisms are used to assign weights to the graph-specific embeddings. However, this process results in high memory and computational demands.

Building upon the insight that a collection of graphs can collectively enhance task performance compared to a single graph, our upcoming chapter introduces an approach to leverage multiple graphs with the intention of mitigating high computational costs. This is achieved through a probabilistic framework, where the underlying distribution

of multiple graphs is learned and utilized to weigh graph-specific embeddings during inference. To optimize the model, we employ an adapted stochastic gradient descent (SGD) approach, utilizing a shared model across multiple graphs to reduce trainable parameters and computational requirements. Depending on how the multiple graphs are generated, this approach implicitly incorporates geometric information into the model. Finally, by exposing the model to diverse graphs, each featuring different geometric properties and/or topologies, and learning their distribution, we can indirectly identify and leverage the "useful" graphs (from a candidate set) for learning.

Chapter 5

Graph Neural Networks with a Distribution of Parametrized Graphs

Traditionally, GNNs have been trained on a single observed graph, including the one presented in Chapter 3. However, relying on just one graph overlooks the variability present in real-world data. By tapping into a collection of graphs with varying topological or geometric characteristics, we can enhance learning and improve the quality of embeddings. Each graph within this collection has the potential to offer distinct and valuable information, contributing to a more comprehensive understanding of the underlying data. Consequently, we transition from a deterministic framework to a probabilistic one.

In this chapter, we introduce an EM framework for graph learning. The framework's objective is to find network parameters that maximize the likelihood of joint distribution between the observed data and the latent variables, which are introduced to generate parameterized graphs. In essence, these latent variables give rise to a collection of parameterized graphs, representing plausible graph topologies and geometries that may emerge from an unknown distribution. Our framework does not assume a statistical model when it comes to handling the unknown distribution such as "being Gaussian". Instead, we adopt Markov Chain Monte Carlo (MCMC) and PAC-Bayesian principles to estimate the distribution of these latent parameters in the E-step. This approach allows for a more flexible and data-driven treatment of variability in graph data. Numerical experiments

demonstrate improvements in performance against baseline models on node classification for heterogeneous graphs and homogeneous graphs and graph regression on chemistry datasets.

To better illustrate our proposed framework, an aspect of our work revolves around tackling the uncertainty inherent in graph structures. This focus is motivated by various studies [40, 119], which highlighted that the provided graph may not always be optimal. For instance, techniques such as graph rewiring and edge weight adjustments have been employed to enhance graph structures and improve learned representations. Additionally, in heterogeneous graph settings, the use of constructs like metapaths for learning has outperformed the original graph structure, implying that the provided graph is suboptimal for learning. Rather than attempting to modify the observed graph directly to optimize its structure, our strategy involves learning the underlying distribution of the observed graph and generating additional instances from that distribution. By doing so, we can tap into the richness of diverse graph collections, thereby enhancing the learning process.

5.1 Introduction

GNNs have facilitated graph representational learning by building upon Graph Signal Processing (GSP) principles and expanding their application in the domains of machine learning. However, GSP and GNNs conventionally rely on a fixed graph shift operator, such as the adjacency or Laplacian matrix, to analyze and learn from graph data, assuming that the given graph is accurate and noise-free. This approach has inherent limitations, considering that graph data is often uncertain.

The uncertainty is due to the existence of multiple potential variations in graph constructions as a universal optimal method does not exist. Furthermore, structural noise, which includes missing or spurious edges, and the absence of informative edge weights, can also contribute to the uncertainty in graph data [119, 120]. It is important to handle this uncertainty as the graph directly influences the results of both GSP and GNNs [121].

Several GNN works have recognized that the provided graph in benchmark datasets

is suboptimal. For example, in [40], a method was introduced to enhance the provided graph by rewiring it at graph bottlenecks. Similarly, in [66] and [67], approaches were developed to reweigh edges, to reduce information flow at cluster boundaries. Another perspective involves considering the given or observed graph as a particular realization of a graph model, as discussed in [119]. In their work, a Bayesian framework was adopted to learn a more robust model that can withstand perturbations in graph topology. These collective efforts underscore the common observation that the observed graph is often imperfect, and determining the optimal graph is a non-trivial task, as it depends on both the physical connections and the edge weights, which regulate the rates of information transmission [122].

Our work aligns with the viewpoint presented in [119]. We conceptualize the observed graph as an individual instance originating from a distribution of graphs, which is influenced by one or more latent parameters. Nevertheless, in contrast to [119] which proposed a Bayesian framework, we propose an EM framework for graph learning and name our model EMGNN. Even though both are probabilistic frameworks, the focus is distinctly different. In the case of the Bayesian framework of [119], the focus is on estimating the posterior distribution of model parameters given the data. As such, model parameters are deemed as random variables trained by a series of characteristically similar graphs. Meanwhile, in our EM framework, we seek to maximize the log-likelihood of the observed data in conjunction with the latent variables. Additionally, we permit the generated graphs to demonstrate more pronounced variations. Our main contributions are:

- We present a general framework for modeling the distribution of graphs to handle uncertainty in graph data. The learned distribution provides valuable insights into our model's behavior.

- We formulate the graph learning problem as a maximum likelihood estimation (MLE) so that tools from statistical learning can be applied. The new objective subsumes the classical objective of minimizing empirical loss if the graph is deterministic.

- We evaluate our model on nine datasets in two distinct applications, and observe

promising performance compared to the respective baseline methods.

- We inspect the learned graph distribution, confirming that it effectively captures the intricacies of heterogeneous graph datasets, thus validating the utility of our model and framework.

The remainder of this chapter is organized as follows: firstly, we offer an overview of how a distribution of graphs can be utilized in the GSP context, laying the foundation for the design of our model. Subsequently, we introduce our proposed method in Section 5.4, followed by a detailed account of the experiments conducted to assess the effectiveness of our model in Section 5.5. Lastly, we draw our conclusions in Section 5.6.

5.2 Signal Processing over a Distribution of Graphs

GNN is closely related to the theory of GSP [123]. Briefly, given an undirected graph G, we consider a fixed graph shift operator S such as its adjacency or Laplacian matrix. A graph signal is a vector $\mathbf{x} = (x_v)_{v \in V}$ that associates a number x_v to each node $v \in V$. Intuitively, applying the linear transformation S to $\mathbf{x}$ is considered as a "shift" of $\mathbf{x}$. If S is the normalized adjacency matrix, then it amounts to the AGGR step of (2.1) for GCN. More generally, if $P(\cdot)$ is a single variable polynomial, then plugging in S results in the matrix $P(S)$, which is called a *convolution filter* in GSP. This notion of convolution appears in [124], and has become widely used since then.

On the signal processing side, [122] has developed a theory that generalizes traditional GSP. The authors propose a signal processing framework assuming a family of graphs are considered simultaneously, to tackle uncertainties in graph constructions. Formally, it considers a distribution μ of graph shift operators S_λ parametrized by λ in a sample space Λ. The work develops corresponding signal processing notions such as Fourier transform, filtering, and sampling. In particular, a convolution takes the form $\mathbb{E}_{\lambda \sim \mu}[P_\lambda(S_\lambda)]$, where $P_\lambda(\cdot)$ is a polynomial and $P_\lambda(S_\lambda)$ is an ordinary convolution with shift S_λ. Our work is based on a similar idea but replaces $P_\lambda(S_\lambda)$ with a more general filter. Furthermore, we introduce an EM framework that is not present in [122] to learn the distribution of graphs.

A PAC-Bayesian framework [125] can be used to learn the distribution μ on Λ. Suppose we have a risk function $R : \Lambda \to \mathbb{R}$. Then we let the density of λ be proportional to the Gibbs posterior $\exp(-\eta R(S_\lambda))p_0(S_\lambda)$, where η is a tunable hyperparameter and $p_0(\cdot)$ is the density of a prior distribution.

5.3 Problem Formulation

5.3.1 Distributions for Different Graph Types

Here, we outline how Λ, a parameter sample space, arises for different graph types, showcasing why the proposed framework is useful for graph-related tasks. Parameters $\lambda \in \Lambda$ can be scalars, vectors, or more general forms, enabling task-specific graph parameterization and diverse graph generation. For instance, a specific λ can indicate the edge weight for an edge type in heterogeneous graphs and be employed to create varied weighted graphs. Details are task-specific and provided in Section 5.5. Though the schemes are simple and intuitive, there may be alternatives for Λ based on other factors.

Heterogeneous graphs are graph structures characterized by the presence of multiple node types and multiple edge types, imparting a greater degree of complexity compared to *homogeneous graphs*, which consist of a single node and edge type. For a heterogeneous graph, the insight is that we assign a parameter to each edge type, whose distribution is to be estimated and used. Intuitively, in a model based on message passing, the parameters for different edge types are interpreted as different information transmission rates. Such information is not observed in the given graph.

It is less obvious how Λ can be constructed for a homogenous graph. Intuitively, we assume that the observed graph contains "noisy" edges and has missing edges (cf. Zhang et al. [119]). Therefore, parameters that are interpreted as probabilities for adding and removing edges from the observed graph, can be introduced (see an example in Fig. 5.1). Though we have different setups, a unified framework to deal with both is proposed in the next section.

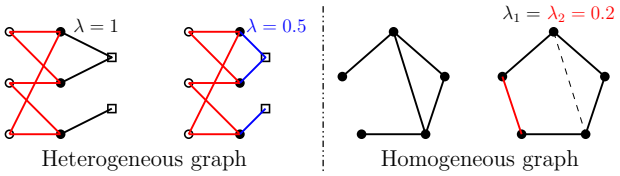


Figure 5.1: For a heterogenous graph, we may use a parameter λ to control the information transmission rate for each edge type. For example, choosing $\lambda = 1$ or $\lambda = 0.5$ for the edge type between "disc" and "square" nodes yields different weighted graphs. Conversely, in a homogeneous example with 5 initial edges, by choosing $\lambda_1 = \lambda_2 = 0.2$, 20% of the initial and missing edges are randomly removed and added, potentially forming a "pentagon".

5.3.2 Maximum Likelihood Estimation

Motivated by the previous subsection, we consider a distribution μ on a parameter (sample) space $\Lambda \subset \mathbb{R}^r$ of graphs $\{G_\lambda, \lambda \in \Lambda\}$, with a fixed set of nodes V. The space Λ can be finite, countably infinite, or even uncountable. For each G_λ, there is a corresponding shift operator S_λ. We usually assume that μ has a density function $p(\cdot)$ w.r.t. a base measure on Λ. For example, if Λ is finite, we can use the discrete counting measure as the base measure. On the other hand, if Λ is a compact interval in $\mathbb{R}$, then we can choose the Lebesgue measure as the base measure.

Assume that each node $v \in V$ is associated with features x_v. They are collectively denoted by $\mathbf{x}$. Our framework depends on a *fixed GNN model architecture* Ψ, e.g., GCN. It outputs the learned embeddings $\mathbf{z} = \Psi(\lambda, \mathbf{x}; \boldsymbol{\theta})$ given the node features $\mathbf{x}$, the graph parameter λ, and the GNN model parameters $\boldsymbol{\theta}$. These in turn are used to determine a vector of labels $\hat{\mathbf{y}}$. For a task-specific loss $\ell(\cdot, \cdot)$ that compares predicted $\hat{\mathbf{y}}$ and true label (vector) $\mathbf{y}$, we may compute $L_{\mathbf{X}}(\lambda, \boldsymbol{\theta}) = \ell(\hat{\mathbf{y}}, \mathbf{y})$. We use $\mathbf{X}$ to denote the full information $\{\mathbf{x}, \mathbf{y}\}$. We interpret $\mathbf{X}$ as a sample from a random variable, denoted by $\mathfrak{X}$, of collective information of features and labels.

An example of Ψ is the model described by (2.1). We may also allow parameters $\boldsymbol{\theta}$ and λ to determine W^k. For example, if $\boldsymbol{\theta} = \{\theta_1^k, \theta_2^k \mid 1 \le k \le K\}$ and λ is a scalar parameter, then one can choose a linear combination $W^k = \theta_1^k + \lambda\theta_2^k$. Moreover, AGGR is determined by the shift S_λ associated with G_λ.

In general, as λ follows an unknown distribution μ, it is hard to find the optimal $\boldsymbol{\theta}$ by minimizing $\mathbb{E}_{\lambda\sim\mu}[L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})]$ directly. On the other hand, the EM algorithm [126] enables

the joint estimation of μ and $\boldsymbol{\theta}$ if we can reformulate the objective as an MLE.

To minimize the loss given $\mathbf{X}$, the parameter $\boldsymbol{\theta}$ is determined by λ and vice versa. Therefore, $\Psi(\cdot, \mathbf{x}; \cdot)$ becomes a random GNN model that depends on $\lambda, \boldsymbol{\theta}$ and input $\mathbf{x}$. We aim to *identify a realization of the random models that makes the observation* $\mathbf{X}$ *likely*, i.e., there is less discrepancy between the estimator labels $\hat{\mathbf{y}}$ and ground truth labels $\mathbf{y}$ measured by the loss $\ell(\hat{\mathbf{y}}, \mathbf{y})$. Motivated by the discussions above, we consider the likelihood function $p(\lambda, \mathbf{X} \mid \boldsymbol{\theta})$ on $\boldsymbol{\theta}$ and formulate the following MLE as our objective:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} p(\mathbf{X} \mid \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\lambda \sim \mu}[p(\lambda, \mathbf{X} \mid \boldsymbol{\theta})]. \tag{5.1}$$

Before discussing the main algorithm in subsequent subsections, we preview the roles of μ and $L_{\mathbf{X}}(\cdot, \cdot)$ in the algorithm. We shall see that the EM algorithm outputs a distribution $\hat{\mu}$ of λ, serving as an estimate of μ, by leveraging the PAC-Bayesian framework [125]. In this framework, the density of λ is proportional to the Gibbs posterior, depending on $\ell(\cdot, \cdot)$. Consequently, $\hat{\mu}$ assigns higher probability density to λ when the loss $L_{\mathbf{X}}(\lambda, \boldsymbol{\theta}^*)$ is lower. Therefore, we are minimizing the given loss as a main component of the algorithm. In the above formulation, $\boldsymbol{\theta}$ is the parameter and μ is unknown before attaining $\boldsymbol{\theta}$. However, once $\boldsymbol{\theta}$ is determined, an estimation of μ is obtained using the Gibbs posterior.

Example 1. *Assume that μ is the delta distribution δ_{λ_0} supported on λ_0, so that the graph G_{λ_0} is deterministic. If we consider the Gibbs posterior, then we have $p(\mathbf{X} \mid \boldsymbol{\theta}) \propto \exp(-\eta \ell(\hat{\mathbf{y}}, \mathbf{y}))$ for a hyperparameter η, where $\hat{\mathbf{y}}$ depends on both $\mathbf{X} = \{\mathbf{x}, \mathbf{y}\}$ and $\boldsymbol{\theta}$. Thus, maximizing $p(\mathbf{X} \mid \boldsymbol{\theta})$ is equivalent to the classical objective of minimizing $\ell(\hat{\mathbf{y}}, \mathbf{y})$.*

5.4 Proposed Method

5.4.1 Expectation-Maximization for GNN

Optimizing (5.1) directly can be challenging, and we utilize the EM algorithm that employs an iterative approach alternating between the E-step and the M-step. Adapted to our setting, the process unfolds as follows:

(a) E-step: Given parameters $\boldsymbol{\theta}^{(t)}$ at the t-th iteration, we compute the expectation as the Q-function

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\lambda \sim p(\cdot \mid \mathbf{X}, \boldsymbol{\theta}^{(t)})}[\log p(\lambda, \mathbf{X} \mid \boldsymbol{\theta})].$$

(b) M-step: $\boldsymbol{\theta}^{(t+1)}$ is updated as $\arg\max Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$.

For the E-step in the t-th iteration, in the same spirit as the PAC-Bayesian framework [125], we apply the Gibbs posterior and assume that

$$p(\lambda, \mathbf{X} \mid \boldsymbol{\theta}) \propto \exp(-\eta^{(t)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta}))\pi_0(\lambda, \mathbf{X}), \tag{5.2}$$

for a tunable hyperparameter $\eta^{(t)}$, while $\pi_0(\cdot)$ is a prior density of the joint $(\lambda, \mathbf{X})$ independent of $\boldsymbol{\theta}$, representing our initial knowledge regarding λ and $\mathbf{X}$. In this expression, $L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})$ implicitly depends on the observations $\mathbf{X}$.

The normalization constant is given by

$$\begin{aligned} C(\boldsymbol{\theta}) &= \int_{(\lambda, \mathbf{X}') \in \Lambda \times \mathfrak{X}} \exp(-\eta^{(t)} L_{\mathbf{X}'}(\lambda, \boldsymbol{\theta}))\pi_0(\lambda, \mathbf{X}') \, \mathrm{d}(\lambda, \mathbf{X}') \\ &= \mathbb{E}_{(\lambda, \mathbf{X}') \sim \pi_0}\big[\exp(-\eta^{(t)} L_{\mathbf{X}'}(\lambda, \boldsymbol{\theta}))\big]. \end{aligned} \tag{5.3}$$

As a prior belief, we treat the observed $\mathbf{X}$ as a typical sample such that the above average is (approximately) the same as the average over graphs by fixing $\mathbf{X}$. We assume that for each fixed $\mathbf{X}$, there exists some prior distribution with density $p_{0,\mathbf{X}}(\cdot)$ on Λ such that:

$$\begin{aligned} &\mathbb{E}_{(\lambda, \mathbf{X}') \sim \pi_0}\big[\exp(-\eta^{(t)} L_{\mathbf{X}'}(\lambda, \boldsymbol{\theta}))\big] \\ &\approx \int_{\lambda \in \Lambda} \exp(-\eta^{(t)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta}))p_{0,\mathbf{X}}(\lambda) \, \mathrm{d}(\lambda) \\ &= \mathbb{E}_{\lambda \sim p_{0,\mathbf{X}}}\big[\exp(-\eta^{(t)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})) \,\big|\, \mathbf{X}\big]. \end{aligned}$$

For simplification, we use the notation $p_{0,\mathbf{X}}(\cdot)$ to represent $p_0(\cdot)$ and $\mathbb{E}_{\lambda \sim p_0}\big[\exp(-\eta^{(t)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})) \,\big|\, \mathbf{X}\big]$

to represent $\mathbb{E}_{\lambda\sim p_0}\big[\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\big]$. We write

$$C(\boldsymbol{\theta}) = \mathbb{E}_{\lambda\sim p_0}\big[\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\big]. \tag{5.4}$$

On the other hand, given $\boldsymbol{\theta}^{(t)}$, from (5.2), we have

$$\begin{aligned} p(\lambda \mid \mathbf{X},\boldsymbol{\theta}^{(t)}) &= \frac{p(\lambda,\mathbf{X}\mid\boldsymbol{\theta}^{(t)})}{p(\mathbf{X}\mid\boldsymbol{\theta}^{(t)})} \\ &\propto \exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}^{(t)}))\frac{\pi_0(\lambda,\mathbf{X})}{p(\mathbf{X}\mid\boldsymbol{\theta}^{(t)})}. \end{aligned}$$

We assume that there is a prior $p'_{0,t}(\cdot)$ such that $p'_{0,t}(\cdot)\propto\frac{\pi_0(\lambda,\mathbf{X})}{p(\mathbf{X}|\boldsymbol{\theta}^{(t)})}$, and which does not depend on $\boldsymbol{\theta}$ but is a function of t. By fixing $\mathbf{X}$, the posterior is written as

$$p(\lambda \mid \mathbf{X},\boldsymbol{\theta}^{(t)}) \propto \exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}^{(t)}))p'_{0,t}(\lambda). \tag{5.5}$$

In our framework, we do not need to estimate the normalization constant for (5.5).

Remark 4. *From the above discussion, we see that priors $p_0(\cdot)$ and $p'_{0,t}(\cdot)$ play important roles. We discuss their choices in Section 5.5 below. It is desirable to have a weaker prior assumption, under which the optimizer can still be readily estimated.*

For the M-step, we analyze the Q-function in more detail. For convenience, we use $p_t(\lambda)$ to denote $p(\lambda \mid \mathbf{X},\boldsymbol{\theta}^{(t)})$.

With (5.4) and (5.5), we can express the Q-function as:

$$\begin{aligned} &Q(\boldsymbol{\theta}\mid\boldsymbol{\theta}^{(t)}) \\ =&\mathbb{E}_{\lambda\sim p_t}\left[\log\frac{\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\pi_0(\lambda,\mathbf{X})}{C(\boldsymbol{\theta})}\right] \\ =&-\eta^{(t)}\mathbb{E}_{\lambda\sim p_t}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] + D - \log C(\boldsymbol{\theta}), \end{aligned}$$

where D is a constant independent of $\boldsymbol{\theta}$.

To estimate $\log C(\boldsymbol{\theta})$, we make the following considerations. First of all, by Jensen's inequality, $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] \le \log C(\boldsymbol{\theta})$. This means that if $\log C(\boldsymbol{\theta})$ is small, then

necessarily so is $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})]$. On the other hand, [127] proposes to use $\mathbb{E}[\log Y] + \frac{\mathrm{var}(Y)}{2\mathbb{E}(Y)^2}$ to approximate $\log\mathbb{E}[Y]$ for a random variable Y. This is derived from the second-order Taylor expansion of $\log Y$ at $\log\mathbb{E}[Y]$. In our case, we have

$$\begin{aligned}\log C(\boldsymbol{\theta}) \approx & -\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] \\ & + \frac{\mathrm{var}\big(\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\big)}{2(\mathbb{E}_{\lambda\sim p_0}[\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))])^2}.\end{aligned} \tag{5.6}$$

If $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})]$ is the dominant component, then we may use $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})]$ as a proxy for $\log C(\boldsymbol{\theta})$, which is more manageable. In Section 5.5.1.3, we shall numerically verify that this is indeed the case for our applications.

Hence, $Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$ is approximated by

$$-\eta^{(t)}\Big(\mathbb{E}_{\lambda\sim p_t}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] - \mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})]\Big) + D.$$

In summary, if we disregard $\eta^{(t)}$ and D, which are independent of $\boldsymbol{\theta}$, we can minimize the following function in the M-step:

$$\begin{aligned}J(\boldsymbol{\theta}) &= \mathbb{E}_{\lambda\sim p_t}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] - \mathbb{E}_{\lambda\sim p_0}[L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})] \\ &= \int_{\lambda\in\Lambda}\big(p_t(\lambda) - p_0(\lambda)\big)L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\,\mathrm{d}\lambda.\end{aligned} \tag{5.7}$$

5.4.2 The Proposed Algorithm: EMGNN

To minimize $J(\boldsymbol{\theta})$ in (5.7), our strategy is to re-express it as an expectation. For this purpose, we introduce a proposal distribution. Let $q(\cdot)$ be the density function of a probability distribution on the sample space Λ whose support includes that of p_0. Then we have:

$$\begin{aligned}J(\boldsymbol{\theta}) &= \int_{\lambda\in\Lambda} q(\lambda)\frac{p_t(\lambda) - p_0(\lambda)}{q(\lambda)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\,\mathrm{d}\lambda \\ &= \mathbb{E}_{\lambda\sim q}\left[\frac{p_t(\lambda) - p_0(\lambda)}{q(\lambda)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\right].\end{aligned}$$

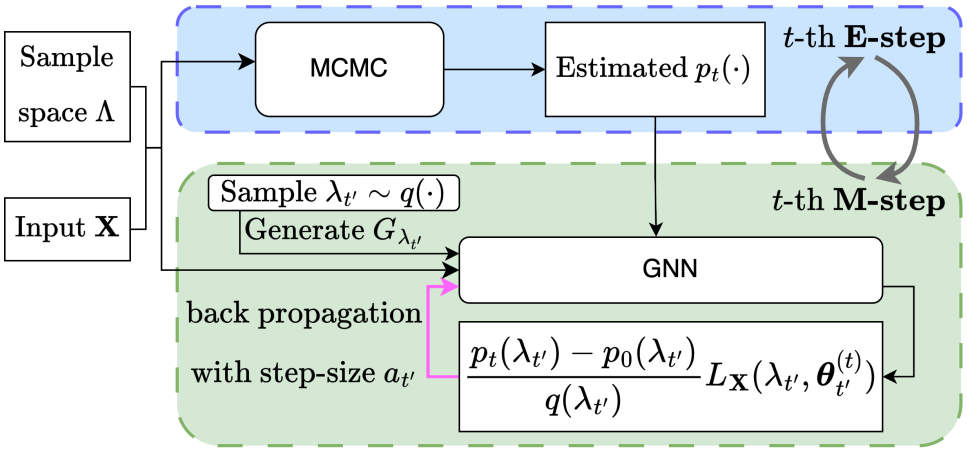


Figure 5.2: Illustration of EMGNN.

We propose to minimize $J(\boldsymbol{\theta})$ by first randomly drawing samples $\Lambda_{T'} = \{\lambda_1, \ldots, \lambda_{T'}\}$ according to the density $q(\cdot)$. Following that, we successively apply gradient descent to $\frac{p_t(\lambda_{t'}) - p_0(\lambda_{t'})}{q(\lambda)} L_{\mathbf{X}}(\lambda_{t'}, \boldsymbol{\theta})$ to update $\boldsymbol{\theta}$. Finally, given (5.5), $p_t(\lambda)$ can be approximated by an empirical distribution if we apply an MCMC method. The overall algorithm is summarized in Algorithm 2 and illustrated in Fig. 5.2.

Remark 5. *In practice, the choices of the prior distributions $p_0(\cdot), q(\cdot)$ and $p'_{0,t}(\cdot)$ are hyperparameters. Moreover, in our experiments, $p'_{0,t}(\cdot)$ is set to be the same for every t. We also discretize the continuous sample space Λ for simplicity in analysis and computation.*

Remark 6. *If we algorithmically plug in the delta distribution supported on λ_0 and $p_0(\lambda_0) = 0$ for $p'_{0,t}(\cdot)$ and $q(\cdot)$ respectively, then EMGNN reduces to the ordinary GNN model on the graph G_{λ_0}.*

Remark 7. *Note that for the coefficient $\frac{p_t(\lambda) - p_0(\lambda)}{q(\lambda)}$, if $p_t(\lambda) < p_0(\lambda)$, then the loss $L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})$ is to be made larger. Intuitively, in this case, a "bad" λ is chosen. For the choice of $q(\cdot)$, in practice, we propose two options in Section 5.5: either the uniform distribution or $q(\cdot) = p_t(\cdot)$. Nonetheless, $q(\cdot)$ can also be other appropriate density functions.*

As we do not minimize $J(\boldsymbol{\theta})$ directly, we justify the proposed approach under additional assumptions. We theoretically analyze the performance of the proposed (randomized) algorithm in lines 5-12 of Algorithm 2, denoted by $\mathcal{A}$. With samples $\Lambda_{T'}$, the algorithm $\mathcal{A}$

Algorithm 2 EMGNN

Input: The observed graph G,
The node features $\mathbf{x}$,
The number of EM iterations T,
The number of epochs per M-step T',
The sample space Λ,
Prior distributions $p_0, p'_{0,t}$ and q,
Function to convert samples to empirical distribution $g(\cdot)$,
Function to generate λ influenced graphs $h(\cdot)$,
A non-increasing step-size $a_{t'}$.
Output: The learned representation $\mathbf{z}$.
Initialization: Pretrain Ψ with G as initial weights.

for $t = 1 : T$ **do**
  AcceptedList$^{(t)} \leftarrow$ MCMC(Λ, $p'_{0,t}$) ▷ E-step
  EmpProbDict$^{(t)} \leftarrow g($AcceptedList$^{(t)})$
  for $t' = 1 : T'$ **do** ▷ M-step
    Sample $\lambda_{t'} \sim q$.
    $G_{\lambda_{t'}} \leftarrow h(\lambda_{t'}, G)$ ▷ $h(\cdot)$ is task-specific. Refer to Section 5.3.1 and Section 5.5.3.
    Update via gradient descent: $\boldsymbol{\theta}^{(t)}_{t'+1} = \boldsymbol{\theta}^{(t)}_{t'} - a_{t'} \nabla (\frac{p_t(\lambda_{t'}) - p_0(\lambda_{t'})}{q(\lambda_{t'})} L_{\mathbf{X}}(\lambda_{t'}, \boldsymbol{\theta}^{(t)}_{t'}))$
    $\boldsymbol{\theta}^{(t)}_{t'+1}, \mathbf{z}_{t'} \leftarrow \Psi(\lambda_{t'}, \mathbf{x}; \boldsymbol{\theta}^{(t)}_{t'})$
  end for
  $\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)}_{T'}$
  $t \leftarrow t + 1$
end for
$\mathbf{z}_{\text{final}} = \mathbb{E}_{\lambda \sim p_T(\cdot)} \left[\Psi(\lambda, \mathbf{x}; \boldsymbol{\theta}^{(T)}_{T'}) \right]$

outputs $\widehat{\boldsymbol{\theta}} = \mathcal{A}(\Lambda_{T'})$. The following expression is considered in algorithm $\mathcal{A}$:

$$J_{\Lambda_{T'}}(\widehat{\boldsymbol{\theta}}) = \frac{1}{T'} \sum_{\lambda_{t'} \in \Lambda_{T'}} \frac{p_t(\lambda_{t'}) - p_0(\lambda_{t'})}{q(\lambda_{t'})} L_{\mathbf{X}}(\lambda_{t'}, \widehat{\boldsymbol{\theta}}).$$

We assume that after translation and scaling by positive constant of $L_{\mathbf{X}}(\lambda, \cdot)$ if necessary, the expression $\frac{p_t(\lambda) - p_0(\lambda)}{q(\lambda)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})$ always belong to $[0, 1]$. The following notions are well-known.

Definition 4. *A differential function f is α-Lipschitz if for all x in the domain of f, we have $\|\nabla f(x)\| \leq \alpha$. It is β-smooth if its gradient is β-Lipschitz.*

Denote by $1_{p_t(\cdot) \geq p_0(\cdot)}(\lambda)$ the indicator that is 1 if $p_t(\lambda) \geq p_0(\lambda)$, and 0 otherwise. Let $b_1 = \mathbb{E}_{\lambda \sim q} 1_{p_t(\cdot) \geq p_0(\cdot)}$. Intuitively, it computes the measure of λ, for which $p_t(\cdot)$ is larger. On the other hand, let $b_2 = \mathbb{E}_{\lambda \sim q} \frac{|p_t(\lambda) - p_0(\lambda)|}{q(\lambda)}$, and $\gamma = \sup_{\lambda \in \Lambda} 1/q(\lambda)$.

Theorem 1. *Assume for any λ, the loss $L_{\mathbf{X}}(\lambda, \cdot)$ is convex, α-Lipschitz and β-smooth. Let b_1, b_2, γ be defined as above. If for every $t' \leq T'$, the non-increasing step-size in the algorithm $\mathcal{A}$ satisfies $a_{t'} \leq \min\{2/(\beta\gamma), c/t'\}$ for a constant c, then there is a constant C independent of T', α such that*

$$\left|\mathbb{E}_{\mathcal{A},\Lambda_{T'}}\left[J_{\Lambda_{T'}}(\widehat{\boldsymbol{\theta}}) - J(\widehat{\boldsymbol{\theta}})\right]\right| \leq \epsilon = C\left(\frac{b_2^2\alpha^2}{T'}\right)^{\frac{1}{\beta\gamma c(1-b_1)+1}}.$$

Proof. The proof is given in the Appendix B. □

Remark 8. *From the result, we see that if b_1 is close to 1, i.e, the set $\{\lambda \mid p_t(\lambda) \geq p_0(\lambda)\}$ has a large measure, then the expected error decays at a rate close to T'^{-1}.*

5.4.3 A Brief Discussion on Testing

As our framework deals with a distribution of graphs, during testing, we acquire our final learned representation as $\mathbf{z}_{\text{final}} = \mathbb{E}_{\lambda\sim p_T}\left[\Psi(\lambda, \mathbf{x}; \boldsymbol{\theta}_{T'}^{(T)})\right]$. The learned model parameters are a particular realization of the possible random models that align with the observed data $\mathbf{X}$ and the multiple graphs influence the final embeddings based on their respective likelihoods. The embedding $\mathbf{z}_{\text{final}}$ is subjected to a softmax operation to obtain $\hat{\mathbf{y}}$ for node classification tasks, while a READOUT function is applied for graph-level tasks.

5.5 Experiments

To validate our algorithm, we study node classification for heterogeneous and homogeneous graphs. Furthermore, we study graph regression using chemical datasets. Hyperparameter and implementation details are provided in the Appendix B.

5.5.1 Heterogeneous Graphs

Heterogeneous graphs are graph structures characterized by the presence of multiple node types and multiple edge types, imparting a greater degree of complexity compared to homogeneous graphs, which consist of a single node and edge type. In these datasets,

we leveraged latent parameter(s) to generate multiple graph instances with varying edge weights for each edge type, representing different information transmission rates.

In the case of a heterogeneous graph with ω edge types, we introduce a vector of latent parameters $\boldsymbol{\lambda} = \{\lambda_1, \ldots, \lambda_\omega\}$, where each $\lambda_i \geq 0$ and $\sum_{i=1}^{\omega} \lambda_i = 1$. The number λ_i is the weight for the i-th edge type. Hence, the sample parameter space is the $(w-1)$-simplex, denoted by Λ. For a chosen $\boldsymbol{\lambda}$, the associated weighted graph $G_{\boldsymbol{\lambda}}$ has adjacency matrix

$$A_{\boldsymbol{\lambda}} = \sum_{i=1}^{\omega} \lambda_i A_i, \tag{5.8}$$

where A_i is the respective edge-type specific adjacency matrix. Note that when any $\lambda_i = 0$, the edges of the associated edge types are removed.

5.5.1.1 Baselines and Datasets

The heterogeneous graph datasets used are the same as those employed in [83] and [18]. These datasets include DBLP, ACM and IMDB. For comprehensive information regarding the datasets, as well as detailed dataset statistics, refer to Section 2.7.3.

We assessed our approach against eight baseline models. Specifically, GAT and GCN are designed for homogeneous graphs, while HAN [81], HGT [128], HetSANN [88], GTN [83], Simple-HGN [87] and SeHGNN [80] are state-of-the-art models developed for heterogeneous graph settings. For our model, we treat edges of different directions as a single type, resulting in two symmetric edge-type specific adjacency matrices. We consider the sample space Λ to span the range $[0.0, 1.0]$ and discretize it in increments of 0.05 to obtain $\widehat{\Lambda}$.

We consider different variants of EMGNN with a GCN backbone, based on choices of $p_0(\cdot), p'_{0,t}(\cdot), q(\cdot)$, as summarized in Table 5.1. In particular, for EMGNN-PD and EMGNN-PH, we set λ_0 for the delta function to be any λ such that $\{\lambda \in \Lambda - \widehat{\Lambda}\}$. Consequently, $p_0(\cdot)$ will be 0 with probability 1 w.r.t $q(\cdot)$ on $\widehat{\Lambda}$. Hence, for these variants, there is no "bad" λ such that the corresponding iteration increases $L_{\mathbf{X}}(\cdot, \cdot)$(Refer to Remark 7). Note that our selection of the uniform and delta distributions is not exhaustive. We opt for the simple generic choice of uniform prior distribution on the sample space to avoid discriminating against any possible value of each parameter (within an appropriate range). Meanwhile,

the delta distribution is incorporated for $p_0(\cdot)$ as a form of ablation study.

Table 5.1: Variants of EMGNN with different choices of $p_0(\cdot), p'_{0,t}(\cdot), q(\cdot)$.

$p_0(\cdot)$	$p'_{0,t}(\cdot)$	$q(\cdot)$	Model
Unif($\widehat{\Lambda}$)	Unif($\widehat{\Lambda}$)	$p_t(\cdot)$	EMGNN-PT
Unif($\widehat{\Lambda}$)	Unif($\widehat{\Lambda}$)	Unif($\widehat{\Lambda}$)	EMGNN-PO
δ_{λ_0}	Unif($\widehat{\Lambda}$)	$p_t(\cdot)$	EMGNN-PD
δ_{λ_0}	Unif($\widehat{\Lambda}$)	Unif($\widehat{\Lambda}$)	EMGNN-PH

5.5.1.2 Results

Table 5.2: Heterogeneous node classification task. Results averaged over ten runs. The best performance is boldfaced and the second-best performance is underlined.

	IMDB		ACM		DBLP	
	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1
GCN	61.91 ± 0.67	60.91 ± 0.57	91.92 ± 0.40	92.00 ± 0.41	94.60 ± 0.31	93.88 ± 0.36
GAT	<u>63.54 ± 1.10</u>	61.87 ± 0.95	92.61 ± 0.36	<u>92.68 ± 0.36</u>	94.48 ± 0.22	93.74 ± 0.27
GTN	60.58 ± 2.10	59.12 ± 1.58	92.12 ± 0.62	92.23 ± 0.60	94.17 ± 0.26	93.59 ± 0.40
HAN	57.83 ± 0.93	56.56 ± 0.77	90.93 ± 0.73	91.01 ± 0.76	92.51 ± 0.24	91.93 ± 0.27
HGT	55.18 ± 1.33	50.82 ± 5.97	90.15 ± 0.89	90.22 ± 0.88	85.55 ± 1.67	84.42 ± 1.56
HetSANN	57.15 ± 0.48	55.19 ± 0.46	90.70 ± 0.62	90.80 ± 0.61	94.71 ± 0.14	93.97 ± 0.16
Simple-HGN	58.91 ± 1.06	58.30 ± 0.34	**92.73 ± 0.21**	92.56 ± 0.42	94.48 ± 0.38	93.69 ± 0.32
SeHGNN	62.13 ± 2.38	60.62 ± 1.95	92.45 ± 0.17	92.51 ± 0.16	94.86 ± 0.14	94.14 ± 0.19
EMGNN-PT	**64.78 ± 1.24**	**63.36 ± 0.80**	<u>92.70 ± 0.26</u>	**92.78 ± 0.26**	**95.06 ± 0.39**	**94.41 ± 0.45**
EMGNN-PO	63.35 ± 0.79	<u>62.25 ± 0.59</u>	92.35 ± 0.38	92.45 ± 0.38	94.95 ± 0.24	94.28 ± 0.28
EMGNN-PD	62.49 ± 0.87	61.55 ± 0.71	92.31 ± 0.43	92.41 ± 0.42	94.89 ± 0.17	94.15 ± 0.23
EMGNN-PH	62.01 ± 0.55	61.15 ± 0.46	92.18 ± 0.52	92.29 ± 0.52	<u>95.02 ± 0.19</u>	<u>94.34 ± 0.20</u>

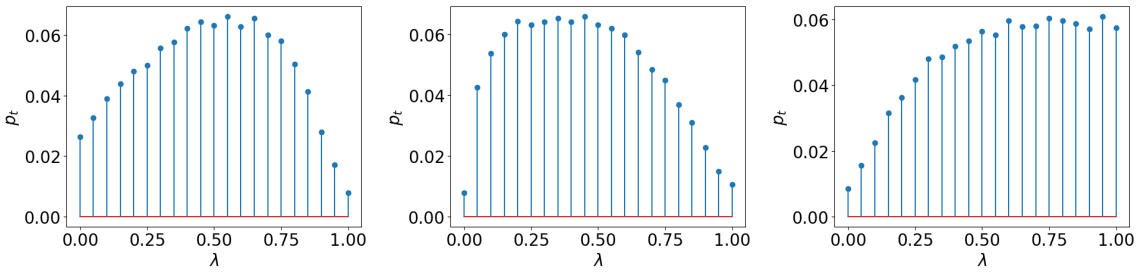
Results are shown in Table 5.2. Similar to recent findings [87], GCN and GAT are observed to perform competitively against models designed for heterogeneous graphs such as GTN under appropriate settings. Meanwhile, EMGNN-PT consistently outperforms other variants in our framework, in both micro and macro F1 scores. In particular, the superior performance of EMGNN-PT, EMGNN-PO, EMGNN-PD and EMGNN-PH compared to GCN indicates the effectiveness of learning with multiple graphs. Moreover, EMGNN-PT outperforming EMGNN-PD, along with EMGNN-PO frequently outperforming EMGNN-PH, indicates that increasing the loss for a "bad" λ is beneficial as it penalizes deviations from desirable graphs.

EMGNN-PT also often surpasses baseline models with attention mechanisms, namely GAT, Simple-HGN, SeHGNN, HAN, HGT, HetSANN and GTN, despite not incorporating

any attention mechanisms. This could be attributed to the construction of multiple graphs, which may form instances whose information is similar to what is achieved with semantic attention. In addition, the model may be able to extract additional useful interactions in other instances of graphs which can enhance its performance.

5.5.1.3 Further Analysis

Ablation study: For EM-GCN[PD] and EM-GCN[PH], λ_0 for the delta function is set to be any $\lambda \in \Lambda \backslash \widehat{\Lambda}$. Consequently, $p_0(\cdot)$ will be 0 with probability 1 w.r.t $q(\cdot)$ on $\widehat{\Lambda}$. Hence, for these variants, there is no "bad" λ such that the corresponding iteration increases $L_{\mathbf{X}}(\cdot, \cdot)$ (cf. Remark 7). From Table 5.2, we see that EM-GCN[PT] outperforms EM-GCN[PD], along with EM-GCN[PO] frequently outperforming EM-GCN[PH]. They indicate that increasing the loss for a "bad" λ is beneficial as it penalizes deviations from desirable graphs.



(a) IMDB (b) DBLP (c) ACM

Figure 5.3: Empirical distribution of $p_t(\cdot)$ that is obtained from the final E-step.

The learned distribution: We examined the learned empirical distributions, depicted in Fig. 5.3. Across all datasets, we notice that the empirical probability of λ falls within the range of approximately $[0.4, 0.6]$ is relatively high. This observation suggests a possible explanation for the decent performance of GCN on a single graph with uniform edge weights.

For IMDB, (5.8) is of the form $\lambda A_{MD} + (1 - \lambda) A_{MA}$. We observed that $\lambda = 1$ has a relatively lower probability compared to $\lambda = 0$. When $\lambda = 1$, it implies that edges in A_{MA} are all removed. This indicates that the MA relation is more crucial than the MD relation. This observation might be linked to the significant difference in edge density

in the edge-type specific adjacency matrices where A_{MA} has three times more non-zero entries than A_{MA}. A similar trend was observed in the ACM dataset, where the disparity in edge density is also substantial. In ACM's case, (5.8) takes the form $\lambda A_{PA} + (1-\lambda)A_{PS}$. The high probability of $\lambda = 1$ indicates that the PS relation is more significant than PA for the task. Consequently, we can infer that edge types with relatively sparse connections have a limited impact on the task.

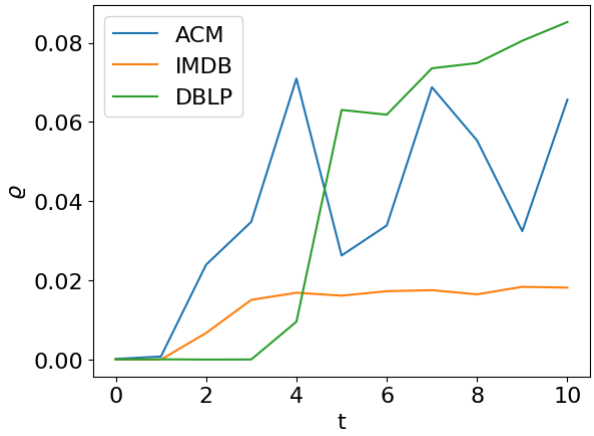


Figure 5.4: Plot of ϱ across t EM iterations

***The Dominant Component of* (5.6)**: In Section 5.4.1, we use $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}\big(L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\big)$ to approximate $\log C(\boldsymbol{\theta})$. To justify this, we provide numerical evidence that the dominant component of (5.6) is $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}\big(L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\big)$. This assertion is supported by assessing the ratio

$$\varrho = \frac{\rho}{-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}\big(L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\big)},$$

where $\rho = \frac{\mathrm{var}\big(\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\big)}{2\Big(\mathbb{E}_{\lambda\sim p_0}\exp(-\eta^{(t)}L_{\mathbf{X}}(\lambda,\boldsymbol{\theta}))\Big)^2}$.

We plot the ϱ values for the heterogeneous graph datasets on EMGNN-PT over multiple t iterations in Fig. 5.4. We found that ϱ consistently exhibits small absolute values, supporting the postulation that $-\eta^{(t)}\mathbb{E}_{\lambda\sim p_0}\big(L_{\mathbf{X}}(\lambda,\boldsymbol{\theta})\big)$ is the main component in (5.6).

5.5.2 Homogeneous Graphs

5.5.2.1 The Experimental Setup and Baslines

For $G = (V, E)$, we follow Section 5.3.1 for the design of Λ. Specifically, we propose to parametrize graphs by a pair $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2\}$ with $\lambda_1, \lambda_2 \in [0, 0.2]$. Here, λ_1 is the probability of randomly removing edges from the graph, to account for possible "noisy edges" in the observed graphs. On the other hand, λ_2 is the probability of randomly introducing edges from a pre-constructed subset E' of all missing edges. More specifically, there are $n(n-1)/2$ possible edges between pairs of distinct nodes of size $|V| = n$. For each pair of nodes v, v' without an edge connection in G, we compute the cosine similarity of their node features. The edge set E' is obtained from including node pairs whose cosine similarities are above a threshold τ (see Appendix B). It is intuitively considered as the set of "likely" missing edges based on feature similarities. Notice that $\boldsymbol{\lambda}$ does not determine a unique graph, we need to slightly modify Algorithm 2 when applying MCMC (see Appendix B).

The most relevant benchmark is BGCN [119] for homogeneous graphs, based on a Bayesian approach to infer the graph distribution. As our construction of Λ involves edge removal and addition, we also consider DropEdge [39] and RSGNN [50], where the former randomly removes edges at each epoch and the latter learns a denser graph using a trained link predictor. The experimental setup follows exactly that from Zhang et al. [119], wherein for each dataset, we evaluate the performance of the algorithms under limited data scenarios where only 10 or 5 labels per class are available.

Table 5.3: Node classification on homogeneous graphs following setup of Zhang et al. [119].

	Cora			Citeseer			Pubmed		
	5 labels	10 labels	20 labels	5 labels	10 labels	20 labels	5 labels	10 labels	20 labels
GCN	74.62 ± 0.54	75.30 ± 0.47	81.37 ± 0.31	54.24 ± 1.26	66.07 ± 0.68	70.19 ± 0.46	69.96 ± 0.65	72.96 ± 0.58	78.45 ± 0.44
DropEdge	74.79 ± 0.56	75.88 ± 0.19	81.46 ± 0.64	54.44 ± 2.54	67.59 ± 1.41	71.23 ± 1.26	71.69 ± 0.50	73.14 ± 0.33	78.50 ± 0.54
RSGNN	**76.80 ± 3.19**	**78.23 ± 2.62**	80.79 ± 0.91	**59.97 ± 1.89**	68.41 ± 0.94	69.73 ± 0.53	70.45 ± 0.78	70.92 ± 0.86	77.55 ± 0.46
BGCN	75.97 ± 0.54	76.52 ± 0.50	81.18 ± 0.48	56.58 ± 0.96	70.61 ± 0.69	72.11 ± 0.40	70.51 ± 1.61	73.36 ± 1.23	76.55 ± 0.65
EM-GCN	74.44 ± 0.76	76.71 ± 0.46	**82.24 ± 0.48**	58.04 ± 2.24	**70.65 ± 1.10**	**72.13 ± 0.96**	**74.03 ± 0.55**	**74.93 ± 0.24**	**78.96 ± 0.38**

5.5.2.2 Results

Based on Table 5.2 and the ablation study in Section 5.5.1.3, we use the EM-GCN[PT] variant for EM-GCN. Results are shown in Table 5.3. Overall, across all datasets, EM-GCN surpasses the performance of BGCN and DropEdge and frequently outperforms RSGNN. In principle, DropEdge and RSGNN do not leverage the potential of a distribution of graphs, which is fundamentally different from our approach.

5.5.2.3 Further Analysis

Heterophilic graphs: Recall that a graph is heterophilic if many edges are connecting nodes with different labels. In particular, nodes from the same class are not grouped. Hence, as BGCN has a clustering mechanism, we expect that it might face challenges for heterophilic graphs. On the other hand, based on the construction of Λ, EM-GCN may generate $\boldsymbol{\lambda}$ such that the associated graph reduces inter-class edges while adding intra-class edges. Hence, we expect EM-GCN should significantly outperform BGCN for heterophilic graphs, which is verified by results in Table 5.4.

EM-GCN is based on the "unsuitable" backbone GCN, which suffers from the same problem as BGCN. Even so, the performance of EM-GCN is much closer to benchmarks ACM-GCN+ [64] and ACMP [129] dedicated to heterophilic graphs. This validates our proposed construction of Λ. Furthermore, our framework can be applied to other backbone models and potentially garner performance improvements. As such, we introduce a variant of our model EM-ACM, with ACM-GCN+ as the backbone model. From Table 5.4, we see that EM-ACM generally outperforms its SOTA backbone. It is reasonable to attribute this to the use of a distribution of graphs.

Robustness: As discussed in Section 5.3.1, our approach might be resistant to errors in the observed graph, as we are not focusing on a single graph. To verify, we consider the Cora graph G and randomly perturb $r\%$ of edges (by adding new edges and removing existing edges) for $5 \leq r \leq 30$. We compare GCN and EM-GCN while observing only the perturbed graph. The results in Fig. 5.5 agree with our speculation that EM-GCN always has a better performance and the gap in accuracies widens as r increases.

Table 5.4: Node classification on heterophilic graphs. The setup and data splits follow Pei et al. [130].

	Texas	Wisconsin	Cornell
GCN	55.14 ± 5.16	51.76 ± 3.06	60.54 ± 5.30
DropEdge	57.57 ± 4.94	57.45 ± 5.47	60.54 ± 5.30
RSGNN	68.38 ± 5.26	68.82 ± 7.25	60.96 ± 6.90
BGCN	57.96 ± 6.77	61.37 ± 4.72	56.48 ± 6.67
ACM-GCN+	86.76 ± 4.26	86.86 ± 2.91	84.05 ± 7.88
ACMP	86.20 ± 3.00	86.10 ± 4.00	**85.40 ± 7.00**
EM-GCN	79.46 ± 4.26	83.73 ± 4.34	77.30 ± 4.10
EM-ACM	**88.38 ± 5.14**	**87.06 ± 2.51**	85.14 ± 6.54

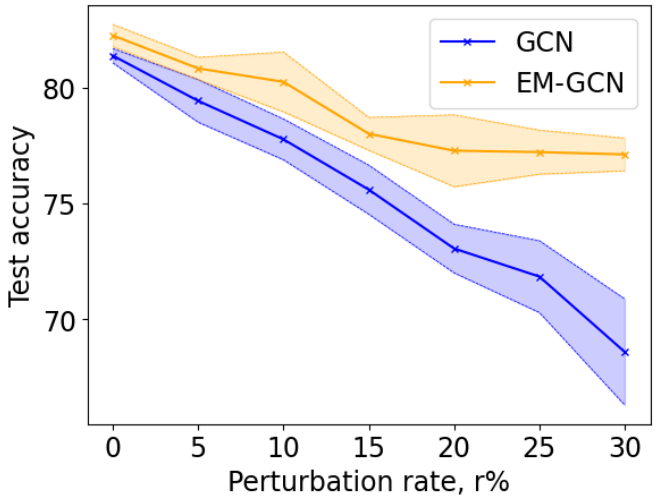


Figure 5.5: Performance comparison for graph perturbation.

5.5.3 Chemical Datasets

Conventional molecular graph representations mirror a molecule's Lewis structure, with atoms as nodes and chemical bonds as edges. This representation falls short in capturing variations in molecular properties resulting from different three-dimensional (3D) spatial arrangements when molecules share the same topology, as seen in cases like cis-trans isomers. Moreover, molecules inherently possess uncertainty due to their quantum mechanical properties, particularly concerning electron orbitals. Hence, using a distribution of graphs for learning in such cases is a sensible choice.

The process of generating different molecular graphs begins with the acquisition of coarse 3D coordinates of the atoms in a molecule using RDKit[1]. Following that, the

[1]A cheminformatics tool. https://www.rdkit.org

interatomic Euclidean distances between all atoms within the molecule are calculated. A parameter, λ is then introduced to define a threshold range, $[0, \lambda]$, for determining node connections and generating multiple graph instances. The notion of employing thresholding based on the interatomic distance between nodes in molecular graphs has been previously documented in works such as [131] and [132]. The former introduced a cut-off distance hyperparameter to construct heterogeneous molecular graphs. Meanwhile, our approach aligns more closely with the latter, where the Vietoris-Rips complex and thresholding are used to form a series of G_λ graphs. However, in [132], they utilized five non-overlapping, manually adjusted intervals for thresholding and adopted a computationally intensive multi-channel configuration to learn from the five generated graphs. There are also works such as [133], where molecules are treated as 3D point clouds and a radius is set to specify interacting vertices.

The graph construction process may involve adding new edges, connecting distant nodes, or removing existing edges. Ideally, the model should prioritize "useful" graph realizations and assign a low probability to less beneficial ones, effectively discarding them.

Table 5.5: Graph regression task on molecular datasets. Average test rmse reported, the lower the better.

	Random split			Scaffold split		
Datasets	FreeSolv	ESOL	Lipophilicity	FreeSolv	ESOL	Lipophilicity
GCN	1.157 ± 0.215	0.652 ± 0.073	0.707 ± 0.030	2.618 ± 0.298	0.876 ± 0.037	0.760 ± 0.009
GAT	1.873 ± 0.522	0.837 ± 0.101	0.704 ± 0.058	2.942 ± 0.591	0.907 ± 0.034	0.777 ± 0.037
Weave	1.497 ± 0.251	0.798 ± 0.088	0.789 ± 0.059	3.129 ± 0.203	1.104 ± 0.063	0.844 ± 0.031
MPNN	1.388 ± 0.404	0.703 ± 0.075	0.640 ± 0.025	2.975 ± 0.775	1.117 ± 0.058	**0.735 ± 0.019**
AttentiveFP	1.275 ± 0.289	0.673 ± 0.085	0.719 ± 0.042	2.698 ± 0.297	0.855 ± 0.029	0.762 ± 0.022
GIN	1.678 ± 0.494	0.792 ± 0.097	0.716 ± 0.073	2.957 ± 0.696	0.990 ± 0.057	0.770 ± 0.021
EMGNN	**0.936 ± 0.162**	**0.606 ± 0.041**	**0.639 ± 0.028**	**2.189 ± 0.128**	**0.834 ± 0.027**	0.743 ± 0.013

5.5.3.1 Baselines and Datasets

MoleculeNet[2] is a popular benchmark for molecular machine learning, encompassing multiple datasets to be tested on a diverse of molecular properties. For our evaluation, we specifically selected the datasets FreeSolv, ESOL, and Lipophilicity, all of which are designed for graph regression tasks. Further elaboration on the chosen datasets can be

[2]https://moleculenet.org/datasets-1

found in Section 2.7.4.

We compared our approach against standard models for molecular properties prediction that do not incorporate transfer learning from a larger dataset such as Zinc15[3]. The selected baseline models for this comparison included Weave [134], MPNN [135], AttentiveFP [136], GIN [38], as well as the standard GCN and GAT models. For EMGNN, a GNN model that generalizes GCN with a degree-1 convolutional filter $P_\lambda(S_\lambda)$ (refer to Section 5.2) is utilized as the backbone of our model. The sample space Λ spans the range $[1, 10]$Å and $\widehat{\Lambda}$ is the discretized space with 0.05 increments.

5.5.3.2 Results

In Table 5.5, the average test root mean square error (rmse) over ten runs with standard deviation is reported for the graph regression task, where the molecular properties of molecular graphs are to be predicted. The result shown is for the case of $q(\cdot) = p_t(\cdot)$. We observe that EMGNN frequently performed better than the baselines. This may be due to EMGNN's training process, which exposes it to diverse graph realizations, allowing it to capture non-covalent interactions that are critical for characterizing the physical properties of molecules. In contrast, the baselines employ the conventional molecular graph representation. We note that our framework does not explicitly incorporate bond angles but it does expose the model to graphs with a broad range of connectivities. This exposure indirectly integrates geometric information, as the latent variable constructs graphs with bond lengths falling within specific ranges. This provides our model with additional 2D information regarding interatomic distances, which may offer insights into the underlying 3D structure.

5.6 Conclusion

In this work, we explored employing a distribution of parametrized graphs for training a GNN in an EM framework. Through a probabilistic framework, we can handle the uncertainty in graph structures and expose our model to various topological and geometric

[3]a database of purchasable drug-like compounds; https://zinc.docking.org/tranches/home/

information. Our approach enables the model to handle multiple graphs where the prediction loss is utilized to estimate the likelihood of the graphs. The model's performance is enhanced as we provide it with a wider array of graphs, which it can then sift through to acquire more valuable information or remove noise.

In the future, it may be worth looking into other possible choices of prior distributions that might garner us better results, possibly following [137]. Also, exploring a weaker prior assumption, as mentioned in Remark 4 would also be an interesting avenue for further research.

Chapter 6

Conclusion and Future Works

In this chapter, we provide a summary of our research works, revisit the research questions, discuss possible improvements, and outline potential directions for future work in GNN that have not been explored in this thesis.

6.1 Conclusion

In conclusion, this thesis has introduced and analyzed three GNN models, namely JSGNN (Chapter 3), SGAT (Chapter 4), and EMGNN (Chapter 5). Each of these models employs geometric concepts to address specific challenges and limitations of existing GNNs in the field of graph representation learning. JSGNN utilized multiple geometric spaces, a hyperbolicity measure, and non-uniform constraints to select the most appropriate space for embedding each node. This approach effectively mitigated distortions in learned node representations that may arise when embedding a node in a space that does not align with its underlying geometry. On the other hand, SGAT addressed the challenge of capturing high-order relations in heterogeneous graphs and circumventing the limitations of existing metapath-based methods. Meanwhile, EMGNN introduced the concept of learning from a distribution of parameterized graphs, overcoming the limitations of learning using a single observed instance of fixed topology and geometry.

Having presented our works in the thesis, we will now revisit the research questions posed in Chapter 1 which we set out to address. Here, we offer some insights and potential

improvements that can serve as interesting directions for future works.

Research Question 1: *Can we enable GNNs to leverage multiple geometric spaces for learning, capitalizing on each space's unique strengths, given the diverse underlying geometries within graphs?*

Prior works, such as GIL and κ-GCN, have already demonstrated the feasibility of using multiple geometric spaces for learning. However, they may have limitations in fully harnessing the strengths of these spaces. Similar to these prior works, our introduced model, JSGNN (Chapter 3), is inspired by the observation that different spaces offer distinct modeling advantages. Additionally, it is well-established that graphs often exhibit complex and diverse structural characteristics across various regions. Nonetheless, JSGNN differs from previous works in terms of the assumptions about how embeddings from different spaces are related. For instance, GIL suggests that embeddings from different spaces can mutually enhance each other, while κ-GCN employs a Cartesian product space, essentially concatenating features from different constant curvature spaces.

GIL's findings have shown that concatenation falls short. They suggest that this is because embeddings learned separately are susceptible to distortion during the learning process due to the diverse graph structures exhibited by nodes. Interestingly, this finding contradicts their model design, where they proposed that embeddings learned from different spaces could mutually enhance each other. The contradiction arises from the realization that, if these embeddings are prone to distortion, mixing them may result in noise and inferior quality embeddings rather than enhancement.

As such JSGNN's approach is to select the embedding of the optimal space for each node and avoid mixing features from different spaces. This is achieved through the utilization of a hyperbolicity measure, which determines each node's underlying neighborhood. The measure guides an attention mechanism equipped with non-uniform constraints, enabling the selection of the most suitable space for each node. Importantly, our model consistently outperforms its single-space building blocks on graphs with varying geometries, demonstrating its efficiency in leveraging the unique strengths of each space.

This is not necessarily observed for GIL and κ-GCN. Our model, however, is currently limited to two geometric spaces - hyperbolic and Euclidean - but we expect this limitation can be overcome in future work. This suggests an affirmative answer to our research question is within reach.

Research Question 2: *Can we develop a GNN for heterogeneous graphs that can capture high-order relations without relying on metapath or metagraph?*

In our pursuit of addressing this question, we introduced SGAT ([18]; Chapter 4). SGAT employs simplicial complexes, a concept previously applied in homogeneous graphs. We recognized the potential advantages of applying this notion in heterogeneous settings due to the inherent complexity implied by the construct of metapaths and metagraphs.

SGAT distinguishes itself from methods based on metapaths or metagraphs by excelling in capturing intricate relationships that involve multiple nodes. This unique capability results in the generation of improved node representations, as evidenced by its enhanced task performance. Notably, SGAT excels in the scenario where noisy and uninformative node features are utilized, demonstrating its efficiency in leveraging the graph's structural information. Furthermore, SGAT effectively circumvents the limitations discussed in Section 4.1.

We can thus conclude that it is feasible to develop a GNN for heterogeneous graphs, capable of capturing high-order relations without relying on metapaths or metagraphs. Our approach, employing simplicial complexes, proves effective and consistently outperforms existing metapath or metagraph-based methods. A later work, SHAN [90] has improved our original proposal by explicitly preserving the original graph structure and employing the hyperbolic space to embed heterogeneous graphs and simplicial complexes which exhibit power-law structures. Potential avenues for future research include optimizing computational efficiency in these models and improving the data preprocessing pipeline to accommodate large-scale graphs.

Research Question 3: *How can we generate and use multiple graphs with diverse topologies and geometric characteristics for learning, considering the potential inaccuracies in the provided graph, to include missing information and reduce noise?*

The EMGNN model addresses this question through a probabilistic framework that introduces latent variables to parameterize and generate multiple graphs. Specifically, it employs an EM approach. In the E-step, we utilize MCMC to learn the distribution of these latent variables, which, in turn, defines the distribution of the parameterized graphs. In the M-step, the model utilizes this learned distribution to obtain the maximum likelihood estimate of the network parameters. This multi-graph learning approach can handle the uncertainty inherent in graph data and offer a range of topological and geometric information to the model.

By applying the principles of PAC-Bayesian learning, the likelihood of the graphs is estimated using task-specific loss. Consequently, graph instances that result in high loss values are assigned low likelihood within the learned distribution. This allows us to effectively filter out poor-performing graph instances and identify those that are well-suited for the task at hand, which may not necessarily correspond to the observed graph.

Our EMGNN model has outperformed other GNN baselines in the node classification task for heterogeneous graphs and graph regression task for chemical datasets. This underscores the efficacy of our approach in leveraging multiple parameterized graphs for improved performance, providing an affirmative answer to this research question. In terms of future research directions, although we currently present various model variants with different manually selected prior distributions, there is an opportunity to advance by loosening the assumptions inherent in the derivation of our model concerning these prior distributions. This opens up avenues for exploring more flexible modeling approaches.

Both JSGNN and SGAT are spatial models that operate in the spatial domain and capitalize on local node connectivity without needing explicit shift operators. This enables their application to large-scale graphs through efficient batching in training and testing.

The main challenge then lies in improving the computation of local hyperbolicity for each node in JSGNN and generating the simplicial complexes in SGAT. Although these aspects have not been optimized due to acceptable speed for small to medium-scale graphs and the ability to store outputs for reuse, multiprocessing or parallelization can accelerate them. Meanwhile, our EM framework can be applied to any model, allowing the framework to be applied to large-scale graphs as long as the base model can do so. Although our EMGNN setup is primarily based on GCN, a spectral GNN which often involves eigendecomposition of the shift operator for spectral convolution, we can also integrate adaptations of GCN specifically designed for large graphs such as [138–140] as our base model.

This thesis was motivated by the idea of exploring mathematical tools and geometric concepts, including geometric hyperbolicity, simplicial complexes, multiple geometric objects (graph topologies) and the EM algorithm, to enhance the design of GNNs and address the limitations observed in existing methods. Our observation that real-world graph data is often characterized by complexity, non-uniformity in terms of geometry, and inherent uncertainty, has also illuminated areas where improvements are possible to achieve our overarching goal: enhancing the quality of learned graph representations and, in turn, achieving better performance in downstream tasks. This has taken us on an enriching journey and led us to develop three distinct GNN models, contributing to the ongoing exploration of graph representation learning.

6.2 Applications

The common focus across all works in this thesis has been on enhancing graph representations, which have broad applications across various domains and practical scenarios where graph-based data is abundant. By improving the representations, we can achieve more accurate and better-performing models. Chapter 5 showcases the applicability of our findings to predicting molecular chemical properties, offering a means to reduce reliance on expensive and time-consuming chemistry lab experiments. This application is particularly relevant in drug discovery, where GNN models have gained traction.

Additionally, Chapters 4 and 5 delve into heterogeneous graphs, which are commonly encountered in real-world scenarios such as fraud detection on digital platforms such as e-commerce, gaming and live-streaming. Fraud activities include spam reviews, fake user registrations, fabricated shopping behaviors, promotion abuse and fraudulent transactions [141–143]. In e-commerce settings, these graphs encompass diverse node types, including users, phone numbers, payment methods, and email addresses. The objective is to make use of structural information to identify "risky" user nodes that could compromise the system's integrity. This application has garnered significant attention, as evidenced by the plethora of industrial papers addressing it [142, 144–147].

In summary, the findings of this thesis have broad applicability across various domains where graph-based data analysis is prevalent, offering the potential for progress across diverse fields, spanning from drug discovery to fraud detection, through improved graph representations.

6.3 Future Works

Lastly, we outline promising areas for future research in GNNs, expanding upon topics not covered in this thesis. These directions offer exciting opportunities for advancing GNN understanding and applications.

Robustness. GNNs have been identified as vulnerable to adversarial attacks, which can manifest in the form of corruption of node attributes or perturbations in the graph topology involving the addition or removal of edges and the introduction of malicious nodes [148, 149]. Works [150–153] in this domain typically aim to discover distinguishing properties that can help discern adversarial nodes or edges from clean ones. Subsequently, they propose strategies based on these properties to remove the adversarial components [154]. More recently, emerging methods [97, 148, 155–157] leverage graph neural Ordinary Differential Equations (ODEs) or Partial Differential Equations (PDEs), a diffusion framework on graphs, to improve the robustness of GNNs. Future research in this direction includes exploring higher-order ODEs and PDEs, along with optimizing diffusion models

for increased memory and time efficiency.

Explainability The demand for explainable predictions in GNNs has grown significantly, especially in critical applications like medical diagnosis [158, 159]. Researchers have adapted eXplainable-AI (XAI) methods to make GNNs interpretable [159]. Recent efforts have focused on developing specialized strategies for GNNs to identify the most influential graph components in model predictions [158, 160–163]. In [164], the typical pipeline of first training the GNN, generating explanations, and then evaluating against ground truth has been suggested to be susceptible to five common pitfalls. Consequently, methods such as [160] have proposed GNNs with *built-in* interpretability, rather than relying on post-hoc explanations. Moreover, while most research concentrates on explaining static GNNs, exploring explanations for temporal GNNs presents a promising avenue.

Author's Publications

Published Conference Papers

1. **See Hian Lee**, Feng Ji and Wee Peng Tay, "Learning On Heterogeneous Graphs Using High-Order Relations," IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Toronto, Canada, 2021, pp. 3175-3179, doi: 10.1109/ICASSP39728.2021.9413417. Available: `https://ieeexplore.ieee.org/document/9413417`.

2. **See Hian Lee**, Feng Ji and Wee Peng Tay, "SGAT: Simplicial Graph Attention Network," in Proc. International Joint Conference on Artificial Intelligence, Jul. 2022. Available: `https://doi.org/10.24963/ijcai.2022/443`.

3. Feng Ji, **See Hian Lee**, Hanyang Meng, Kai Zhao, Jielong Yang, and Wee Peng Tay, "Leveraging label non-uniformity for node classification in graph neural networks," in Proc. International Conference on Machine Learning, Hawaii, USA, Jul. 2023. Available: `https://arxiv.org/pdf/2305.00139.pdf`.

Papers Pending Review

1. **See Hian Lee**, Feng Ji, and Wee Peng Tay, "Node-Specific Space Selection via Localized Geometric Hyperbolicity in Graph Neural Networks," submitted for review in the Pattern Recognition Journal. [Preprint available at `https://arxiv.org/abs/2303.01724`].

2. **See Hian Lee**, Feng Ji, Kelin Xia, and Wee Peng Tay, "Graph Neural Networks with a Distribution of Parametrized Graphs," submitted for review in International

Conference on Machine Learning (ICML). [Preprint available at https://arxiv.org/abs/2310.16401].

Appendix A

SGAT

R.1 Preliminaries

Here, we discuss some properties of simplicial complexes and the notion of upper adjacency, which we use in our proposed SGAT model. For further details, readers can consult [115].

R.1.1 Simplicial Complex

A simplicial complex is a set consisting of vertices, edges and other higher order counterparts, all of which are known as k-simplices, $k \geq 0$, defined as follows.

Definition 5 (k-simplex)**.** *A standard (unit) k-simplex is defined as*

$$\sigma^k = \left\{ (\tilde{v}_0, \ldots, \tilde{v}_k) \in \mathbb{R}^{k+1}_{\geq 0} \ : \ \sum_{i=0}^{k} \tilde{v}_i = 1 \right\}. \tag{A.1}$$

Any topological space that is homeomorphic to the standard k-simplex and thus, share the same topological characteristics is called a k-simplex.

As an example, let $v_0, \ldots, v_k$ be vertices of a graph embedded in a Euclidean space so that they are affinely independent (i.e., $v_1 - v_0, \ldots, v_k - v_0$ are linearly independent). Then the set

$$\left\{ \sum_{i=0}^{k} \alpha_i v_i \ : \ \alpha_i \geq 0, \ \sum_{i=0}^{k} \alpha_i = 1 \right\} \tag{A.2}$$

is a k-simplex. In particular,the vertices $\{v_i\}$ are instances of σ^0 while the edges (v_i, v_j),

$i \neq j$, are instances of σ^1. Subsequently, it is possible to geometrically interpret 0-simplices σ^0 as vertices, 1-simplices σ^1 as edges, 2-simplices σ^2 as triangles, 3-simplices σ^3 as tetrahedron and so on.

A face of a k-simplex is a $(k-1)$-simplex that we obtain from (A.1) by restricting a fixed coordinate in $(\tilde{v}_0, \ldots, \tilde{v}_k)$ to be zero. Specifically, a face of a k-simplex is a set containing elements of the form $(\tilde{v}_0, \ldots, \tilde{v}_{j-1}, \tilde{v}_{j+1}, \ldots, \tilde{v}_k)$. A k-simplex has $k+1$ faces.

A simplicial complex is a class of topological spaces that encodes higher-order relationships between vertices. The formal definition is as below.

Definition 6 (Simplicial complex)**.** *A simplicial complex χ is a finite set of simplices such that the following holds.*

- *Every face of a simplex in χ is also in χ.*
- *The non-empty intersection of any two simplices σ_1, σ_2 in χ is a face of both σ_1 and σ_2.*

It is also possible to produce a concrete geometric object for each simplicial complex χ. The geometric realisation of a simplicial complex is a topological space formed by glueing simplices together along their common faces [165]. For instance, a simplicial complex of dimension 1, consists of two kinds of simplices σ^0 and σ^1. By glueing σ^1 with common σ^0, we obtain a graph in the usual sense. This means that χ^0 is the set of vertices V in the graph and χ^1 is the set of edges E in the graph.

R.1.2 Simplicial Adjacencies

Four kinds of adjacencies can be identified between simplices. They are boundary adjacencies, co-boundary adjacencies, lower-adjacencies and upper-adjacencies [92]. In this work, we utilize only upper-adjacency so that our SGAT model is equivalent to GAT when $K = 1$. We say that two k-simplices in χ^k are upper-adjacent if they are faces of the same $(k+1)$-simplex. The upper adjacency matrix $A^k \in \mathbb{R}^{|\chi^k| \times |\chi^k|}$ of $\chi^k = \{\sigma_1^k \ldots \sigma_{|\chi^k|}^k\}$ indicates whether pairs of k-simplices are upper-adjacent. The upper adjacency matrix of 0-simplices (nodes), A^0, is the usual adjacency matrix of a graph.

Appendix B

EMGNN

R.1 Hyperparameters and Code

The models were trained on a server equipped with four NVIDIA RTX A5000 GPUs for hardware acceleration.

Heterogeneous node classification. The Simple-HGN model was obtained from the CogDL library [166], SeHGNN from their code repository (`https://github.com/ICT-GIMLab/SeHGNN`), while HetSANN and HGT are from the OpenHGNN library [167]. Meanwhile, GAT was sourced from the DGL library [168] and GCN is from `https://github.com/tkipf/pygcn`.

The hidden units are set to 64 for all models. The hyperparameters of SeHGNN and Simple-HGN are as in the respective repository. For fair comparison, GAT and GCN are evaluated on the entire graph ignoring the types. Specifically for our model, the number of MCMC iterations N_{mc} is set to be 15000, the T denoting the number of EM iterations and T' is tuned by searching on the following search spaces: $[10, 15, 20, 25, 30]$.

Homogeneous node classification. The BGCN model was obtained from its code repository at `https://github.com/huawei-noah/BGCN/tree/master`. Likewise, for RSGNN, the model was sourced from its code repository located at `https://github.com/EnyanDai/RSGNN`. As for the ACM-GCN+ model, it was acquired from its code repository (`https://github.com/SitaoLuan/ACM-GNN`).

The hyperparameters used for evaluating the models align with those specified in their respective repositories if provided.

Unlike heterogeneous graphs, during implementation, once $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ is determined in MCMC, we still need to draw a graph $G_{\boldsymbol{\lambda}}$ according to $\boldsymbol{\lambda}$. The graph $G_{\boldsymbol{\lambda}}$ is stored for later use.

In the context of our model, N_{mc} is set to be 15000, the T and T' is tuned by searching on the following search spaces: $[10, 15, 20, 25, 30]$. The choice of the threshold τ (see Section 5.5.2) is tabulated in Table R.1. Notice that the threshold τ is chosen such that E' for each dataset is neither too small or too big.

Table R.1: Statistics and parameters of homogeneous graph datasets.

	# Nodes	# Edges	# Features	# Classes	Threshold τ
Cora	2708	5429	1433	7	0.5
Citeseer	3327	4732	3703	6	0.6
Pubmed	19717	44338	500	3	0.9
Texas	183	295	1703	5	0.6
Cornell	183	280	1703	5	0.6
Wisconsin	251	466	1703	5	0.6

Graph regression on molecular datasets. The experiments on this task were facilitated using the DGL-LifeSci library [169] which provided the code and hyperparameters for the baseline models. For fair comparisons, all the models were evaluated using the canonical features as documented at `https://lifesci.dgl.ai/generated/dgllife.utils.CanonicalAtomFeaturizer.html`. Specifically for our model, N_{mc} is set to 18000, T and T' is tuned by searching on the following search spaces: $[10, 15, 20, 25, 30]$.

R.2 Proof of Theorem 1

The proof is a refinement of the proof of [170, Theorem 3.12]. As $L_{\mathbf{X}}$ is assumed to be convex, we observe that in the expression for $J_{\Lambda_{T'}}(\widehat{\boldsymbol{\theta}})$, a term is convex if $p(\lambda_i) \geq p_0(\lambda_i)$ and concave otherwise. Therefore, we need to separate these two cases when performing gradient descent.

We follow the proof of [170, Theorem 3.12], and highlight necessary changes. By [170, Theorem 2.2], it suffices to show that the algorithm $\mathcal{A}$ is ϵ-uniformly stable [170, Definition 2.1]. Let $\Lambda_1 = \{\lambda_{1,1}, \ldots, \lambda_{1,T'}\}$ and $\Lambda_2 = \{\lambda_{2,1}, \ldots, \lambda_{2,T'}\}$ be two sample sequences of λ that differ in only a single sample. Consider the gradient updates $\Gamma_{1,1}, \ldots, \Gamma_{1,T'}$ and $\Gamma_{2,1}, \ldots, \Gamma_{2,T'}$. Let $\widehat{\boldsymbol{\theta}}_{1,t'}$ and $\widehat{\boldsymbol{\theta}}_{2,t'}, t' \leq T'$ be the corresponding outputs of the algorithm $\mathcal{A}$.

Introduce $f(\lambda, \boldsymbol{\theta}) = \frac{p(\lambda)-p_0(\lambda)}{q(\lambda)} L_{\mathbf{X}}(\lambda, \boldsymbol{\theta})$. As $L_{\mathbf{X}}(\lambda, \cdot)$ is convex, α-Lipschitz and β-smooth, $f(\lambda, \boldsymbol{\theta})$ is $\alpha\gamma$-Lipschitz, $\beta\gamma$-smooth. Moreover, it is convex if $p(\lambda) \geq p_0(\lambda)$.

Write $\delta_{t'}$ for $\|\widehat{\boldsymbol{\theta}}_{1,t'} - \widehat{\boldsymbol{\theta}}_{2,t'}\|$. Using the fact that $f(\lambda, \cdot)$ is $(\alpha\gamma)$-Lipschitz, by [170, Lemma 3.11], we have for any $t_0 \leq T'$

$$\begin{aligned}
&\mathbb{E}\big(|f(\lambda, \widehat{\boldsymbol{\theta}}_{1,T'}) - f(\lambda, \widehat{\boldsymbol{\theta}}_{2,T'})|\big) \\
&\leq \frac{t_0}{T'} + \alpha\gamma \mathbb{E}(\delta_{T'} \mid \delta_{t_0} = 0).
\end{aligned}$$

We need an estimation of $\mathbb{E}(\delta_{T'} \mid \delta_{t_0} = 0)$. For convenience, for any $t' \geq t_0$, let $\Delta_{t'} = \mathbb{E}(\delta_{t'} \mid \delta_{t_0} = 0)$.

Observe that at step t', with probability $1 - 1/n$, the samples selected are the same in both Λ_1 and Λ_2. Moreover, with probability $(1 - 1/n)b_1$, the common sample is convex, so we can apply [170, Lemma 3.7.2]. With probability $(1 - 1/n)(1 - b_1)$, the common sample is non-convex, and [170, Lemma 3.7.1] is applicable. With probability $1/n$, the selected samples are different and $\Gamma_{1,t'}$ and $\Gamma_{2,t'}$ are respectively $\frac{|p(\lambda)-p_0(\lambda)|}{q(\lambda)}\alpha a_{t'}$-bounded, by [170, Lemma 3.3].

Therefore, by linearity of expectation and [170, Lemma 2.5], we may estimate:

$$\begin{aligned}
\Delta_{t+1} &\leq (1 - \frac{1}{n})\big(b_1 + (1 - b_1)(1 + a_{t'}\beta\gamma)\big)\Delta_{t'} + \frac{1}{n}\Delta_{t'} \\
&\quad + \frac{\alpha a_{t'}}{n}\mathbb{E}_{\lambda_{1,t'}, \lambda_{2,t'}}\big(\frac{|p(\lambda_{1,t'}) - p_0(\lambda_{1,t'})|}{q(\lambda_{1,t'})} \\
&\quad + \frac{|p(\lambda_{2,t'}) - p_0(\lambda_{2,t'})|}{q(\lambda_{2,t'})}\big) \\
&= (1 - \frac{1}{n})\big(b_1 + (1 - b_1)(1 + a_{t'}\beta\gamma)\big)\Delta_{t'} \\
&\quad + \frac{1}{n}\Delta_{t'} + \frac{2b_2\alpha a_{t'}}{n}
\end{aligned}$$

$$
\begin{aligned}
&\leq (1-\frac{1}{n})\big(b_1+(1-b_1)(1+\frac{c\beta\gamma}{t'})\big)\Delta_{t'} \\
&+\frac{1}{n}\Delta_{t'}+\frac{2b_2\alpha c}{nt'} \\
&=\big(1+(1-\frac{1}{n})(1-b_1)\frac{c\beta\gamma}{t'}\big)\Delta_{t'}+\frac{2c(b_2\alpha)}{nt'} \\
&\leq \exp\big((1-\frac{1}{n})(1-b_1)\frac{c\beta\gamma}{t'}\big)\Delta_{t'}+\frac{2c(b_2\alpha)}{nt'} \\
&=\exp\Big((1-\frac{1}{n})\frac{c\big((1-b_1)\beta\gamma\big)}{t'}\Big)\Delta_{t'}+\frac{2c(b_2\alpha)}{nt'}.
\end{aligned}
$$

Then, by the same argument as in the proof of [170, Theorem 3.12], we have the algorithm $\mathcal{A}$ is ϵ-uniformly stable for any

$$
\epsilon \leq C\big(\frac{b_2^2\alpha^2}{T'}\big)^{\frac{1}{\beta\gamma c(1-b_1)+1}},
$$

for some constant C independent of T' and α.

Bibliography

[1] OpenAI, "Chatgpt [large language model]," 2023. [Online]. Available: https://chat.openai.com/chat

[2] J. Tang, J. Li, Z. Gao, and J. Li, "Rethinking graph neural networks for anomaly detection," in *International Conference on Machine Learning*, 2022.

[3] Y. Li, Y. Ji, S. Li, S. He, Y. Cao, X. Li, J. Shi, Y. Yang, and Y. Liu, "Relevance-aware anomalous users detection in social network via graph neural network," 2021.

[4] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. ICML'17. JMLR.org, 2017, p. 1263–1272.

[5] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. Hoesel, H. Schopmans, T. Sommer, and P. Friederich, "Graph neural networks for materials science and chemistry," *Communications Materials*, vol. 3, 11 2022.

[6] W. Gong and Q. Yan, "Graph-based deep learning frameworks for molecules and solid-state materials," *Computational Materials Science*, vol. 195, p. 110332, 2021.

[7] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '18, 2018, p. 974–983.

[8] Y. Chen, M. Yang, Y. Zhang, M. Zhao, Z. Meng, J. Hao, and I. King, "Modeling scale-free graphs with hyperbolic geometry for knowledge-aware recommendation,"

in *Proceedings of the 15th ACM International Conference on Web Search and Data Mining*, ser. WSDM '22, 2022, p. 94–102.

[9] W. Ju, Z. Fang, Y. Gu, Z. Liu, Q. Long, Z. Qiao, Y. Qin, J. Shen, F. Sun, Z. Xiao, J. Yang, J. Yuan, Y. Zhao, X. Luo, and M. Zhang, "A comprehensive survey on deep graph representation learning," 2023.

[10] S. Khoshraftar and A. An, "A survey on graph representation learning methods," 2022.

[11] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1105–1114.

[12] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. Association for Computing Machinery, 2014, p. 701–710.

[13] X. Shen, S. Pan, W. Liu, Y.-S. Ong, and Q.-S. Sun, "Discrete network embedding," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18, 2018, p. 3549–3555.

[14] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 1476–1481.

[15] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[16] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, "Tri-party deep network representation," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, p. 1895–1901.

[17] S. H. Lee, F. Ji, and W. P. Tay, "Node-specific space selection via localized geometric hyperbolicity in graph neural networks," *arXiv*, 2023. [Online]. Available: https://arxiv.org/abs/2303.01724

[18] S. H. Lee, F. Ji, and W. P. Tay, "Sgat: Simplicial graph attention network," in *International Joint Conference on Artificial Intelligence*, 2022. [Online]. Available: https://arxiv.org/abs/2207.11761

[19] S. H. Lee, F. Ji, K. Xia, and W. P. Tay, "Graph neural networks with a distribution of parametrized graphs," 2023.

[20] G. Shobha and S. Rangaswamy, "Chapter 8 - machine learning," in *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications*, ser. Handbook of Statistics, V. N. Gudivada and C. Rao, Eds. Elsevier, 2018, vol. 38, pp. 197–228. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169716118300191

[21] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, p. 53, 2021.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. on Neural Networks and Learning Systems*, vol. 32, pp. 4–24, 2019.

[24] N. Kaur, G. Kunapuli, S. Joshi, K. Kersting, and S. Natarajan, "Neural networks for relational data," in *Inductive Logic Programming: 29th International Conference, ILP 2019, Plovdiv, Bulgaria, September 3–5, 2019, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2019, p. 62–71. [Online]. Available: https://doi.org/10.1007/978-3-030-49210-6_6

[25] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *International Conference on Learning Representations*, 2017.

[26] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 40–48.

[27] Y. Tian, C. Zhang, Z. Guo, X. Zhang, and N. Chawla, "Learning MLPs on graphs: A unified view of effectiveness, robustness, and efficiency," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=Cs3r5KLdoj

[28] R. Ando and T. Zhang, "Learning on graph with laplacian regularization," in *Advances in Neural Information Processing Systems*, B. Schölkopf, J. Platt, and T. Hoffman, Eds., vol. 19. MIT Press, 2006. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2006/file/d87c68a56bc8eb803b44f25abb627786-Paper.pdf

[29] H. Yang, K. Ma, and J. Cheng, "Rethinking graph regularization for graph neural networks," 2020.

[30] Y. Hu, H. You, Z. Wang, Z. Wang, E. Zhou, and Y. Gao, "Graph-mlp: Node classification without message passing in graph," 2021.

[31] B. Jin, G. Liu, C. Han, M. Jiang, H. Ji, and J. Han, "Large language models on graphs: A comprehensive survey," 2024.

[32] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 13–18 Jul 2020, pp. 1725–1735.

[33] Q. Kang, K. Zhao, Y. Song, S. Wang, and W. P. Tay, "Node embedding from neural Hamiltonian orbits in graph neural networks," in *Proc. International Conference on Machine Learning*, Haiwaii, USA, Jul. 2023.

[34] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *International Conference on Learning Representations*, 2022.

[35] S. H. Lee, F. Ji, and W. P. Tay, "Learning on heterogeneous graphs using high-order relations," *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3175–3179, 2021.

[36] K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay, "Graph neural convection-diffusion with heterophily," in *Proc. International Joint Conference on Artificial Intelligence*, Macao, China, Aug 2023.

[37] Z. Chen, L. Chen, S. Villar, and J. Bruna, "Can graph neural networks count substructures?" in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Curran Associates Inc., 2020.

[38] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=ryGs6iA5Km

[39] Y. Rong, W. Huang, T. Xu, and J. Huang, "Dropedge: Towards deep graph convolutional networks on node classification," *International Conference on Learning Representations*, 2020.

[40] J. Topping, F. D. Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *International Conference on Learning Representations*, 2022.

[41] A. Deac, M. Lackenby, and P. Veličković, "Expander graph propagation," 2022.

[42] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lió, and M. Bronstein, "Weisfeiler and lehman go topological: Message passing simplicial networks," in

Proc. of the 38th International Conference on Machine Learning (ICML), 2021, pp. 1026–1037.

[43] B. Weisfeiler and A. Lehman, *A reduction of a graph to a canonical form and an algebra arising during this reduction*. Nauchno-Technicheskaya Informatsia, 1968.

[44] C. K. Joshi, C. Bodnar, S. V. Mathis, T. Cohen, and P. Lio, "On the expressive power of geometric graph neural networks," 2023. [Online]. Available: https://openreview.net/forum?id=Rkxj1GXn9_

[45] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020.

[46] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *International Conference on Learning Representations*, 2016.

[47] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 1025–1035.

[48] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=S1e2agrFvS

[49] H. Maron, H. Ben-Hamu, N. Shamir, and Y. Lipman, "Invariant and equivariant graph networks," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Syx72jC9tm

[50] E. Dai, W. Jin, H. Liu, and S. Wang, "Towards robust graph neural networks for noisy graphs with sparse labels," in *Proc. of the 15th ACM International WSDM Conference*, 2022.

[51] F. Ji, S. H. Lee, H. Meng, K. Zhao, J. Yang, and W. P. Tay, "Leveraging label

non-uniformity for node classification in graph neural networks," in *International Conference on Machine Learning*, 2023.

[52] J. You, R. Ying, and J. Leskovec, "Position-aware graph neural networks," in *International Conference on Machine Learning*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:174800449

[53] S. Nishad, S. Agarwal, A. Bhattacharya, and S. Ranu, "Graphreach: Position-aware graph neural network using reachability estimations," 2021.

[54] Q. Sun, J. Li, H. Yuan, X. Fu, H. Peng, C. Ji, Q. Li, and P. S. Yu, "Position-aware structure learning for graph topology-imbalance by relieving under-reaching and over-squashing," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, ser. CIKM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1848–1857. [Online]. Available: https://doi.org/10.1145/3511808.3557419

[55] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=wTTjnvGphYj

[56] W. Park, W. Chang, D. Lee, J. Kim, and S. won Hwang, "Grpe: Relative positional encoding for graph transformer," 2022.

[57] E. Hwang, V. Thost, S. S. Dasgupta, and T. Ma, "An analysis of virtual nodes in graph neural networks for link prediction (extended abstract)," in *The First Learning on Graphs Conference*, 2022. [Online]. Available: https://openreview.net/forum?id=dI6KBKNRp7

[58] L. Haonan, S. H. Huang, T. Ye, and G. Xiuyan, "Graph star net for generalized multi-task learning," 2019.

[59] T. Pham, T. Tran, H. Dam, and S. Venkatesh, "Graph classification via deep learning with virtual nodes," 2017.

[60] K. Ishiguro, S. ichi Maeda, and M. Koyama, "Graph warp module: an auxiliary module for boosting the power of graph neural networks in molecular graph analysis," 2019.

[61] J. Li, D. Cai, and X. He, "Learning graph-level representation for drug discovery," 2017.

[62] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20. Curran Associates Inc., 2020.

[63] D. Jin, Z. Yu, C. Huo, R. Wang, X. Wang, D. He, and J. Han, "Universal graph convolutional networks," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=MSXDyfli9vy

[64] S. Luan, C. Hua, Q. Lu, J. Zhu, M. Zhao, S. Zhang, X.-W. Chang, and D. Precup, "Revisiting heterophily for graph neural networks," *Advances in neural information processing systems*, vol. 35, pp. 1362–1375, 2022.

[65] X. Zheng, Y. Liu, S. Pan, M. Zhang, D. Jin, and P. S. Yu, "Graph neural networks for graphs with heterophily: A survey," 2022.

[66] H. Li, J. Cao, J. Zhu, Y. Liu, Q. Zhu, and G. Wu, "Curvature graph neural network," *Information Sciences*, 2021.

[67] Z. Ye, K. S. Liu, T. Ma, J. Gao, and C. Chen, "Curvature graph network," *Proceedings of the 8th International Conference on Learning Representations (ICLR 2020)*, April 2020.

[68] G. Bachmann, G. Bécigneul, and O.-E. Ganea, "Constant curvature graph convolutional networks," *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[69] H. Cho, B. DeMeo, J. Peng, and B. Berger, "Large-margin classification in hyperbolic space," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 89. PMLR, 16–18 Apr 2019, pp. 1832–1840.

[70] M. Yang, M. Zhou, Z. Li, J. Liu, L. Pan, H. Xiong, and I. King, "Hyperbolic graph neural networks: A review of methods and applications," 2022.

[71] I. Chami, Z. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 4869–4880, 2019.

[72] Q. Liu, M. Nickel, and D. Kiela, "Hyperbolic graph neural networks," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019, p. 8230–8241.

[73] Y. Zhang, X. Wang, C. Shi, X. Jiang, and Y. F. Ye, "Hyperbolic graph attention network," *IEEE Transactions on Big Data*, 2021.

[74] Y. Zhang, X. Wang, C. Shi, N. Liu, and G. Song, "Lorentzian graph convolutional networks," in *Proceedings of the Web Conference 2021*. New York, NY, USA: Association for Computing Machinery, 2021, p. 1249–1261.

[75] W. Chen, X. Han, Y. Lin, H. Zhao, Z. Liu, P. Li, M. Sun, and J. Zhou, "Fully hyperbolic neural networks," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, May 2022, pp. 5672–5686.

[76] S. Zhu, S. Pan, C. Zhou, J. Wu, Y. Cao, and B. Wang, "Graph geometry interaction learning," *Advances in Neural Information Processing Systems*, 2020.

[77] L. Sun, Z. Zhang, J. Ye, H. Peng, J. Zhang, S. Su, and P. S. Yu, "A self-supervised mixed-curvature graph neural network," *Association for the Advancement of Artificial Intelligence (AAAI)*, 2022.

[78] R. Hussein, D. Yang, and P. Cudré-Mauroux, "Are meta-paths necessary? revisiting heterogeneous graph embeddings," in *Proc. of the 27th ACM International Conference on Information and Knowledge Management*, 2018, p. 437–446. [Online]. Available: https://doi.org/10.1145/3269206.3271777

[79] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. of the 23rd ACM SIGKDD International Conference on Knowl. Discovery and Data Mining*, 2017, p. 135–144. [Online]. Available: https://doi.org/10.1145/3097983.3098036

[80] X. Yang, M. Yan, S. Pan, X. Ye, and D. Fan, "Simple and efficient heterogeneous graph neural network," in *AAAI Conference on Artificial Intelligence*, 2023.

[81] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *The World Wide Web Conference*, 2019, p. 2022–2032. [Online]. Available: https://doi.org/10.1145/3308558.3313562

[82] X. Fu, J. Zhang, Z. Meng, and I. King, "MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding," in *Proc. of The Web Conference 2020*, 2020, p. 2331–2341. [Online]. Available: https://doi.org/10.1145/3366423.3380297

[83] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *Proc. of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[84] Y. Fang, X. Zhao, P. Huang, W. Xiao, and M. de Rijke, "M-hin: Complex embeddings for heterogeneous information networks via metagraphs," in *Proc. of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR'19, 2019, p. 913–916. [Online]. Available: https://doi.org/10.1145/3331184.3331281

[85] A. Sankar, X. Zhang, and K. C.-C. Chang, "Meta-gnn: Metagraph neural network for semi-supervised learning in attributed heterogeneous information networks," in *Proc. of the 2019 IEEE/ACM International Conference*

on Advances in Social Networks Analysis and Mining, 2019. [Online]. Available: https://doi.org/10.1145/3341161.3342859

[86] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 793–803.

[87] Q. Lv, M. Ding, Q. Liu, Y. Chen, W. Feng, S. He, C. Zhou, J. Jiang, Y. Dong, and J. Tang, "Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21, 2021, p. 1150–1160.

[88] H. Hong, H. Guo, Y. Lin, X. Yang, Z. Li, and J. Ye, "An attention-based graph neural network for heterogeneous structural learning," 2019.

[89] S. Ebli, M. Defferrard, and G. Spreemann, "Simplicial neural networks," *NeurIPS Workshop in Topological Data Analysis and Beyond*, 2020. [Online]. Available: https://arxiv.org/abs/2010.03633

[90] J. Li, Y. Sun, and M. Shao, "Multi-order relations hyperbolic fusion for heterogeneous graphs," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, ser. CIKM '23. Association for Computing Machinery, 2023, p. 1358–1367.

[91] W. Yu and J. McCann, "Random walk with restart over dynamic graphs," in *IEEE International Conference on Data Mining*. IEEE, 2016, pp. 589–598.

[92] S. Barbarossa and S. Sardellitti, "Topological signal processing over simplicial complexes," *IEEE Trans. on Signal Processing*, vol. PP, pp. 1–1, 03 2020.

[93] J. M. Lee, *Introduction to Riemannian Manifolds*. Springer Cham, 2018.

[94] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney, "Tree-like structure in large

social and information networks," in *IEEE 13th International Conference on Data Mining*, 2013, pp. 1–10.

[95] M. Bridson and A. Haefliger, *Metric Spaces of Non-Positive Curvature*. Springer, 1999.

[96] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 3438–3445, 2020.

[97] B. P. Chamberlain, J. Rowbottom, M. I. Gorinova, S. D. Webb, E. Rossi, and M. M. Bronstein, "GRAND: Graph neural diffusion," *The Symbiosis of Deep Learning and Differential Equations*, 2021.

[98] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen, "Decoupling the depth and scope of graph neural networks," *Advances in Neural Information Processing Systems*, 2021.

[99] J. Liu, M. Yang, M. Zhou, S. Feng, and P. Fournier-Viger, "Enhancing hyperbolic graph embeddings via contrastive learning," 2022.

[100] C. Villani, *Optimal Transport: Old and New*. Springer Science & Business Media, 2008.

[101] J. Gao, X. Huang, and J. Li, "Unsupervised graph alignment with wasserstein distance discriminator," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 426–435.

[102] M. Rowland, J. Hron, Y. Tang, K. Choromanski, T. Sarlos, and A. Weller, "Orthogonal estimation of wasserstein distances," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 89. PMLR, 16–18 Apr 2019, pp. 186–195.

[103] Y. Chen, C. Li, and Z. Lu, "Computing wasserstein-p distance between images with linear cost," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 509–518.

[104] S. Kolouri, K. Nadjahi, U. Simsekli, R. Badeau, and G. Rohde, "Generalized sliced wasserstein distances," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[105] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jul 2017, pp. 5425–5434.

[106] B. P. Chamberlain, J. Rowbottom, D. Eynard, F. Di Giovanni, D. Xiaowen, and M. M. Bronstein, "Beltrami flow and neural diffusion on graphs," *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[107] S. Liu, R. Ying, H. Dong, L. Li, T. Xu, Y. Rong, P. Zhao, J. Huang, and D. Wu, "Local augmentation for graph neural networks," *International Conference on Machine Learning*, 2022.

[108] W. Feng, J. Zhang, Y. Dong, Y. Han, H. Luan, Q. Xu, Q. Yang, E. Kharlamov, and J. Tang, "Graph random neural networks for semi-supervised learning on graphs," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2020, p. 22092–22103.

[109] B. Xiong, S. Zhu, N. Potyka, S. Pan, C. Zhou, and S. Staab, "Pseudo-riemannian graph convolutional networks," *Advances in Neural Information Processing Systems*, 2022.

[110] C. Shi, B. Hu, W. X. Zhao, and P. S. Yu, "Heterogeneous information network embedding for recommendation," *IEEE Trans. on Knowledge and Data Engineering*, vol. 31, no. 2, p. 357–370, Feb. 2019. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2833443

[111] C. Yang, Y. Feng, P. Li, Y. Shi, and J. Han, "Meta-graph based HIN spectral embedding: methods, analyses and insights," in *ICDM*, 2018.

[112] E. Bunch, Q. You, G. Fung, and V. Singh, "Simplicial 2-complex convolutional neural networks," *NeurIPS 2020 Workshop on Topological Data Analysis and Beyond*, 2020. [Online]. Available: https://openreview.net/forum?id=TLbnsKrt6J-

[113] T. Roman, A. Nayyeri, B. Fasy, and R. Schwartz, "A simplicial complex-based approach to unmixing tumor progression data," *BMC bioinformatics*, vol. 16, p. 254, 12 2015.

[114] C. Giusti, R. Ghrist, and D. S. Bassett, "Two's company, three (or more) is a simplex," *Journal of Computational Neuroscience*, vol. 41, no. 1, pp. 1–14, 2016. [Online]. Available: https://doi.org/10.1007/s10827-016-0608-6

[115] A. Hatcher, *Algebraic topology*. Cambridge: Cambridge Univ. Press, 2000. [Online]. Available: https://cds.cern.ch/record/478079

[116] GUDHI Project, *GUDHI User and Reference Manual*. GUDHI Editorial Board, 2015. [Online]. Available: http://gudhi.gforge.inria.fr/doc/latest/

[117] K. Borgwardt, E. Ghisu, F. Llinares-López, L. O'Bray, and B. Rieck, *Graph Kernels: State-of-the-Art and Future Challenges*. Foundations and Trends® in Machine Learning, 2020, vol. 13.

[118] M. Horn, E. D. Brouwer, M. Moor, Y. Moreau, B. Rieck, and K. Borgwardt, "Topological graph neural networks," *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=oxxUMeFwEHd

[119] Y. Zhang, S. Pal, M. Coates, and D. Üstebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*. AAAI Press, 2019. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33015829

[120] M. Dong and Y. Kluger, "Towards understanding and reducing graph structural noise for gnns," in *Proceedings of the 40th International Conference on Machine Learning*, ser. ICML'23. JMLR.org, 2023.

[121] R. Li, X. Yuan, M. Radfar, P. Marendy, W. Ni, T. O'Brien, and P. Casillas-Espinosa, "Graph signal processing, graph neural network and graph learning on biological data: A systematic review," *IEEE Reviews in Biomedical Engineering*, Oct 2021.

[122] F. Ji, W. P. Tay, and A. Ortega, "Graph signal processing over a probability space of shift operators," *IEEE Transactions on Signal Processing*, vol. 71, pp. 1159–1174, 2023.

[123] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[124] P. V. Michael Defferrard, Xavier Bresson, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. of the 29th International Conference on Neural Information Processing Systems*, 2016.

[125] B. Guedj, "A primer on PAC-Bayesian learning," *arXiv preprint arXiv:1901.05353*, 2019.

[126] C. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[127] Y. W. Teh, D. Newman, and M. Welling, "A collapsed variational bayesian inference algorithm for latent dirichlet allocation," in *Proc. of the 19th International Conference on Neural Information Processing Systems*, 2006.

[128] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proceedings of the International World Wide Web Conference*, 2020, pp. 2704–2710.

[129] Y. Wang, K. Yi, X. Liu, Y. G. Wang, and S. Jin, "ACMP: Allen-Cahn message passing with attractive and repulsive forces for graph neural networks," in

International Conference on Learning Representations, 2023. [Online]. Available: https://openreview.net/forum?id=4fZc_79Lrqs

[130] H. Pei, B. Wei, K. C.-C. Chang, Y. Lei, and B. Yang, "Geom-gcn: Geometric graph convolutional networks," 2020.

[131] Z. Shui and G. Karypis, "Heterogeneous molecular graph neural networks for predicting molecule properties," in *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE Computer Society, Nov 2020, pp. 492–500.

[132] C. Shen, J. Luo, and K. Xia, "Molecular geometric deep learning," 2023.

[133] N. Thomas, T. Smidt, S. Kearnes, L. Yang, L. Li, K. Kohlhoff, and P. Riley, "Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds," 2018.

[134] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of Computer-Aided Molecular Design*, pp. 595–608, 2016.

[135] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 06–11 Aug 2017, pp. 1263–1272.

[136] Z. Xiong, D. Wang, X. Liu, F. Zhong, X. Wan, X. Li, Z. Li, X. Luo, K. Chen, H. Jiang, and M. Zheng, "Pushing the boundaries of molecular representation for drug discovery with the graph attention mechanism," *Journal of Medicinal Chemistry*, pp. 8749–8760, 2020.

[137] R. Yang and J. Berger, *A Catalog of Noninformative Priors*, ser. Discussion papers. Institute of Statistics and Decision Sciences, Duke University, 1996. [Online]. Available: https://books.google.com.sg/books?id=mghZHQAACAAJ

[138] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 257–266. [Online]. Available: https://doi.org/10.1145/3292500.3330925

[139] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. Association for Computing Machinery, 2018, p. 1416–1424.

[140] J. Chen, T. Ma, and C. Xiao, "FastGCN: Fast learning with graph convolutional networks via importance sampling," in *International Conference on Learning Representations*, 2018.

[141] X. Liang, Z. Yang, B. Wang, S. Hu, Z. Yang, D. Yuan, N. Z. Gong, Q. Li, and F. He, "Unveiling fake accounts at the time of registration: An unsupervised approach," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3240–3250. [Online]. Available: https://doi.org/10.1145/3447548.3467094

[142] Z. Li, H. Wang, P. Zhang, P. Hui, J. Huang, J. Liao, J. Zhang, and J. Bu, "Live-streaming fraud detection: A heterogeneous graph neural network approach," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3670–3678. [Online]. Available: https://doi.org/10.1145/3447548.3467065

[143] P. Agarwal, M. Srivastava, V. Singh, and C. Rosenberg, "Modeling user behavior with interaction networks for spam detection," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2437–2442. [Online]. Available: https://doi.org/10.1145/3477495.3531875

[144] T. Pourhabibi, K.-L. Ong, B. H. Kam, and Y. L. Boo, "Fraud detection: A systematic literature review of graph-based anomaly detection approaches," *Decision Support Systems*, vol. 133, p. 113303, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923620300580

[145] Y. Luo, G. Wang, Y. Liu, J. Yue, W. Cheng, and B. Fei, "Faf: A risk detection framework on industry-scale graphs," in *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, ser. CIKM '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 4717–4723. [Online]. Available: https://doi-org.remotexs.ntu.edu.sg/10.1145/3583780.3615477

[146] Y. Dou, Z. Liu, L. Sun, Y. Deng, H. Peng, and P. S. Yu, "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 315–324. [Online]. Available: https://doi.org/10.1145/3340531.3411903

[147] L. Wang, P. Li, K. Xiong, J. Zhao, and R. Lin, "Modeling heterogeneous graph network on fraud detection: A community-based framework with attention mechanism," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1959–1968. [Online]. Available: https://doi.org/10.1145/3459637.3482277

[148] Y. Song, Q. Kang, S. Wang, K. Zhao, and W. P. Tay, "On the robustness of graph neural diffusion to topology perturbations," in *Advances in Neural Information Processing Systems (NeurIPS)*, New Orleans, USA, Nov. 2022.

[149] B. Xie, H. Chang, Z. Zhang, X. Wang, D. Wang, Z. Zhang, R. Ying, and W. Zhu, "Adversarially robust neural architecture search for graph neural networks," 2023.

[150] X. Zhang and M. Zitnik, "Gnnguard: Defending graph neural networks against adversarial attacks," in *Proceedings of the 34th International Conference on Neural*

Information Processing Systems, ser. NIPS'20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[151] N. Entezari, S. A. Al-Sayouri, A. Darvishzadeh, and E. E. Papalexakis, "All you need is low (rank): Defending against adversarial attacks on graphs," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, ser. WSDM '20. Association for Computing Machinery, 2020, p. 169–177.

[152] H. Wu, C. Wang, Y. Tyshetskiy, A. Docherty, K. Lu, and L. Zhu, "Adversarial examples for graph data: Deep insights into attack and defense," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, ser. IJCAI'19. AAAI Press, 2019, p. 4816–4823.

[153] D. Zhu, Z. Zhang, P. Cui, and W. Zhu, "Robust graph convolutional networks against adversarial attacks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. Association for Computing Machinery, 2019, p. 1399–1407.

[154] F. Mujkanovic, S. Geisler, S. Günnemann, and A. Bojchevski, "Are defenses for graph neural networks robust?" in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=yCJVkELVT9d

[155] K. Zhao, Q. Kang, Y. Song, R. She, S. Wang, and W. P. Tay, "Adversarial robustness in graph neural networks: A Hamiltonian energy conservation approach," in *Advances in Neural Information Processing Systems*, New Orleans, USA, Dec. 2023.

[156] Q. KANG, Y. Song, Q. Ding, and W. P. Tay, "Stable neural ODE with lyapunov-stable equilibrium points for defending against adversarial attacks," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=9CPc4EIr2t1

[157] Y. Song, Q. Kang, and W. P. Tay, "Error-correcting output codes with ensemble diversity for robust learning in neural networks," in *AAAI Conference on Artificial Intelligence*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:234353506

[158] H. Cho, Y. Oh, and E. Jeon, "Seen: Sharpening explanations for graph neural networks using explanations from neighborhoods," 2021.

[159] Y. Li, J. Zhou, S. Verma, and F. Chen, "A survey of explainable graph neural networks: Taxonomy and evaluation metrics," 2023.

[160] Z. Zhang, Q. Liu, H. Wang, C. Lu, and C. Lee, "Protgnn: Towards self-explaining graph neural networks," *arXiv preprint arXiv:2112.00911*, 2021.

[161] W. Xia, M. Lai, C. Shan, Y. Zhang, X. Dai, X. Li, and D. Li, "Explaining temporal graph models through an explorer-navigator framework," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=BR_ZhvcYbGJ

[162] D. Luo, W. Cheng, D. Xu, W. Yu, B. Zong, H. Chen, and X. Zhang, "Parameterized explainer for graph neural network," 2020.

[163] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnnexplainer: Generating explanations for graph neural networks," 2019.

[164] L. Faber, A. K. Moghaddam, and R. Wattenhofer, "When comparing to ground truth is wrong: On evaluating gnn explanation methods," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, ser. KDD '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 332–341. [Online]. Available: https://doi.org.remotexs.ntu.edu.sg/10.1145/3447548.3467283

[165] F. Ji, G. Kahn, and W. P. Tay, "Signal processing on simplicial complexes with vertex signals," *IEEE Access*, pp. 1–1, 2022.

[166] Y. Cen, Z. Hou, Y. Wang, Q. Chen, Y. Luo, Z. Yu, H. Zhang, X. Yao, A. Zeng, S. Guo, Y. Dong, Y. Yang, P. Zhang, G. Dai, Y. Wang, C. Zhou, H. Yang, and J. Tang, "Cogdl: A comprehensive library for graph deep learning," in *Proceedings of the ACM Web Conference 2023 (WWW'23)*, 2023.

[167] H. Han, T. Zhao, C. Yang, H. Zhang, Y. Liu, X. Wang, and C. Shi, "Openhgnn: An open source toolkit for heterogeneous graph neural network," in *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, ser. CIKM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3993–3997. [Online]. Available: https://doi.org/10.1145/3511808.3557664

[168] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," *arXiv preprint arXiv:1909.01315*, 2019.

[169] M. Li, J. Zhou, J. Hu, W. Fan, Y. Zhang, Y. Gu, and G. Karypis, "Dgl-lifesci: An open-source toolkit for deep learning on graphs in life science," *ACS Omega*, 2021.

[170] M. Hardt, B. Recht, and Y. Singer, "Train faster, generalize better: stability of stochastic gradient descent," in *Proceedings of the 33th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 48. PMLR, Jun 2016, pp. 1225–1234.