DEEP LEARNING ON GRAPH-STRUCTURED DATA

by

TONG ZEKUN
(B.S., Xidian University)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

INDUSTRIAL SYSTEMS ENGINEERING

in the

GRADUATE DIVISION

of the

NATIONAL UNIVERSITY OF SINGAPORE

2022

Supervisor:
Professor Chee Yeow Meng

Examiners:
Associate Professor Huang Zhiyong
Assistant Professor Li Xiaobo

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Tong Zekun

Tong Zekun

07 February 2023

To my dear parents

Acknowledgments

Upon completion of this Ph.D. thesis, I would like to extend my deepest gratitude to those who have supported me over the past four years. The opportunity to earn a doctorate is made possible by their professional guidance and kind patience.

I would like to express my sincere gratitude to Prof. Chee Yeow Meng and Prof. Andrew Lim for their valuable suggestions and supervision during the course of my Ph.D. degree. I would also like to thank Prof. David S. Rosenblum, Prof. Chen Caihua, Prof. Li Chongshou and Prof. Tang Jing for their treasured help in my research path. Moreover, I would like to thank Dr. Liang Yuxuan for his hands-on guidance which was really influential in shaping my research topic. Meanwhile, I would like to express appreciation to my co-authors and discussants: Dr. Wu Yuwei, Mr. Li Xinke, Mr. Dai Yongxing, Mr. Sun Changsheng, Dr. Ding Henghui, Mr. Cai Xu and Mr. Wang Jingyang. This thesis is inseparable from their abundant advice.

In addition, I wish to gratefully mention my teammates, colleagues, and friends who have been extremely helpful to me in my research and life journey. They are, Dr. Zhao Huangjie, Dr. Zhao Yue, Dr. Huang Yuming, Dr. Wang Shixiong, Dr. Li Jingwen, Mr. Hu Kanxin, Mr. Lu Zedi, Mr. Cong Qianhao, Mr. Yuan Yiliang, Dr. Che Yuxin, Dr. Pan Binbin, Dr. Wei Xiaoyang, Ms. Zhao Jingyi, Mr. Yabang Zhao, Mr. Ma Yining, Mr. Chen Zhirui, Dr. Zhang Hui, Mr. Xie Jinbin, Dr. Zhu Yuqin, and many other friends.

Finally, I would like to express my gratitude to my parents and my girlfriend, who were always there cheering me up and stood by me through up-and-downs during the past few years.

Contents

Abstract

Deep Learning on Graph-structured Data

by

Tong Zekun

Doctor of Philosophy in Industrial Systems Engineering

National University of Singapore

Deep learning methods are becoming widely attractive for tackling difficult real-world problems. It can not only handle grid-like data, such as images, but also process non-Euclidean data, such as graphs. Graph-structured data is ubiquitous, from abstract molecular structures to figurative road networks, it records connections between entities with an efficient data structure. As a generalized deep learning architecture for graphs, Graph Neural Networks (GNNs) are widely utilized to process this graph-structured data in a variety of real-world tasks, including traffic prediction, social network analysis, drug discovery, and etc.

GNNs can derive variants suitable for different graph types, such as directed and undirected, homogeneous and heterogeneous, and static and dynamic graphs, by adjusting the model structure. Among these graph types, the use of GNNs in directed graphs is still in its infancy. For directed graphs, edges describe the relationships and interactions between distinct nodes, particularly the orientation, which is a specific attribute. Existing GNN approaches and frameworks are still insufficient to process directed graphs in terms of spectral-based convolution, model structure, and learning methods. To expand the field of graph deep learning into directed graphs, we propose several methods to address these limitations.

Firstly, we extend spectral-based graph convolution to directed graphs using first- and second-order proximity, which can not only retain the connection properties of the directed graph, but also expand the receptive field of the convolution operation. A new directed graph convolutional network (DGCN) model is then designed to learn representations on the directed graph, leveraging both the first- and second-order proximity information. We empirically show the fact that graph convolution working only with DGCNs can encode more useful information from graph and help achieve better performance when generalized to other models.

Secondly, we theoretically extend spectral-based graph convolution to directed graphs and derive a simplified form using *personalized* PageRank. Specifically, we present Directed Graph Inception Convolutional Networks (DiGCN), which utilizes directed graph convolution and k^{th}-order proximity to achieve larger receptive fields and learn multi-scale features in directed graphs. We empirically show that DiGCN can encode more structural information from directed graphs than GCNs and help achieve better performance when generalized to other models.

Finally, we design a directed graph data augmentation method called Laplacian perturbation and theoretically analyze how it provides contrastive information without changing the directed graph structure. Moreover, we present a Directed Graph Contrastive Learning (DiGCL) framework, which dynamically learns from all possible contrastive views generated by the Laplacian perturbation. Then we train it using multi-task curriculum learning to progressively learn from multiple easy-to-difficult contrastive views. We empirically show that our model can retain more structural features of directed graphs than other graph contrastive learning models because of its ability to provide complete contrastive information.

Our works in this thesis expand the use of GNNs to directed graphs in three ways: spectral-based convolution, model structure, and learning method. Given the effectiveness and efficiency demonstrated in experiments, our proposed works outperform other state-of-the-art methods on citation networks and co-purchase datasets. In future, we will further investigate the graph deep learning with various graph types under more complex application scenarios.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The use of deep learning methods to tackle difficult real-world problems is growing in popularity [93]. All kinds of image-related tasks, including classification [92], semantic segmentation [17], and object detection [139, 59], have benefited from the use of these deep learning approaches, and the results have been encouraging. In addition to image recognition, deep learning techniques have shown promising results in the area of natural language processing [23, 111], specifically in the form of neural machine translation [4]. Data for such tasks often lend themselves to grid-based representations. This enables the use of kernel-based operations such as convolution and pooling in the form of local filters that scan each position on the input. Convolutional Neural Networks (CNNs) are a type of deep learning neural network that are commonly used in processing grid-based data [2]. Unlike traditional hand-crafted filters, the local filters used in convolutional layers are trainable. By learning the weights of these trainable filters, the networks can automatically decide what kind of features to extract, thereby eliminating the need for manual feature extraction [174]. However, the real-world task has its charms, not least of which is that the data does not always appear grid-like and instead is distributed in an irregular domain, such as graph-structured data [63].

Graph-structured data is ubiquitous and exists widely in various real-world problem scenarios. Figure 1.1 provides an illustration of graph-structured data in real-world. We can group these scenarios into two main categories: *explicit structure scenarios* and *implicit structure scenarios*. Explicit structure scenarios imply that

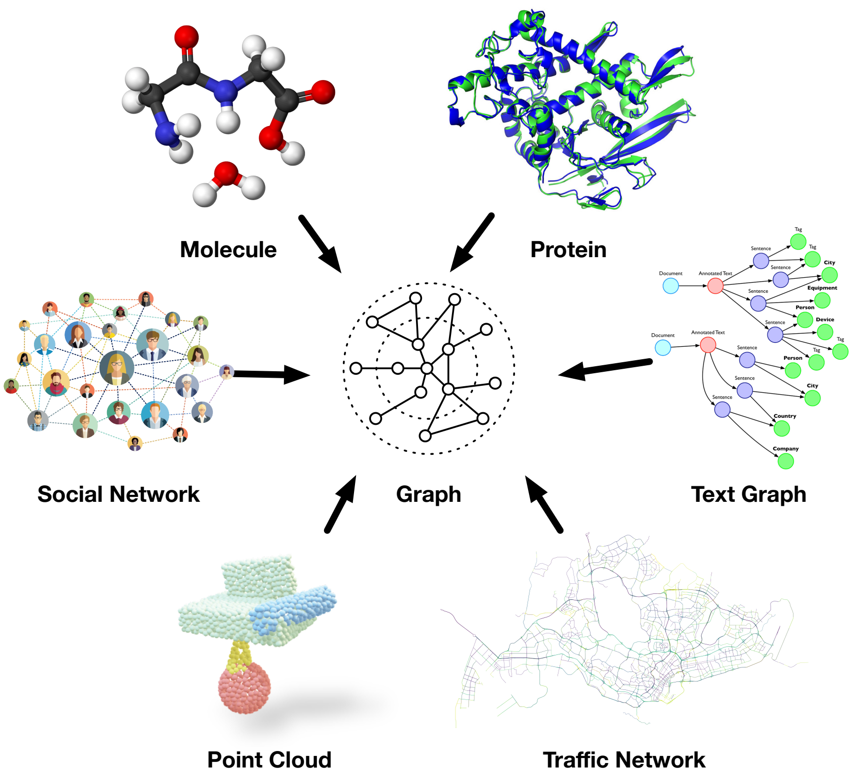


Figure 1.1: Graph-structured data in real world.

the data to be analyzed is naturally organized into graph structures, such as traffic networks [150, 108, 6], knowledge graphs [171, 175, 218], recommender systems [137, 29, 154], transaction networks [184, 169, 21] and etc. Implicit structure scenarios require artificially mining and organizing into graph structures based on the original data. For example, convert connections between images/point cloud into graph [196, 146], or connections between texts [74, 217].

As an extension of deep learning beyond grid-like domains, Graph Neural Networks (GNNs) [55, 28] are intended to handle non-Euclidean graph structures that are intractable to previous deep learning techniques. They use graph data as input and effectively learn the underlying pairwise relations among data nodes, which are widely used in numerous problems to process graph-structured data. We list some examples of real-world problems using GNNs to learn from graph data as follows.

- **Chemistry.** Several GNN models have been proposed to solve chemistry-related problems, which involve molecular systems [44, 143, 32], crystalline materials [197, 14], surface chemistry [125, 3], heterogeneous catalysis [50], etc.

The structures are described in terms of topological relationships, where atoms are represented as nodes, and the relationships between them are represented as edges.

- **Biology.** Prediction of disease association [128, 172], drug development and discovery [76, 106], prediction of interactions of biological entities [30, 37] are common areas where GNNs are currently used in bioinformatics [215].
- **Traffic Networks.** Traffic network is one of the most exposed, common, and visible graph structures in people's daily life. GNN-based model can be used for a wide range of traffic prediction problems, for example, traffic flow forecast [150, 108], traffic speed forecast [208, 210], human/vehicle trajectory prediction [99, 116], taxi supply and demand forecast [42, 6], etc.
- **Recommender Systems.** Most of the information in a recommender system has a graph structure, such as social relationships [36], knowledge graphs [171, 175], user-item interaction graphs [80], and etc. GNNs are capable of capturing higher-order interaction and can effectively integrate edge information from social relationships and knowledge graphs [190].
- **Computer Vision (CV).** An image can be represented as a spatial map, and image regions are frequently spatially and semantically interdependent. Video can also be represented as spatio-temporal graphs, with each node representing a region of interest in the video and the edges representing the relationships between such regions [188]. GNNs can be utilized naturally to extract patterns from these graphs and facilitate associated CV tasks, such as image object detection [199], image classification [67], video classification [176], etc. GNNs are also widely used in 3D CV, for example in point clouds [183, 104].
- **Natural Language Processing (NLP).** Graph is a natural method to capture the relationships between various components of the text, such as entities, sentences, and documents, to exploit the underlying linguistic and semantic structures of the text [188]. For NLP tasks that involve labeling words or phrases, they can be converted to a node classification task [204, 107]. In addition, predicting the relationships between two text elements is also a vital problem [53, 95].

1.2 Motivations

Due to the diversity of graph types, such as directed/undirected graphs, homogeneous/heterogeneous graphs, etc., GNNs need to design different networks for the structural characteristics of different input graph data [188]. Among these graph types, learning from directed graph data to solve practical problems is gaining popularity, such as neural architecture representation and automatic design [13, 31], causality inference [79, 62] and the implication graph of SAT solvers [194]. However, the presence of rich directed structural features and asymmetric connectivity relations in **directed graphs** poses a challenge for applying deep graph neural networks to directed graphs. In this thesis, we are dedicated to explore this new area, extending GNNs to directed graphs.

Graph Convolutional Networks (GCNs) are the most versatile branch of GNNs. They are variations of the classic CNNs, which are typically used for image and video processing. The illustration is shown in Figure 1.2. The key difference is that GCNs use graph-based convolutional layers to process the data, rather than traditional grid-based convolutional layers. This allows GCNs to capture complex relationships and dependencies between nodes in a graph, making them well suited for tasks such as node classification and link prediction.

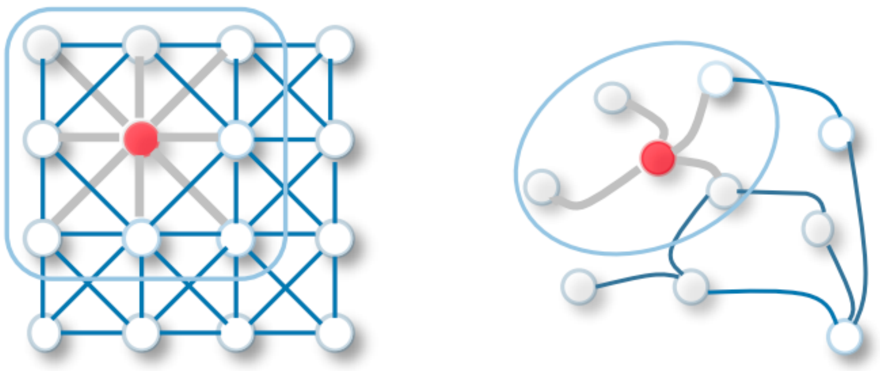


Figure 1.2: Illustration of CNNs (left) and GNNs (right) [192]. CNNs are specifically tailored to process regular, structured data in Euclidean spaces, while GNNs are more generalized versions that can handle irregular or non-Euclidean structured data with variable node connections and unordered nodes.

However, most spectral-based GCNs work only with undirected graphs [193]. They transform directed graphs to undirected by relaxing their direction structure [88, 187], i.e. adding edges to symmetric the adjacency matrices. It will mislead message passing schemes to aggregate features with incorrect weights and discard distinctive direction structure. In a citation graph, later articles can cite earlier ones, but not vice versa. Several works define the structure's motifs and inheritance relationship [118, 84]. These methods require learning templates or rules and cannot handle complex structures beyond their definitions. Hence, we hope to design efficient, generalized and theoretically guaranteed directed graph convolutional networks.

In addition to theoretical extensions for directed graphs, the network structure has room for improvement. During each convolution operation, in most existing spectral-based GCN models, only 1-hop node features are considered, capturing only first-order information. When extracting local features, it is natural to consider direct links, but this is not always the case. Many legitimate relationships in real-world graph may not be encoded via first-order links [156]. For example, social networking members who share similar interests may not always communicate with each other. In other words, it is likely that the features obtained by first-order proximity are insufficient. Although it is possible to obtain more information by stacking multiple layers of GCNs, this strategy frequently results in feature dilution and overfitting issues as models become deeper [84, 97]. Some works [140, 198, 1], inspired by the Inception Network for image classification [155], broaden their layers to obtain larger receptive fields and enhance their learning capacities. However, they employ a fixed adjacency matrix in a single layer, which makes it more challenging to capture multi-scale features. A scalable neighborhood would provide additional scale information, particularly for nodes belonging to communities of varying sizes. Therefore, how to design the model network structure so that it can be applied to directed graphs is also an urgent issue to be solved.

Moreover, existing GNNs [113, 160, 161] are trained end-to-end under supervision. This training scheme shows excellent performance due to sufficient labeled data. Consequently, several Graph Contrastive Learning (GCL) works [207, 136, 57] are proposed based on GNNs and Contrastive Learning [58] to make use of rich unlabeled data. However, these methods encounter problems when processing directed graphs, particularly with the data augmentation method and the contrastive learning framework. First, most of the data augmentation methods used in GCL [207, 229, 228] do not take into account the directed graph structure and may discard distinctive direction information. For example, the idea of dropping nodes/edges [228, 207, 180] is borrowed from random erasing used in images [220], which disregards the disparity between nodes and edges in various graph structures. Furthermore, conventional contrastive learning frameworks are not optimized for directed graphs and can only learn from a limited number of contrastive views [207, 229, 57]. However, due to the complexity of directed graphs, a small number of contrastive views is insufficient to fully comprehend their structural characteristics [160]. We want to design self-supervised models applicable to directed graphs for efficient contrastive learning of unlabeled data based on the aforementioned challenges.

1.3 Objectives and Contributions

In this thesis, we focus on extending the application of deep graph neural networks to directed graphs. From the motivations mentioned above, our objectives can be summarized as: (1) to theoretically expand the graph spectral-based convolution into directed graphs, (2) to design graph neural network structures that can be used in directed graphs, and (3) to explore how to learn structural features from directed graphs in the absence of supervision information.

To achieve the above research objectives, we first study the limitations of current graph convolutional networks in directed graphs and construct a second-order proximity network to learn their structural information. Second, we theoretically extend the spectral-based graph convolution to directed graphs and build an Inception network to learn multi-scale features. Third, we investigate self-supervised graph learning and propose a novel data augmentation approach and a contrastive learning framework for encoding directed graph features. The main contributions of this thesis are summarized below.

Second-order Graph Convolution: We extend spectral-based graph convolution to directed graphs using first- and second-order proximity, which can not only retain the connection properties of the directed graph, but also expand the receptive field of the convolution operation. A new GCN model, called DGCN, is then designed to learn representations on the directed graph, leveraging both the first- and second-order proximity information. We empirically show the fact that GCNs working only with DGCNs can encode more useful information from graph and help achieve better performance when generalized to other models. Moreover, extensive experiments on citation networks and co-purchase datasets demonstrate the superiority of our model over the state-of-the-art methods.

PageRank-based Graph Convolution: We theoretically extend spectral-based graph convolution to directed graphs and derive a simplified form using *personalized* PageRank. Specifically, we present Directed Graph Inception Convolutional Networks (DiGCN), which utilizes directed graph convolution and k^{th}-order proximity to achieve larger receptive fields and learn multi-scale features in directed graphs. We empirically show that DiGCN can encode more structural information from directed graphs than GCNs and help achieve better performance when generalized to other models. Moreover, experiments on various benchmarks demonstrate its superiority over state-of-the-art methods.

Contrastive Graph Learning: We design a directed graph data augmentation method called Laplacian perturbation and theoretically analyze how it provides contrastive information without changing the directed graph structure. Moreover, we present a directed graph contrastive learning framework which dynamically learns from all possible contrastive views generated by the Laplacian perturbation. Then we train it using multi-task curriculum learning to progressively learn from multiple easy-to-difficult contrastive views. We empirically show that our model can retain more structural features of directed graphs than other GCL models because of its ability to provide complete contrastive information. Experiments on various benchmarks reveal our dominance over the state-of-the-art approaches.

1.4 Organization

The roadmap for this thesis is outlined as follows and the main body of this thesis is organized as shown in Figure 1.3.

- **Chapter** 2 reviews related research works in graph neural networks. In this chapter, we outline the general pipeline and main modules of Graph Neural Networks, and further discuss the most important parts: computational modules and learning methods. For each of these two modules, we address the work and give some preliminaries related to graph convolutional networks and graph contrastive learning, respectively.

- **Chapter** 3 studies the limitations of the current graph convolution networks applied to directed graphs and proposes a new network structure based on second-order proximity.
- **Chapter** 4 is concerned with theoretical extensions of spectral-based graph convolution and builds an inception network to learn multi-scale features in directed graphs.
- **Chapter** 5 presents a study on self-supervised learning of directed graph features using data augmentation and contrastive learning in the absence of data labels.
- **Chapter** 6 concludes the thesis and outlines some potential future work.

As shown in Figure 1.3, the three methods we propose are closely related. All three methods are designed to handle directed graph data. Among them, the method DGCN in Chapter 3 is the first graph convolution network for directed graphs. This method leverages first- and second-order proximity to expand the convolution receptive field and retain directed features of the graph. However, DGCN mainly improves the model structure and aggregation method by utilizing some manually defined neighborhood aggregation methods to compensate for information loss caused by converting directed graphs to undirected graphs by increasing second-order neighbors. Hence, this method has theoretical limitations. To bridge the theoretical gap, we present another novel approach in Chapter 4, DiGCN. DiGCN simplifies the spectral-based graph convolution theoretically and extends it to directed graphs by defining k^{th}-order proximity.

DGCN and DiGCN are trained end-to-end under supervision. This training scheme shows excellent performance by virtue of enough labeled data. However, a large amount of data exists in an untagged form. To be able to utilize this data, we propose DiGCL in Chapter 5, a self-supervised learning framework for directed graphs. Based on the theory of *directed graph Laplacian matrix* proposed by DiGCN, DiGCL introduces a data augmentation method that utilizes Laplacian perturbation and curriculum learning to learn from contrastive views.

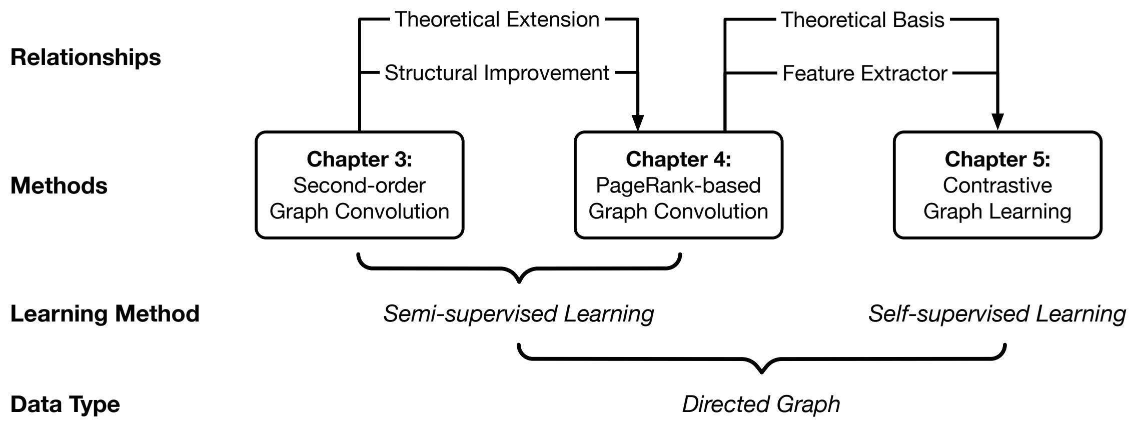


Figure 1.3: The organization of this thesis between three main methods.

In summary, the method DiGCN in Chapter 4 extends the theory and improves the structure based on the method DGCN in Chapter 3, and provides the theoretical basis and feature extractor for the method DiGCL in Chapter 5. Regarding the model learning methods, the methods in Chapter 3 and Chapter 4 use semi-supervised learning, while Chapter 5 uses a self-supervised learning training method.

Throughout this thesis, we will use a unified terminology system. Particularly, we use bold capital letters (*e.g.,* $\mathbf{X}$) and bold lowercase letters (*e.g.,* $\mathbf{x}$) to denote matrices and vectors, respectively. We use non-bold letters (*e.g.,* x) to represent scalars and Greek letters (*e.g.,* λ) as parameters.

Chapter 2

Literature Review and Preliminaries

In this chapter, we review the literature relevant to our topic in three sections. In the first section, we outline the general pipeline and main modules of Graph Neural Networks, and further discuss the most important parts: computational modules and learning methods. In the second section, we address the work related to graph convolutional networks, which is divided into two main types: spatial-based and spectral-based methods. In the third section, we summarize the work on graph contrastive learning, especially the different types of data augmentation schemes. We also give a data augmentation framework and categorize the different data augmentation schemes into this framework.

2.1 Graph Neural Networks

Due to the heterogeneity and complexity of graph-structured data, the analysis and processing of graph-structured data has proven to be a difficult but necessary task [88, 193]. In recent years, Graph Neural Networks (GNNs) have emerged to directly learn graph-structured data. GNNs convert graph-structured data into feature representations and achieve outstanding performance on tasks such as node classification, link prediction, and graph clustering [188]. This shows that GNNs can learn the intrinsic patterns and deeper semantic features of graph-structured data. We introduce the general pipeline for designing GNNs in this section and detail it for the modules that will be covered in this thesis.

2.1.1 General Pipeline

GNNs have diverse functions and widely varying structures. We summarize the general pipeline of GNNs from the designer's perspective [193]. The illustration is shown in Figure 2.1.

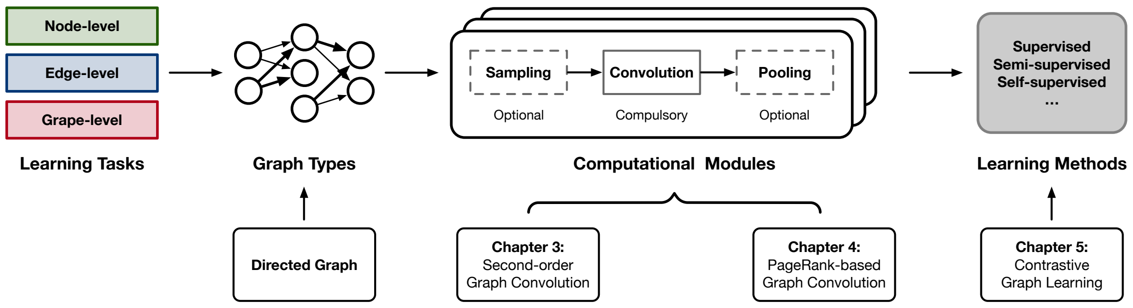


Figure 2.1: General pipeline for a GNN model and the connection with our methods.

When designing a GNN, the first step is to determine the type of **learning task**, whether it is node-level, edge-level, or graph-level. After that, we determine the **graph type** that accommodates the application scenarios and the structure of the actual data. Then, we design the **computational module** based on the determined graph type. The computational modules determine how the GNNs learn information from the graph structure and also how good the output features are. The last thing that needs to be designed is **learning method**. For different practical task scenarios, we need to specify the training mode, whether it is supervised, unsupervised, or self-supervised. Notice that the actual design of GNNs is more intricate, with inter-layer operations, joint learning with external knowledge, hybrid models, etc. Here, only general design concepts are discussed.

Our work focuses on optimizing the graph neural network so that it can adapt to a broader range of graph structures and training patterns. The modules improved by our proposed methods are listed in Figure 2.1. The input data of our methods is directed graph. The method **DGCN** proposed in Chapter 3 and the method **DiGCN** proposed in Chapter 4 mainly improve computational modules, while the method **DiGCL** proposed in Chapter 5 focuses on optimizing learning methods.

2.1.2 Learning Tasks

We usually abstract complex graph problems in the real world into generalized graph tasks. For graph learning tasks, there are typically three types:

- **Node-level tasks** concentrate on nodes and include classification [88, 187], regression [9], and clustering [163] of nodes. Node regression predicts a continuous value for each node, while node classification attempts to classify nodes into multiple classes. The objective of node clustering is to divide the nodes into several distinct groups, with similar nodes grouped together.
- **Edge-level tasks** are edge classification [85] and link prediction [213, 216]. These tasks require the model to categorize edge types and predict whether an edge exists between two nodes.
- **Graph-level tasks** including graph classification [35], regression [119], and matching [102, 39], require the model to learn graph representation from local and global features.

2.1.3 Graph Types

Complex graph types could provide additional information about nodes and their connections. Graphs are typically classified as [193]:

- **Directed/Undirected Graphs.** Directed graphs contain edges that are all directed from one node to another, providing more information than undirected graphs. An undirected graph can be viewed as a special directed graph [160].
- **Homogeneous/Heterogeneous Graphs.** In homogeneous graphs, nodes and edges are of the same type, whereas in heterogeneous graphs, they are of different types. Node/edge types play important roles in heterogeneous graphs and careful consideration is needed when performing modeling [211].
- **Static/Dynamic Graphs.** When input characteristics or the topology of the graph change over time, the graph is considered dynamic. Time-series information is an essential and important part of the dynamic graph [178]. This type of graph is commonly found in traffic networks.

Note that these categories are orthogonal, which means these types can be combined; for example, a dynamic undirected heterogeneous graph is possible. Other graph types, such as hypergraphs and signed graphs, are designed for specific tasks [193]. In this thesis, we mainly consider the extension of GNNs in directed graphs.

2.1.4 Computational Modules

In this section, we introduce the core part of GNNs, the computational modules, and divide them into three categories according to their functions: convolution, sampling, and pooling operation.

Convolution Operation. The convolution operation is used to propagate information between nodes and their neighbors so that the aggregated information could capture both feature and topological information. It is the most important part in GNNs and directly determines how the model learns the features from the graph. It is also the component that numerous methods [28, 88, 165, 187] attempt to enhance. We provide a comprehensive summary of graph convolution, the most closely related topic of our work in Section 2.2.

Sampling Operation. Normally, GNNs aggregate messages from neighboring nodes in each layer for every node. The size of supporting neighbors will increase exponentially with model depth if multiple GNN layers are stacked. Sampling is an efficient and effective method for addressing this neighbor explosion issue. Meanwhile, when dealing with large graphs, it is impossible to store and process all neighborhood information for each node, the sampling operation is required to carry out the feature propagation [223]. Three are three types of graph sampling methods: node sampling [54, 15, 206], layer sampling [16, 75], and subgraph sampling [22, 209]. These different levels of sampling methods can be used individually or in combination to get the best results.

Pooling Operation. For the graph-level task, GNNs need to aggregate the global information of the graph (including all nodes and all edges), and get a vector that can represent the whole graph by aggregating the global information, and subsequently perform classification and regression operations on the vector. Therefore, the graph pooling is designed to extract high-level subgraph or graph representations. There are currently several works designing hierarchical pooling layers on graphs through self-attention [94], top-k sampling [41] and eigendecomposition [112].

2.1.5 Learning Methods

Depending on the usage and data characteristics, there are various learning methods such as supervised, unsupervised, semi-supervised, weakly-supervised and self-supervised learning. Here, we focus on semi-supervised and self-supervised learning, which is closely related to our work.

2.1.5.1 Semi-supervised Learning

In many practical applications of ML, it is easy to find a large number of unclassified labeled samples, but it requires special equipment or an expensive and time-consuming experimental procedure to manually label the samples with class labels [226], resulting in a very small number of samples with class labels and a surplus of unlabeled samples. Therefore, an attempt is made to add a large number of unlabeled samples to a limited number of labeled samples for learning, with the expectation of improving learning performance, resulting in semi-supervised learning [225]. It is intermediate between supervised learning (all labeled data) and unsupervised learning (no labeled data) and solves the problems of model generalization of supervised learning and model inaccuracy of unsupervised learning.

Semi-supervised learning on graph data is beneficial because models using graph data can not only use a small number of given node labels, but can also make full use of structural information to infer unlabeled node types. Semi-supervised learning is used in numerous graph tasks. For example, node classification task aims to predict non-existing node properties (known as the target propert) based on other node properties [88, 187, 90], link prediction task focuses on estimating the probability of links between nodes in a graph [65, 43], and graph classification task seeks to predict target graph labels in a collection of graphs each with an attached categorical label [98, 149, 231]. These simplified but generalized graph learning tasks become the baseline tasks to measure the capability of graph neural networks [33, 70]. In our experiments, we mainly use semi-supervised node classification and link prediction tasks as standard tasks for evaluating models.

2.1.5.2 Self-supervised Learning

Self-supervised learning is a popular area of research that aims to improve the feature extraction ability of models by designing auxiliary tasks (proxy tasks) for unlabeled data to exploit the representational properties of the data itself [77]. During the learning process, the supervision information is obtained from the data itself automatically without the need for manual annotation [109]. Self-supervised learning enables the model to learn more informative representations from unlabeled data and achieve better performance [165, 57], generalization [72, 136, 71], and robustness [207, 83] on a variety of downstream tasks.

We can group graph self-supervised learning into three main categories: generation-based, auxiliary property-based and contrast-based methods [109]. This categorization is based on the network structure and the design of the auxiliary tasks. The following is a discussion of the various types of graph self-supervised learning work.

Generation-based Methods. Generation-based methods are generally designed to reconstruct the original data and use the original data as their supervision information. This category can be linked back to Autoencoder [64], which learns to compress data vectors into low-dimensional representations using an encoder network and then attempts to reconstruct the input vectors using a decoder network. Similar to autoencoders, generation-based graph self-supervised methods usually take the full graph or subgraph as input and learn the graph structure and representation through the process of reconstructing the graph structure. As one of the representative works, VGAE [87] employs an inference model-based encoder to estimate the mean and deviation with two parallel output layers. The Kullback-Leibler divergence is calculated between the prior distribution and the estimated distribution to measure reconstruction loss. To learn additional generative representations for graph data, SIG-VAE [56] takes hierarchical variational inference into account, similar to VGAE. An alternative to reconstructing the entire graph is to recover the masked edges. Denoising Link Reconstruction [73] eliminates existing edges at random and then attempts to recover the discarded links using a decoder based on pairwise similarity.

Auxiliary Property-based Methods. Auxiliary property-based methods obtain node-, link- and graph- level supervision information from original graphs to guide the training process of the model. These methods have a similar training paradigm to supervised learning because they both employ supervision information for learning [109]. Their distinction lies in the method of labeling: In supervised learning, the manual label is human-annotated, which frequently incurs high costs; however, in auxiliary property-based methods, the pseudo label is generated automatically and at little cost. There are primarily two subcategories of auxiliary property-based methods: (1) auxiliary property classification, which uses classification-based proxy tasks to train the encoder [151, 230, 132], and (2) auxiliary property regression, which performs self-supervised learning using regression-based proxy tasks [82, 81].

Contrast-based Methods. Contrast-based methods are based on the idea of mutual information (MI) maximization [66], which acquires knowledge by foreseeing concordance between two augmented instances. Several Graph Contrastive Learning (GCL) [207, 229, 136, 57] works are proposed based on Graph Neural Networks (GNNs) [88, 54, 90] and Contrastive Learning (CL) [19, 162, 58]. As revitalization of the CL used in the visual representation learning [58, 19], GCL forces views generated from the same instance closer while views from different instances apart. As the closely related area, we summarize the contrast-based work in Section 2.3.

The properties of self-supervised methods for graphs vary. Generation-based methods are simple to implement because the reconstruction task is straightforward to construct, but recovering large-scale graphs can be memory intensive. While the design of decoders and loss functions can be kept simple with auxiliary property-based methods, selecting useful auxiliary properties often requires expertise in the relevant domain. Contrast-based methods have more flexible designs and border applications than other categories [109]. Thus, we focus on contrast-based graph self-supervised learning methods (graph contrastive learning) in this thesis.

2.2 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are promising tools to effectively learn from graph-structured data [88, 54, 165]. They follow an iterative neighborhood aggregation (or message passing) scheme to obtain structural knowledge in the neighborhood nodes [20]. Formally, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its adjacency matrix can be denoted as $\mathbf{A} = \{0, 1\}^{n \times n}$, where $n = |\mathcal{V}|$. The nodes are described by the feature matrix $\mathbf{X} \in \mathbb{R}^{n \times c}$, with the feature dimension c per node. Considering a K-layer GCN $f(\cdot)$, the propagation of the k^{th} layer is represented as

$$\boldsymbol{h}_i^{(k)} = \text{COMB}^{(k)}\left(\boldsymbol{h}_i^{(k-1)}, \text{AGG}^{(k)}\left(\left\{\boldsymbol{h}_{i'}^{(k-1)} : i' \in \mathcal{N}(i)\right\}\right)\right), \tag{2.1}$$

where $\mathcal{N}(i)$ is the neighborhood set of node i and $\boldsymbol{h}_i^{(k)}$ is its feature at the k^{th} layer with $\boldsymbol{h}_i^{(0)} = \boldsymbol{x}_i$. AGG$(\cdot)$ controls how features are aggregated from neighbor. Note that the neighborhood set is customizable and is not limited to direct adjacency (existence of directed edges), e.g., in Chapter 3, we have second-order proximity/-connection, and in APPNP [90] any node can be connected. Besides, COMB$(\cdot)$ determines how the front and current-layer features are combined. After the K-layer propagation, we can summarize output embedding of $\mathcal{G}$ through the READOUT function

$$f(\mathcal{G}) = \text{READOUT}\left(\left\{\boldsymbol{h}_i^{(k)} : v_i \in \mathcal{V}, k \in K\right\}\right). \tag{2.2}$$

Above we present a general framework of GCNs. Next, we introduce a few most representative instantiations or variants of GCNs. We can divide GCNs into two categories: spatial-based [54, 165] and spectral-based [88, 101, 28] methods. Spatial-based GCNs operate directly on the nodes and edges of a graph and use a convolution operation to aggregate information. Spectral-based GCNs operate on the graph's Laplacian matrix and use a spectral filtering operation to transform the graph signal before using a spatial convolution operation to aggregate information.

2.2.1 Spatial-based Methods

In spatial-based methods, the topology of the graph is used to define the convolutions directly on the graph. The most difficult aspect of spatial approaches is defining the convolution operation with neighborhoods of varying sizes and preserving the local invariance. In other words, the spatial-based methods mainly require the design of AGG, including neighborhood set $\mathcal{N}(i)$, and COMB functions to make them efficient to aggregate the features of neighbor nodes and propagate to other nodes.

GraphSAGE [54] proposes a general inductive framework that can generate node embeddings for unknown data in an efficient manner. It includes two components: sampling and aggregation. GraphSAGE initially samples the neighbors based on the connection information between nodes, and then continuously aggregates the information of neighbor nodes using a multi-layer aggregation function, which is utilized to predict node labels.

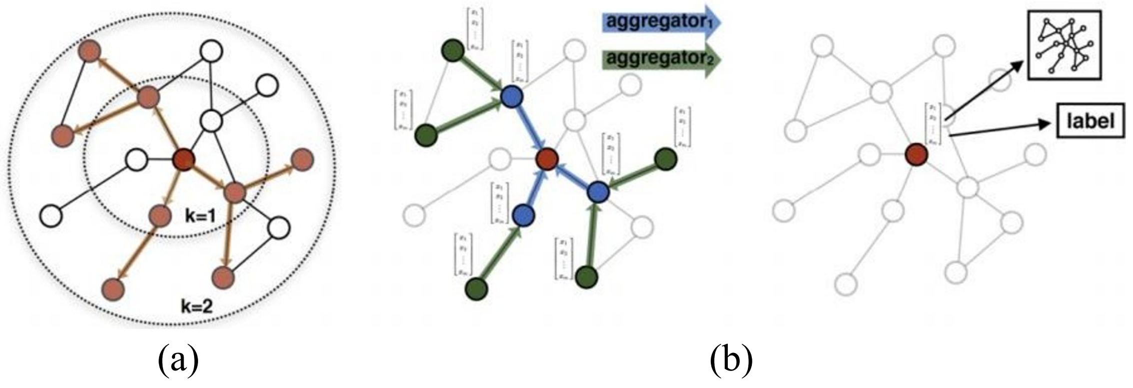


Figure 2.2: Illustration of GraphSAGE components [54]: (a) neighborhood sampling and (b) information aggregation.

As shown in Figure 2.2(a), the k parameter controls the size of the neighborhood used. A value of 1 only considers adjacent nodes as similar, while a value of 2 also includes nodes at a distance of 2. However, increasing k to 2 also allows nodes at a distance of 4 to influence each other's embeddings through a shared intermediary node, potentially introducing unwanted information sharing.

Once the neighborhood has been determined, a method for sharing information among the neighboring nodes needs to be established. Aggregation functions, also known as aggregators, take in the neighborhood as input and combine the embeddings of each neighbor using weights to create a neighborhood embedding. These functions are used to aggregate information from the node's neighborhood. An illustration of this process is shown in Figure 2.2(b). The weights used by the aggregator can be either learned or fixed, depending on the specific function being used. It offers four functions for aggregating nodes: mean, GCN, LSTM, and pooling, reflecting the diversity of spatial-based methods in the selection of aggregation functions. GraphSAGE with a mean aggregator is an inductive variant of GCN.

Several models [165, 212] have been proposed to generalize the attention operator to graphs. Attention-based operators assign different weights for neighbors, so that they could alleviate noises and achieve better results compared to the direct method of aggregation using original weights. The attention mechanism is embedded within the network's propagation stage in a Graph Attention Network (GAT) [165]. Each node's hidden states are calculated using a self-attention strategy in which the node pays attention to its neighbors.

In addition to GAT, Gated Attention Network (GaAN) [212] also uses a multi-headed attention mechanism. GaAN's attention aggregator uses key-valued attention and dot-product attention, whereas GAT's attention aggregator calculates attention coefficients using fully connected layers. Moreover, GaAN assigns different weights to different attention heads. This aggregator is called the gated attention aggregator.

The spatial-based methods circumvent graph theory and obviate the need to transform the graph signal from the spatial domain to the spectral domain. These methods define convolution operations directly on the spatial domain, which is more intuitive than spectral graph convolution but lacks theoretical support. Next, we present spectral-based methods.

2.2.2 Spectral-based Methods

Spectral-based GCNs utilize a spectral representation of the graphs in their work. These methods define the convolution operator in the spectral domain and have theoretical background in graph signal processing [147]. In spectral methods, a graph signal is first converted to the spectral domain using the graph Fourier transform, followed by the convolution operation. After convolution, the obtained signal is transformed back using the inverse graph Fourier transform [193]. We break down spectral-based graph convolution into two categories based on the type of graph, specifically undirected and directed graph convolution.

2.2.2.1 Undirected Graph Convolution

We begin with graph convolutional networks (GCN) [88], the most widely used type of graph neural network due to its versatility [103, 177]. GCN has multi-layers that stacks first-order Chebyshev polynomials as a graph convolution layer and learns graph representations using a nonlinear activation function. Using Chebyshev polynomials to replace the time-consuming Laplacian eigenvalue decomposition of graph spectral convolution is proposed by ChebNet [28]. Unlike GCN, the polynomial of ChebNet can be K^{th} order, while GCN does a simplification and uses only the first order. Please refer to Section 3.2.2 for the detailed definition of graph spectral analysis and the derivation from ChebNet to GCN.

GCN [88] can be grouped into the general framework in the previous section. The AGG function in GCN is defined as the weighted average of the neighbor node representations. The COMB function is defined as the summation of the aggregated messages and the node representation itself, in which the node representation is normalized by its own degree [188]. However, since spectral convolution requires a semi-positive definite Laplacian operator, this limits spectral convolution to undirected graphs.

There are also numerous variants of GCN, which are manifested in simplifying the convolution process [187], using different network structures [140], adding different convolution scales [1], etc. SGC [187] removes nonlinear layers by using 2-hop adjacency matrix as replacement and collapses weight matrices to reduce computational consumption. SIGN [140] proposes an inception-like structure which uses SGC as basic block and concatenates these block of different size together into a FC layer. N-GCN [1] inspires from random walk, it builds a multi-scale GCN which uses different powers of adjacency matrices as input to achieve extract feature from different K-hop neighborhoods. It can gain the information from the k^{th} step from current node, which the same idea with K-hop. However, its K-hop method is only applicable to undirected graphs.

Klicpera *et al.* [90] proposes APPNP model which utilize a propagation based on personalized PageRank to handle feature oversmoothing problem. Besides, the PPRGo [11] increases the efficiency of APPNP by incorporating multi-hop neighborhood information in a single step. Note that no matter APPNP or PPRGo, the basic form of their propagation matrices is $\mathbf{A}_{\mathrm{ppnp}} = \alpha \left(\mathbf{I} - (1-\alpha)\mathbf{A}_{\mathrm{u}}\right)^{-1}$, α is the teleport probability of personalized PageRank, which is quite different from our directed graph Laplacian in Equation 5.1. Furthermore, they use symmetric matrix $\mathbf{A}_{\mathrm{u}}$ in the propagation, which means they are not adaptive to directed graphs.

2.2.2.2 Directed Graph Convolution

The majority of research on directed network analysis has been spectral in nature and centered on symmetrization-based techniques [112, 126, 142], but edge directionality itself can contain vital information [160, 216]. Hermitian clustering [26] and motif-based methods [164] have been used to discover imbalanced flows in directed graphs. There has been a recent uptick in the use of GCNs to extract structure information from directed edges [61].

Several GCN works have tried to learn the complex directed graph structure by defining motifs [118], using magnetic Laplacian [216], inheritance relationships [84], redefining the propagation scheme from Markov process view [113] and flow imbalance measures [60]. Ma *et al.* [113] use the directed Laplacian operator to solve the directed graph problem. However, it is defined on strongly connected directed graphs, which is not universally applicable to any directed graphs. DIGRAC [60] uses imbalance objectives and evaluation metrics to perform directed graph clustering based on flow imbalance measures.

MagNet [216] utilizes a complex Hermitian matrix called the magnetic Laplacian. This matrix encodes both undirected geometric structure in the magnitude of its entries and directional information in the phase. This paper constructs the adjacency matrix of a directed graph in the form of complex numbers to preserve the directionality of the graph. Compared to real-valued Laplacian matrices, this approach is better able to represent the directed nature of the graph. However, experimental results show that the performance of node classification is not as good as link prediction and that the best results for certain node classification experiments are obtained when the imaginary part of the complex number is zero, i.e. when only the real part is retained, treating the directed graph as an undirected graph.

It is worth mentioning that our proposed methods DGCN [161] in Chapter 3 and DiGCN [160] in Chapter 4 are the first few works to explore directed graph convolution, and are also the baselines for the aforementioned MagNet and DIGRAC.

2.3 Graph Contrastive Learning

Given the success of GCNs in extracting features from graph-structured data, it is natural to explore ways to improve the representations learned by these networks. One promising approach is to utilize Graph Contrastive Learning (GCL), which aims to learn robust representations by contrasting different views of the same graph. Using GCNs as the underlying model in the contrastive learning process allows for representations that are specifically tailored to the structure of the graph, resulting in improved performance on tasks related to graphs. In the following, we will delve deeper into the details of graph contrastive learning and explore how it can be integrated with GCNs to enhance the learned representations.

GCL is an extension of popular Contrastive Learning (CL) methods [19, 162, 58] on the graph. Several studies [57, 207, 228, 229] have shown the potential of GCL in learning generalized representation on graph-structured data. They force views generated from the same instance closer while views from different instances apart using *InfoNCE-like* [123] objective function.

We list a few of representative GCL works as follows. DGI [166] employs two discriminators to directly measure mutual information (MI) between input and representations of both nodes and edges without data augmentation. GMI [133] generalizes the concept of MI computations from vector space to graph domain where the calculation of MI from two aspects of node features and topological structure is indispensable. MVGRL [57] proposes to learn both node and graph-level representations by performing graph diffusion augmentation and contrasting augmented graph representations. GraphCL [207] proposes a GCL framework combinations of several graph augmentations to incorporate various priors and study the impact of various graph augmentations on multiple datasets. This method is primarily used for contrastive learning at the graph-level. GRACE [228] presents a GCL method with adaptive augmentation that incorporates various priors for topological and semantic aspects of the graph, which mainly focuses on the node-level contrastive learning.

Previous works [157, 57, 203] have found the powerful capabilities of using multiple contrastive views at the same time in contrastive learning. However, their methods require to set contrastive views in advance, which means the views are fixed during training [57, 157]. Unlike them, we do not preset the parameters to fix the contrastive views but let them dynamically change during training. We compared the existing GCL methods as shown in Table 2.1.

For design graph contrastive learning models, one of the crucial steps is to generate good contrasting views through data augmentation. Thus, we summarize a framework for data augmentation that systematically subdivides different data augmentation methods into three levels, enabling us to pick the right data augmentation method for different problems as needed.

Table 2.1: Comparison of different GCL methods.

Method	Graph Type	Data Augmentation	Views
MVGRL [57]	Undirected	Graph diffusion	Fixed
GCC [136]	Undirected	Graph sampling with restart	Fixed
GraphCL [207]	Undirected	Node dropping Edge perturbation Sample subgraph Attribute masking	Fixed
GRACE [228]	Undirected	Removing edge Feature masking	Fixed
Ours (DiGCL)	Directed	Laplacian perturbation	Easy to difficult

2.3.1 Data Augmentation Framework

We can summarize the graph data augmentation as:

$$\hat{\mathcal{G}} = \mathcal{T}_\Theta(\mathcal{G}) = \mathcal{T}_\Theta(\mathbf{A}, \mathbf{X}), \tag{2.3}$$

which aims to find a set $\mathcal{T}_\Theta$ that takes the graph structure $\mathbf{A}$ and node feature $\mathbf{X}$ into consideration and output the augmented graph $\hat{\mathcal{G}}$. There may be several kinds of augmentation functions in the set $\mathcal{T}$. If we wish to classify the numerous data augmentations, we need to consider not only their perturbations to the original graph, but also the impact they will have on the GCN-based encoder. Thus, we divide them into three categories according to the impact of data augmentations on the encoder's perturbed parts. The general formula of GCNs has been described in Equation 2.1 and Equation 2.2, and the illustration of graph data augmentation is shown in Figure 2.3

2.3.1.1 Topology-level Augmentation

The common point of generating contrasting views from the topology level is to obtain a different structure of the graph by perturbing $\mathbf{A}$ while keeping the original semantic information. We can further break it down into two categories:

Local Aggregation. Several data augmentations can be treated as perturbing the aggregation scheme AGG$(\cdot)$ among their neighbors $\mathcal{N}(i)$. This aspect focuses on adjusting the weights for the node and its neighbors and assumes that the semantic

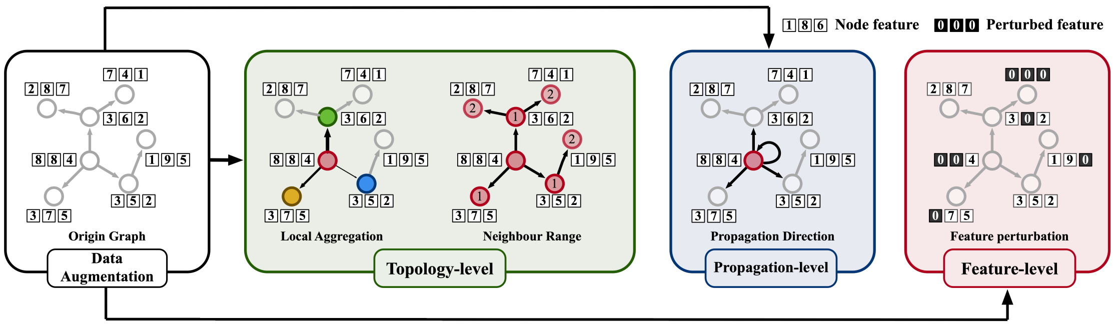


Figure 2.3: Illustration of Graph Data Augmentation.

meaning of $\mathcal{G}$ has certain robustness to the local aggregation changes. We can write the perturbed graph as

$$\hat{\mathcal{G}}_{\text{la}} = (\nu\mathbf{I} + \xi\mathcal{W}(\mathbf{A}), \mathbf{X}), \tag{2.4}$$

where ν, ξ are the aggregation weights and $\mathcal{W}$ is the adjacency perturbation function.

Neighbour Range. Another form of topological augmentation is perturbing the range of neighbours $\mathcal{N}(i)$. Since first-order neighbours often contain rich local information, after expanding the range of neighbours, we are able to get farther global information [160, 187]. This motivates many methods, i.e., neighbour hop [132], to make first and higher-orders as contrasting views, that is, the local and global contexts. It implies that changing range of neighbour does not affect the semantic meaning of $\mathcal{G}$. Formally. this type of augmentation can be written as:

$$\hat{\mathcal{G}}_{\text{nr}} = (\nu\mathbf{I} + \textstyle\sum_{j=1}^{\hat{k}} \xi_j \mathbf{A}^j, \mathbf{X}) = (\mathcal{K}(\mathbf{A}), \mathbf{X}), \tag{2.5}$$

where $\mathcal{K}(\cdot)$ is a polynomial function and $\hat{k}$ is the order of neighbours. By combining the above two augmentations, We can obtain a generalized formula for the Topology-level augmentation as

$$\hat{\mathcal{G}}_{\text{topo}} = (\mathcal{K}(\mathcal{W}(\mathbf{A})), \mathbf{X}). \tag{2.6}$$

2.3.1.2 Propagation-level Augmentation

Most current data augmentations ignore the reverse direction of feature propagation, which means that features can be propagated back to itself or re-initiated propagating with a certain likelihood [90]. The underlying prior is the perturbation propagation does not affect the expressiveness of the model much. We can think of Propagation-level augmentation as the original Laplacians multiplied by the perturbed inverse Laplacians. Formally, we can define this kind of augmentation as

$$\hat{\mathcal{G}}_{\text{prop}} = (\mathcal{K}(\mathbf{A})/\mathcal{Q}(\mathbf{A}), \mathbf{X})\,, \tag{2.7}$$

where $\mathcal{Q}(\cdot)$ is a polynomial function, and the bias of $\mathcal{Q}$ is 1 to keep the original propagation when there is no perturbation. This type of augmentation is not common used in GCL due to the high computational cost of matrix inverse.

2.3.1.3 Feature-level Augmentation

Feature-level augmentation prompts models to recover missing or interrupted vertex features using their context information, i.e., the remaining features. We can define this augmentation as adding noise to the node feature via randomly perturbing a part of feature matrix $\mathbf{X}$. Formally, we write the perturbed graph as:

$$\hat{\mathcal{G}}_{\text{feat}} = (\mathbf{A}, \mathcal{M}(\mathbf{X})), \tag{2.8}$$

where $\mathcal{M}(\cdot)$ is a feature perturbation function, *e.g.,* masking features or adding Gaussian noise [57].

2.3.2 Graph Data Augmentation

Good data augmentations are the prerequisite for contrastive learning [207, 167, 181]. Most current data augmentation methods, such as dropping nodes or edges [207, 228, 229, 179] and attribute masking [71, 207, 182] are migrated from visual representation methods [189, 220] with graph structural improvements. Other methods learn graph structure features by combining with traditional graph algorithms to go self-supervised, for example, subgraph sampling [207, 136, 180], graph diffusion based on PageRank/Heat kernel [57], and multi-hop neighbor prediction [132]. We divide graph data augmentation into two subsections based on the type of graph, discussing undirected and directed graph data augmentation, respectively.

2.3.2.1 Undirected Graph Data Augmentation

We category the existing data augmentations for undirected graph in Table 2.2, and give a detailed explanation of the categorization.

Table 2.2: Summary of existing data augmentations. "**LA**" means Local Aggregation and "**NR**" stands for Neighbour Range. The sampling-based approaches subgraph sampling and node dropping would lead to the loss of nodes, affecting both the adjacency matrix **A** and the feature matrix **X**.

Method	Category	Perturbed Part
Edge perturbation	Topology **(LA)**	$\mathrm{AGG}(\cdot)$
Neighbour hop	Topology **(NR)**	$\mathcal{N}(i)$
Graph diffusion	Propagation	$\mathrm{COMB}(\cdot)$
Feature masking	Feature	$\mathbf{X}$
Subgraph sampling	Topology & Feature	$\mathbf{A}$ & $\mathbf{X}$
Node dropping	Topology & Feature	$\mathbf{A}$ & $\mathbf{X}$

Edge Perturbation. Perturbation of edges is a common and efficient way to augment graph. For computational consumption reasons, we generally remove existing edges randomly on top of the original graph instead of adding edges randomly. For removing edges [207, 228], we first sampling a random edge masking matrix $\mathbf{N} = \{0, 1\}^{n\times n}$, whose entry is drawn from a Bernoulli distribution $\mathcal{B}(1 - p_e)$, i.e., $\mathbf{N} \sim \mathcal{B}(1 - p_e)$. p_e is the edge perturbation probability. We can obtain the perturbed adjacency matrix $\hat{\mathbf{A}}$ as

$$\mathcal{W}_{\text{edge}}(\mathbf{A}) = \hat{\mathbf{A}} = \mathbf{A} \oplus \mathbf{N}, \tag{2.9}$$

where $\oplus$ is the exclusive OR (XOR) operation. We obtain the augmented graph as:

$$\hat{\mathcal{G}}_{\text{edge}} = (\mathcal{W}_{\text{edge}}(\mathbf{A}), \mathbf{X}). \tag{2.10}$$

This kind of data augmentation belongs to Local Aggregation of Topology-level when we take $\nu = 0$ and $\xi = 1$.

Neighbour Hop. Designing this form of data augmentation scheme is relatively straightforward and only requires consideration of the neighbors at different orders. Formally, the k^{th}-order perturbed graph $\hat{\mathcal{G}}_{\text{nh}}$ as:

$$\hat{\mathcal{G}}_{\text{nh}} = (0 \cdot \mathbf{I} + 0 \cdot \mathbf{A} + \ldots + 1 \cdot \mathbf{A}^k, \mathbf{X}). \tag{2.11}$$

When using this method, we just need to select the different k for comparison. This approach is part of Neighbour Range in Topology-level category.

Graph Diffusion. A representative method in the **Propagation-level** category is graph diffusion from MVGRL [57], which uses Personalized PageRank kernel [90] with teleport probability α as data augmentation. We can rewrite this graph diffusion using our framework as:

$$\hat{\mathcal{G}}_{\text{gd}} = \left(\frac{\alpha}{\mathbf{I} - (1-\alpha)\tilde{\mathbf{A}}}, \mathbf{X}\right), \tag{2.12}$$

where $\tilde{\mathbf{A}}$ is the normalized adjacency matrix $\mathbf{A}$.

Feature Masking. The underlying assumption of **Feature-level** augmentation is that perturbing partial node attributes does not affect the model predictions much. For feature masking [228, 207], it can be done by sampling. Assuming a Bernoulli distribution $\mathcal{B}(1-p_f)$, where p_f denotes the probability of each feature being modified, we draw a random matrix $\mathbf{M} = \{0,1\}^{n\times b}$ from $\mathcal{B}(1-p_f)$, i.e., $\mathbf{M} \sim \mathcal{B}(1-p_f)$, and b is the feature dimension. We then acquire the perturbed feature matrix as

$$\mathcal{M}_{\text{mask}}(\mathbf{X}) = \hat{\mathbf{X}} = \mathbf{X} \oplus \mathbf{M}. \tag{2.13}$$

We can obtain the augmented graph as:

$$\hat{\mathcal{G}}_{\text{fm}} = (\mathbf{A}, \mathcal{M}_{\text{mask}}(\mathbf{X})) \tag{2.14}$$

Graph Sampling. There are several variants of graph sampling based methods, such as subgraph sampling, node dropping (full graph sampling), etc. These methods assume that the graph structure will have some redundant information and enable the network to learn the intrinsic structural information by comparing the augmented graph with different sampling rates.

Similar to graph pooling, both subgraph and node sampling methods result in losing some nodes. This means that they not only affect the adjacency matrix $\mathbf{A}$, but also change the feature matrix $\mathbf{X}$. We think of such methods as composites of two categories: Topology-level and Feature-level.

The graph sampling approaches are more commonly used for graph-level contrastive learning [207, 57], such as small chemical molecules. Since such methods inevitably lose part of the nodes, this leads to their inapplicability to node-level graph contrastive learning.

2.3.2.2 Directed Graph Data Augmentation

However, most of these data augmentation methods for undirected graph are not optimized for directed graphs, and thus they cannot efficiently obtain structure information that can be used for self-supervised learning. The performance of the existing methods on the directed graphs can be seen in Table 5.8. To the best of our knowledge, there is no data augmentation scheme specifically designed for directed graphs. The Laplacian perturbation proposed in our work DiGCL [159] is the first data augmentation scheme currently designed specifically for directed graphs.

2.4 Summary

In this chapter, we present an extensive review of Graph Neural Networks. We start from the general design pipeline of GNNs and summarize the work involved in the different modules. We also summarize the advantages and disadvantages of various techniques and discuss the shortcomings for existing work on directed graphs. Moreover, we closely analyze the work in the areas of Graph Convolutional Networks and Graph Contrastive Learning, two areas closely related to the later chapters, and give some preliminaries. Motivated by the related background and shortcomings of related methods, we will present a more detailed description of how we (propose to) solve these problems in the subsequent chapters.

Chapter 3

Second-order Graph Convolution

In this chapter, we present DGCN, a novel graph convolutional network for directed graphs. DGCN uses first- and second-order proximity to generalize spectral-based GCNs to directed graphs, It can retain directed features of graphs and expand the convolution receptive field to extract and leverage surrounding information. Our approach outperforms other models in semi-supervised classification tasks through experimental validation on various real-world datasets.

3.1 Introduction

Graph structure is a common form of data in real-world problems. It has a very strong ability to represent complex structures and can easily express entities and their relationships. Graph Convolutional Networks (GCNs) [55, 28, 88, 54, 165, 200] are CNNs variants on graphs and effectively learn underlying pairwise relations among data nodes. Different GCN variants greatly promote their use in various real-world tasks, including but not limited to social networks [18], quantum chemistry [103], text classification [204] and image recognition [177].

One of the main reasons that a GCN can achieve such good results on a graph is that it makes full use of the structure of the graph. It captures rich information from the neighborhood of object nodes through the links instead of being limited to a specific distance range. General GCN models provide a neighborhood aggregation scheme for each node to gain a representation vector, and then learn a mapping function to predict the node attributes [69]. Since spatial-based methods need to traverse surrounding nodes when aggregating features, they usually add significant overhead to computation and memory usage [187]. On the contrary, spectral-based methods use matrix multiplication instead of traversal search, which greatly improves the training speed. Thus, we focus mainly on the spectral-based method here.

Although the above methods have achieved improvement in many aspects, there are two major shortcomings with existing spectral-based GCN methods.

First, spectral-based GCNs are limited to apply to undirected graphs [193]. For directed graphs, the only way is to relax the graph structure from a directed to an undirected graph by symmetrizing the adjacency matrix. In this way we can get the semi-definite Laplacian matrix, but at the same time we also lose the unique structure of the directed graph. For example, in a citation graph, later published articles can cite former published articles, but the reverse is not true. This is a unique time series relationship. If we transform it into an undirected graph, we lose this part of the information. Although we can represent the original directed graph in another form, such a temporal graph learned by combination of Recurrent Neural Networks (RNNs) and GCNs [129], we still want to dig more structural features from the original without adding extra components.

Second, in most existing spectral-based GCN models, during each convolution operation, only 1-hop node features are taken into account (using the same adjacency matrix), which means they only capture the *first-order* information between nodes. It is natural to consider direct links when extracting local features, but this is not always the case. In some real-world graph data, many legitimate relationships may not be encoded via first-order links [156]. For instance, people in social networking communities share common interests, but they do not necessarily contact each other. In other words, the features we get by *first-order* proximity are likely to be insufficient. Although we can obtain more information by stacking multiple layers of GCNs, multi-layer networks will introduce more trainable parameters, making them prone to overfitting when the label rate is small or needing extra labeled information, which is not label efficient [100]. Therefore, we need a more efficient feature extraction method.

To address these issues, we leverage *second-order* proximity between nodes as a complement to existing methods, which inspire from the *hub* and *authority* web model [89, 221]. By using *second-order* proximity, the directional features of the directed graph can be retained. Additionally, the receptive field of the graph convolution can be expanded to extract more features. Different from *first-order*

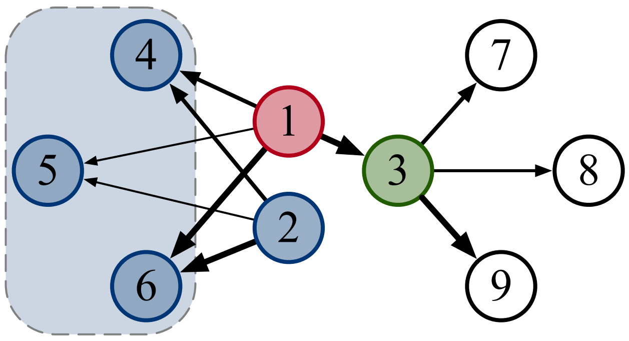


Figure 3.1: A simple directed graph example. Line width indicates the weight of the edges. The node v_1 has *first-order* proximity with v_3. It also has *second-order* proximity with v_2, because they have shared neighbors $\{v_4, v_5, v_6\}$. Both v_2 and v_3's features should be considered when aggregating v_1's feature.

proximity, judging whether two nodes have *second-order* proximity does not require nodes to have paths between them, as long as they share common neighbors, which can be considered as *second-order* proximity. In other words, nodes with more shared neighbors have stronger *second-order* proximity in the graph. This general notion also appears in sociology [48], psychology [138] and daily life: people who have a lot of common friends are more likely to be friends. A simple example is shown in Figure 3.1. When considering the neighborhood feature aggregation of v_1 in the $layer_1$, we need to consider v_3, because it has a *first-order* connection with v_1, but we also need to aggregate v_2's features, due to the high *second-order* similarity with v_1 in sharing three common neighbors.

In this chapter, we present a new spectral-based model on directed graphs, **<u>D</u>**irected **<u>G</u>**raph **<u>C</u>**onvolutional **<u>N</u>**etworks (**DGCN**), which utilizes *first* & *second-order* proximity to extract graph information. We not only consider basic *first-order* proximity to obtain neighborhood information, but also take *second-order* proximity into account, which is a complement for the sparsity of *first-order* information in real-world data. What's more, we verify this efficiency by extending *Feature and Label Smoothness* measurements [69] to our application scope. Through experiments, we empirically show that our model exhibits superior performance against baselines while the number of parameters and computational consumption is significantly reduced.

In summary, our study has the following contributions:

1. We present a novel graph convolutional network called DGCN, which can be applied to the directed graphs by utilizing *first- and second-order* proximity. To our knowledge, this is the first-ever attempt that enables spectral-based GCNs to generalize to directed graphs.

2. We define *first- and second-order* proximities on the directed graph, which are designed for expanding the convolution operation receptive field, extracting and leveraging graph information. Meanwhile, we empirically show that this method has both feature and label efficiency, while also demonstrating powerful generalization capability.

3. We experiment with semi-supervised classification tasks on various real-world datasets, which validates the effectiveness of *first- and second-order* proximity and the improvements obtained by DGCNs over other models.

3.2 Undirected Graph Convolution and its Limitations

The spectral-based graph convolution is based on spectral graph theory [68], which examines the characteristics of graphs by analyzing the eigenvalues and eigenvectors of related matrices such as the adjacency matrix and the graph Laplacian and its variations. In this section, we first go through the currently used spectral-based graph convolution defined on the undirected graphs and then study the limitations of its application to directed graphs.

3.2.1 Undirected Graph Laplacian

Formally, we define the underlying data structure, graph, as follows:

Definition 1. ***Graph*** *[135]. A general graph has n vertices or nodes is define as* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V}$ *is node set and* $\mathcal{E}$ *is edge set. Each edge* $(u, v) \in \mathcal{E}$ *is an ordered pair between node u and v. The nodes are described by the feature matrix* $\mathbf{X} \in \mathbb{R}^{n \times c}$*, with the number of features c per node. A graph is directed when has any* $(u, v) \not\equiv (v, u)$*. The adjacency matrix of the directed graph can be denoted as*

$\mathbf{A} = \{0,1\}^n$, *where*

$$\mathbf{A}(u,v) := \begin{cases} 1 & \textit{if } (u,v) \in \mathcal{E} \\ 0 & \textit{otherwise} \end{cases}. \tag{3.1}$$

When each edge has edges in their opposite direction $(u,v) \equiv (v,u)$*, the graph is undirected. We mark the undirected graph as* $\mathcal{G}_u$*, and its adjacency matrix is defined as* $\mathbf{A}_u$*, where*

$$\mathbf{A}_u(u,v) := \begin{cases} 1 & \textit{if } (u,v) \in \mathcal{E} \textit{ or } (v,u) \in \mathcal{E} \\ 0 & \textit{otherwise} \end{cases}. \tag{3.2}$$

At the center field of spectral graph theory as well as a number of important machine learning algorithms, such as spectral clustering [168], lies a matrix called the graph Laplacian. This matrix is also the key to spectral-based graph convolution. The undirected graph Laplacian matrix is defined as follows:

Definition 2. ***Undirected Graph Laplacian Matrix.*** *Given an undirected graph* $\mathcal{G}_u$*, the undirected graph Laplacian is defined as*

$$\mathbf{L}_u = \mathbf{D}_u - \mathbf{A}_u, \tag{3.3}$$

where $\mathbf{D}_u \in \mathbb{Z}^{n\times n}$ *is a diagonal matrix of node degree, where*

$$\mathbf{D}_u(u,v) := \begin{cases} \text{degree}\,(u) & \textit{if } u = v \\ 0 & \textit{otherwise} \end{cases}. \tag{3.4}$$

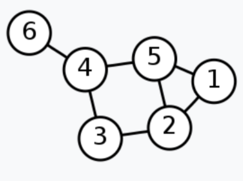
Graph	Degree matrix	Adjacency matrix	Laplacian matrix
	$\begin{pmatrix} 2&0&0&0&0&0\\0&3&0&0&0&0\\0&0&2&0&0&0\\0&0&0&3&0&0\\0&0&0&0&3&0\\0&0&0&0&0&1 \end{pmatrix}$	$\begin{pmatrix} 0&1&0&0&1&0\\1&0&1&0&1&0\\0&1&0&1&0&0\\0&0&1&0&1&1\\1&1&0&1&0&0\\0&0&0&1&0&0 \end{pmatrix}$	$\begin{pmatrix} 2&-1&0&0&-1&0\\-1&3&-1&0&-1&0\\0&-1&2&-1&0&0\\0&0&-1&3&-1&-1\\-1&-1&0&-1&3&0\\0&0&0&-1&0&1 \end{pmatrix}$

Figure 3.2: Simple example of an undirected graph and its Laplacian matrix [185].

A simple example of an undirected graph and its Laplacian matrix is shown in Figure 3.2. It should be noted that this Laplacian matrix requires the use of a real symmetric adjacency matrix, so directed graphs cannot be directly applied to this definition. We will define the Laplacian matrix for directed graphs in subsequent Chapter 4.

3.2.2 Undirected Graph Convolution

$\mathbf{L}_\mathrm{u}$ is symmetric and positive semi-definite [49]. It can be normalized as $\mathbf{L}_\mathrm{u}^{'} = \mathbf{I} - \mathbf{D}_\mathrm{u}^{-\frac{1}{2}}\mathbf{A}_\mathrm{u}\mathbf{D}_\mathrm{u}^{-\frac{1}{2}} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{I}$ is an identity matrix, $\mathbf{U}$ is the matrix of eigenvectors of $\mathbf{L}_\mathrm{u}^{'}$ and $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues. The spectral convolution on the undirected graph is defined as the multiplication of node scalar $x \in \mathbb{R}^n$ with a filter $g_\theta = \mathrm{Diag}(\theta)$ parameterized by $\theta \in \mathbb{R}^n$ in the Fourier domain [28]:

$$g_\theta * x = \mathbf{U}\left(\mathbf{U}^T g_\theta \odot \mathbf{U}^T x\right) = \mathbf{U} g_\theta \mathbf{U}^T x, \tag{3.5}$$

where $*$ represents convolution operation and $\odot$ is the element-wise Hadamard product. Here we can understand g_θ as a function of the eigenvalues $\mathbf{\Lambda}$ of $\mathbf{L}_\mathrm{u}^{'}$.

Graph convolutions can be further approximated by K^{th} Chebyshev polynomials [28] to reduce computation-consuming:

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta_k' T_k(\tilde{\mathbf{L}_\mathrm{u}^{'}})x, \tag{3.6}$$

where $\tilde{\mathbf{L}_\mathrm{u}^{'}} = 2\mathbf{L}_\mathrm{u}^{'}/\lambda_\mathrm{max} - \mathbf{I}$, λ_max denotes the largest eigenvalue of $\mathbf{L}_\mathrm{u}^{'}$ and $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0(x) = 1$ and $T_1(x) = x$. The K-polynomial filter $g_{\theta'}$ parameterized by $\theta' \in \mathbb{R}^K$ shows its good localization in the node domain through integrating the node features within the K-hop neighborhood[214]. Our model obtains node features in a different way, which will be explained in Section 3.3.4.

Kipf *et al.* employ Graph Convolutional Networks (GCNs)[88], which is a first-order approximation of ChebNet. They assume $k = 1$ and $\lambda_{max} \approx 2$. Besides, they set θ as shared parameters over the whole graph, *i.e.*, $\theta = \theta_0' = -\theta_1'$ in Equation 3.6 to simplify graph convolution and get the following GCN convolution g_θ as:

$$g_\theta * x \approx \theta\left(\mathbf{I} + \mathbf{D}_\mathrm{u}^{-\frac{1}{2}}\mathbf{A}_\mathrm{u}\mathbf{D}_\mathrm{u}^{-\frac{1}{2}}\right)x. \tag{3.7}$$

They also use renormalization trick converting $\mathbf{I} + \mathbf{D}_\mathrm{u}^{-\frac{1}{2}}\mathbf{A}_\mathrm{u}\mathbf{D}_\mathrm{u}^{-\frac{1}{2}}$ to $\mathbf{D}_\mathrm{u}^{'-\frac{1}{2}}\mathbf{A}_\mathrm{u}^{'}\mathbf{D}_\mathrm{u}^{'-\frac{1}{2}}$, where $\mathbf{A}_\mathrm{u}^{'} = \mathbf{A}_\mathrm{u} + \mathbf{I}$ and $\mathbf{D}_\mathrm{u}^{'}(u,v) = \sum_{v\in\mathcal{V}} \mathbf{A}_\mathrm{u}^{'}(u,v)$, to alleviate numerical instabilities and exploding/vanishing gradients problems. The final graph convolution layer is defined as follows:

$$\mathbf{Z}_\mathrm{u}^{'} = \left(\mathbf{D}_\mathrm{u}^{'-\frac{1}{2}}\mathbf{A}_\mathrm{u}^{'}\mathbf{D}_\mathrm{u}^{'-\frac{1}{2}}\right)\mathbf{X}\mathbf{\Theta}. \tag{3.8}$$

Here, $\mathbf{X} \in \mathbb{R}^{n\times c}$ is the c-dimensional node feature vector, $\mathbf{\Theta} \in \mathbb{R}^{c\times d}$ is the filter parameters matrix and $\mathbf{Z}^{'}_{\mathrm{u}} \in \mathbb{R}^{n\times d}$ is the convolved result with d output dimension.

However, the derivations are based on the premise that the graph is undirected, *i.e.,* the Laplacian matrices are symmetric. The way they use to deal with directed graph is to relax it into an undirected graph, thereby constructing a symmetric Laplacian matrix.

3.2.3 Usage Limitations on Directed Graph

Although relaxing directed graph to undirected can be used for spectral convolution, it is not able to represent the actual structure of the directed graph. For instance, as we mention in the Section 3.1, there is a time limit for citations: previously published papers cannot cite later ones. If the citation network is relaxed to an undirected graph, this restriction will no longer exist.

Since the graph Laplacian and von Neumann entropy are intrinsically linked [205], we can use the proprieties of von Neumann entropy to analyze what the difference would be when converting a directed graph to an undirected graph. We start with defining the von Neumann entropy of directed graphs.

Definition 3. ***Approximate von Neumann entropy for directed graphs****.* *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its approximate von Neumann entropy [205] is defined as*

$$\mathcal{H}^{\mathrm{D}}_{\mathrm{VN}}(\mathcal{G}) = 1 - \frac{1}{n} - \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{out}\,d_v^{out}} + \frac{d_u^{in}}{d_v^{in}d_u^{out^2}}\right) - \sum_{(u,v)\in\tilde{\mathcal{E}}}\frac{1}{d_u^{out}\,d_v^{out}}\right\}, \tag{3.9}$$

where $\tilde{\mathcal{E}} = \{(u,v) \mid (u,v) \in \mathcal{E} \text{ and } (v,u) \notin \mathcal{E}\}$. $d_u^{\mathrm{in}} = \sum_{v\in\mathcal{V}} \mathbf{A}(v,u)$ and $d_u^{out} = \sum_{v\in\mathcal{V}} \mathbf{A}(u,v)$ are the in- and out-degree of the node u.

To simplify the derivation process, we normalize the above von Neumann entropy. The normalization is done in relation to the graph size, which removes part of the size dependence. In particular, we compute the quantity

$$\begin{aligned}\mathcal{J}^{\mathrm{D}}_{\mathrm{VN}}(\mathcal{G}) &= |\mathcal{V}|\left|\mathcal{H}^{D}_{\mathrm{VN}}(\mathcal{G}) - \left(1 - \frac{1}{|\mathcal{V}|}\right)\right| \\ &= \frac{1}{2n}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{\mathrm{out}}\,d_v^{\mathrm{out}}} + \frac{d_u^{\mathrm{in}}}{d_v^{\mathrm{in}}d_u^{\mathrm{out}^2}}\right) - \sum_{(u,v)\in\tilde{\mathcal{E}}}\frac{1}{d_u^{\mathrm{out}}\,d_v^{\mathrm{out}}}\right\}\end{aligned} \tag{3.10}$$

as a normalized quantity which captures variations in the in-degree and out-degree statistics. It has the same manner as the approximate von Neumann entropy $\mathcal{H}_{\mathrm{VN}}^{D}(\mathcal{G})$. It is worth noting that the monotonicity features of the normalized quantity and the original entropy are opposed.

We want to investigate whether the simple transformation of directed graphs to undirected graphs has any effect on the loss of graph structure information. We assume that node $p, q \in \mathcal{V}$ and $(p, q) \in \tilde{\mathcal{E}}$, thus $(p, q) \in \mathcal{E}$ and $(q, p) \notin \mathcal{E}$. We gradually transform the original directed graph into an undirected graph by converting a directed edge (p, q) into an undirected edge, *i.e.,* adding (q, p) into $\mathcal{E}$. Since both (p, q) and (q, p) are $\in \mathcal{E}$, $(p, q) \notin \tilde{\mathcal{E}}$. We define the graph after adding the edge (q, p) as $\mathcal{G}' = (\mathcal{V}, \mathcal{E} \cup \{(q, p)\})$ and the normalized quantity can be calculated as

$$\begin{aligned}
\mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}') &= \frac{1}{2n}\left\{\sum_{(u,v)\in\mathcal{E}\cup\{(q,p)\}}\left(\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}} + \frac{d_u^{\mathrm{in}}}{d_v^{\mathrm{in}} d_u^{\mathrm{out}^2}}\right) - \sum_{(u,v)\in\tilde{\mathcal{E}}/\{(p,q)\}}\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}}\right\} \\
&= \frac{1}{2n}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}} + \frac{d_u^{\mathrm{in}}}{d_v^{\mathrm{in}} d_u^{\mathrm{out}^2}}\right) + \sum_{(u,v)\in\{(q,p)\}}\left(\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}} + \frac{d_u^{\mathrm{in}}}{d_v^{\mathrm{in}} d_u^{\mathrm{out}^2}}\right)\right. \\
&\quad \left. - \sum_{(u,v)\in\tilde{\mathcal{E}}}\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}} + \sum_{(u,v)\in\{(p,q)\}}\frac{1}{d_u^{\mathrm{out}}\, d_v^{\mathrm{out}}}\right\}.
\end{aligned} \tag{3.11}$$

After adding an edge, we can calculate the change in the von Neumann entropy as

$$\begin{aligned}
\mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}) - \mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}') = \frac{1}{2n}&\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}) d_v^{\mathrm{out}}(\mathcal{G})} + \frac{d_u^{\mathrm{in}}(\mathcal{G})}{d_v^{\mathrm{in}}(\mathcal{G}) d_u^{\mathrm{out}}(\mathcal{G})^2}\right)\right. \\
&- \sum_{(u,v)\in\mathcal{E}\cup\{(q,p)\}}\left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} + \frac{d_u^{\mathrm{in}}(\mathcal{G}')}{d_v^{\mathrm{in}}(\mathcal{G}') d_u^{\mathrm{out}}(\mathcal{G}')^2}\right) \\
&\left. - \sum_{(u,v)\in\tilde{\mathcal{E}}}\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}) d_v^{\mathrm{out}}(\mathcal{G})} + \sum_{(u,v)\in\tilde{\mathcal{E}}/\{(p,q)\}}\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')}\right\}
\end{aligned}$$

$$\begin{aligned}\mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}) - \mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}') = \frac{1}{2n}\Bigg\{ & \sum_{(u,v)\in\mathcal{E}} \left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}) d_v^{\mathrm{out}}(\mathcal{G})} + \frac{d_u^{\mathrm{in}}(\mathcal{G})}{d_v^{\mathrm{in}}(\mathcal{G}) d_u^{\mathrm{out}}(\mathcal{G})^2} \right) \\ & - \sum_{(u,v)\in\mathcal{E}} \left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} + \frac{d_u^{\mathrm{in}}(\mathcal{G}')}{d_v^{\mathrm{in}}(\mathcal{G}') d_u^{\mathrm{out}}(\mathcal{G}')^2} \right) \\ & - \sum_{(u,v)\in\{(q,p)\}} \left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} + \frac{d_u^{\mathrm{in}}(\mathcal{G}')}{d_v^{\mathrm{in}}(\mathcal{G}') d_u^{\mathrm{out}}(\mathcal{G}')^2} \right) \\ & - \sum_{(u,v)\in\tilde{\mathcal{E}}} \frac{1}{d_u^{\mathrm{out}}(\mathcal{G}) d_v^{\mathrm{out}}(\mathcal{G})} + \sum_{(u,v)\in\tilde{\mathcal{E}}} \frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} \\ & - \sum_{(u,v)\in\{(p,q)\}} \frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} \Bigg\}. \end{aligned} \tag{3.12}$$

The graphs in real scenes tend to have a large scale, and adding an edge has minimal impact on the other nodes of the original graph. Therefore, we ignore the effect of adding an edge on the in/out degree of the original set of edges here. We can simplify the above equation as

$$\begin{aligned}\mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}) - \mathcal{J}_{\mathrm{VN}}^{\mathrm{D}}(\mathcal{G}') \approx \frac{1}{2n}\Bigg\{ & - \sum_{(u,v)\in\{(q,p)\}} \left(\frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} + \frac{d_u^{\mathrm{in}}(\mathcal{G}')}{d_v^{\mathrm{in}}(\mathcal{G}') d_u^{\mathrm{out}}(\mathcal{G}')^2} \right) \\ & - \sum_{(u,v)\in\{(p,q)\}} \frac{1}{d_u^{\mathrm{out}}(\mathcal{G}') d_v^{\mathrm{out}}(\mathcal{G}')} \Bigg\} < 0. \end{aligned} \tag{3.13}$$

After turning a directed edge into an undirected edge, the von Neumann entropy decreases. This means that by removing the link directions, the obtained graph loses part of the structural information contained in the directed graph. We would lose more and more structural information as we gradually remove the directed structure. Ye *et al.* [205] have the same point and use the generated graphs to experimentally corroborate this idea. The existing undirected graph convolution used on directed graphs suffers from the limitation of not being able to learn complete information about the directed graph structure.

In order to solve this problem, we propose a spectral-base GCN model for directed graph that leverages First- and Second-order Proximity in the following sections.

3.3 Second-order Directed Graph Convolution

In this section, we present our spectral-based GCN model $f(\mathbf{X}, \mathbf{A})$ for directed graphs that leverages the First- and Second-Order Proximity, called DGCN. We provide the mathematical motivation of directed graph convolution and consider a multi-layer Graph Convolutional Network which has the following layer-wise propagation rule, where

$$\begin{cases} \hat{\mathbf{A}}_F & = & \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \tilde{\mathbf{A}}_F \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \\ \hat{\mathbf{A}}_{S_{in}} & = & \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{in}} \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \\ \hat{\mathbf{A}}_{S_{out}} & = & \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{out}} \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \end{cases}, \tag{3.14}$$

and

$$\mathbf{H}^{(l+1)} = \Gamma\left(\sigma(\hat{\mathbf{A}}_F \mathbf{H}^{(l)} \mathbf{\Theta}^{(l)}), \sigma(\hat{\mathbf{A}}_{S_{in}} \mathbf{H}^{(l)} \mathbf{\Theta}^{(l)}), \sigma(\hat{\mathbf{A}}_{S_{out}} \mathbf{H}^{(l)} \mathbf{\Theta}^{(l)})\right). \tag{3.15}$$

Here, $\tilde{\mathbf{A}}_F$ is the normalized First-order Proximity matrix with self-loop and $\tilde{\mathbf{A}}_{S_{in}}$, $\tilde{\mathbf{A}}_{S_{out}}$ are the normalized Second-order Proximity matrices with self-loop, which are defined in Section 3.3.1. $\Gamma(\cdot)$ is a fusion function combines the proximity matrices together defined in Section 3.3.2. $\mathbf{\Theta}^{(l)}$ is a shared trainable weight matrix and $\sigma(\cdot)$ is an activation function. $\mathbf{H}^{(l)}$ is the matrix of activation in the l^{th} layer and $\mathbf{H}^{(0)} = \mathbf{X}$.

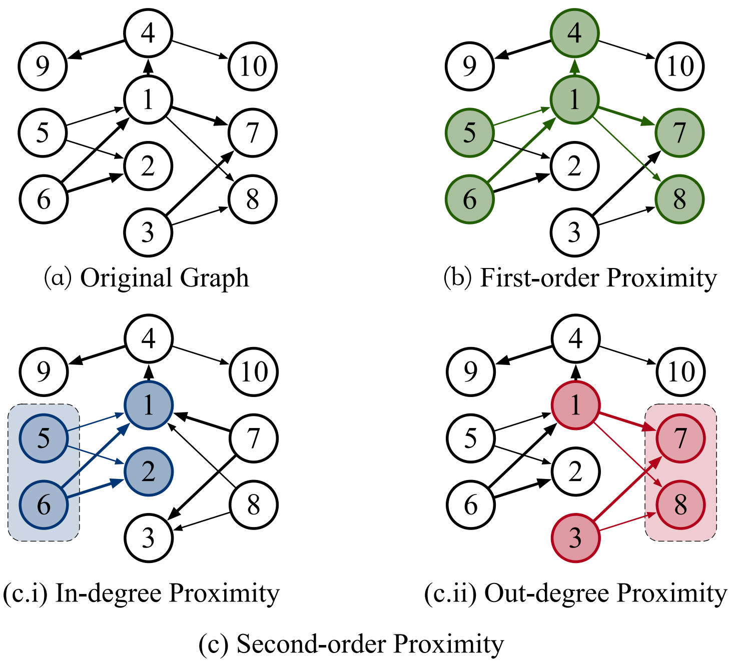
3.3.1 First- and Second-order Proximity

To conduct the feature extraction, we not only obtain the node's features from its directly adjacent nodes, but also extract the hidden information from *second-order* neighbor nodes. Different from other methods considering K-hop neighborhood information[28, 1], we define *First- and Second-order Proximity* in directed graphs and show schematically descriptions in Figure 3.3.

3.3.1.1 First-order Proximity

The first-order proximity refers to the local pairwise proximity between the nodes in a graph, which is formally defined as follows:

Definition 4. ***First-order Proximity***. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for node $u, v \in \mathcal{V}$, if edge $(u, v) \in \mathcal{E}$ or $(v, u) \in \mathcal{E}$, we define that node u and v to be first-order proximities.*



(a) Original Graph
(b) First-order Proximity
(c.i) In-degree Proximity
(c.ii) Out-degree Proximity
(c) Second-order Proximity

Figure 3.3: First- and second-order proximity examples in a directed graph. The first-order proximity of node 1 in Subgraph(b) is node $\{4, 5, 6, 7, 8\}$, but node 1 is not a first-order proximity to node 2, because the edges $\{1 \to 2\}$ or $\{2 \to 1\}$ do not exist. In Subgraph(c.i), node 1 and node 2 have second-order in-degree proximity, because they share common neighbors $\{1 \leftarrow (5, 6) \to 2\}$; while node 1 and node 3 have second-order out-degree proximity, because of $\{1 \to (7, 8) \leftarrow 3\}$ in Subgraph(c.ii).

The illustration of first-order proximity is shown in Figure 3.3(b). Each node in the graph has its own first-order proximity neighbor nodes, and we can emulate the form of the adjacency matrix to define a first-order proximity matrix to capture the properties of the first-order proximity. We define the *first-order* proximity matrix as follows:

Definition 5. ***First-order Proximity Matrix.*** *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the* first-order *proximity matrix $\mathbf{A}_F$ between node u and v for each edge (u, v) in the graph is defined as*

$$\mathbf{A}_F(u, v) = \mathbf{A}^{sym}(u, v), \tag{3.16}$$

where $\mathbf{A}^{sym}$ is the symmetric matrix of adjacency matrix $\mathbf{A}$. If there is no edge from node u to v or v to u, $\mathbf{A}_F(u, v)$ is equal to 0.

In Figure 3.3(b), it is easy to find that node 1 has *first-order* proximity with node 4. Note that the *first-order* proximity is relaxed for directed graphs. We use the symmetric matrix to replace to original one, which is inevitable losing some directed informations. For this part of the missing information, we will use another way to retain it, which is the *second-order* proximity. This problem does not exist for undirected graphs because its weights matrix is symmetric.

3.3.1.2 Second-order Proximity

The second-order proximity assumes that if two nodes share common neighbors tend to be similar. Formally, we define second-order proximity as

Definition 6. ***Second-order Proximity***. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for node $u, v \in \mathcal{V}$, if there exist any node $i \in \mathcal{V}$ and edges $\{(i,u),(i,v)\} \in \mathcal{E}$, we say that node u and v to be second-order in-degree proximities. Similarly, if there exist any node $i \in \mathcal{V}$ and edges $\{(u,i),(v,i)\} \in \mathcal{E}$, we define that node u and v to be second-order out-degree proximities.*

Second-order proximity relationships can assist in identifying nodes that possess similarity through shared neighborhood nodes, despite not being directly connected. The degree of second-order proximity between nodes is determined by the number of links with their shared neighborhood nodes. In this case, we build second-order proximity matrices, so that similar nodes can be connected with each other with different connection weights. The specific definition is as follows:

Definition 7. ***Second-order Proximity Matrices***. *In a graph $\mathcal{G}$, for node u and v, we define the second-order in-degree proximity matrix $\mathbf{A}_{S_{in}}(u,v)$ and out-degree proximity matrix $\mathbf{A}_{S_{out}}(u,v)$:*

$$\mathbf{A}_{S_{in}}(u,v) = \sum_{i \in \mathcal{V}} \frac{\mathbf{A}_{i,u}\mathbf{A}_{i,v}}{\sum_{j \in \mathcal{V}} \mathbf{A}_{i,j}} \tag{3.17}$$

and

$$\mathbf{A}_{S_{out}}(u,v) = \sum_{i \in \mathcal{V}} \frac{\mathbf{A}_{u,i}\mathbf{A}_{v,i}}{\sum_{j \in \mathcal{V}} \mathbf{A}_{j,i}}. \tag{3.18}$$

Since $\mathbf{A}_{S_{in}}(u,v)$ sums up the node degrees which array to both u and v, *i.e.*, $\sum_i \mathbf{A}\{u \leftarrow i \rightarrow v\}$, it can best reflect the similarity of the in-degree between node u and v. The larger the $\mathbf{A}_{S_{in}}(u,v)$, the higher the similarity of the second-order in-degree. Similarly, $\mathbf{A}_{S_{out}}(u,v)$ measures the second-order out-degree proximity by accumulating the node degrees from both u and v, *i.e.,* $\sum_i \mathbf{A}\{u \rightarrow i \leftarrow v\}$. If no shared nodes linked from/to u and v, we set their second-order proximity as 0. A visualization of these two proximities are shown in Figure 3.3(c) and the pseudocode is shown in Algorithm 1.

The second-order proximity of node u and v is pair-wise, thus, $\mathbf{A}_{S_{in}}(u, v) = \mathbf{A}_{S_{in}}(v, u)$ and $\mathbf{A}_{S_{out}}(u, v) = \mathbf{A}_{S_{out}}(v, u)$, *i.e.,* $\mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ are symmetric.

Algorithm 1: First- and Second-order Proximity computation procedure

```
Input: graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
       graph adjacency matrix: $\mathbf{A}$;
       feature matrix: $\mathbf{X}$
Output: First- and Second-order proximity matrices: $\mathbf{A}_F$, $\mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$
1:  for $u \in \mathcal{V}$ do
2:      for $v \in \mathcal{V}$ do
3:          $\mathbf{A}_F(u, v) \leftarrow \mathbf{A}^{\text{sym}}(u, v)$
4:          $\text{SUM}_{\text{in}} \leftarrow 0$
5:          $\text{SUM}_{\text{out}} \leftarrow 0$
6:          for $i \in \mathcal{V}$ do
7:              if $(i, u)$ & $(i, v) \in \mathcal{E}$ then
8:                  $\text{SUM}_{\text{in}} \leftarrow \text{SUM}_{\text{in}} + \frac{\mathbf{A}_{i,u}\mathbf{A}_{i,v}}{\sum_j \mathbf{A}_{i,j}}$
9:              if $(u, i)$ & $(v, i) \in \mathcal{E}$ then
10:                 $\text{SUM}_{\text{out}} \leftarrow \text{SUM}_{\text{out}} + \frac{\mathbf{A}_{u,i}\mathbf{A}_{v,i}}{\sum_j \mathbf{A}_{j,i}}$
11:         $\mathbf{A}_{S_{in}}(u, v) \leftarrow \text{SUM}_{\text{in}}$
12:         $\mathbf{A}_{S_{out}}(u, v) \leftarrow \text{SUM}_{\text{out}}$
13: return $\mathbf{A}_F, \mathbf{A}_{S_{in}}, \mathbf{A}_{S_{out}}$
```

3.3.2 Second-order Directed Graph Convolution

In the previous section, we define the first-order and second-order proximity, and have obtained three proximity symmetric matrices $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$. Similar to the authors that define the graph convolution operation on undirected graphs in Section 3.2, we use first- and second-order proximity matrices to achieve graph convolution of directed graphs.

In Equation 3.7, the adjacency matrices $\mathbf{A}_{\text{un}}$ of the graph stores the information of the graph and provides the receptive field for the filter $\boldsymbol{\Theta}$, so as to realize the transformation from the graph signal $\mathbf{X}$ to the convolved signal $\mathbf{Z}'$. It is worth noting that the first- and second-order proximity matrices we have defined have similar functions: first-order proximity provides a '1-hop' like receptive field, and second-order proximity provides a '2-hop' like receptive field. In addition to this, we need to show that the first- and second-order proximity matrices have the same properties as $\mathbf{A}_{\text{un}}$ and are able to perform spectral graph convolution. We have the following theorem of first- and second-order proximity matrices:

Theorem 1. *The Laplacian matrices of the first- and second-order proximities* $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ *and* $\mathbf{A}_{S_{out}}$ *are positive semi-definite matrices.*

Proof. According to Definition 5 and 7, $\mathbf{A}_F, \mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ are symmetric. We can consider these matrices as adjacency matrices for undirected graphs $\mathcal{G}_{\text{un},F}$, $\mathcal{G}_{\text{un},S_{in}}$ and $\mathcal{G}_{\text{un},S_{out}}$, respectively. Unlike $\mathcal{G}_{\text{un}}$, the edges of these undirected graphs are weighted, i.e., the adjacency matrix no longer has only 0 or 1 values. We mark this kind of graph as weighted undirected graph $\bar{\mathcal{G}} = (\bar{\mathcal{V}}, \bar{\mathcal{E}})$ with edge weights $\bar{w}(u,v), \forall (u,v) \in \bar{\mathcal{E}}$. The weight matrix $\bar{\mathbf{W}} = (\bar{w}(u,v)) \in \mathbb{R}^{n \times n}$ where $w(u,v) = 0$ if $(u,v) \notin \bar{\mathcal{E}}$ and weighted degree matrix $\bar{\mathbf{D}} = \text{diag}(\bar{d}(u))$, where $\bar{d}(u) = \sum_{(u,v)\in\bar{\mathcal{E}}} \bar{w}(u,v)$. The weighted Laplacian is formulated as

$$\bar{\mathbf{L}} = \bar{\mathbf{D}} - \bar{\mathbf{W}}. \tag{3.19}$$

Let's let $e_u \in \{0,1\}^n$ be the standard basis vectors (1 in the u-th coordinate, 0 's else where). The weighted Laplacian can be written in this format [186]:

$$\bar{\mathbf{L}} = \bar{\mathbf{D}} - \bar{\mathbf{W}} = \sum_{(u,v)\in\bar{\mathcal{E}}} \bar{w}(u,v)(e_u - e_v)(e_u - e_v)^T. \tag{3.20}$$

Since $(e_u - e_v)(e_u - e_v)^T$ is positive semi-definite, $\bar{\mathbf{L}}$ is a sum of non-negative coefficients with positive semi-definite matrices, which implies $\bar{\mathbf{L}}$ is positive semi-definite. So for $\mathcal{G}_{\text{un},F}$, $\mathcal{G}_{\text{un},S_{in}}$ and $\mathcal{G}_{\text{un},S_{out}}$ which belong to the weighted undirected graph, their Laplacian matrices are all positive semi-definite. $\square$

According to Definition 3.2, graph Laplacian matrix is defined on undirected graphs and requires the adjacency matrix to be symmetric, we cannot directly use the definition of the undirected graph Laplacian matrix for graph convolution operations on directed graphs. Theorem 1 gives us the theoretical basis for spectral graph analysis using first- and second-order proximity matrices. For any directed graph, we can use the above method to convert it into proximity metrics and then perform graph convolution operations.

Proximity Convolution

We define the first-order proximity convolution $f_F(\mathbf{X}, \tilde{\mathbf{A}}_F)$, second-order in- and out-degree proximity convolution $f_{S_{in}}(\mathbf{X}, \tilde{\mathbf{A}}_{S_{in}})$ and $f_{S_{out}}(\mathbf{X}, \tilde{\mathbf{A}}_{S_{out}})$:

$$\begin{cases} \mathbf{Z}_F & = & f_F(\mathbf{X}, \tilde{\mathbf{A}}_F) & = & \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \tilde{\mathbf{A}}_F \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \mathbf{X}\mathbf{\Theta} \\ \mathbf{Z}_{S_{in}} & = & f_{S_{in}}(\mathbf{X}, \tilde{\mathbf{A}}_{S_{in}}) & = & \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{in}} \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \mathbf{X}\mathbf{\Theta} \\ \mathbf{Z}_{S_{out}} & = & f_{S_{out}}(\mathbf{X}, \tilde{\mathbf{A}}_{S_{out}}) & = & \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{out}} \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \mathbf{X}\mathbf{\Theta} \end{cases}, \tag{3.21}$$

where adjacency matrix $\tilde{\mathbf{A}}$ ($\mathbf{A}$ added self-loop) is used in the definition to derive $\tilde{\mathbf{A}}_F$, $\tilde{\mathbf{A}}_{S_{in}}$ and $\tilde{\mathbf{A}}_{S_{out}}$. $\tilde{\mathbf{D}}_F = \text{Diag}(\sum_v^n \tilde{\mathbf{A}}_F(u, v))$, $\tilde{\mathbf{D}}_{S_{in}} = \text{Diag}(\sum_v^n \tilde{\mathbf{A}}_{S_{in}}(u, v))$ and $\tilde{\mathbf{D}}_{S_{out}} = \text{Diag}(\sum_v^n \tilde{\mathbf{A}}_{S_{out}}(u, v))$.

It can be seen that $\mathbf{Z}_F, \mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$ not only obtain rich first- and second-order neighbor feature information, but $\mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$ also retain the directed graph structure information. Based on these facts, we further design a fusion method to integrate the three signals together, so as to retain the characteristics of the directed structure while obtaining the surrounding information.

Fusion Operation

Directed graph fusion operation Γ is a signal fusion function of the first-order proximity convolution output $\mathbf{Z}_F$, second-order in- and out-degree proximity convolution outputs $\mathbf{Z}_{S_{in}}$ and $\mathbf{Z}_{S_{out}}$:

$$\mathbf{Z} = \Gamma(\mathbf{Z}_F, \mathbf{Z}_{S_{in}}, \mathbf{Z}_{S_{out}}). \tag{3.22}$$

Fusion function Γ can be various, such as normalization functions, summation functions, and concatenation. In practice, we find concatenation fusion has the best performance. A simple example is:

$$\mathbf{Z} = \text{Concat}(\mathbf{Z}_F, \lambda \mathbf{Z}_{S_{in}}, \mu \mathbf{Z}_{S_{out}}), \tag{3.23}$$

where Concat$(\cdot)$ is the concatenation of matrices, λ and μ are weights to control the importance between different proximities. For example, in a graph with fewer second-order neighbors, we can reduce the values of λ and μ and use more first-order information. λ and μ can be set manually or trained as learnable parameters.

For a given features matrix $\mathbf{X}$ and a directed adjacency matrix $\mathbf{A}$, after taking all the steps above, we can get the final directed graph result $\mathbf{Z} = f(\mathbf{X}, \mathbf{A})$.

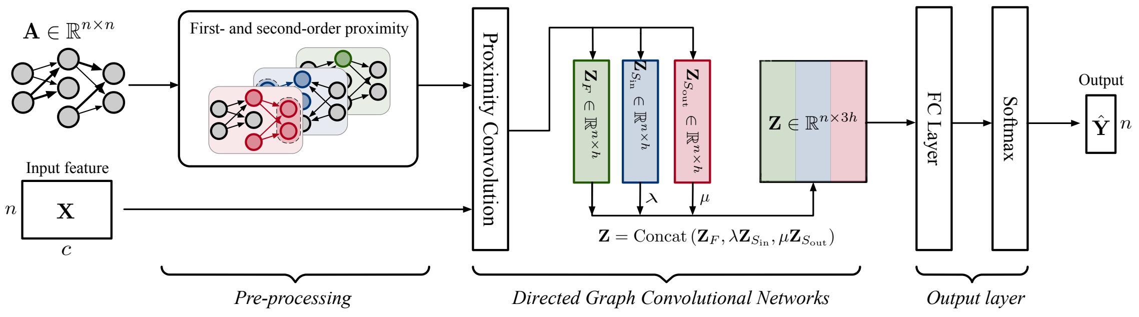

$\mathbf{A} \in \mathbb{R}^{n \times n}$
First- and second-order proximity
Input feature
n
$\mathbf{X}$
c
Proximity Convolution
$\mathbf{Z}_F \in \mathbb{R}^{n \times h}$
$\mathbf{Z}_{S_{\text{in}}} \in \mathbb{R}^{n \times h}$
$\mathbf{Z}_{S_{\text{out}}} \in \mathbb{R}^{n \times h}$
λ
μ
$\mathbf{Z} \in \mathbb{R}^{n \times 3h}$
$\mathbf{Z} = \text{Concat}\left(\mathbf{Z}_F, \lambda \mathbf{Z}_{S_{\text{in}}}, \mu \mathbf{Z}_{S_{\text{out}}}\right)$
FC Layer
Softmax
Output
$\hat{\mathbf{Y}}$
n
Pre-processing
Directed Graph Convolutional Networks
Output layer

Figure 3.4: The schematic depiction of DGCN for semi-supervised learning. Model inputs are an adjacent matrix $\mathbf{A}$ and a features matrix $\mathbf{X}$, while outputs are labels of predict nodes $\hat{\mathbf{Y}}$.

3.3.3 Implementation

In the previous section, we proposed a simple and flexible model on the directed graph, which can extract the surrounding information efficiently and retain the directed structure. In this section, we will implement our model to solve semi-supervised node classification task. More specifically, how to mine the similarity between node class using directed adjacency matrix $\mathbf{A}$ when there is no graph structure information in node feature matrix $\mathbf{X}$.

Definition 8. ***Semi-Supervised Node Classification***[1]. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$, and node feature matrix $\mathbf{X} \in \mathbb{R}^{n \times c}$, where $n = |\mathcal{V}|$ is the number of nodes and c is the feature dimension. Given a subset of nodes $\mathcal{V}_L \subset \mathcal{V}$, where nodes in $\mathcal{V}_L$ have observed labels and generally $|\mathcal{V}_L| << |\mathcal{V}|$. The task is using the labeled subset $\mathcal{V}_L$, node feature matrix $\mathbf{X}$ and adjacency matrix $\mathbf{A}$ predict the unknown label in $\mathcal{V}_{UL} = \mathcal{V} \setminus \mathcal{V}_L$.*

For this task, we build a two layer network model on directed graph with a directed adjacency matrix $\mathbf{A}$ and node feature matrix $\mathbf{X}$, is schematically depicted in Figure 3.4. In the first step, we calculate the first- and second-order proximity matrices $\hat{\mathbf{A}}_F$, $\hat{\mathbf{A}}_{S_{in}}$ and $\hat{\mathbf{A}}_{S_{out}}$ according to Equation 3.15 in the preprocessing stage. Our model can be written in the following form of forward propagation as

$$\hat{\mathbf{Y}} = f(\mathbf{X}, \mathbf{A}) = \text{Softmax}\left(\text{Concat}\left(\text{ReLU}\begin{pmatrix} \hat{\mathbf{A}}_F \mathbf{X}\boldsymbol{\Theta}^{(0)} \\ \lambda \hat{\mathbf{A}}_{S_{in}} \mathbf{X}\boldsymbol{\Theta}^{(0)} \\ \mu \hat{\mathbf{A}}_{S_{out}} \mathbf{X}\boldsymbol{\Theta}^{(0)} \end{pmatrix} \boldsymbol{\Theta}^{(1)}\right)\right). \tag{3.24}$$

In this formula, the first layer is the directed graph convolution layer. Three different proximity convolutions share a same filter weight matrix $\boldsymbol{\Theta}^{(0)} \in \mathbb{R}^{c \times h}$, which can transform the input dimension c to the embedding size h. After feature matrix $\mathbf{X}$ through the first layer, there will be three different convolved results. Then we use a fusion function to concatenate them together, λ and μ are variable weights to trade off first- and second-order feature embedding. The second layer is a fully connected layer, which we use to change feature dimension from $3h$ to d. d is the output dimension. $\boldsymbol{\Theta}^{(1)} \in \mathbb{R}^{3h \times d}$ is an embedding-to-output weight matrix. The softmax activation function is defined as $\text{Softmax}(z_i) = \frac{1}{\mathcal{Z}} \exp(z_i)$ with $\mathcal{Z} = \sum_i \exp(z_i)$ and applied row-wise. We use labeled examples to evaluate the cross-entropy error for semi-supervised node classification task:

$$L_{\varepsilon}(\mathbf{Y}, \hat{\mathbf{Y}}) = -\sum_{l \in \mathbf{V}_L} \sum_{i=1}^{d} \mathbf{Y}_{li} \ln \hat{\mathbf{Y}}_{li} \tag{3.25}$$

where $\mathbf{Y}_l$ is the actual class and $\hat{\mathbf{Y}}_l$ is the predict class of node l. $\mathcal{V}_L$ is the subset of $\mathcal{V}$ which is labeled. The pseudocode of DGCN is shown in Algorithm 2.

3.3.4 Discussion

3.3.4.1 Time and Space Complexity

For the graph convolution defined in Equation 3.21, we can use a sparse matrix to store directed adjacency matrix $\mathbf{A}$. Because we use full batch training in this task, full dataset has to be loaded into memory for every iteration. The memory space cost is $\mathcal{O}(|\mathcal{E}|)$, which means it is linear with the number of edges.

At the same time, we use the sparse matrix and the density matrix to multiply during the convolution operation. The multiplication of the sparse matrix can be considered to be linearly related to the number of edges $|\mathcal{E}|$. In the semi-supervised

Algorithm 2: DGCN for semi-supervised node classification procedure

```
    Input: graph: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;
    graph adjacency matrix: $\mathbf{A}$; features matrix: $\mathbf{X}$;
    node class label: $\mathbf{Y}$;
    weight matrices: $\mathbf{\Theta}$;
    concat weight: $\lambda$, $\mu$
    Output: Predict class matrix $\hat{\mathbf{Y}}$
 1: Initialize $\mathbf{\Theta}$ ;
     Pre-processing
 2: $\tilde{\mathbf{A}} \leftarrow \mathbf{A} + \mathbf{I}$
 3: $\tilde{\mathbf{A}}_F$, $\tilde{\mathbf{A}}_{S_{in}}$, $\tilde{\mathbf{A}}_{S_{out}} \leftarrow$ Algo1($\tilde{\mathbf{A}}$)
 4: $\tilde{\mathbf{D}}_F \leftarrow$ RowNorm($\tilde{\mathbf{A}}_F$)
 5: $\tilde{\mathbf{D}}_{S_{in}} \leftarrow$ RowNorm($\tilde{\mathbf{A}}_{S_{in}}$)
 6: $\tilde{\mathbf{D}}_{S_{out}} \leftarrow$ RowNorm($\tilde{\mathbf{A}}_{S_{out}}$)
     Second-order Directed Graph Convolution
 7: $\mathbf{Z}_F \leftarrow \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \tilde{\mathbf{A}}_F \tilde{\mathbf{D}}_F^{-\frac{1}{2}} \mathbf{X\Theta}$
 8: $\mathbf{Z}_{S_{in}} \leftarrow \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{in}} \tilde{\mathbf{D}}_{S_{in}}^{-\frac{1}{2}} \mathbf{X\Theta}$
 9: $\mathbf{Z}_{S_{out}} \leftarrow \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \tilde{\mathbf{A}}_{S_{out}} \tilde{\mathbf{D}}_{S_{out}}^{-\frac{1}{2}} \mathbf{X\Theta}$
10: $\mathbf{Z} = \mathrm{ReLU}(\mathrm{Concat}(\mathbf{Z}_F, \lambda\mathbf{Z}_{S_{in}}, \mu\mathbf{Z}_{S_{out}}))$
     Downstream Task
11: $\hat{\mathbf{Y}} = \mathrm{Softmax}(\mathrm{FC}(\mathbf{Z}))$
12: loss $\leftarrow L_\varepsilon(\mathbf{Y}, \hat{\mathbf{Y}})$
13: SGD(loss)
14: return $\hat{\mathbf{Y}}$
```

classification task, we need to multiply with $\mathbf{\Theta}^0 \in \mathbb{R}^{c \times h}$ and $\mathbf{\Theta}^1 \in \mathbb{R}^{3h \times d}$. Thus, we can obtain the computational complexity of the model as $\mathcal{O}(|\mathcal{E}|chd)$.

3.3.4.2 First- & Second-order Proximities Evaluation

Furthermore, to measure our model's ability to extract surrounding information, we define two indicators[69, 171] on the directed graph: Feature Smoothness, which is used to evaluate how much surrounding information we can obtain and Label Smoothness for evaluating how useful the obtained information is.

First, for a node in a directed graph, we need to know what its surroundings, i.e., define the connectivity between a node and its surroundings. Similar to defining the the first- and second-order proximities in directed graph, we define the first- and second-order edges as follows.

Definition 9. ***First & second-order edges in directed graph.*** *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For an order pair (u, v), where u and $v \in \mathcal{V}$. If $(u, v) \in \mathcal{E}$, we say that the order pair (u, v) is a first-order edge. If there exists a node $i \in \mathcal{V}$ that satisfies the order pairs (i, u) and $(i, v) \in \mathcal{E}$ or (u, i) and $(v, i) \in \mathcal{E}$, we define the order pair (u, v) as the second order edge. The edge set has both first & second-order edge of $\mathcal{G}$ denoted by $\mathcal{E}'$.*

For a node, if the features of its surrounding nodes are more different from its own features, it suggests that it can get more different types of information from the surrounding nodes. Intuitively, when we want to measure how much different information can a node obtain from its surroundings, we will consider measuring how frequently the different features of the surrounding nodes appear. Inspired by graph smoothness, which is an effective measure of the signal frequency in graph signal processing [222, 69], we define the feature smoothness on a directed graph as

Definition 10. ***Feature Smoothness.*** *The Feature Smoothness λ_f over node feature space is defined as follows*

$$\lambda_f = \frac{\left\| \sum_{u \in \mathcal{V}} \left(\sum_{(u,v) \in \mathcal{E}'} (\mathbf{x}_u - \mathbf{x}_v)^2 \right) \right\|_1}{|\mathcal{E}'| \cdot c}, \tag{3.26}$$

where $||\cdot||_1$ is the Manhattan norm, c is the node feature dimension and $\mathbf{x}_u$ is node feature of node u.

According to Definition 10, when λ_f is large, the feature signals of a graph have higher frequency, meaning that feature vectors $\mathbf{x}_u$ and $\mathbf{x}_v$ are more likely to be dissimilar for two connected nodes u and v in the graph. This indicates that nodes with dissimilar features tend to be connected. Intuitively, in a graph with high frequency feature sets, the context of a node can gain more information from its surrounding. Hou *et al.* [69] theoretically prove the same point from the perspective of information loss in Theorem 4: a large λ_f means that a GNN model can obtain more information from graph data.

For the node classification task, we want to determine how useful the information obtained from the surrounding nodes is. We consider that the information obtained is valid when the current node label is consistent with the surrounding node labels, otherwise, it is invalid. Based on this, we define Label Smoothness λ_l as

Definition 11. ***Label Smoothness***. *The Label Smoothness λ_l is defined as*

$$\lambda_l = \frac{\sum_{(u,v)\in\mathcal{E}'} \mathbb{I}\left(u \simeq v\right)}{|\mathcal{E}'|}, \tag{3.27}$$

where $\mathbb{I}(\cdot)$ is an indictor function and we define $u \simeq v$ if the label of node u and v are the same.

According to Definition 11, when λ_l is large, nodes with different labels tend to be interconnected, resulting in a greater negative impact on the task. Conversely, when λ_l is small, a node receives more positive information from its surrounding.

By analyzing the above-mentioned indicators and their respective variations, we can determine whether a graph model is capable of learning more useful information from the graph structure. The experimental results in Section 3.4.3 demonstrate that our model, in comparison to other methods for undirected graphs, is able to extract more valuable information from directed graphs.

3.3.4.3 Generalization to other Graph Models

Our method using first-and second-order proximity to improve the convolution receptive field and retain directed information has strong generalization ability. In most spectral-based models, we can use these proximity matrices to replace the original adjacency matrix. Take Simplifying Graph Convolutional Networks (SGC)[187] as an example, we can generalize our method to the SGC as follows:

$$\hat{\mathbf{Y}}_{\mathrm{S}'} = \mathrm{Softmax}\left(\mathbf{S}'\mathbf{X}\mathbf{\Theta}\right), \tag{3.28}$$

where we use the concatenation of first- and second-order proximity matrices $\mathbf{A}_F$,$\mathbf{A}_{S_{in}}$ and $\mathbf{A}_{S_{out}}$ to replace the origin K-th power of adjacency matrix $\mathbf{S}^K$ and $\mathbf{S}$ is the simplified adjacency matrix defined in SGC[187]. We mark the matrix after concatenation as $\mathbf{S}'$. Experimental results in Section 3.4.3 show that integrating our method can not only make the SGC model applicable to directed graphs, but also improve accuracy.

3.3.4.4 Relation with K-hop Methods

Our work considers not only the first-order relationship, but also the second-order ones when extracting surrounding information. The first-order proximity has the similar function to the 1-hop, which is to obtain the information of directly connected points. However, the reason why we do not define our second-order relationship as 2-hop is that it does not need node u and node v to have a 2 degree path directly.

For the K-hop method, and $K = 2$, they need a K degree path from u to v, *i.e.,* $\{u \rightarrow i \rightarrow v\}$ in directed graph. However, in our method, the second-order pattern diagram is transformed into $\{u \rightarrow i \leftarrow v\}$ and $\{u \leftarrow i \rightarrow v\}$, which is obvious that we get information from the shared attributes among nodes, not from the path. What's more, when evaluating the second-order proximity of nodes, we do not use the weights of the connecting edges between them, but use the sum of the normalized weights of their shared nodes.

3.4 Experiments

In this section, we evaluate the effectiveness of our model using experiments. We test on citation and co-purchase networks, and then evaluate the performance on directed and undirected dataset.

3.4.1 Datasets and Baselines

We use the several datasets to evaluate our model. In the citation network datasets: Cora-Full [10], Cora-ML [10], CiteSeer [144] , DBLP [127] and PubMed [121], nodes represent articles, while edges represent citation between articles. These datasets also include bag-of-words feature vectors for each article. In addition to the citation network, we also use the Amazon Co-purchase Network: Am-Photo and Am-Computers [145], where nodes represent goods, while edges represent two kinds of goods that are often purchased together. Bag-of-words encoded product reviews product category are also given as features, and class labels are given by the product category. In the above datasets, except DBLP and PubMed are undirected data we obtained, the rest are directed.

The origin Cora-ML has 70 classes, we combine the 2 classes that cannot perform the dataset split to the nearest class. Label rate is the fraction of nodes in the training set per class. We use 20 labeled nodes per class to calculate the label rate. The detailed datasets description is shown in Table 3.1.

We compare our model to five state-of-the-art models that can be divided into two main categories: 1) **spectral-based** GNNs including ChebNet [28], Graph

Table 3.1: Datasets Details

Datasets	Nodes	Edges	Classes	Features	Label rate
Cora-Full	19793	65311	68	8710	7.07%
Cora-ML	2995	8416	7	2879	4.67%
CiteSeer	3312	4715	6	3703	3.62%
DBLP	17716	105734	4	1639	0.45%
PubMed	18230	79612	3	500	0.33%
Am-Photo	7650	143663	8	745	2.10%
Am-Computer	13752	287209	10	767	1.45%

Convolutional Networks (GCNs) [88], Simplifying Graph Convolutional Networks (SGC) [187] and 2) **spatial-based** GNNs containing GraphSage [54] and Graph Attention Networks (GAT) [165].

For all baseline models, we use their model structure in the original papers, which including layer number, activation function selection, normalization and regularization selection, etc. It is worth noting that GraphSage has three variants in the original article using different aggregators: **mean**, **meanpool** and **maxpool**. In this chapter, we use **mean** as its aggregator as it performs best [145]. Besides, we set the mini-batch size to 512 on Am-Photo and Am-Computer and 16 on other datasets. For GCN, we set the size of hidden layer to 64 on Am-Photo and Am-Computer and 16 on other datasets. We also fix the number of attention heads to 8 for GAT, power number to 2 for SGC and $k = 2$ in ChebNets, as proposed in the respective papers.

3.4.2 Experimental Setup

We implement the DGCN and all baseline models using the python library of PyTorch and DGL. All the experiments are conducted on a server with one GPU (NVIDIA GTX-2080Ti), two CPUs (Intel Xeon E5 * 2) and Ubuntu 18.04 System.

Our Method Setup We train the two-layer DGCN model built in Section 3.3.3 for semi-supervised node classification task and provide additional experiments to explore the effect of model layers on the accuracy. We use full batch training, and each iteration will use the whole dataset. For each epoch, we initialize weights according to Glorot and Bengio[46] and initialize biases with zeros. We use Adam[86] as optimizer with a learning rate of 0.01. Validation set is using for hyperparameter optimization, which have weights(λ, μ) of first- and second-order proximity, dropout rate for all layer, L2 regularization factor for the DGCN layer and embedding size.

Dataset Split The split of the dataset will greatly affect the performance of the model[145, 90]. Especially for a single split, not only will it cause overfitting problems during training, but it is also easy to get misleading results. In our experiments, we will randomly split the data set and perform multiple experiments to obtain stable and reliable results. What's more, we also test the model under different sizes of training set in Section 3.4.3. For train/validation/test splitting, we choose 20 labels per class for training set, 500 labels for validation set and rest for test set, which follows the split in GCN[88], which marked as ***Label Split***.

3.4.3 Experimental Results

Semi-Supervised Node Classification

The comparison results of our model and baselines on seven datasets are reported in Table 3.2. Reported numbers denote classification accuracy in percent. Except DBLP and PubMed, all other datasets are directed. We train all models for a maximum of 500 epochs and early stop if the validation accuracy does not increase for 50 consecutive epochs in each dataset split, then calculate mean test accuracy and standard deviation averaged over 10 random train/validation/test splits with 5 random weight initializations. We use the following settings of hyerparameters for all datasets: drop rate is 0.5; L2 regularization is $5 \cdot 10^{-4}$; $\lambda = \mu = 1$ (we will explain the reasons in later section). Besides, we choose embedding size as 128 for Co-purchase Network: Am-Photo and Am-Computer, and 64 for others.

Our method achieved the state-of-the-art results on all datasets except Cora-Full. Although SGC achieves the best results on Cora-Full, its performances on other datasets are not outstanding. Our method achieves best results on both directed (Cora-ML, CiteSeer) and undirected (DBLP, PubMed) datasets. Our method is not significantly improved compared to GCN on the Am-Photo and Am-Computer, mainly because our model has only one convolutional layer while GCN uses two convolutional layers. The single layer network representation capability is not enough to handle large nodes graph.

Table 3.2: Mean test accuracy and standard deviation in percent. Underlined bold font indicates best results.

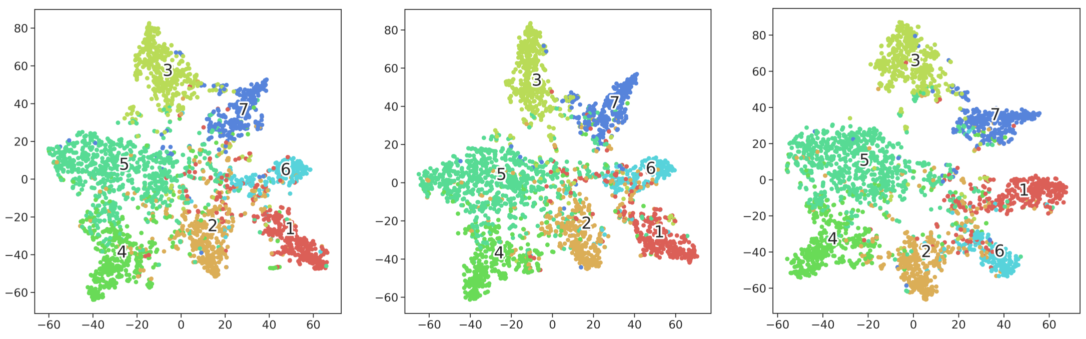
Label Split	Cora-Full	Cora-ML	CiteSeer	DBLP	PubMed	Am-Photo	Am-Computer
ChebNet	58.0 ± 0.5	79.2 ± 1.4	59.7 ± 4.0	64.0 ± 2.8	74.6 ± 2.5	82.5 ± 2.4	72.9 ± 3.0
GCN	59.1 ± 0.7	81.7 ± 1.2	64.7 ± 2.3	71.5 ± 2.7	76.8 ± 2.2	90.4 ± 1.5	81.9 ± 1.9
SGC	**61.2 ± 0.6**	80.3 ± 1.1	61.4 ± 3.4	69.2 ± 2.8	75.8 ± 2.8	89.4 ± 1.4	80.2 ± 1.2
GraphSage	58.1 ± 0.7	80.2 ± 1.6	62.8 ± 2.1	68.1 ± 2.5	75.2 ± 3.2	89.8 ± 1.9	80.4 ± 2.5
GAT	60.8 ± 0.6	81.5 ± 1.0	63.7 ± 2.0	71.8 ± 2.6	76.5 ± 2.3	90.0 ± 1.3	81.2 ± 2.5
DGCN	60.8 ± 0.6	**82.0 ± 1.4**	**65.4 ± 2.3**	**72.5 ± 2.5**	**76.9 ± 1.9**	**90.8 ± 1.1**	**82.0 ± 1.7**

First- & Second-order Proximities Evaluation

Table 3.3: Smoothness values for First- and Second-order Proximity on different datasets. 1^{st} represents first-order proximity, $1^{st}\&2^{nd}$ represents first- and second-order proximity, λ_f means Feature Smoothness and λ_l means Label Smoothness.

Smoothness	CORA-ML	CITESEER	DBLP	PUBMED
1^{st} $\lambda_f(10^{-4})$	3.759	8.719	3.579	3.135
$1^{st}\&2^{nd}$ $\lambda_f(10^{-4})$	9.789	54.720	36.810	20.480
1^{st} λ_l	0.577	0.489	0.656	0.605
$1^{st}\&2^{nd}$ λ_l	0.393	0.574	0.459	0.547

Table 3.3 reports the two smoothness values of CORA-ML, CITESEER, DBLP and PUBMED. After adding second-order proximity, the feature smoothness of CITESEER increases from 8.719×10^{-4} to 54.720×10^{-4}, while the label smoothness increases from 0.4893 to 0.5735. This change shows that the second-order proximity is very effective on this dataset, which helps increase the quantity and improve the quality of information from the surrounding. The label smoothness of other datasets decreases slightly, while their feature smoothness significantly increase. In other words, the second-order proximity widens the receptive field, thus greatly increases the amount of information obtained.



(a) GCN (b) Ours w/ 1st Proximity (c) Ours w/ 1st&2nd Proximity

Figure 3.5: 2D t-SNE[114] visualizations of the first convolutional layer feature outputs on CORA-ML dataset. (a) The data of different classes (denote by colors) are distributed more clearly and compactly in our model feature map.

Besides, the second-order proximity preserves the directed structure information which helps it to filter out valid information. The t-SNE results shown in Figure 3.5 also show that second-order proximity can help the model achieve better results.

Generalization to other Model (SGC)

In order to test the generalization ability of our model, we design an experiment according to the scheme proposed in Section 3.3.4.3. We use the concatenation of first- and second-order proximity matrices to replace the origin K-th power of adjacency matrix. The generalized SGC model is denoted by **SGC+DGCN**. In addition, we set the power time $K = 2$ to the origin SGC model. We follow the experimental setup described in the previous section, and the results are summarized in Table 3.4. Obviously, our generalized model outperforms the original model on all datasets, not only significantly improves classification accuracy, but also has more stable performance (with smaller standard deviations). Our method has good generalization ability because it has a simple structure that can be plugged into existing models easily while providing a wider receptive field by the second-order proximity matrices to improve model performance.

Table 3.4: Accuracy of origin SGC and generalized SGC. Underlined bold font indicates best results.

Label Split	Cora-ML	CiteSeer	DBLP	PubMed
SGC	80.3 ± 1.1	61.4 ± 3.4	69.2 ± 2.8	75.8 ± 2.8
SGC+DGCN	$\underline{\mathbf{82.3 \pm 1.4}}$	$\underline{\mathbf{63.8 \pm 2.0}}$	$\underline{\mathbf{71.1 \pm 2.3}}$	$\underline{\mathbf{76.5 \pm 2.3}}$

Effects of Model Depth

We investigate the effects of model depth (number of convolutional layers) on classification performance. To prevent overfitting with only one layer, we increase the difficulty of the task and set the training set size per class to 10, validation set size to 500 and the rest as test set. The other settings are the same with previous. Results are summarized in Figure 3.6. Obviously, for the datasets experimented here, the best results are obtained by a 1- or 2-layer model and test accuracy does not increase when the model goes deeper.

The main reason is overfitting. The increase in the number of model layers not only greatly increases the amount of parameters, but also widens the receptive field of the convolution operation. When DGCN has only one layer, we only need to obtain information from surrounding connected nodes and nodes of shared 1-hop neighbors. When the DGCN changes to K layers, we need consider both K-hop

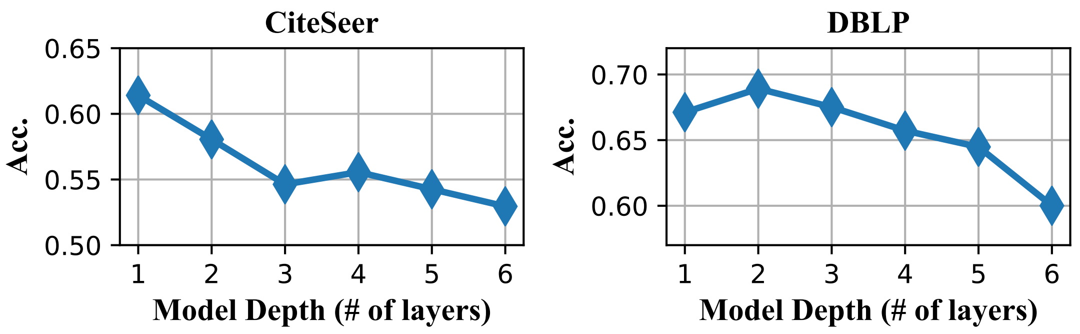


Figure 3.6: Effects to classification accuracy when DGCN goes deeper on CiteSeer and DBLP.

neighborhoods and the points that share the K-hop neighbors. For a simple semi-supervised learning task, deep DGCN obtains too much information, which easily leads to overfitting.

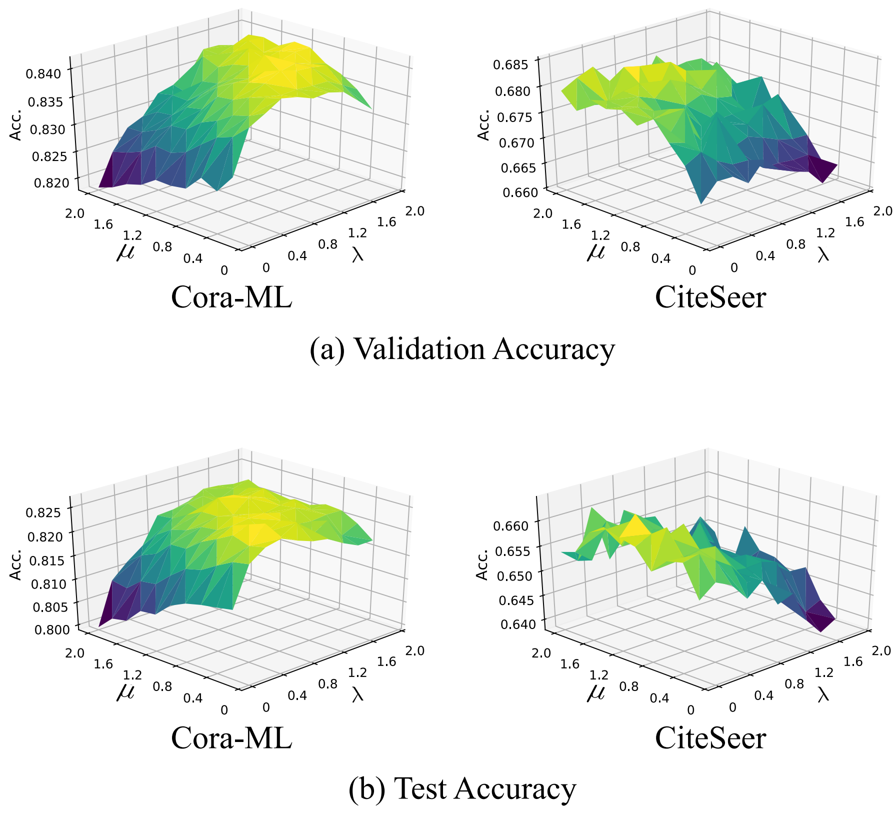


Figure 3.7: Accuracy in validation and test set for different weights (λ and μ) selection on Cora-ML and CiteSeer.

Weights Selection of First- and Second-order Proximity

We set two hyperparameters λ and μ defined in Section 3.3.2 to adjust the first- and second-order proximity weights when concatenating them. Figure 3.7 shows the accuracy in validation and test set with different weights. We set λ and μ to change within $(0, 2]$. We find that when the hyperparameters take the boundary values, the accuracies decreases significantly. When the values of the two hyperparameters are close, the accuracies of the model will increase.

This is because the second-order in-degree and out-degree proximity matrix not only represent the relationship between the nodes' shared neighbors, but also encode the structure information of the graph, which needs both in- and out-degree matrix. Besides, the unbalance of second-order in- and out-degree makes the optimal hyperparameters combination differ for datasets. Therefore, we use a combination $\lambda = \mu = 1$ that can achieve balanced performance.

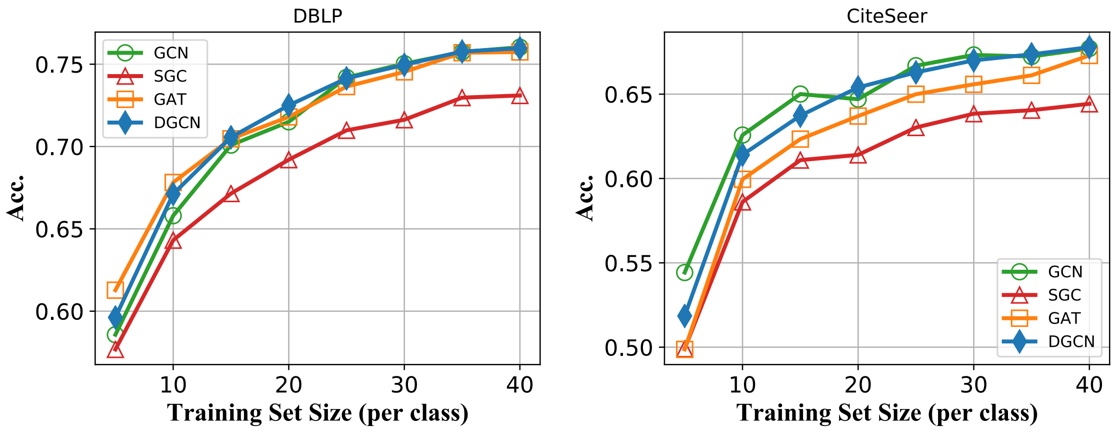


Figure 3.8: Accuracy for different training set sizes (number of labeled nodes per class) on DBLP and CiteSeer.

Effects of Training Set Size

Since the label rates for real world datasets are often small, it is important to study the performance of the model on small training set. Figure 3.8 shows how the number of training nodes per class to affect accuracy of different models on Cora-ML and DBLP. These four methods perform similarly under small training set size. As the amount of data increases, the accuracy improves greatly. Our method does not perform as well as GCN on CiteSeer and GAT on DBLP respectively. This can be attributed to the second-order proximity and model structure. In the case of less training data, the second-order proximity matrices will become very sparse, which makes it unable to supplement sufficient information. And our model has only one layer of convolution structure, which is not effective when we can not get enough information. On the country, GAT uses eight fixed attention heads and GCN uses two convolutional layers to aggregate node features.

3.5 Summary

In this chapter, we present a novel graph convolutional networks DGCN, which can be applied to the directed graphs. We define *first- and second-order* proximity on the directed graph to enable the spectral-based GCNs to generalize to directed graphs. It can retain directed features of graph and expand the convolution operation receptive field to extract and leverage surrounding information. Besides, we empirically show that this method helps increase the quantity and improve quality of information obtained. Finally, we use semi-supervised classification tasks and extensive experiments on various real-world datasets to validate the effectiveness and generalization capability of *first- and second-order* proximity and the improvements obtained by DGCNs over other models.

Chapter 4

PageRank-based Graph Convolution

In the previous chapter, we introduce the first directed graph convolution network, DGCN, which leverages first- and second-order proximity to expand the convolution receptive field and retain directed features of the graph. However, DGCN mainly improves the model structure and aggregation method by utilizing some manually defined neighborhood aggregation methods. It compensates for information loss caused by converting directed graphs to undirected graphs by increasing second-order neighbors. Hence, this method has theoretical limitations. In this chapter, we present another novel approach, DiGCN, for directed graph learning. DiGCN simplifies the spectral-based graph convolution theoretically and extends it to directed graphs by defining k^{th}-order proximity. In brief, DiGCN in Chapter 4 extends the theory and improves the structure based on the DGCN in Chapter 3.

4.1 Introduction

A majority of spectral-based GCNs transform directed graphs to undirected by relaxing its direction structure [88, 187], i.e., trivially adding edges to symmetrize the adjacency matrices. It will not only mislead message passing scheme to aggregate the features with incorrect weights but also discard distinctive direction structure [182], such as irreversible time-series relationships. Besides, there are several works that learn the specific structure by defining motifs [118], inheritance relationship [84] and second-order proximity[161]. However, these methods have to stipulate learning templates or rules in advance, and is not capable to deal with complex structures beyond their definitions.

Besides, most of the existing spectral-based GCNs enhance their capabilities of feature extraction by stacking a number of graph convolutional layers [97, 38]. However, it often leads to feature dilution as well as overfitting problem when models become deep [84, 97]. Inspired by Inception Network for image classification [155], some works [140, 198, 1] widen their layers to obtain larger receptive fields and increase learning abilities. However, they use the fixed adjacency matrix in one layer, which increases the difficulty to capture multi-scale features. A scalable neighborhood would be desirable to provide more scale information, especially for nodes belonging to communities with different sizes. Moreover, choosing a proper receptive field scheme to fuse multi-scale features together can help handle complex structures in directed graphs.

To address these issues, we first extend the spectral-based graph convolution to directed graphs by leveraging the inherent connections between graph Laplacian and stationary distributions of PageRank [124]. Since the original directed graph is not necessarily irreducible and aperiodic, the corresponding Markov chain does not have unique stationary distribution. To solve this problem, we add a chance of teleporting back to every node based on PageRank. However, the derived directed graph Laplacian is too dense, and it is extremely time-consuming to perform convolution. Thus, referring to *personalized* PageRank [5], we introduce an extra auxiliary node as the teleport connected with every node to simplify fully-connected links in PageRank. The simplified directed graph Laplacian can dramatically reduce the number of edges without changing the properties (irreducible and aperiodic). In addition, we theoretically analyze its properties and find that our Laplacian is the intermediate form between the undirected and directed graph, and the degree of conversion is determined by the teleport probability α.

Moreover, inspired by the Inception Network [155], we exploit k^{th}-order proximity between two nodes in a directed graph, which is determined through the shared k^{th}-order neighborhood structures of these two nodes. This does not require direct k^{th}-hop paths between them. By using this method, we design scalable receptive fields, which not only allows us to learn features of different sizes within one convolutional layer but also get larger receptive fields. This notion of proximity also appears in network analysis (HITS[89, 221]), psychology [138] and daily life: people who have a lot of common friends are more likely to be friends. In this way, we avoid yielding unbalanced receptive fields caused by the asymmetric paths in directed graphs. Besides that, to obtain r-range receptive field, our model only requires stacking $\lceil \log_k r \rceil$ layers instead of r GCN layers in conventional approaches.

Through experiments, we empirically show that **<u>Di</u>**rected **<u>G</u>**raph Inception **<u>C</u>**onvolutional **<u>N</u>**etworks (**DiGCN**) outperforms against competitive baselines. Additionally, our directed graph convolution is superior to GCN's convolution in the mainstream directed graph benchmarks, especially over 20% accuracy on Cora-ML dataset. Our implement is available at `https://github.com/flyingtango/DiGCN`.

4.2 PageRank-based Directed Graph Convolution

In this section, we first give the definition of directed graph Laplacian based on PageRank, which is too dense to perform convolution well. We then simplify it by *personalized* PageRank and analyze its properties. Finally, we give the definition of directed graph convolution based on the above operations.

4.2.1 Directed Graph Laplacian based on PageRank

Formally, given a directed graph (directed graph) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its adjacency matrix can be denoted as $\mathbf{A} = \{0,1\}^{n\times n}$, where $n = |\mathcal{V}|$. The nodes are described by the feature matrix $\mathbf{X} \in \mathbb{R}^{n\times c}$, with the number of features c per node. GCN [88] proposes the *spectral graph convolution* as $\mathbf{Z}_u = \hat{\mathbf{A}}_u \mathbf{X} \boldsymbol{\Theta}$, where $\mathbf{Z}_u \in \mathbb{R}^{n\times d}$ is the convolved result with output dimension d, $\boldsymbol{\Theta} \in \mathbb{R}^{c\times d}$ is trainable weight and $\hat{\mathbf{A}}_u$ is the normalized self-looped version of undirected adjacency matrix $\mathbf{A}_u$ (see [88]). GCN and its variants need the **undirected** symmetric adjacency matrix $\mathbf{A}_u$ as input, therefore, they transform asymmetric $\mathbf{A}$ to symmetric form by relaxing direction structure of directed graphs, e.g., let $\mathbf{A}_u = (\mathbf{A} + \mathbf{A}^T)/2$ in their experiments[1].

Noticing the inherent connections between graph Laplacian and stationary distributions of Markov Chains [122], we can use the properties of Markov Chains to help us solve the problem in directed graphs. Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a random walk on $\mathcal{G}$ is a Markov process with transition matrix $\mathbf{P}_{rw} = \mathbf{D}^{-1}\mathbf{A}$, where the diagonal degree matrix $\mathbf{D}(i,i) = \sum_j \mathbf{A}(i,j)$. The $\mathcal{G}$ may contain isolated

[1]There are various ways to construct an undirected graph from a directed graph. In this chapter, we consider one of the most commonly used methods that averaging edge weights when combination.

nodes in the periphery or could be formed into bipartite graph. Thus, $\mathbf{P}_{rw}$ is not necessarily *irreducible* and *aperiodic*, we can not guarantee this random walk has unique stationary distribution.

Definition 12. ***Irreducible and Aperiodic****: Given an input $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{G}$ is irreducible iff for any two vertices $v_i, v_j \in \mathcal{V}$, there is an integer $k \in \mathbb{Z}^+$, s.t. $A_{ij}^k > 0$. Meanwhile, $\mathcal{G}$ is aperiodic iff the greatest common divisor of the lengths of its cycles is one. The random walk $\mathbf{P}$ defined on $\mathcal{G}$ has the same irreducible and aperiodic properties with $\mathcal{G}$.*

To satisfy the irreducible and aperiodic properties and thus obtain an unique stationary distribution, an intuitive solution is to make all nodes in the graph connected, and PageRank is exactly what we need. PageRank is an algorithm that utilizes the concepts of Markov Chains [122] or more generally *Perron-Frobenius* Theory [7] for non-negative matrices, applied in the context of internet-based networks and linked information resources [34]. The key idea behind PageRank with a "teleport probability" is that it allows for the possibility that a person browsing the web may randomly "teleport" to a different page by following a link to an external site. This idea of being able to randomly connect to arbitrary nodes can help us deal with the problem of disconnected points in the graph. We define PageRank as

Definition 13. ***PageRank Matrix****: Let $\mathbf{N}$ be a random row-normalized non-negative matrix and let $\alpha \in (0, 1)$ be a constant. Denote further by $\mathbf{R}$ the matrix in which all entries are $1/n$. The PageRank matrix $\mathbf{M}$ is defined as*

$$\mathbf{M} = (1 - \alpha) \cdot \mathbf{N} + \alpha \cdot \mathbf{R}, \tag{4.1}$$

where α is the teleport probability [12].

We slightly modify the random walk to PageRank in the graph $\mathcal{G}$ for the which adds a small chance of teleporting back to every node, and define the directed graph with PageRank as follows:

Definition 14. ***Directed Graph with PageRank****: Given an input directed graph $\mathcal{G}$, the directed graph with PageRank $\mathcal{G}_{pr}$ is generated by adding a small chance of teleporting back from one node to other nodes. The transition matrix of $\mathcal{G}_{pr}$ is* *defined as* $\mathbf{P}_{pr} = (1-\alpha)\mathbf{P}_{rw} + \frac{\alpha}{n}\mathbf{1}^{n\times n}$, *where* $\alpha \in (0,1)$ *and be controlled to keep the probability* $\frac{\alpha}{n}$ *in a small range.*

Figure 4.1 shows the conversion from the original graph $\mathcal{G}$ to $\mathcal{G}_{pr}$ with PageRank. It is easy to prove $\mathbf{P}_{pr}$ is *irreducible* and *aperiodic*, thus, it has a unique left eigenvector π_{pr} (also called Perron vector) which is strictly positive with eigenvalue 1 according to *Perron-Frobenius* Theory [7].

The row-vector π_{pr} corresponds to the stationary distribution of $\mathbf{P}_{pr}$ and we have $\pi_{pr}(i) = \sum_{i,i\to j} \pi_{pr}(i)\mathbf{P}_{pr}(i,j)$. That is, the probability of finding the walk at vertex i is the sum of all the incoming probabilities from vertices j that have a directed edges to i. Thus, π_{pr} has analogy property with nodes degree matrix $\tilde{\mathbf{D}}_u$ in undirected graph that reflecting the connectivity between nodes [45]. Using this property, we define the directed graph Laplacian $\mathbf{L}_{pr}$ using PageRank in symmetric normalized format [24] as follows:

$$\mathbf{L}_{pr} = \mathbf{I} - \frac{1}{2}\left(\mathbf{\Pi}_{pr}^{\frac{1}{2}}\mathbf{P}_{pr}\mathbf{\Pi}_{pr}^{-\frac{1}{2}} + \mathbf{\Pi}_{pr}^{-\frac{1}{2}}\mathbf{P}_{pr}^{T}\mathbf{\Pi}_{pr}^{\frac{1}{2}}\right), \tag{4.2}$$

where we use $\mathbf{\Pi}_{pr} = \frac{1}{||\pi_{pr}||_1}\operatorname{Diag}(\pi_{pr})$ to replace $\tilde{\mathbf{D}}_u$ in the undirected graph Laplacian [88]. In contrast to $\mathbf{P}_{pr}$, this matrix is symmetric. Likewise, another work [113] also employs this idea to solve directed graph problem. However, it is defined on the strongly connected directed graphs, which is not universally applicable to any directed graphs. Our method can easily generalize to it by $\alpha \to 0$.

Adding a chance of teleporting back to every node guarantees π_{pr} exists and makes $\mathbf{L}_{pr}$ a $\mathbb{R}^{n\times n}$ dense matrix at the same time. Using this Laplacian matrix leads to greatly increase computational overhead of convolution operation and memory requirement of $\mathcal{O}(n^2)$ for training (see time usage in Section 4.4.2.2). To deal with it, we propose a simplified sparse Laplacian using *personlized* PageRank.

4.2.2 Approximate Directed Graph Laplacian based on Personalized PageRank

It is not difficult to find that the generated $\mathbf{L}_{pr}$ by adding a chance of teleporting back to every node has a lot of pseudo edges that should not exist. These dense edges increase the computational overhead while helping the directed graph to become strongly connected. To solve this issue, we reconsider the equation $\mathbf{P}_{pr} =$

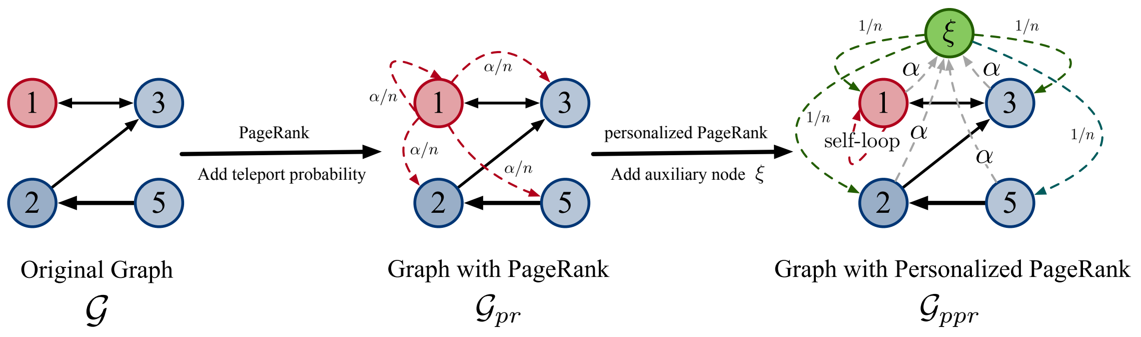


Figure 4.1: Illustration of conversion from original graph with PageRank-based methods. Note that for the sake of brevity, the connections are not fully shown. Blue nodes $2, 3, 5$ will also have the same pattern of connections like the red node 1.

$(1-\alpha)\mathbf{P}_{rw} + \frac{\alpha}{n}\mathbf{1}^{n\times n}$ of PageRank. Instead of viewing it as a combination of the random walk $\mathbf{P}_{rw}$ with a fully-connected teleport transition matrix, we can also view it as a *personalized* PageRank matrix using all the nodes as teleports. Personalized PageRank [78, 5, 130], as a variant of PageRank, focuses on the relative significance of a target node with respect to a source node in a graph [170]. In other words, different nodes can have personalization values connected to some subset of graph nodes instead of all nodes. To retain properties while sparse the Laplacian, we design an auxiliary node scheme using *personalized* PageRank.

4.2.2.1 Using Auxiliary Node as Teleport

Instead of using all the nodes of the graph itself as teleport points, our approach is to artificially add an auxiliary point for passing connectivity. More precisely, we introduce an auxiliary node $\xi \notin \mathcal{V}$ as the *personalized* PageRank teleport set $\mathcal{T} = \{\xi\}$ shown in the green node in Figure 4.1. In addition to this, we also add a self-loop to each node. This widely used trick not only enables the graph to be aperiodic, but also increases the probability of retaining its own features in message passing [88]. Formally, we define the graph after taking the above two operations as:

Definition 15. ***Directed Graph with Personalized PageRank****: Given an input $\mathcal{G}$, the directed graph with personalized PageRank $\mathcal{G}_{ppr}$ is generated by two steps: 1) adding a self-loop to each node; 2) adding an auxiliary node $\xi \notin \mathcal{V}$ as the personalized PageRank teleport set $\mathcal{T} = \{\xi\}$ and let each node in $\mathcal{G}$ has a α possibility of linking to ξ. The transition matrix $\mathbf{P}_{ppr}$ of the graph $\mathcal{G}_{ppr}$ is defined as* *follows:*

$$\mathbf{P}_{ppr} = \begin{bmatrix} (1-\alpha)\tilde{\mathbf{P}} & \alpha\mathbf{1}^{n\times 1} \\ \\ \frac{1}{n}\mathbf{1}^{1\times n} & 0 \end{bmatrix}, \quad \mathbf{P}_{ppr} \in \mathbb{R}^{(n+1)\times(n+1)}, \tag{4.3}$$

where $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$*,* $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}^{n\times n}$ *denotes the adjacency matrix with added self-loops and* $\tilde{\mathbf{D}}(i,i) = \sum_j \tilde{\mathbf{A}}(i,j)$*.*

Adding self-loops makes $\mathbf{P}_{ppr}$ aperiodic due to the greatest common divisor of the lengths of its cycles is one. Meanwhile, each node in $\mathcal{V}$ has a α possibility of linking to ξ and ξ has a $1/n$ possibility of teleporting back to every node in $\mathcal{V}$, which guarantees $\mathbf{P}_{ppr}$ to be irreducible. Thus, $\mathbf{P}_{ppr}$ has a unique left eigenvector $\pi_{ppr} \in \mathbb{R}^{n+1}$ which is strictly positive with eigenvalue 1.

4.2.2.2 Approximate Directed Graph Laplacian

Note that there are $n+1$ nodes in the graph $\mathcal{G}_{ppr}$, including n original points and an auxiliary node. Exploring the properties of the graph $\mathcal{G}_{ppr}$ is not what we are after, our target is finding the Laplacian matrix of $\tilde{\mathbf{P}}$ for spectral analysis. However, $\tilde{\mathbf{P}}$ is not necessarily irreducible, which means the eigenvector $\tilde{\pi} \in \mathbb{R}^n$ with the largest eigenvalue of $\tilde{\mathbf{P}}$ is not unique. Therefore, we need to find the approximate stationary distribution of $\tilde{\mathbf{P}}$.

As stated in the previous section, $\mathbf{P}_{ppr}$ has a unique left eigenvector $\pi_{ppr} \in \mathbb{R}^{n+1}$ which is strictly positive with eigenvalue 1. We can split π_{ppr} into two parts: $\pi_{ppr} = (\pi_{appr}, \pi_\xi)$, where $\pi_{appr} \in \mathbb{R}^n$ is the unique stationary distribution of the first n points and $\pi_\xi \in \mathbb{R}^1$ is the unique stationary distribution of the auxiliary node ξ. We find that π_{appr} can converge to stationary distribution of $\tilde{\mathbf{P}}$ according to Theorem 2.

Theorem 2. *For the first n nodes in the graph* $\mathcal{G}_{ppr}$ *except the auxiliary node, we have its unique left eigenvector* π_{appr} *and adjacency matrix* $\tilde{\mathbf{P}}$ *with added self-loops defined in Defination 15. When teleport probability* $\alpha \to 0$*,* $\pi_{appr}\tilde{\mathbf{P}} - \pi_{appr} \to 0$*. That is,* π_{appr} *converges to stationary distribution of* $\tilde{\mathbf{P}}$*.*

Proof. We start the proof from Equation $\pi_{ppr}\mathbf{P}_{ppr} = \pi_{ppr}$, leading to

$$\begin{bmatrix} \pi_{appr} & \pi_{\xi} \end{bmatrix} \begin{bmatrix} (1-\alpha)\tilde{\mathbf{P}} & \alpha\mathbf{1}^{n\times 1} \\ \\ \frac{1}{n}\mathbf{1}^{1\times n} & 0 \end{bmatrix} = \begin{bmatrix} \pi_{appr} & \pi_{\xi} \end{bmatrix}. \tag{4.4}$$

Therefore,

$$\begin{aligned} (1-\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{1}{n}\pi_{\xi}\mathbf{1}^{1\times n} &= \pi_{appr} \\ \alpha\pi_{appr}\mathbf{1}^{n\times 1} &= \pi_{\xi} \end{aligned}, \tag{4.5}$$

then,

$$(1-\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{\alpha}{n}\pi_{appr} = \pi_{appr} \tag{4.6}$$

and

$$\pi_{appr}\tilde{\mathbf{P}} - \pi_{appr} = \frac{n-1}{n}\frac{\alpha}{1-\alpha}\pi_{appr}. \tag{4.7}$$

Clearly, π_{appr} is upper bounded by $||\pi_{appr}||_{\infty} \leqslant 1$. Therefore, when $\alpha \to 0$, $\pi_{appr}\tilde{\mathbf{P}} - \pi_{appr} \to 0$. The proof is concluded. □

Thus, we can control α in a small range and use the stationary distribution of $\mathbf{P}_{ppr}$ to approximate the stationary distribution of $\tilde{\mathbf{P}}$. The Equation 4.2 can be simplified to

$$\mathbf{L}_{appr} = \mathbf{I} - \frac{1}{2}\left(\tilde{\mathbf{\Pi}}^{\frac{1}{2}}\tilde{\mathbf{P}}\tilde{\mathbf{\Pi}}^{-\frac{1}{2}} + \tilde{\mathbf{\Pi}}^{-\frac{1}{2}}\tilde{\mathbf{P}}^{T}\tilde{\mathbf{\Pi}}^{\frac{1}{2}}\right) \approx \mathbf{I} - \frac{1}{2}\left(\mathbf{\Pi}_{appr}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{appr}^{-\frac{1}{2}} + \mathbf{\Pi}_{appr}^{-\frac{1}{2}}\tilde{\mathbf{P}}^{T}\mathbf{\Pi}_{appr}^{\frac{1}{2}}\right), \tag{4.8}$$

where $\tilde{\mathbf{\Pi}} = \frac{1}{||\tilde{\pi}||_1}\operatorname{Diag}(\tilde{\pi})$ and $\mathbf{\Pi}_{appr} = \frac{1}{||\pi_{appr}||_1}\operatorname{Diag}(\pi_{appr})$. Note that $\mathbf{L}_{appr}$ retains the graph's sparsity of $\mathcal{O}(|\mathcal{E}|)$.

4.2.2.3 Generalization of Approximate Directed Graph Laplacian

As presented in Theorem 2, we can obtain the approximate Laplacian matrix for the directed graph when α tends to 0. In addition to this, we find that our method can be generalized to other forms under different conditions shown in Theorem 3.

Theorem 3. *Given an input graph $\mathcal{G}$ and its approximate directed graph Laplacian matrix $\mathbf{L}_{appr}$. When teleport probability $\alpha \to 1$, $\mathbf{\Pi}_{appr} \to \frac{1}{n}\cdot\mathbf{I}^{n\times n}$ and $\mathbf{L}_{appr} \to \mathbf{I} - \frac{1}{2}(\tilde{\mathbf{P}} + \tilde{\mathbf{P}}^{T})$. Specially, if $\mathcal{G}$ is undirected, then $\mathbf{L}_{appr} \to \mathbf{I} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$.*

Proof. From the Equation 4.5 above, we obtain

$$\alpha\pi_{appr}\mathbf{1}^{n\times 1} = \pi_{\xi}, \tag{4.9}$$

and $\pi_{ppr} = (\pi_{appr}, \pi_\xi)$ is the stationary distribution of $\mathbf{P}_{ppr}$, thus, $\pi_{appr}\mathbf{1}^{n\times 1} = 1 - \pi_\xi$ and

$$\pi_\xi = \frac{\alpha}{1+\alpha}. \tag{4.10}$$

Then, we have

$$(1-\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n} = \pi_{appr}. \tag{4.11}$$

Therefore,

$$\begin{aligned}\frac{\pi_{appr}}{||\pi_{appr}||_1} &= \frac{(1-\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n}}{1-\pi_\xi} \\ &= \frac{(1-\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n}}{\frac{1}{1+\alpha}} \\ &= (1-\alpha)(1+\alpha)\pi_{appr}\tilde{\mathbf{P}} + \frac{1}{n}\alpha\mathbf{1}^{1\times n}\end{aligned}. \tag{4.12}$$

Since π_{appr} is stationary distribution and $\tilde{\mathbf{P}}$ is transition matrix, $||\pi_{appr}\tilde{\mathbf{P}}||_\infty \leqslant ||\pi_{appr}||_\infty||\tilde{\mathbf{P}}||_\infty \leqslant 1$. It is easy to show when $\alpha \to 1$, $\frac{\pi_{appr}}{||\pi_{appr}||_1} \to \frac{1}{n}\mathbf{1}^{1\times n}$ and $\mathbf{\Pi}_{appr} = \frac{1}{||\pi_{appr}||_1}\operatorname{Diag}(\pi_{appr}) \to \frac{1}{n}\cdot\mathbf{I}^{n\times n}$. Besides, for $\mathbf{L}_{appr}$ as follows

$$\mathbf{L}_{appr} \approx \mathbf{I} - \frac{1}{2}\left(\mathbf{\Pi}_{appr}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{appr}^{-\frac{1}{2}} + \mathbf{\Pi}_{appr}^{-\frac{1}{2}}\tilde{\mathbf{P}}^T\mathbf{\Pi}_{appr}^{\frac{1}{2}}\right), \tag{4.13}$$

when $\alpha \to 1$,

$$\begin{aligned}\mathbf{L}_{appr} &\to \mathbf{I} - \frac{1}{2}\left(\frac{1}{\sqrt{n}}\tilde{\mathbf{P}}\sqrt{n} + \frac{1}{\sqrt{n}}\tilde{\mathbf{P}}^T\sqrt{n}\right) \\ &\to \mathbf{I} - \frac{1}{2}\left(\tilde{\mathbf{P}} + \tilde{\mathbf{P}}^T\right)\end{aligned} \tag{4.14}$$

Meanwhile, in the case that $\mathcal{G}$ is undirected, $\tilde{\mathbf{P}}$ is symmetric and $\tilde{\mathbf{P}} = \tilde{\mathbf{P}}^T = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$, $\mathbf{L}_{appr}$ coverages to:

$$\mathbf{L}_{appr} \to \mathbf{I} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}. \tag{4.15}$$

The proof is concluded. $\square$

We show in Theorem 3 and Equation 4.16 that two common used undirected Laplacian matrices are special cases of our method under certain conditions: the trivial-symmetric form $\mathbf{I} - \frac{1}{2}(\tilde{\mathbf{P}} + \tilde{\mathbf{P}}^T)$ mentioned in Section 4.2.1 and random-walk normalized form $\mathbf{I} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ [25].

$$\mathbf{L}_{appr} \xrightarrow{\alpha \to 1} \underbrace{\mathbf{I} - \frac{1}{2}\left(\tilde{\mathbf{P}} + \tilde{\mathbf{P}}^T\right)}_{\text{trivial-symmetric form}} \xrightarrow[\tilde{\mathbf{P}}=\tilde{\mathbf{P}}^T=\tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}]{\mathcal{G}\text{ is undirected}} \underbrace{\mathbf{I} - \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}}_{\text{random walk form}} \quad . \tag{4.16}$$

When α tends to 1, the form of our method is closer to the form of undirected graph Laplacian. That is to say α can control the degree of conversion from a directed form to an undirected form. The smaller α retains the more directed properties, and vice versa.

4.2.3 Directed Graph Convolution

As we have defined the directed graph Laplacian in Equation 4.8 and it is symmetric, we can follow the spectral analysis in GCN [88] to derive the definition of the directed graph convolution as:

$$\mathbf{Z} = \frac{1}{2}\left(\mathbf{\Pi}_{appr}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{appr}^{-\frac{1}{2}} + \mathbf{\Pi}_{appr}^{-\frac{1}{2}}\tilde{\mathbf{P}}^T\mathbf{\Pi}_{appr}^{\frac{1}{2}}\right)\mathbf{X}\mathbf{\Theta}, \tag{4.17}$$

where $\mathbf{Z} \in \mathbb{R}^{n\times d}$ is the convolved result with d output dimension, $\mathbf{X} \in \mathbb{R}^{n\times c}$ is node feature matrix and $\mathbf{\Theta} \in \mathbb{R}^{c\times d}$ is trainable weight. Note that we carry out row normalization to the input weighted adjacency matrix. This propagation scheme has complexity $\mathcal{O}(|\mathcal{E}|cd)$ which is same with GCN [88], as directed graph Laplacian is sparse and can be calculated during preprocessing. The pseudocode of directed graph convolution is shown in Algorithm 3.

Algorithm 3: Directed graph convolution procedure

```
   Input: Adjacency matrix: $\mathbf{A}$, features matrix: $\mathbf{X}$, teleport probability $\alpha$,
          learnable weights: $\mathbf{\Theta}$
   Output: Convolution result $\mathbf{Z}$
1: Initialize $\mathbf{\Theta}$;
2: $\tilde{\mathbf{A}} \leftarrow \mathbf{A} + \mathbf{I}^{n\times n}$ ;
3: $\tilde{\mathbf{P}} \leftarrow \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ ;
4: $\mathbf{P}_{ppr} \leftarrow \mathrm{AddAuxNode}(\tilde{\mathbf{P}}, \alpha)$;
5: $\pi_{ppr} \leftarrow \mathrm{LeftEVD}(\mathbf{P}_{ppr})$;
6: $\pi_{appr} \leftarrow \pi_{ppr}(1:n)$;
7: $\mathbf{\Pi}_{appr} \leftarrow \frac{1}{||\pi_{appr}||_1}\mathrm{Diag}(\pi_{appr})$;
8: $\mathbf{Z} \leftarrow \frac{1}{2}\left(\mathbf{\Pi}_{appr}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{appr}^{-\frac{1}{2}} + \mathbf{\Pi}_{appr}^{-\frac{1}{2}}\tilde{\mathbf{P}}^T\mathbf{\Pi}_{appr}^{\frac{1}{2}}\right)\mathbf{X}\mathbf{\Theta}$;
9: return $\mathbf{Z}$
```

4.3 Directed Graph Inception Network

In this section, first, we introduce k^{th}-order Proximity as scalable receptive field and then we present **DiGCN**, a multi-scale inception network, to learn from features of different size in directed graphs.

4.3.1 Scalable Receptive Field based on k^{th}-order Proximity

We start by explaining the feature spreading ways in GCNs. Xu et al. [201] have shown that the information of node i spreads to node j in an analogous random walk manner, which means path is the way of feature transmission and the size of receptive field is determined by the length of the path in a graph. However, in directed graph, long paths only exist between a few points and are often not bidirectional, which is not conducive to obtaining global features. Meanwhile, different communities have various node degrees of in and out, which may cause unbalanced receptive fields (paths) in directed graphs. To solve this problem, we propose k^{th}-order Proximity in directed graphs which not only obtains the node's features from its directly adjacent nodes, but also extract the hidden information from k^{th}-order neighbor nodes. That is, if two nodes share common neighbors, they tend to be similar.

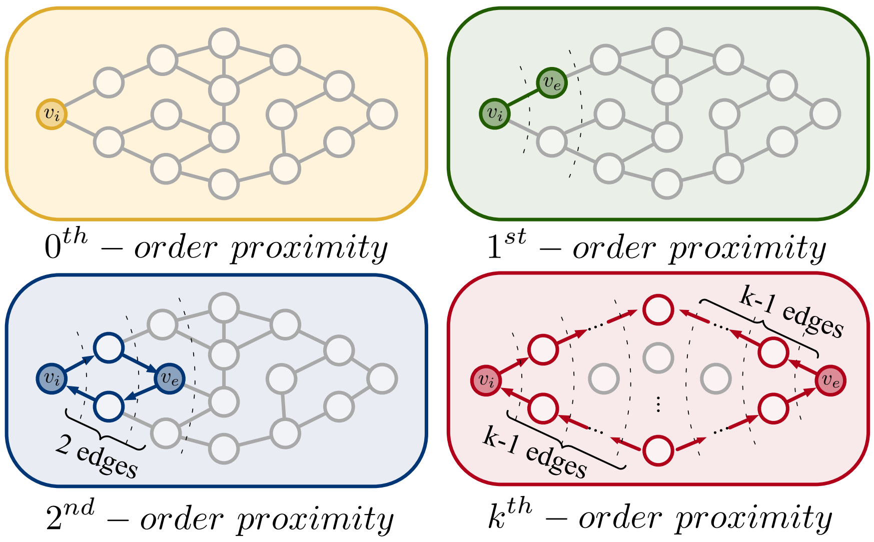

v_i
0^{th} − order proximity
v_i
v_e
1^{st} − order proximity
v_i
v_e
2 edges
2^{nd} − order proximity
v_i
v_e
k-1 edges
k-1 edges
k^{th} − order proximity

Figure 4.2: Illustration of k^{th} -order proximity.

Definition 16. k^{th}***-order Proximity***. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for $k \geqslant 2$, if $\exists\, e \in \mathcal{V}$ and a path between node i and j ($i, j \in \mathcal{V}$) in this form: $v_i \underbrace{\rightarrow \cdots \rightarrow}_{k-1\ \ edges} v_e \underbrace{\leftarrow \cdots \leftarrow}_{k-1\ \ edges} v_j$, we define this path as k^{th}-order meeting path $\mathcal{M}_{i,j}^{(k)}$. Similarity, the k^{th}-order diffusion path ${\mathcal{D}_{i,j}}^{(k)}$ is $v_i \underbrace{\leftarrow \cdots \leftarrow}_{k-1\ \ edges} v_e \underbrace{\rightarrow \cdots \rightarrow}_{k-1\ \ edges} v_j$. If there exist both ${\mathcal{M}_{i,j}}^{(k)}$ and ${\mathcal{D}_{i,j}}^{(k)}$ between node i and j, we think they are k^{th}-order proximity and e is their k^{th}-order common neighbor. Note that one node is 0^{th}-order proximity with itself and 1^{st}-order proximity with its directly connected neighbors.*

The schematic of k^{th} -order proximity shows in Figure 4.2. For a directed graph $\mathcal{G}$, we use k^{th}-order proximity to generate k receptive fields based on the input adjacency matrix $\mathbf{A}$. Multi-layer networks can be implemented by stacking Inception blocks. Based on Definition 16, we build a k^{th}-order proximity matrix to connect similar nodes together.

Definition 17. k^{th}***-order Proximity Matrix***. *In order to model the k^{th}-order proximity, we define the k^{th}-*order *proximity matrix $\mathbf{P}^{(k)}(k \in \mathbb{Z})$ in the graph $\mathcal{G}$:*

$$\mathbf{P}^{(k)} = \begin{cases} \mathbf{I} & k = 0 \\ \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}} & k = 1 \\ \text{Intersect}\left((\mathbf{P}^{(1)})^{k-1} (\mathbf{P}^{(1)^T})^{k-1}, (\mathbf{P}^{(1)^T})^{k-1} (\mathbf{P}^{(1)})^{k-1} \right)/2 & k \geqslant 2 \end{cases}. \tag{4.18}$$

$\tilde{\mathbf{A}}$ is the adjacency matrix with self-loops of $\mathcal{G}$ and $\tilde{\mathbf{D}}$ is corresponding diagonalized degree matrix. Intersect$(\cdot)$ *denotes the element-wise intersection of matrices that only when the corresponding positions have both meeting and diffusion paths, the sum operation is performed, otherwise, it is 0.*

The k^{th}-order proximity matrix $\mathbf{P}^{(k)}$ is symmetric if $k \geqslant 2$ because of the intersection operation. k represents distance between two similar nodes, that is, the size of the receptive fields. We can get scalable receptive fields by setting different k.

4.3.2 Multi-scale Inception Network Structure

Based on the proposed k^{th}-order proximity matrix, we define the multi-scale directed graph convolution as:

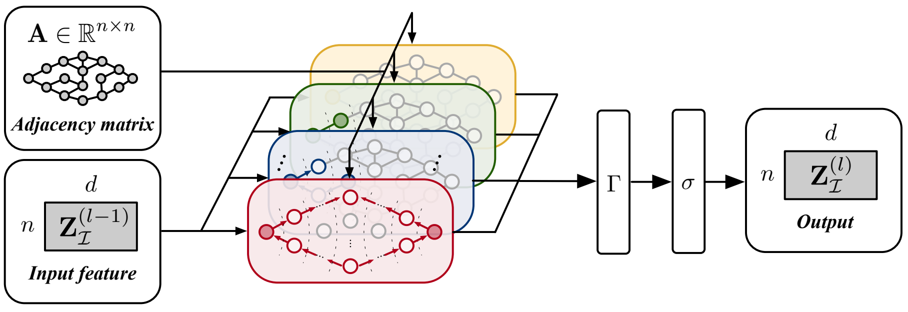
$$\mathbf{Z}^{(k)} = \begin{cases} \mathbf{X}\mathbf{\Theta}^{(0)} & k=0 \\ \frac{1}{2}\left(\mathbf{\Pi}^{(1)\frac{1}{2}}\mathbf{P}^{(1)}\mathbf{\Pi}^{(1)-\frac{1}{2}} + \mathbf{\Pi}^{(1)-\frac{1}{2}}\mathbf{P}^{(1)T}\mathbf{\Pi}^{(1)\frac{1}{2}}\right)\mathbf{X}\mathbf{\Theta}^{(1)} & k=1 \\ \mathbf{W}^{(k)-\frac{1}{2}}\mathbf{P}^{(k)}\mathbf{W}^{(k)-\frac{1}{2}}\mathbf{X}\Theta^{(k)} & k \geqslant 2 \end{cases} . \tag{4.19}$$

$\mathbf{Z}^{(k)} \in \mathbb{R}^{n\times d}$ are convolved results with d output dimension, $\mathbf{X} \in \mathbb{R}^{n\times c}$ is node feature matrix, $\mathbf{W}^{(k)}$ is diagonalized weight matrix of $\mathbf{P}^{(k)}$ and $\mathbf{\Theta}^{(k)} \in \mathbb{R}^{c\times d}$ is trainable weight. Note that when $k = 1$, $\mathbf{Z}^{(1)}$ is calculated by directed graph convolution with $\mathbf{P}^{(1)}$ in Section 4.2.3 and $\mathbf{\Pi}^{(1)}$ is the corresponding approximate diagonalized eigenvector.

Inspired by the **Inception module** proposed in [155], we build the multi-scale directed graph Inception network. We can compare $\mathbf{P}^{(k=0)}$ to 1×1 convolution kernel and treat $\mathbf{Z}^{(k=0)}$ as a skip connection term carrying non-smoothed features. Moreover, $\mathbf{Z}^{(k\geqslant 1)}$ is designed to encode multi-scale directed structure features. Finally, we use fusion operation Γ to fusion multi-scale features together as an Inception block $\mathbf{Z}_{\mathcal{I}}$:

$$\mathbf{Z}_{\mathcal{I}} = \sigma(\Gamma(\mathbf{Z}^{(0)}, \mathbf{Z}^{(1)}, ..., \mathbf{Z}^{(k)})), \tag{4.20}$$

where σ is activation function. Fusion function Γ can be various, such as normalization, summation and concatenation. In practice, we use Γ to keep the feature dimensions unchanged, that is keeping $\mathbf{Z}_{\mathcal{I}} \in \mathbb{R}^{n\times d}$ for stacking the same block. The schematic of Inception block shows in Figure 4.3. We convolute the k^{th}-order proximity matrices with input feature matrix $\mathbf{Z}_{\mathcal{I}}^{(l-1)}$ and gain the output $\mathbf{Z}_{\mathcal{I}}^{(l)}$ after fusion. We encapsulate this process as an Inception block shown in the Figure 4.3, where $l \in \mathbb{Z}^+$ is the number of layers and the initial input $\mathbf{Z}_{\mathcal{I}}^{(0)} = \mathbf{X}$. The pseudocode of DiGCN is shown in Algorithm 4.



$\mathbf{A} \in \mathbb{R}^{n\times n}$
Adjacency matrix
d
n
$\mathbf{Z}_{\mathcal{I}}^{(l-1)}$
Input feature
Γ
σ
d
n
$\mathbf{Z}_{\mathcal{I}}^{(l)}$
Output

Figure 4.3: Illustration of Inception block of l^{th} layer.

We notice that a recent work SIGN uses a similar Inception structure to handle large scale graph learning [140]. Differently, they use SGC [187] as basic block which is not applicable to directed graphs and concatenate these block of different size together into a FC layer. However, concatenation is a kind of features fusion method, and in some cases, the effect of concatenation is not as good as summation, we illustrate this in Section 4.4.2.2.

Algorithm 4: DiGCN procedure

```
    Input: Adjacency matrix: $\mathbf{A}$, features matrix: $\mathbf{X}$;
    width of Inception block $k$, fusion function $\Gamma$;
    activatation function: $\sigma$, teleport probability $\alpha$ ;
    learnable weights: $\{\mathbf{\Theta}^0, \mathbf{\Theta}^1, ..., \mathbf{\Theta}^k\}$
    Output: Convolution result $\mathbf{Z}_{\mathcal{I}}$
 1: Initialize $\{\mathbf{\Theta}^0, \mathbf{\Theta}^1, ..., \mathbf{\Theta}^k\}$;
 2: for $i \leftarrow 0$ to $k$ do
 3:     if $i = 0$ then
 4:         $\mathbf{P}^{(i)} \leftarrow \mathbf{I}^{n\times n}$;
 5:     else if $i = 1$ then
 6:         $\tilde{\mathbf{A}} \leftarrow \mathbf{A} + \mathbf{I}^{n\times n}, \tilde{\mathbf{D}} \leftarrow \mathrm{RowSum}(\tilde{\mathbf{A}})$, $\mathbf{P}^{(i)} \leftarrow \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$;
 7:     else
 8:         $\mathbf{P}^{(i)} \leftarrow \mathrm{Intersect}((\mathbf{P}^{(1)})^{k-1}(\mathbf{P}^{(1)^T})^{k-1}, (\mathbf{P}^{(1)^T})^{k-1}(\mathbf{P}^{(1)})^{k-1})/2$;
 9:         $\mathbf{W}^{(i)} \leftarrow \mathrm{RowSum}(\mathbf{P}^{(i)})$;
10: for $j \leftarrow 0$ to $k$ do
11:     if $j = 0$ then
12:         $\mathbf{Z}^{(0)} \leftarrow \mathbf{X}\mathbf{\Theta}^{(0)}$;
13:     else if $j = 1$ then
14:         $\mathbf{Z}^{(1)} \leftarrow \mathrm{DirectedGraphConv}(\mathbf{P}^{(1)}, \alpha)$;
15:     else
16:         $\mathbf{Z}^{(j)} \leftarrow \mathbf{W}^{(j)^{-\frac{1}{2}}}\mathbf{P}^{(j)}\mathbf{W}^{(j)^{-\frac{1}{2}}}\mathbf{X}\mathbf{\Theta}^{(j)}$;
17: $\mathbf{Z}_{\mathcal{I}} \leftarrow \sigma(\Gamma(\mathbf{Z}^{(0)}, \mathbf{Z}^{(1)}, ..., \mathbf{Z}^{(k)}))$;
18: return $\mathbf{Z}_{\mathcal{I}}$
```

Generalization to other Models. Our method using k^{th}-order proximity to improve the convolution receptive field has strong generalization ability. In most spectral-based models, we can use our Inception block to replace the original layer (see Table 4.1). Our Inception block can generalizate to other models only need to modify some parameters.

Table 4.1: Generalization of Inception block. $\mathbf{\Lambda}$ is Laplacian eigenvalue matrix defined in ChebNet [28] and $\mathbf{A}_u$ is the symmetric form of $\mathbf{A}$.

	Undir.	Dir.	Adj	Scale Range	Weights	Fusion Method
ChebNet [28]	✓		$\mathbf{\Lambda}$	$[0, 1, .., k]$	$\mathbf{\Theta}^0, ..., \mathbf{\Theta}^k$	Sum
GCN [88]	✓		$\mathbf{A}_u$	1	$\mathbf{\Theta}$	None
SGC [187]	✓		$\mathbf{A}_u$	k	$\mathbf{\Theta}$	None
SIGN [140]	✓		$\mathbf{A}_u$	$[0, 1, .., k]$	$\mathbf{\Theta}^0, ..., \mathbf{\Theta}^k$	Concate
Ours (DiGCN)	✓	✓	$\mathbf{A}$	$[0, 1, .., k]$	$\mathbf{\Theta}^0, ..., \mathbf{\Theta}^k$	Any Γ

Taking SGC [187] as an example, we can generalize our method to the SGC by replacing the origin k^{th} -power of adjacency matrix by Inception block. Experimental results in Section 4.4.2.2 show that integrating our method can help improve accuracy.

Time and Space Complexity. For directed graph convolution defined in Equation 4.17, we can use a sparse matrix to store Laplacians. And since we use full batch training, the memory space cost for one adjacency matrix is $\mathcal{O}(|\mathcal{E}|)$.

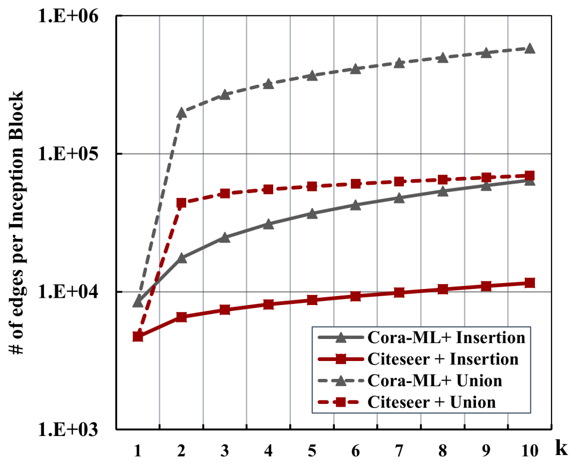


Figure 4.4: Number of edges per Inception block.

For the Inception block defined in Equation 4.20, due to the asymmetry of the directed graphs mentioned in Section 4.3.1, long paths normally exist between a few points and are not bidirectional. Thus, using k^{th}-order proximity will get an unbalanced receptive field and introduce computational complexity. Intersection and union of meeting and diffusion paths both can handle the unbalancing problem. We compare the number of edges per Inception block with different k on Cora-ML [10] and CiteSeer [144] shown in Figure 4.4 and find that the edges in k^{th}-order matrix will not increase exponentially and intersection does help to reduce memory consumption. Thus, the memory space cost for one Inception block in practical is $\mathcal{O}(k|\mathcal{E}|)$. However, the worst case does exist when the input graph is undirected and strongly connected. Though it is unsuitable to our model, which mainly treats reducible directed graphs, the worst space complexity is $\mathcal{O}(k|\mathcal{V}|^2)$.

We can calculate eigenvalue decomposition in the Equation 4.8 during the preprocessing and store the results, therefore the computational complexity is $\mathcal{O}(|\mathcal{V}|^3)$. At the same time, we use the sparse matrix multiplication. Thus, we can obtain the complexity of convolution as $\mathcal{O}(k|\mathcal{E}|cd)$.

4.4 Experiments

We conduct extensive experiments to evaluate the effectiveness of our model.

4.4.1 Experimental Settings

Here, we detail datasets, the baseline settings and model implementation in experiments. We implement the DiGCN and all the baseline models using the python library of PyTorch [2], Pytorch-Geometric [40] and DGL 0.3 [3]. All the experiments are conducted on a server with one GPU (NVIDIA RTX-2080Ti), two CPUs (Intel Xeon E5 * 2) and Ubuntu 18.04 System.

4.4.1.1 Datasets and Splitting

We use several **directed graph** datasets including citation networks: CORA-ML [10] and CITESEER [144], and Amazon Co-purchase Networks: AM-PHOTO and AM-COMPUTER [145]. The split of the datasets will greatly affect the performance of the models [145, 90]. Especially for a single split, not only will it cause overfitting problems during training, but it is also easy to get misleading results. Thus, in our experiments, we randomly split the datasets and perform multiple experiments to obtain stable and reliable results. For train/validation/test split, following the rules in GCN [88], we choose 20 labels per class for training set, 500 labels for validation set and rest for test set.

[2]https://pytorch.org
[3]https://www.dgl.ai

We use four open access datasets to test our method. Label rate is the fraction of nodes in the training set per class. We use 20 labeled nodes per class to calculate the label rate.

Table 4.2: Datasets Details

Datasets	Nodes	Edges	Classes	Features	Label rate
Cora-ML	2995	8416	7	2879	4.67%
CiteSeer	3312	4715	6	3703	3.62%
Am-Photo	7650	143663	8	745	2.10%
Am-Computer	13752	287209	10	767	1.45%

4.4.1.2 Baselines

We compare our model to eight state-of-the-art models that can be divided into four main categories: 1) **spectral-based** GNNs including ChebNet [28], GCN [88], SGC [187] , APPNP [90] and InfoMax [166]; 2) **spatial-based** GNNs containing GraphSage [54] and GAT [165]; 3) **directed** GNNs including DGCN [161] (we do not use [113] because it only apply for strongly connected graph which needs cropping the original dataset to meet its settings); 4) **graph Inception** having SIGN [140].

For all baseline models, we use their model structure in the original papers, including layer number, activation function selection, normalization and regularization selection, etc. It should be noted that GraphSage has three variants in the original article using different aggregators: **mean**, **meanpool**, and **maxpool**. In this chapter, we use **mean** as its aggregator since it performs best [145]. Detailed hyper-parameter settings are shown in Table 4.3.

4.4.2 Semi-supervised Node Classification

Node classification is a common task used to measure graph models. In this chapter, we adopt the task of *Semi-supervised Node Classification in Directed Graphs* to verify the learning ability of models. Compared with the common experiments for undirected graphs [88, 187, 180], the challenge is that the given adjacency matrix **A** is asymmetric, which means message passing has its direction. Based on the method proposed above, we build several models to deal with this problem.

Table 4.3: The hyper-parameters of baselines.

Model	layers	lr	weight decay	dropout	hidden dimension	Others
ChebNet [28]	2	0.01	5e-4	0.5	CORA-ML & CITESEER:16 others:64	num-hop=2
GCN [88]	2	0.01	5e-4	0.5	64	-
SGC [187]	1	0.1	5e-4	0.5	-	power-times=2
APPNP [90]	2	0.01	5e-4	0.5	64	$\alpha = 0.1$
InfoMax [166]	1	0.001	5e-4	0	2048	max-LR-iter=150
GraphSage [54]	2	0.005	5e-4	0.6	CORA-ML & CITESEER:16 others:64	mean
GAT [165]	2	0.005	5e-4	0.6	CORA-ML & CITESEER:8 others:32	heads=16
DGCN [161]	2	0.01	5e-4	0.5	64	concatenation
SIGN [140]	2	0.1	5e-4	0.5	64	$k = 2$

4.4.2.1 Implementation

In this section, we implement our model to solve directed graph semi-supervised node classification task. More specifically, how to mine the similarity between node class using adjacency matrix $\mathbf{A}$ when there is no graph structure information in node feature matrix $\mathbf{X}$. The task definition can be seen in Definition 8.

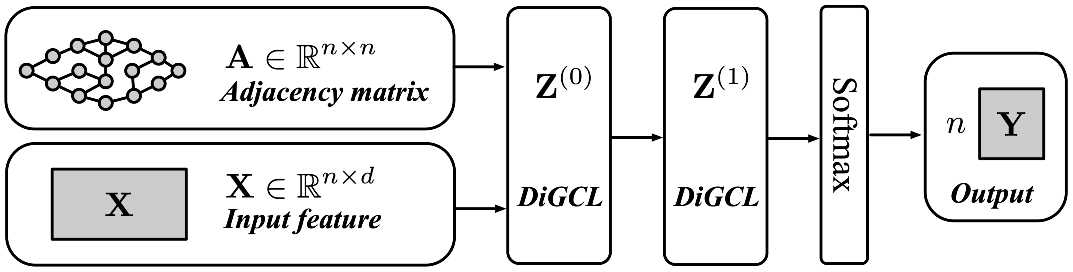


Figure 4.5: The schematic depiction of model using directed graph convolution.

For this task, we first build a two layer network model on directed graphs which only use directed graph convolution. We schematically show the model in Figure 4.5 and use DiGCL to represent directed graph convolution layer. Model inputs are the adjacent matrix $\mathbf{A}$ and features matrix $\mathbf{X}$, while outputs are labels of predict nodes $\mathbf{Y}$. Our model can be written in the following form of forward propagation:

$$\begin{array}{rcl} \hat{\mathbf{A}} & = & \frac{1}{2}\left(\mathbf{\Pi}_{appr}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{appr}^{-\frac{1}{2}} + \mathbf{\Pi}_{appr}^{-\frac{1}{2}}\tilde{\mathbf{P}}^{T}\mathbf{\Pi}_{appr}^{\frac{1}{2}}\right) \\ \mathbf{Y} & = & \mathrm{softmax}\left(\hat{\mathbf{A}}(\mathrm{ReLU}(\hat{\mathbf{A}}\mathbf{X}\mathbf{\Theta}^{(0)})\mathbf{\Theta}^{(1)}\right) \end{array} . \tag{4.21}$$

Moreover, we build the DiGCN model using k^{th}-order proximity as an Inception block, which can be written in the following form of forward propagation:

$$\begin{array}{rcl} \mathbf{Z}_{\mathcal{I}}^{(l)} & = & \sigma(\Gamma(\mathbf{Z}^{(0)}, \mathbf{Z}^{(1)}, ..., \mathbf{Z}^{(k)})^{(l-1)}) \\ \mathbf{Y}_{\mathcal{I}} & = & \mathrm{softmax}\left(\mathbf{Z}_{\mathcal{I}}^{(l)}\right) \end{array} , \tag{4.22}$$

where $l \in \mathbb{Z}^{+}$ is the number of layers and $(\cdot)^{(l)}$ represents the weights of l^{th} layer. The activation function σ and the fusion function Γ are chosen differently according to the experiments in the chapter. We show the model in Figure 4.6, model inputs are an adjacent matrix $\mathbf{A}$ and a features matrix $\mathbf{X}$, while outputs are labels of predict nodes $\mathbf{Y}_{\mathcal{I}}$.

We perform a grid search on the hyperparameters: lr in range [0.001, 0.1], weight decay in range [1e-5, 1e-3] and dropout rate in range [0.3,0.8] and use all labeled

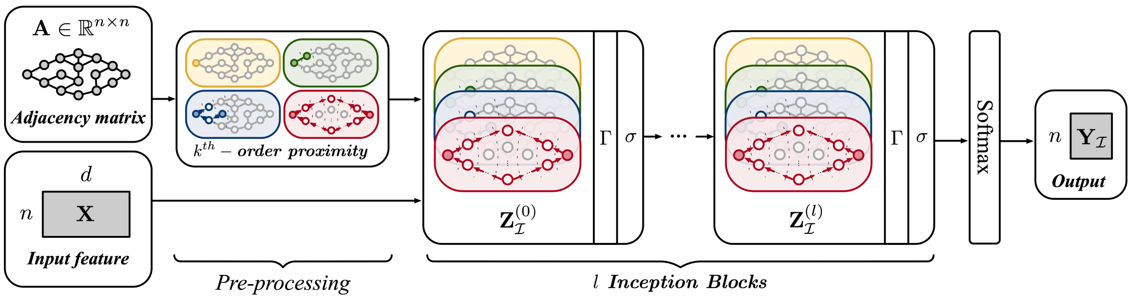

$\mathbf{A} \in \mathbb{R}^{n\times n}$
Adjacency matrix
d
n
$\mathbf{X}$
Input feature
k^{th} – order proximity
Pre-processing
$\mathbf{Z}_{\mathcal{I}}^{(0)}$
Γ
σ
$\mathbf{Z}_{\mathcal{I}}^{(l)}$
l Inception Blocks
Softmax
n
$\mathbf{Y}_{\mathcal{I}}$
Output

Figure 4.6: The schematic depiction of DiGCN for semi-supervised learning.

examples to evaluate the cross-entropy error for semi-supervised node classification task. The val accuracy on Cora-ML and Am-Photo with the number of layers and hidden dimension are shown in the Figure 4.7(a,b) respectively. Detailed hyper-parameter settings of out models are shown in Table 4.4.

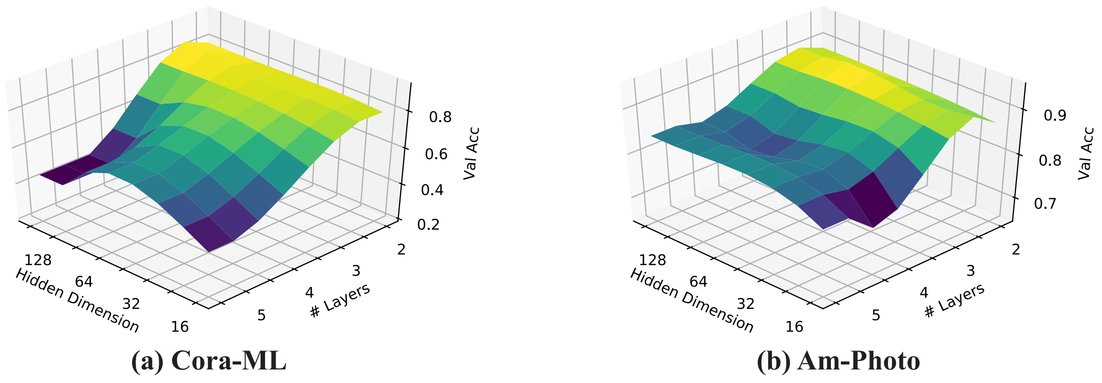


Figure 4.7: Val accuracy on Cora-ML and Am-Photo.

Table 4.4: The hyper-parameters of DiGCN model. "w/ pr" means directed graph convolution using $\mathbf{L}_{pr}$; "w/ appr" means directed graph convolution using $\mathbf{L}_{appr}$; "w/o IB" means using 1^{st}-order proximity directed graph convolution only; "w/ IB" means using Inception block.

Our models	layers	lr	weight decay	dropout	hidden dim	Others
w/pr w/o IB	2	0.05	1e-4	0.5	64	$\alpha = 0.1$
w/appr w/o IB	2	0.05	1e-4	0.5	64	$\alpha = 0.1$
w/appr w/ IB	3	0.01	5e-4	0.6	32	$\alpha = 0.1$, $k = 2$

4.4.2.2 Experimental Results

Overall accuracy. The performance comparisons between our model and baselines on four datasets are reported in Table 4.5. We train all models for a maximum of 1000 epochs and early stop if the validation accuracy does not increase for 200 consecutive epochs, then calculate mean test accuracy with STD in percent (%) averaged over 20 random dataset splits with random weight initialization.

It can be seen easily that our methods achieves the state-of-the-art results on all datasets. Notice that spectral-based models including ChebNet, GCN, SGC and InfoMax, do not perform well on directed graph datasets compared to their good performance in undirected graphs. This is mainly because these models have limited ability to obtain features from the surroundings using asymmetric adjacency matrices. However, APPNP is an exception. It allows features to randomly propagate with a certain teleport probability, which breaks through the path limitation and achieves good results in directed graphs. The spatial-based method and ours have similar results, which shows that both methods have good suitability for directed graphs. Moreover, DGCN performs well on the most datasets, however, it uses both in- & out-degree proximity matrix to obtain structural features in directed graphs, which leads to out of memory on Am-Computer. Meanwhile, SIGN uses SGC as the basic module, thus, even if Inception method is used, SIGN does not perform well in directed graphs (see analysis in Section 4.3.2).

Training time. With the same training settings, we measure the convergence speed of models by average training time per run in second (s) in Table 4.5. Apparently, our models have similar results. $\mathbf{L}_{pr}$ can only be applied to moderately sized graphs, while $\mathbf{L}_{appr}$ scales to large graphs. Compared with the spectral-based methods, the overall speed of our model without Inception block is similar to SGC since the Laplacian is precomputed. On Am-Photo and Am-Computer with large scales, our model is 30% faster on average than GCN. The accuracy of our model improves significantly while the speed decreases after using Inception, which is consistent with complexity analysis in Section 4.3.2.

Ablation study. We validate the effectiveness of the components and the resulting ACC are shown in Table 4.5. Comparing model with $\mathbf{L}_{appr}$ and model with $\mathbf{L}_{pr}$, we find that the approximate method can not only achieve the similar accuracy but also save training time and memory. Meanwhile, we find that the combination of $\mathbf{L}_{appr}$ and Inception block brings significant improvement in accuracy. This substantially validate that scalable receptive fields do help to learn features from neighborhood.

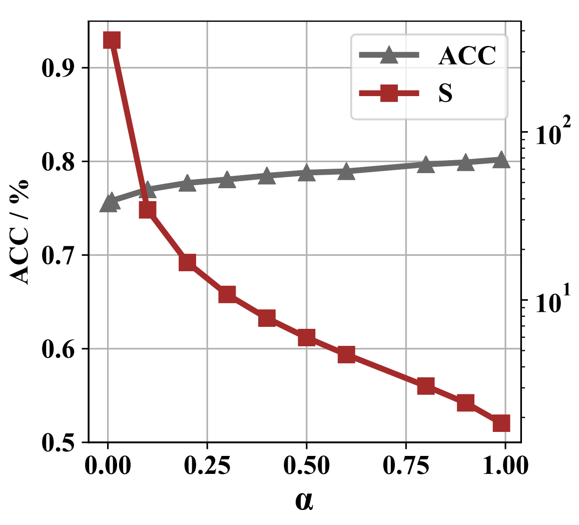
Table 4.5: Overall accuracy and training time of node classification task."w/ pr" means using $\mathbf{L}_{pr}$; "w/ appr" means using $\mathbf{L}_{appr}$; "w/o IB" means using directed graph convolution only; "w/ IB" means using Inception block. The best results are highlighted with **bold** and the second is highlighted with underline.

Model	CORA-ML		CITESEER		AM-PHOTO		AM-COMPUTER	
	acc	time	acc	time	acc	time	acc	time
ChebNet [28]	64.02 ± 1.5	7.23	56.46 ± 1.4	7.45	80.91 ± 1.0	10.52	73.25 ± 0.8	16.96
GCN [88]	53.11 ± 0.8	4.48	54.36 ± 0.5	4.80	53.20 ± 0.4	4.86	60.50 ± 1.6	5.04
SGC [187]	51.14 ± 0.6	1.92	44.07 ± 3.5	3.58	71.25 ± 1.3	2.31	76.17 ± 0.1	3.68
APPNP [90]	70.07 ± 1.1	6.84	65.39 ± 0.9	6.94	79.37 ± 0.9	6.72	63.16 ± 1.4	6.47
InfoMax [166]	58.00 ± 2.4	4.11	60.51 ± 1.7	4.85	74.40 ± 1.2	31.80	47.32 ± 0.7	41.96
GraphSage [54]	72.06 ± 0.9	6.22	63.19 ± 0.7	6.21	87.57 ± 0.9	8.52	79.29 ± 1.3	14.49
GAT [165]	71.91 ± 0.9	6.02	63.03 ± 0.6	6.12	$\underline{89.10 \pm 0.7}$	8.83	79.45 ± 1.5	14.66
DGCN [161]	75.02 ± 0.5	6.53	$\underline{66.00 \pm 0.4}$	6.84	83.66 ± 0.8	36.29	OOM	-
SIGN [140]	66.47 ± 0.9	2.81	60.69 ± 0.4	2.96	74.13 ± 1.0	5.33	69.40 ± 4.8	4.97
Ours								
w/ pr w/o IB	$\underline{77.11 \pm 0.5}$	39.13	64.77 ± 0.6	47.19	OOM*	-	OOM	-
w/ appr w/o IB	77.01 ± 0.4	2.71	64.92 ± 0.3	2.69	88.72 ± 0.3	2.95	$\underline{85.55 \pm 0.4}$	4.23
w/ appr w/ IB	**80.28 ± 0.5**	6.38	**66.11 ± 0.7**	6.42	**90.02 ± 0.5**	11.77	**85.94 ± 0.5**	26.63

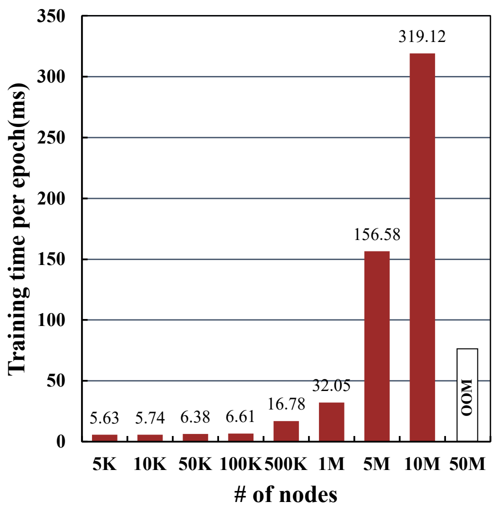
* OOM stands for out of memory (see efficiency analysis in Section 4.2.1)

Effect of teleport probability α. Figure 4.8(a) shows the effect of the hyperparameter α on the test accuracy and structural retention. Referring to the assumption in Section 4.2.2, we define structural retention as $\mathcal{S} = \mathrm{KL}(\pi_{appr}, \pi_{appr}\tilde{\mathbf{P}})^{-1}$, where KL means KL divergence. We use $\mathcal{S}$ to measure how much directed structural information retained after approximation. The smaller $\mathcal{S}$, the less information remain. According to Theorem 2 that α needs to be close to 0 to retain directed graph structural information in Laplacian, however, we find that a higher α improves accuracy slightly. In view of this, we choose $\alpha \in [0.05, 0.2]$ to balance structural retention and accuracy. α should be adjusted for the different datasets [90], but in order to maintain consistency, we take $\alpha = 0.1$ in all experiments.

Training time at different graph scales. We report results for the mean training time in millisecond (ms) per epoch for 200 epochs on simulated random graphs using directed graph convolution. We construct a simple random graph with N nodes and assign $2N$ edges uniformly at random. We take the node index matrix as input feature matrix and give the same label for every node. Figure 4.8(b) summarizes the results and shows that our model can handle about 10 million nodes in one GPU (11GB).



(a)



(b)

Figure 4.8: (a) effect of α to ACC and structural retention $\mathcal{S}$; (b) training time per epoch on random directed graphs with different size.

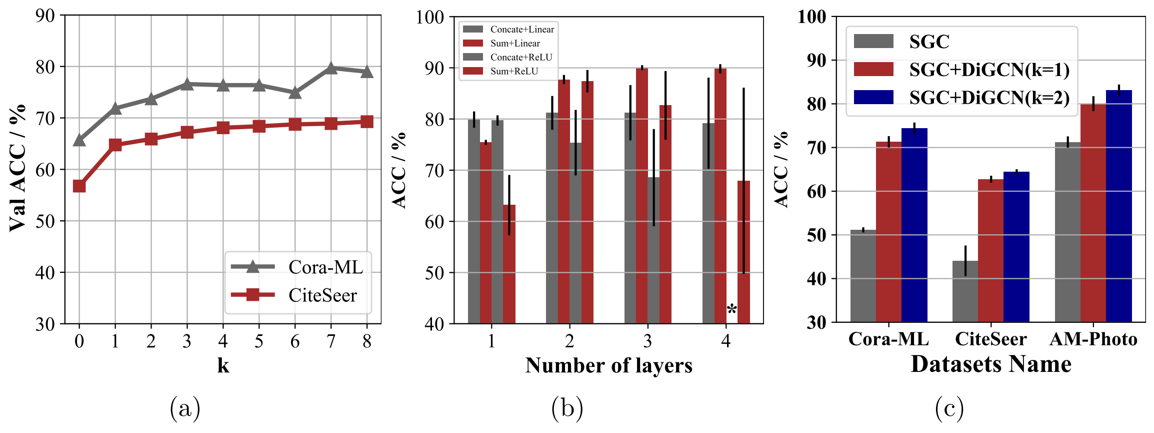


Figure 4.9: (a) effects of model width k ; (b) fusion and activation functions on Am-Photo in different layers (* stands for unable to converge); (c) generalization to SGC using different k.

Depth and width of Inception block. How to balance the model depth and width becomes a vital issue for Inception block. Figure 4.9(a) shows that for a single layer model, the improvement in val accuracy is not significant for $k > 2$, which means larger receptive field cannot obtain more effective information on small-scale dataset CiteSeer. Then, we keep $k = 2$ and carry out grid search on model depth and choose layer=3. To intuitively compare the impact of depth under the same receptive range, we choose GAT as baseline, which can obtain various range by adjusting the head size without stacking layers. From Table 4.6, we can observe that larger receptive fields help our model to perform better. It is consistent that using a moderate number of layers is enough to effectively learn features.

Table 4.6: Results at various depths. Our model sets $k = 2$, and uses $\mathbf{L}_{appr}$ as Laplacian and Sum as the fusion operation. "Range" means range of the receptive field. The best results are highlighted with **bold**.

Model	Range	Setting	Cora-ML	CiteSeer	Am-Photo	Am-Computer
GAT	2	head=2	71.33 ± 1.4	**63.33 ± 0.7**	**81.12 ± 1.5**	**75.12 ± 3.2**
Ours		layer=1	**72.14 ± 1.0**	62.89 ± 0.4	75.43 ± 0.5	64.17 ± 0.5
GAT	4	head=4	71.65 ± 1.0	63.30 ± 0.7	86.03 ± 1.0	77.57 ± 1.5
Ours		layer=2	**76.62 ± 0.5**	**63.98 ± 0.5**	**87.71 ± 0.9**	**82.36 ± 0.7**
GAT	8	head=8	71.62 ± 0.8	63.17 ± 0.6	87.04 ± 1.0	78.22 ± 1.7
Ours		layer=3	**80.28 ± 0.5**	**66.11 ± 0.7**	**90.02 ± 0.5**	**85.94 ± 0.5**
GAT	16	head=16	71.91 ± 0.9	63.03 ± 0.6	89.10 ± 0.7	79.45 ± 1.5
Ours		layer=4	**79.95 ± 0.8**	**64.00 ± 1.0**	**89.81 ± 0.9**	**83.36 ± 0.7**

Fusion operation and activation function. We show results in Figure 4.9(b) and use Sum and Concate to represent Summation and Concatenation respectively. We find that the choices of fusion and activation functions need to match the complexity of the model. When the model is shallow, Concate performs better due to more parameters. Sum achieves better results in deep model because it requires fewer parameters, which helps prevent the model from overfitting. Since we use k^{th}-order proximate IB, linear combinations can achieve stable results on smaller datasets. Using a non-linear activation function (ReLU) may result in models that are too complex to learn features effectively.

Generalization to other model (SGC). To test the generalization ability of Inception block, we use it to replace the origin layer in SGC [187]. The generalized SGC model is denoted by **SGC+DiGCN**. Moreover, we test the case of $k = 1$ and $k = 2$ to the generalized model and the results are shown in Figure 4.9(c). Obviously, whether $k = 1$ or $k = 2$ our generalized model outperforms the original model on all datasets, which shows that the multi-scale receptive field helps the model obtain more surrounding information. Meanwhile, our method has good generalization ability because of its simple structure that can be plugged into models easily.

4.4.3 Link Prediction

Link prediction is another common task used to measure the capability of graph models. It is a task that predicting the existence of a edge between two nodes in a graph. In this experiment, the dataset we use is directed graph datasets, *i.e.,* the edges to be predicted are directed.

4.4.3.1 Implementation

We implement the link prediction task using Pytorch-Geometric [4]. The ratio of positive validation edges is 0.05 and the ratio of positive test edges is 0.1. We do not use early stop and obtain the mean and std that are calculated for 20 random dataset splits and a maximum number of epochs of 500. The hyper-parameters of two models are shown in Table 4.7.

[4]https://github.com/rusty1s/pytorch_geometric

Table 4.7: The hyper-parameters of link prediction models.

Models	layers	lr	hidden dim	output dim	activation function	Others
GCN	2	0.01	128	64	ReLU	-
DiGCN	2	0.01	128	64	ReLU	$\alpha = 0.1$

4.4.3.2 Experimental Results

We use link prediction task to show that our model is able to obtain more structural information. In this task, we split the edges of a directed graph into positive and negative train/val/test edges and compare the results with GCN over 20 runs for a maximum of 500 epochs. Figure 4.10 shows that that our model (w/ appr and w/o IB) outperforms GCN for link prediction on all datasets. This is mainly because we take the edge direction into account when calculating $\mathbf{L}_{appr}$, which allows us to obtain more accurate structural information in directed graphs.

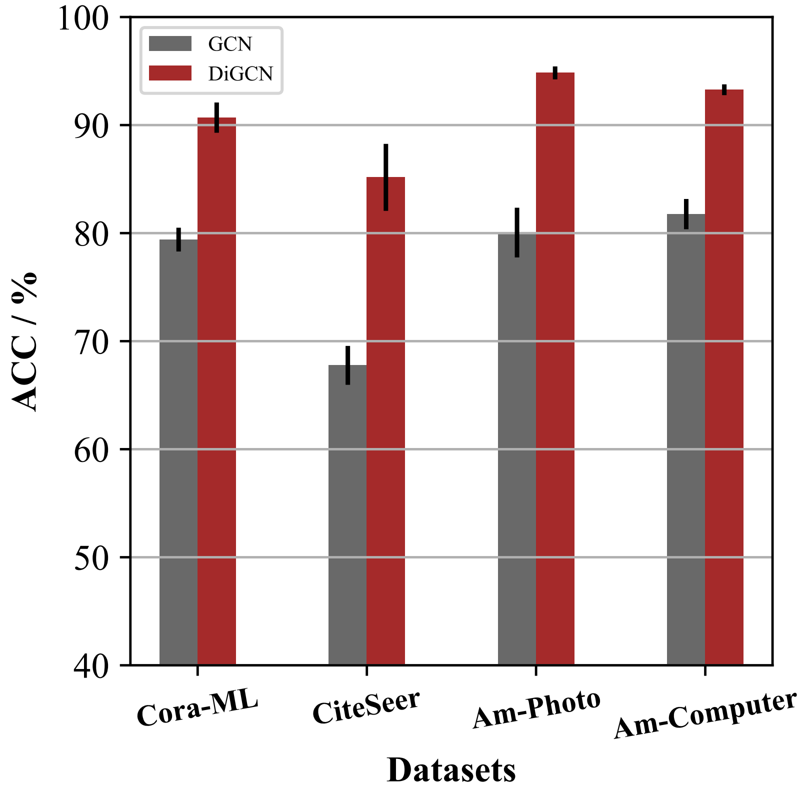


Figure 4.10: Link prediction results on different datasets

4.5 Summary

In this chapter, we present a novel Directed Graph Inception Convolutional Networks (**DiGCN**), which can effectively learn directed graph representation. We theoretically extend spectral-based graph convolution to directed graph and further simplify it. Besides, we define k^{th}*-order proximity* and design the directed graph Inception networks to learn multi-scale features. This simple and scalable model can not only learn directed graph structure, but also get hidden information through k^{th}-order proximity relationship. Finally, we use several tasks on various real-world datasets to validate the effectiveness and generalization capability of our model. The results show that our model outperforms several state-of-the-art methods.

Chapter 5

Contrastive Graph Learning

In Chapter 3, we introduce DGCN, the first GCNs for learning from directed graphs. In Chapter 4, we build DiGCN, which combines spectral-based graph convolution with directed graph inception networks to learn multi-scale features and hidden information through k^{th}-order proximity relationships. DGCN and DiGCN are trained end-to-end under supervision. This training scheme shows excellent performance by virtue of enough labeled data. However, a large amount of graph data exists in an untagged form. To utilize this vast amount of unused data, we propose DiGCL, a self-supervised learning framework for directed graphs. Based on the theoretical definition of *directed graph Laplacian matrix* proposed by DiGCN, DiGCL introduces a data augmentation method that utilizes Laplacian perturbation and curriculum learning to learn from contrastive views. The results show that DiGCL can retain more structural features and provide adequate contrastive information, outperforming other existing graph contrastive learning models.

5.1 Introduction

To make use of rich unlabeled data, several Graph Contrastive Learning (GCL) [153, 207, 229, 136, 57, 227] works are proposed based on GNNs and Contrastive Learning (CL) [19, 162, 58]. They utilize data augmentation methods to generate contrastive views from the original graphs, then force views generated from the same instance (node or graph) closer while views from different instances apart using *InfoNCE-like* [123] objective function. However, current GCL methods encounter some issues in processing directed graphs, mainly in *data augmentation method* and *contrastive learning framework*.

Firstly, most *data augmentation* methods used in GCL [207, 229, 228] do not take the directed graph structure into account and may discard distinctive direction information. For example, the idea of dropping nodes/edges [228, 207, 180] is borrowed from random erasing used in images [220] , which overlooks the discrepancy of nodes and edges in different graph structures. Moreover, by grasping contrastive information through changing graph structure, part of the distinctive direction information, *e.g.,* irreversible time-series relationships, will inevitably be lost. The message passing scheme will also be mislead, making it difficult for GNN-based encoders to learn from directed graphs. There is a lack of data augmentation methods that are specifically designed for directed graphs to retain the original directed graph structure while providing enough contrastive information.

Besides, common *contrastive learning frameworks* are not optimized for directed graphs, they can only learn from a limited number of contrastive views [207, 229, 57]. However, due to the complex structure of directed graphs, it is insufficient to use a small number of contrastive views to fully understand their structural characteristics [160]. Furthermore, since the contrastive views have to be determined before training, it becomes another problem to select the ideal views for the downstream task during pre-processing, about which we actually have no knowledge in advance [158, 195]. Hand-picking contrastive views also cause a decrease in generalization. Several works [157, 203] try to obtain more information by increasing the number of contrastive views. However, they still have to pre-define the views and trade off between the number of views and the training difficulty.

To address these two issues, we first propose a directed graph data augmentation method called **Laplacian perturbation**. Since the contrastive views are passed to the encoder in the form of Laplacian matrix, a desirable data augmentation method is to perturb the Laplacian matrix without changing the directed graph structure. To achieve this, we adopt the approximate directed graph Laplacian [160] where a teleport probability is introduced to control the degree of approximation. Thus, by adding a perturbation term to this probability, we can simply augment the Laplacian matrix without altering the directed graph structure. As it is time-consuming to calculate the Laplacian matrix, we then speed it up using the power method [117]. In addition, from the theoretical analysis, we find that the Laplacian perturbation is essentially a perturbation to the directed graph entropy. The perturbation term essentially determines the magnitude of the entropy perturbation error, which affects the magnitude of the contrastive information.

To learn the complete structural characteristics of directed graphs, we design a **directed graph contrastive learning framework** to learn from contrastive views generated by Laplacian perturbation. First, we introduce a generalized dynamic-view contrastive objective which maximizes the sum of the mutual information between the representations of all contrastive views. This objective allows the contrastive views to change dynamically during training and motivates the encoder to learn a generalized representation from all views provided by data augmentation. However, this dynamic-view objective function is hard to optimize. Thus, we leverage the multi-task curriculum learning strategy [134, 141, 51, 120] to divide multiple contrastive views into sub-tasks with various difficulties and progressively learn from easy-to-difficult sub-tasks. In this way, we can learn all views step-by-step to reach the final objective. Moreover, learning from all contrastive views eliminates the need to adjust the data augmentation parameters anymore.

We empirically show that our **Di**rected **G**raph **C**ontrastive **L**earning (**DiGCL**) outperforms the competitive baselines in the settings of unsupervised and supervised learning. Systematic analysis is also carried out to analyze the performance of various augmentations on the mainstream benchmarks and the impact of different pacing functions on the performance of directed graph contrastive learning.

5.2 Directed Graph Data Augmentation

In this section, we first design a directed graph data augmentation scheme named Laplacian perturbation. As it is time-consuming to calculate the Laplacian matrix, we then speed it up using the power method [117]. Finally, we analyze the proposed data augmentation scheme theoretically. The data augmentation illustration is in Figure 5.1 and the pseudocode is given in Algorithm 5.

5.2.1 Directed Graph Laplacian and its Approximation

Formally, let a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its adjacency matrix can be denoted as $\mathbf{A} = \{0,1\}^{n\times n}$, where $n = |\mathcal{V}|$. The nodes are described by the feature matrix $\mathbf{X} \in \mathbb{R}^{n\times c}$, with the number of features c per node. Intuitively, the data augmentation can be performed at the topological- and feature-level, operating on $\mathbf{A}$ and $\mathbf{X}$, respectively [207, 229]. The distinctive attribute of directed graphs is the directionality of the edges, which leads us to focus on *topological node-level* data augmentation.

The augmented directed graph is fed to the GNN-based encoder in the form of Laplacian matrix to learn the representation. Since $\mathcal{G}$ may contain isolated nodes or could be formed into bipartite graph, it is not appropriate to trivially use directed graph Laplacian [24]. To relax this constraint, we use its approximate form: the approximate directed graph Laplacian matrix [160], which is defined as

$$\mathbf{L}_{\text{appr}}(\mathcal{G}, \alpha) = \mathbf{I} - \frac{1}{2}\left(\mathbf{\Pi}_{\text{appr}}^{\frac{1}{2}}\tilde{\mathbf{P}}\mathbf{\Pi}_{\text{appr}}^{-\frac{1}{2}} + \mathbf{\Pi}_{\text{appr}}^{-\frac{1}{2}}\tilde{\mathbf{P}}^T\mathbf{\Pi}_{\text{appr}}^{\frac{1}{2}}\right), \tag{5.1}$$

where $\mathbf{\Pi}_{\text{appr}} = \frac{1}{||\pi_{\text{appr}}||_1}\operatorname{Diag}(\pi_{\text{appr}})$. The approximate eigenvector π_{appr} is defined as

$$(1-\alpha)\pi_{\text{appr}}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n} = \pi_{\text{appr}}, \tag{5.2}$$

where $\tilde{\mathbf{P}} = \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$, $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}^{n\times n}$ denotes the transition matrix with added self-loops and the diagonal degree matrix $\tilde{\mathbf{D}}(u,u) = \sum_{v\in\mathcal{V}}\tilde{\mathbf{A}}(u,v)$. Tong *et al.* [160] add self-loops to the original directed graph to ensure $\mathcal{G}$ to be *aperiodic* and redefine the transition matrix based on *personalized* PageRank with the teleport probability $\alpha \in (0,1)$ to guarantee the redefined matrix to be *irreducible*. Note that we follow the [160] and set $\alpha = 0.1$ in this chapter.

According to Equation 5.1, existing data augmentation methods [207, 229], *e.g., node/edge dropping, subgraph sampling*, need to obtain the contrastive information by changing $\tilde{\mathbf{P}}$. Their use of sampling-based methods inevitably corrupts the directed graph structure and thus misleads the message passing in the GNN-based encoder. A desirable data augmentation method on $\mathbf{L}_{\text{appr}}$ is to perturb π_{appr} reasonably without altering the directed graph structure $\tilde{\mathbf{P}}$[1].

5.2.2 Directed Graph Data Augmentation with Laplacian Perturbation

From Equation 5.2, it is easy to find that the eigenvector π_{appr} depends on the $\tilde{\mathbf{P}}$ and α. In other words, we can shift the teleport probability α without changing the

[1]Note that adding self-loops to transform $\mathbf{P}$ into $\tilde{\mathbf{P}}$ is a common trick [88, 187], and we ignore the effect of this operation on the directed graph structure.

directed graph structure $\tilde{\mathbf{P}}$, thus altering the eigenvector π_{appr} and finally perturbing the Laplacian matrix $\mathbf{L}_{\text{appr}}$.

Definition 18. ***Laplacian perturbation***. *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the teleport probability α, we define the Laplacian perturbation opertation $\Phi(\cdot)$ on the $\mathbf{L}_{\text{appr}}(\tilde{\mathbf{P}}, \alpha)$ as*

$$\Phi_{\Delta\alpha}(\mathcal{G}, \alpha) = \mathbf{L}_{\text{appr}}(\mathcal{G}, \alpha + \Delta\alpha), \tag{5.3}$$

where $\Delta\alpha$ is a perturbation term that satisfies $\Delta\alpha \geqslant 0$ and $\alpha + \Delta\alpha \in (0, 1)$.

Through this operation, the directed graph structure and the sparsity of the Laplacian matrix are maintained, which means there is no training burden to the subsequent GNN-based encoder. However, using this operation for data augmentation is very time-consuming, since the time complexity of computing Equation 5.2 is $\mathcal{O}(n^2)$. To deal with it, we design an accelerating algorithm for computing this approximate eigenvector based on the power method [117].

We take advantage of the sparsity of the transition matrix $\tilde{\mathbf{P}}$ to compute the approximate eigenvector π_{appr} and rewrite Equation 5.2 to the form of discrete-time Markiv chain at time t as

$$\pi_{\text{appr}}^{t+1} = (1 - \alpha)\pi_{\text{appr}}^{t}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n}. \tag{5.4}$$

As stated in Tong *et al.* [160], π_{appr}^{t} has a special property that $\pi_{\text{appr}}^{t}\mathbf{1}^{n\times 1} = \frac{1}{1+\alpha}$. We then multiply the last term of Equation 5.4 with $(1+\alpha)\pi_{\text{appr}}^{t}\mathbf{1}^{n\times 1}$ to

$$\pi_{\text{appr}}^{t+1} = (1-\alpha)\pi_{\text{appr}}^{t}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}[(1+\alpha)\pi_{\text{appr}}^{t}\mathbf{1}^{n\times 1}]\mathbf{1}^{1\times n} = (1-\alpha)\pi_{\text{appr}}^{t}\tilde{\mathbf{P}} + \frac{\alpha}{n}\pi_{\text{appr}}^{t}\mathbf{1}^{n\times n}. \tag{5.5}$$

We can solve Equation 5.5 iteratively using the power method [117]. Therefore, the complexity decreases to $\mathcal{O}(nk)$, where k is the number of iteration times. Due to page limit, we do not discuss the number of iterations and convergence rate here. Please refer to [105] for more details. We take $k = 100$ and the tolerance as 1e−6 throughout. It can be seen easily that for large-scale graphs, our method can significantly improve the speed and makes it possible to do Laplacian perturbation during the training, which is the basis for the framework proposed in Section 5.3. In addition, we compare the running time of our fast Laplacian perturbation with different data augmentation methods in Section 4.4.2.2.

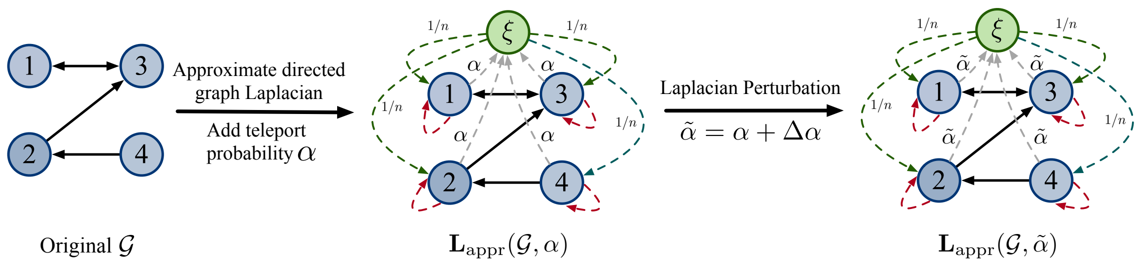

Original $\mathcal{G}$
Approximate directed graph Laplacian
Add teleport probability α
$\mathbf{L}_{\text{appr}}(\mathcal{G},\alpha)$
Laplacian Perturbation
$\tilde{\alpha} = \alpha + \Delta\alpha$
$\mathbf{L}_{\text{appr}}(\mathcal{G},\tilde{\alpha})$
ξ
$1/n$
α
$\tilde{\alpha}$

Figure 5.1: Illustration of Laplacian perturbation. ξ is the auxiliary node defined in [160] and the red dotted lines represent adding the self-loops.

```
Algorithm 5: Laplacian Perturbation $\Phi(\cdot)$ Procedure
    Input: Directed graph adjacency matrix: $\mathbf{A}$, teleport probability $\alpha$,
           perturbation term $\Delta\alpha$
    Output: Perturbed Laplacian $\hat{\mathbf{L}}_{\text{appr}}$
1: $\tilde{\mathbf{A}} \leftarrow \mathbf{A} + \mathbf{I}^{n\times n}$ ;
2: $\tilde{\mathbf{P}} \leftarrow \tilde{\mathbf{D}}^{-1}\tilde{\mathbf{A}}$ ;
3: $\hat{\alpha} = \alpha + \Delta\alpha$ ;
4: $\hat{\pi}_{\text{appr}} \leftarrow (1-\hat{\alpha})\hat{\pi}_{\text{appr}}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\hat{\alpha}}{1+\hat{\alpha}}\mathbf{1}^{1\times n}$;
5: $\hat{\mathbf{\Pi}}_{\text{appr}} \leftarrow \frac{1}{||\hat{\pi}_{\text{appr}}||_1}\text{Diag}(\hat{\pi}_{\text{appr}})$;
6: $\hat{\mathbf{L}}_{\text{appr}} \leftarrow \mathbf{I} - \frac{1}{2}\left(\hat{\mathbf{\Pi}}^{\frac{1}{2}}_{\text{appr}}\tilde{\mathbf{P}}\hat{\mathbf{\Pi}}^{-\frac{1}{2}}_{\text{appr}} + \hat{\mathbf{\Pi}}^{-\frac{1}{2}}_{\text{appr}}\tilde{\mathbf{P}}^{T}\hat{\mathbf{\Pi}}^{\frac{1}{2}}_{\text{appr}}\right)$;
7: return $\hat{\mathbf{L}}_{\text{appr}}$
```

5.2.3 Justification of Laplacian Perturbation

Noticing the inherent connections between graph Laplacian and von Neumann entropy [205], we can use the proprieties of von Neumann entropy to analyze the Laplacian perturbation operation quantitatively. We start with defining the von Neumann entropy of directed graphs.

Definition 19. ***Directed graph von Neumann entropy.*** *Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, its von Neumann entropy [205] based on the $\mathbf{L}_{\text{appr}}$ is defined as*

$$\tilde{\mathcal{H}}_{\text{VN}}(\mathcal{G},\alpha) = 1 - \frac{1}{n} - \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{out}\,d_v^{out}} + \frac{\pi_{\text{appr}}(u)}{\pi_{\text{appr}}(v)d_u^{out2}}\right) - \sum_{(u,v)\in\tilde{\mathcal{E}}}\frac{1}{d_u^{out}\,d_v^{out}}\right\}, \tag{5.6}$$

where $\tilde{\mathcal{E}} = \{(u,v) \mid (u,v) \in \mathcal{E}$ and $(v,u) \notin \mathcal{E}\}$. $d_u^{\text{in}} = \sum_{v\in\mathcal{V}}\mathbf{A}(v,u)$ and $d_u^{out} = \sum_{v\in\mathcal{V}}\mathbf{A}(u,v)$ are the in- and out-degree of the node u.

Since the Laplacian perturbation does not involve the nodes/edges and only changes the teleport probability connecting to the auxiliary node ξ as shown in Figure 5.1, the number of nodes n and the degrees of nodes d_u (edge distribution) will not change. On the contrary, the approximate eigenvector π_{appr} has been changed. According to Equation 5.6, the essence of this operation is a perturbation to the directed graph entropy. We will further introduce the perturbation error to quantify this impact.

Definition 20. ***Perturbation error***. *Given the perturbation term $\Delta\alpha$, we define the perturbation error of directed graph von Neumann entropy caused by Laplacian perturbation as $\Delta\tilde{\mathcal{H}}_{\text{VN}}(\alpha, \alpha+\Delta\alpha) = \tilde{\mathcal{H}}_{\text{VN}}(\mathcal{G}, \alpha) - \tilde{\mathcal{H}}_{\text{VN}}(\mathcal{G}, \alpha+\Delta\alpha)$.*

From the above definition, it is easy to find out that the perturbation error is a function of α and $\Delta\alpha$. α is typically fixed, e.g., set to 0.1. Then with α fixed, we would like to figure out the monotonicity of the perturbation error with respect to $\Delta\alpha$, which helps us to explore the effect of the error on the von Neumann entropy. We prove the monotonicity of the perturbation error in Theorem 4.

Theorem 4. ***Monotonicity of the perturbation error***. *The perturbation error $\Delta\tilde{\mathcal{H}}_{\text{VN}}$ increases monotonically with the Laplacian perturbation term $\Delta\alpha$.*

Proof. The perturbation error is defined in Definition 20 as

$$\Delta\tilde{\mathcal{H}}_{\text{VN}}(\alpha, \alpha+\Delta\alpha) = \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{\pi_{\text{appr}}^{\alpha+\Delta\alpha}(u)}{\pi_{\text{appr}}^{\alpha+\Delta\alpha}(v)d_u^{\text{out}^2}} - \frac{\pi_{\text{appr}}^{\alpha}(u)}{\pi_{\text{appr}}^{\alpha}(v)d_u^{\text{out}^2}}\right)\right\}. \tag{5.7}$$

We start out the proof from Equation 5.2, leading to

$$(1-\alpha)\pi_{\text{appr}}^{\alpha}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha}{1+\alpha}\mathbf{1}^{1\times n} = \pi_{\text{appr}}^{\alpha}, \tag{5.8}$$

and the approximate eigenvector component for node u is

$$\pi_{\text{appr}}^{\alpha}(u) = (1-\alpha)\sum_{i,(i,u)\in\mathcal{E}}\pi_{\text{appr}}^{\alpha}(i)\tilde{\mathbf{P}}(i,u) + \frac{1}{n}\frac{\alpha}{1+\alpha}. \tag{5.9}$$

In the [205], they assume that the eigenvector component is proportional to the in-degree of the corresponding node when the neighborhood of this node has similar out-degree and in-degree, *i.e.,*

$$\frac{\sum_{i,(i,u)\in\mathcal{E}} \pi^{\alpha}_{\text{appr}}(i)\tilde{\mathbf{P}}(i,u)}{\sum_{i,(i,v)\in\mathcal{E}} \pi^{\alpha}_{\text{appr}}(i)\tilde{\mathbf{P}}(i,v)} \approx \frac{d^{\text{in}}_u}{d^{\text{in}}_v} = \frac{cd^{\text{in}}_u}{cd^{\text{in}}_v}, \tag{5.10}$$

where the constant c controls the d^{in}_u and $\pi^{\alpha}_{\text{appr}}(i)\tilde{\mathbf{P}}(i,u)$ at the same scale. Meanwhile, they experimentally verify that even under this strong assumption, the calculated von Neumann entropy does not show significant errors [205]. Thus, we adopt their assumption and simply Equation 5.10 to

$$\sum_{i,(i,u)\in\mathcal{E}} \pi^{\alpha}_{\text{appr}}(i)\tilde{\mathbf{P}}(i,u) \approx cd^{\text{in}}_u. \tag{5.11}$$

We further let $0 < \alpha_1 < \alpha_2 < 1$ and take them into Equation 5.7. We can derive $\Delta\tilde{\mathcal{H}}_{\text{VN}}(\alpha_1,\alpha_2)$ as

$$\begin{aligned}
&= \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}} \frac{1}{d^{\text{out}^2}_u}\left(\frac{(1-\alpha_2)cd^{\text{in}}_u + \frac{1}{n}\frac{\alpha_2}{1+\alpha_2}}{(1-\alpha_2)cd^{\text{in}}_v + \frac{1}{n}\frac{\alpha_2}{1+\alpha_2}} - \frac{(1-\alpha_1)cd^{\text{in}}_u + \frac{1}{n}\frac{\alpha_1}{1+\alpha_1}}{(1-\alpha_1)cd^{\text{in}}_v + \frac{1}{n}\frac{\alpha_1}{1+\alpha_1}}\right)\right\}\\
&= \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}} \frac{1}{d^{\text{out}^2}_u}\left(\frac{n(1-{\alpha_2}^2)cd^{\text{in}}_u + \alpha_2}{n(1-{\alpha_2}^2)cd^{\text{in}}_v + \alpha_2} - \frac{n(1-{\alpha_1}^2)cd^{\text{in}}_u + \alpha_1}{n(1-{\alpha_1}^2)cd^{\text{in}}_v + \alpha_1}\right)\right\}\\
&= \frac{1}{2n^2}\left\{\sum_{(u,v)\in E} \frac{1}{d^{\text{out}^2}_u}\left(\frac{(n(1-{\alpha_2}^2)\alpha_1 c(d^{\text{in}}_u - d^{\text{in}}_v) - n(1-{\alpha_1}^2)\alpha_2 c(d^{\text{in}}_u - d^{\text{in}}_v)}{(n(1-{\alpha_2}^2)cd^{\text{in}}_v + \alpha_2)(n(1-{\alpha_1}^2)cd^{\text{in}}_v + \alpha_1)}\right)\right\}\\
&= \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}} \frac{d^{\text{in}}_u - d^{\text{in}}_v}{d^{\text{out}^2}_u}\left(\frac{nc(1-{\alpha_2}^2)\alpha_1 - nc(1-{\alpha_1}^2)\alpha_2}{(n(1-{\alpha_2}^2)cd^{\text{in}}_v + \alpha_2)(n(1-{\alpha_1}^2)cd^{\text{in}}_v + \alpha_1)}\right)\right\}\\
&= \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}} \frac{d^{\text{in}}_u - d^{\text{in}}_v}{d^{\text{out}^2}_u}\left(\frac{nc(1/\alpha_2 - \alpha_2) - nc(1/\alpha_1 - \alpha_1)}{(n(1-{\alpha_2}^2)cd^{\text{in}}_v + \alpha_2)(n(1-{\alpha_1}^2)cd^{\text{in}}_v + \alpha_1)/(\alpha_1\alpha_2)}\right)\right\}.
\end{aligned} \tag{5.12}$$

Since $d^{\text{in}}_v \leqslant n$ and $d^{\text{out}}_u \leqslant n$,

$$\begin{aligned}
\Delta\tilde{\mathcal{H}}_{\text{VN}}(\alpha_1,\alpha_2) &\geqslant \frac{1}{2n^2}\sum_{(u,v)\in\mathcal{E}} (d^{\text{in}}_u - d^{\text{in}}_v)\underbrace{\left(\frac{\frac{c}{n}(1/\alpha_2 - \alpha_2 - 1/\alpha_1 + \alpha_1)}{(n^2(1-{\alpha_2}^2)c + \alpha_2)(n^2(1-{\alpha_1}^2)c + \alpha_1)/(\alpha_1\alpha_2)}\right)}_{constant\ \ C}\\
&\geqslant \frac{C}{2n^2}\sum_{(u,v)\in\mathcal{E}} (d^{\text{in}}_u - d^{\text{in}}_v).
\end{aligned} \tag{5.13}$$

As $0 < \alpha_1 < \alpha_2 < 1$, the constant $C < 0$. And the edges point from node u to node v, thus the term $\sum_{(u,v)\in\mathcal{E}}(d^{\text{in}}_u - d^{\text{in}}_v) < 0$. Therefore,

$$\Delta\tilde{\mathcal{H}}_{\text{VN}}(\alpha_1,\alpha_2) > 0. \tag{5.14}$$

Clearly, the perturbation error $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}}$ increases monotonically with the Laplacian perturbation term $\Delta\alpha$. The proof is concluded. □

Based on the monotonicity, we further give the upper and lower bounds of $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}}$.

Theorem 5. ***Bounds on the perturbation error****. Given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the teleport probability α, the inequality*

$$0 < \Delta\tilde{\mathcal{H}}_{\mathrm{VN}}(\alpha, \alpha+\Delta\alpha) < \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{out^2}} - \frac{\pi^{\alpha}_{\mathrm{appr}}(u)}{\pi^{\alpha}_{\mathrm{appr}}(v)d_u^{out^2}}\right)\right\} \tag{5.15}$$

holds. When the perturbation term $\Delta\alpha = 0$, $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}} = 0$ and when $\Delta\alpha \to 1-\alpha$, the perturbation error $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}}$ towards the upper bound.

Proof. From Theorem 4, $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}}$ increases monotonically with the Laplacian perturbation term $\Delta\alpha$. Thus, when $\Delta\alpha = 0$, $\Delta\tilde{\mathcal{H}}_{\mathrm{VN}} = 0$. For the upper bound of the perturbation error, we start with Equation 5.2 that

$$(1-\alpha-\Delta\alpha)\pi^{\alpha+\Delta\alpha}_{\mathrm{appr}}\tilde{\mathbf{P}} + \frac{1}{n}\frac{\alpha+\Delta\alpha}{1+\alpha+\Delta\alpha}\mathbf{1}^{1\times n} = \pi^{\alpha+\Delta\alpha}_{\mathrm{appr}}. \tag{5.16}$$

Since π_{appr} is the stationary distribution and $\tilde{\mathbf{P}}$ is transition matrix, $||\pi_{appr}\tilde{\mathbf{P}}||_\infty \leqslant ||\pi_{appr}||_\infty||\tilde{\mathbf{P}}||_\infty \leqslant 1$. It is easy to observe that when $\alpha+\Delta\alpha \to 1$, $\pi^{\alpha+\Delta\alpha}_{\mathrm{appr}} \to \frac{1}{2n}\mathbf{1}^{1\times n}$, which means $\pi^{\alpha+\Delta\alpha}_{\mathrm{appr}}(u), \pi^{\alpha+\Delta\alpha}_{\mathrm{appr}}(v)$ are equivalent as $\alpha+\Delta\alpha \to 1$. Thus,

$$\Delta\tilde{\mathcal{H}}_{\mathrm{VN}}(\alpha, \alpha+\Delta\alpha) \to \frac{1}{2n^2}\left\{\sum_{(u,v)\in\mathcal{E}}\left(\frac{1}{d_u^{\mathrm{out}^2}} - \frac{\pi^{\alpha}_{\mathrm{appr}}(u)}{\pi^{\alpha}_{\mathrm{appr}}(v)d_u^{\mathrm{out}^2}}\right)\right\}, \tag{5.17}$$

when $\alpha+\Delta\alpha \to 1$. The proof is concluded. □

We show in Theorem 4 that Laplacian perturbation can provide contrastive information in various magnitude for the encoder to learn the representations by changing $\Delta\alpha$. Meanwhile, since it does not need to alter the structure and the number of nodes n is generally high, the perturbation error will be in a very small range as shown in Theorem 5. This can help the encoder to focus more on the directed graph structure rather than just learning from the errors.

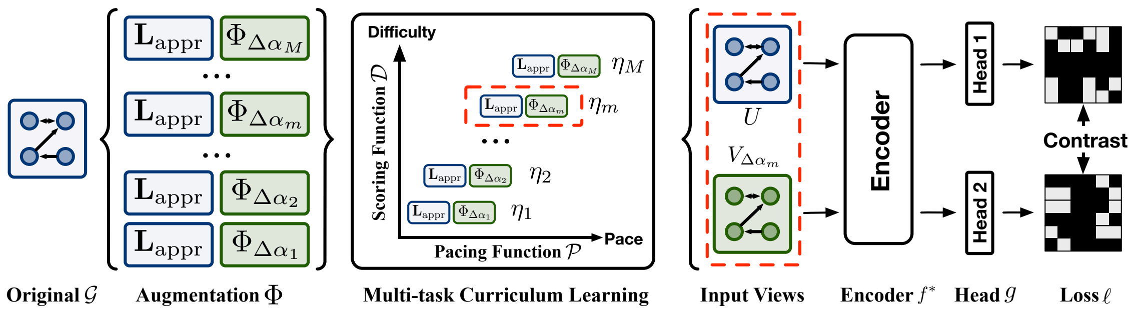


Figure 5.2: Illustration of our DiGCL model using Laplacian perturbation. For a directed graph, we first generate M different pairs of contrastive views by Laplacian perturbation. The different contrastive view pairs are then scored by a scoring function and mapped to different training paces by a pacing function. Finally, the arranged contrastive view pairs are input into a shared encoder to progressively learn the unsupervised graph representation with contrastive loss.

5.3 Directed Graph Contrastive Learning

In this section, we introduce a new directed graph contrastive learning framework (DiGCL) with a more generalized objective, and then we present multi-task curriculum learning scheme to help DiGCL progressively learn from multiple easy-to-difficult contrastive views. The model illustration is in Figure 5.2 and the pseudocode is given in Algorithm 6.

5.3.1 Learning with Dynamic-view Contrastive Objective

As we have defined the directed graph data augmentation in Equation 5.3, we follow the common GCL paradigm using our Laplacian perturbation.

Fixed-view Objective. GCL seeks to maximize the mutual information (MI) between the representations of augmented graphs under different views [207, 229]. Based on it, we design a framework for directed graph contrastive learning, which contains three steps. (1) First, given a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the teleport probability α, we generate two correlated views using for the $\mathcal{G}$: $U_{\Delta\alpha_1} = \Phi_{\Delta\alpha_1}(\mathcal{G}, \alpha)$ and $V_{\Delta\alpha_2} = \Phi_{\Delta\alpha_2}(\mathcal{G}, \alpha)$ as a contrastive pair. This generation is through executing Laplacian perturbation Φ on the input $\mathcal{G}$ with the parameters $\Delta\alpha_1, \Delta\alpha_2$ separately. (2) Second, we take a GNN-based encoder $f(\cdot)$ to extract representation $\mathbf{H}_{\Delta\alpha_1} = f(U_{\Delta\alpha_1})$, $\mathbf{H}_{\Delta\alpha_2} = f(V_{\Delta\alpha_2})$ of two contrastive views. (3) Finally, the encoder is trained with the contrastive objective as

$$\max_f I\left(\mathbf{H}_{\Delta\alpha_1}; \mathbf{H}_{\Delta\alpha_2}\right) = \max_f I\left(f(\Phi_{\Delta\alpha_1}(\mathcal{G}, \alpha)); f(\Phi_{\Delta\alpha_2}(\mathcal{G}, \alpha))\right), \tag{5.18}$$

where $I(\cdot)$ is the mutual information. This framework lets the encoder $f(\cdot)$ learn the representation of two different views $U_{\Delta\alpha_1}$ and $V_{\Delta\alpha_2}$ while preserving as much MI as possible.

As shown in Equation 5.18, the view generation is controlled by two perturbation terms $\Delta\alpha_1$ and $\Delta\alpha_2$, which may affect the objective function. The current GCLs manually select these parameters in advance, *e.g.,* grid search, and fix contrastive views during training. This parameter selection strategy could make the encoder obtained by Equation 5.18 learn only the contrastive information in some particular views and lead to a reduced generalization of the model [195].

Dynamic-view Objective. To solve it, we propose a more generalized objective to learn from dynamic changing contrastive views. We rewrite Equation 5.18 as

$$\max_{f^*} I\left(\mathbf{H}_{\Delta\alpha_1}; \mathbf{H}_{\Delta\alpha_2}\right) = \max_{f^*} \mathop{\mathbb{E}}_{\Delta\alpha_1, \Delta\alpha_2} I\Big(f^*(\Phi_{\Delta\alpha_1}(\mathcal{G}, \alpha)); f^*(\Phi_{\Delta\alpha_2}(\mathcal{G}, \alpha))\Big), \tag{5.19}$$

which requires the obtained optimal encoder f^* works well with $\forall\Delta\alpha_1, \Delta\alpha_2 \in [0, 1-\alpha)$ to learn a balanced and generalized representation. Note that *InfoMin* [158] designs a similar strategy that iteratively maximizes the MI in the worst views. In contrast, our objective maximizes the average MI in all views because finding the worst views is related to the downstream task, about which we have zero prior knowledge. Besides, unlike other contrastive learning methods that also use multiple contrastive views [157, 57, 203], there is no fixed number of views in our approach, and we do not preset the parameters to fix the contrastive views but let them dynamically change during training.

However, maximizing Equation 5.19 is not easy. Varying the perturbation term brings changes to the MI of views, which may result in large variance during the training process and unable to converge [158]. To increase the stability, we empirically make one of the views unperturbed in this chapter, *i.e.,* $\Delta\alpha_1 = 0$. Then, we simplify Equation 5.19 as

$$\max_f \mathop{\mathbb{E}}_{\Delta\alpha} I\Big(f(\mathbf{L}_{\text{appr}}(\mathcal{G}, \alpha)); f(\Phi_{\Delta\alpha}(\mathcal{G}, \alpha))\Big). \tag{5.20}$$

We mark this view as U and design a multi-task training scheme to optimize proposed objective[1].

5.3.2 Training using Multi-task Curriculum Learning

To tackle this problem, we propose a training scheme using curriculum learning, which utilizes *prior knowledge about the difficulty of the learning tasks* to learn from easy-to-difficult contrastive views.

Multi-task Curriculum Learning. Curriculum Learning facilitates the optimization on such a complex objective by scheduling the sub-objectives in a certain order [8, 134]. Inspired by it, we regard the process of training DiGCL as a multi-task problem [134, 141, 51, 52, 120, 96] and decompose it into M sub-tasks $\eta_1, \dots, \eta_M$. Thus, the objective Equation 5.20 can be rewritten as

$$\max_{f^*} \frac{1}{M} \sum_{m=1}^{M} \mathop{\mathbb{E}}_{\Delta\alpha_m} I\Big(f^*(\mathbf{L}_{\text{appr}}(\mathcal{G}, \alpha)); f^*(\Phi_{\Delta\alpha_m}(\mathcal{G}, \alpha))\Big), \tag{5.21}$$

where $\Delta\alpha_m \in \{\Theta_{1,\dots,M}\}$ is a uniform partition of $\Theta = [0, 1-\alpha)$ and the sub-task η_m is associated with the $\Delta\alpha_m$. Then we consider sub-tasks that have less difficulty to learn in the optimization process as **easy** sub-tasks and vice versa as **difficult** sub-tasks. We quantify this difficulty into *difficulty score* by means of a scoring function $\mathcal{D}(\cdot)$. Based on it, we learn sub-tasks sequentially in a certain order decided by the pacing function $\mathcal{P}(\cdot)$ to solve the main multi-task problem. Via information transmission by a shared encoder, the previously solved easy sub-tasks will assist in solving the next difficult sub-tasks, while the latter can fine-tune the encoder learned by the former.

Scoring Function. The scoring function $\mathcal{D}(\cdot)$ measures how difficult the sub-task η_m is, and can be any function $\mathcal{D}(\cdot) : \{\eta_1, \dots, \eta_M\} \to \{d_1, \dots, d_M\}$, where d denotes the difficulty score. In each sub-task, there exists a pair of contrastive views as defined in Equation 5.21. We empirically assume that the difficulty d_m of the sub-task η_m depend on how difficult it is for the encoder $f(\cdot)$ to learn the contrastive information from this sub-task. Referring to Definition 20, we can find Laplacaian perturbation provides contrastive information to the encoder, and the perturbation

[1]The objective of fixing a view is not exactly the same as the original one, and we ignore this trick's impact.

Name	Expression $\mathcal{P}_{(d_a,d_b)}(l)$
log	$d_a + (d_b - d_a)\left(1 + 1/3\log\left(\frac{l}{L} + e^{-3}\right)\right)$
exp	$d_a + \frac{d_b - d_a}{e^3 - 1}\left(\exp\left(\frac{3l}{L}\right) - 1\right)$
linear	$d_a + (d_b - d_a)\frac{l}{L}$

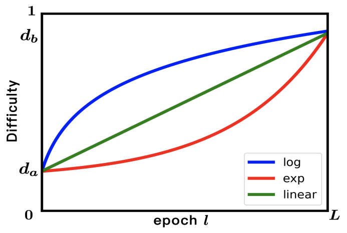


Figure 5.3: (left) pacing function definitions for the three families of pacing functions used throughout; (right) the plot of pacing function curves from each family. l is the training epoch.

term $\Delta\alpha$ controls how much of this information is. Therefore, we can define the scoring function $\mathcal{D}(\cdot)$ to calculate the difficulty d_m of a sub-task η_m as

$$d_m = \mathcal{D}(\eta_m) = 1 - \frac{\Delta\alpha_m}{1-\alpha}. \tag{5.22}$$

The higher this score represents the smaller the perturbation error, the less contrastive information provided, and the less the encoder is able to distinguish the difference between the two views, resulting in a more difficult sub-task. This idea coincides with the design of the discriminator in GAN [219, 47].

Pacing Function. The pacing function decides the learning sequence of sub-tasks and can be any function $\mathcal{P}(\cdot) : \{1, \dots, L\} \to \{d_1, \dots, d_M\}$, where L denotes all the learning iterations. We consider three function families [191]: logarithmic, exponential, and linear. Table in Figure 5.3(left) illustrates the pacing functions used in our experiments, which are parameterized by (d_a, d_b). Here d_a is the initial difficulty and d_b is the difficulty at the end of training, thus any pacing function with $d_a = d_b$ is equivalent to fixed view contrastive learning. We mark their corresponding perturbation terms as $(\Delta\alpha_a, \Delta\alpha_b)$. Considering the score in Equation 5.22 is defined in a continuous domain, we set the step size of the pacing function to 1, the finest-grained unit, *i.e.,* $L = M$. This can help the model to switch between different training views in a more delicate way.

It is desired to find the optimal solution over the entire definition domain of $\Delta\alpha \in [0, 1-\alpha)$, we set the start point $\Delta\alpha_a = 0.9 - \alpha$ (0.9 is used because the boundary value cannot be obtained) and ending point $\Delta\alpha_b = 0$ in this chapter, *i.e.,* $d_a = 1/9, d_b = 1$.

Contrastive loss. After discussing how to measure expectations in the objective, our attention shifts to maximizing the MI. Several works have investigated the lower bound of MI in contrastive learning, here we adopt *InfoNCE* [123]. For i^{th} node u_i in view U, the node-wise objective is defined as

$$\ell\left(u_i, v_i\right) = -\log \frac{\exp\left(\mathcal{S}\left(\boldsymbol{z}_U^i, \boldsymbol{z}_{V_{\Delta\alpha_m}}^i\right)/\tau\right)}{\sum_{j=1}^{n} \exp\left(\mathcal{S}\left(\boldsymbol{z}_U^i, \boldsymbol{z}_{V_{\Delta\alpha_m}}^j\right)/\tau\right)}, \tag{5.23}$$

where v_i is its corresponding positive node in view $V_{\Delta\alpha_m}$ and other nodes in view $V_{\Delta\alpha_m}$ is negative nodes of u_i. $\boldsymbol{z}_U^i$ is the projection of node feature that $\boldsymbol{z}_U^i \in \mathbf{Z}_U = g(\mathbf{H}_U)$. A non-linear transformation $g(\cdot)$ named projection head maps augmented representations $\mathbf{H}$ to another lower dimension where the contrastive loss is calculated. $\mathcal{S}(\cdot)$ denotes cosine similarity function and τ denotes the temperature parameter. The final loss is computed across all positive node pairs in the views of one pace. Note that GRACE [228] proposes similar *InfoNCE-like* loss function. Different from it, we do not treat intra-view nodes (nodes in U except u_i) as the negative samples since GNN-based encoder is already able to learn intra-graph structure well, we want to focus on inter-graph contrastive information.

Algorithm 6: DiGCL Training Procedure

```
Input: Directed graph: $\mathcal{G}$, teleport probability $\alpha$, scoring function: $\mathcal{D}$,
       pacing function: $\mathcal{P}$, encoder: $f^*(\cdot)$, projection head: $g(\cdot)$, data
       augmentation function: $\Phi(\cdot)$, loss function: $\ell(\cdot)$, number of
       iterations: $L$, initial difficulty $d_a$, ending difficulty $d_b$
Output: Trained Encoder $f^*(\cdot)$
1:  Initialize $f^*(\cdot)$, $g(\cdot)$;
2:  for $l \leftarrow 0$ to $L$ do
3:      $d_m = \mathcal{P}_{(d_a,d_b)}(l)$ ;
4:      $\Delta\alpha \leftarrow \mathcal{D}^{-1}(d_m)$ ;
5:      $U \leftarrow \mathbf{L}_{\text{appr}}(\mathcal{G}, \alpha)$ ;
6:      $V \leftarrow \Phi_{\Delta\alpha}(\mathcal{G}, \alpha)$ ;
7:      $\mathbf{H}_U \leftarrow f(U)$ ;
8:      $\mathbf{H}_V \leftarrow f(V)$ ;
9:      $\mathbf{Z}_U \leftarrow g(\mathbf{H}_U)$ ;
10:     $\mathbf{Z}_V \leftarrow g(\mathbf{H}_V)$ ;
11:     loss $\leftarrow \ell(\mathbf{Z}_U, \mathbf{Z}_V)$ ;
12:     SGD(loss) ;
13: return $f^*(\cdot)$
```

5.4 Experiments

We conduct extensive experiments to evaluate the effectiveness of our model. We implement the DiGCL and all baseline models using the python library of PyTorch, PyG [40] and DGL [173]. All the experiments are conducted on a server with one 12GB NVIDIA TITAN V GPU, two Intel Xeon E5 CPUs and Ubuntu 18.04 System. Our implement can be obtained at `https://github.com/flyingtango/DiGCL`.

5.4.1 Experimental Settings

5.4.1.1 Experimental Task

There are two main tasks to evaluate the capability of GCL, one is *node classification* [57, 228] and the other is *graph classification* [207, 136]. There are different at the instance scale. Since the datasets for graph classification are mostly small molecules, the graph structure is simple and undirected, and the effectiveness of our model cannot be measured. Therefore we mainly use the **Node Classification in Directed Graphs** [160] to measure each module in this chapter.

Compared with the common experiments for undirected graphs [88], the challenge of node classification in directed graph is that the given adjacency matrix $\mathbf{A}$ is asymmetric, which means message passing has its direction. We will also use datasets of undirected graphs for controlled trials. The task definition is at Definition 8.

For self-supervised methods, the task requires to use the adjacency matrix $\mathbf{A}$ and the node feature matrix $\mathbf{X}$ to learn node representation without labels. Specifically, after the model has unsupervisedly learned the node feature representation, simple classical classification algorithms, such as logistic regression, SVM, and etc., can be used to categorize the nodes from the node representation, which is a semi-supervised step. In this chapter, all experiments in semi-supervised learning are set up the same, including the division of the datasets and the number of trial repetitions.

Besides, to verify the generalizability of our approach, we also perform the graph classification task on three undirected graph datasets in the experiments. The definition and experiment setup are the same as for GRACE[228] and GCA [229].

5.4.1.2 Datasets and Splitting

We use several widely-used datasets including directed graph datasets: Cora-ML [10], CiteSeer [144] and Am-Photo [145]; undirected graph datasets: PubMed [121] and DBLP [127]. The split of the datasets would have a significant effect on models' results [145]. Thus, we divide the datasets randomly and conduct multiple tests to achieve consistent outcomes. For train/validation/test split, following the rules in GCN [88], we choose 20 labels per class for training set, 500 labels for validation set, and rest for the test set.

Table 5.1: Datasets Details for Node Classification

Datasets	Graph type	Nodes	Edges	Classes	Features	Label rate
Cora-ML [10]	Directed	2995	8416	7	2879	4.67%
CiteSeer [144]	Directed	3312	4715	6	3703	3.62%
Am-Photo [145]	Directed	7650	143663	8	745	2.10%
PubMed [121]	Undirected	18230	79612	3	500	0.33%
DBLP [127]	Undirected	17716	105734	4	1639	0.45%

We use five open access datasets in the task of node classification. Label rate is the fraction of nodes in the training set per class. We use 20 labeled nodes per class to calculate the label rate.

Besides, we use the following datasets for graph classification task: MUTAG [91] containing mutagenic compounds, PTC [91] containing compounds tested for carcinogenicity, and IMDB-Bin [202] connecting actors/actresses (nodes) based on movie appearances (edges).

Table 5.2: Dataset Details for Graph Classification

Datasets	Graphs	Average nodes per graph	Average edges per graph	Classes
MUTAG	188	17.93	19.79	2
PTC	344	14.29	14.69	2
IMDB-Bin	1000	19.77	193.06	2

5.4.1.3 Baselines

We compare our model to the 11 SOTA models divided into four main categories: 1) **supervised** models for undirected graph with GCN [88], GAT [165] and [90];

2) **supervised** models for directed graph with MagNet [216] and DiGCN [160]; 3) **self-supervised** models without augmentations including DGI [166] and GMI [133]; 4) **contrastive learning** models containing MVGRL [57], GraphCL [207], GRACE [228], and GCA [229]. The baseline methods are given below:

Table 5.3: The implementations of the baselines on the node classification task.

Model	Training Type	Implementation
GCN [88]	Supervised	`https://pyg.org/`
GAT [165]	Supervised	
APPNP [90]	Supervised	
MagNet [216]	Supervised	`https://github.com/matthew-hirn/magnet`
DiGCN [160]	Supervised	`https://github.com/flyingtango/DiGCN`
DGI [166]	Self-supervised	`https://github.com/PetarV-/DGI`
GMI [133]	Self-supervised	`https://github.com/zpeng27/GMI`
MVGRL [57]	Self-supervised	`https://github.com/kavehhassani/mvgrl`
GraphCL [207]	Self-supervised	`https://github.com/Shen-Lab/GraphCL`
GRACE [228]	Self-supervised	`https://github.com/CRIPAC-DIG/GRACE`
GCA [229]	Self-supervised	`https://github.com/CRIPAC-DIG/GCA`

For all baseline models, we use their model structure in the original papers, including layer number, activation function selection, normalization and regularization selection, etc. We implement GCN, GAT, and APPNP using PyG [40]. Note that for DiGCN, we do not use its inception module but only use the directed graph convolution. Detailed hyper-parameter settings are shown in Table 5.6.

To ensure the generality of the model, we have minimized the variation of hyper-parameters. Our implementation is based on the GRACE code, with improvements to the topological data augmentation and the model training scheme. For the feature-level perturbation part, we also apply the dropping feature method used in GCA, GRACE and MVGRL, with the same parameters as in GRACE. We initialize our model with Glorot initialization [46] and use Adam optimizer [86] in all datasets. The initial learning rate is set to 0.001 and the weight decay factor is set to 1e-5 on all datasets. We set the number of layers used in the GCN encoder as 2. As stated in Section 5.3.2, we fixed the initial and ending difficulty as 0.8 and 0 to obtain the complete contrastive information. The detailed parameter settings are shown in

Table 5.4.

Table 5.4: The hyperparameters of our models.

Our models	layer	lr	weight-decay	hidden dim	init $\Delta\alpha$	end $\Delta\alpha$	epochs
CORA-ML	2	0.001	1e-5	128	0.8	0	600
CITESEER	2	0.001	1e-5	128	0.8	0	300
AM-PHOTO	2	0.001	1e-5	512	0.8	0	2000
PUBMED	2	0.001	1e-5	256	0.8	0	600
DBLP	2	0.001	1e-5	256	0.8	0	600

For the graph classification task, we follow the setting in the [57] and only change the data augmentation and the pacing function. The hyperparameters are as follow.

Table 5.5: The hyperparameters on graph classification task.

Method	Hyperparameters	MUTAG	PTC	IMDB-BIN
MVGRL	layer	4	4	2
	batches	256	128	256
	epochs	20	20	20
	$\alpha = 0.2$	0.1	0.1	0.1
MVGRL+DiGCN	layer	4	4	2
	batches	256	128	256
	epochs	20	20	20
	$\Delta\alpha$	$0.8 \to 0$	$0.8 \to 0$	$0.8 \to 0$

Table 5.6: The hyperparameters of baselines for node classification task.

Model	layer	lr	weight-decay	hidden dimension	Others
GCN	2	0.01	5e-4	64	-
GAT	2	0.005	5e-4	CORA-ML & CITESEER:8 others:32	heads=16
APPNP	2	0.01	5e-4	64	$\alpha = 0.1$
MagNet	2	5e-3	5e-4	64	$K = 1$, $q = 0.1$
DiGCN	2	0.01	5e-4	64	$\alpha = 0.1$
DGI	1	0.001	0	512	max-LR-iter=150
GMI	1	0.001	0	512	$\alpha = 0.8$, $\beta = 1$, $\gamma = 1$
MVGRL	1	0.001	0	512	$\alpha = 0.2$, $t = 5$
GraphCL	1	0.001	0	512	drop rate=0.2
GRACE	2	0.001	1e-5	CORA-ML & CITESEER:128 others:256	augmentation parameters are consistent with the paper
GCA	2	0.001	1e-5	CORA-ML & CITESEER:128 others:256	augmentation parameters are consistent with the paper

5.4.2 Experimental Results

Accuracy of node classification. The performance comparisons between our model and baselines models on five datasets are reported in Table 5.7. We use a two-layer GCN as our encoder and first train our model in an unsupervised manner to obtain the embedding. Then we take a simple ℓ_2-regularized logistic regression as the classifier [166]. For curriculum learning scheme, we select the log pacing function and the start and ending difficulties are set in Section 5.3.2. We train all models according to their default settings, then calculate mean test accuracy with standard deviation (STD) in percent (%) averaged over 20 random dataset splits with random weight initialization.

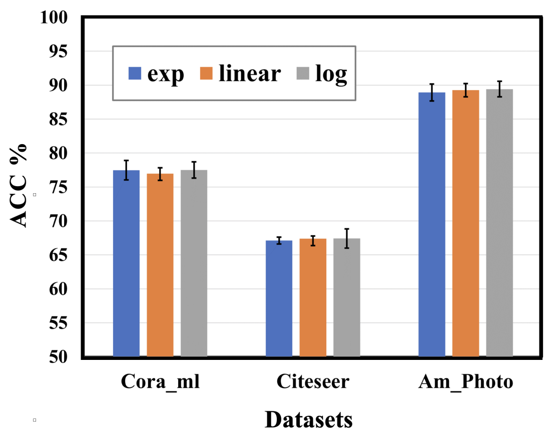
It can be seen easily that our methods achieve the state-of-the-art results on all datasets. In general, the unsupervised methods including MVGRL, GRACE, and GCA, do not perform well on directed graphs compared to their good performance in undirected graphs. This is mainly due to the models' data augmentation methods are not applicable to digaphs, resulting in the inability to learn contrastive information from complex directed structures. Notice that GraphCL performs relatively mediocre in unsupervised methods, most notably because it focuses mainly on graph-level unsupervised methods, and does not apply well to node-level contrastive learning task. Since DGI and GMI do not require data augmentation to provide contrastive information, they perform very well on both undirected and directed graphs, which shows good suitability for different graph structure. Moreover, supervised methods such as GCN, GAT and APPNP are inferior to DiGCN and MagNet, which are specifically designed for directed graphs, in terms of performance. Since our Laplacian perturbation uses the approximate Laplacian matrix proposed by DiGCN, we compare their performance. It is not difficult to find that our model outperforms DiGCN on all datasets, which shows that contrastive learning can learn good encoders by performing a certain data augmentation in an unsupervised manner.

Table 5.7: Accuracy (%) of node classification task with STD. "No Curr" means do not use curriculum learning and we set two fixed views as $\Delta\alpha_a = \Delta\alpha_b = 1 - \alpha$. "Random" means random order, "Anti Curr" means using anti-curriculum order and "Curr" indicates using curriculum order. The best results are highlighted with **bold** and the second are marked with <u>underline</u>. OOM means out of memory on a 12GB GPU.

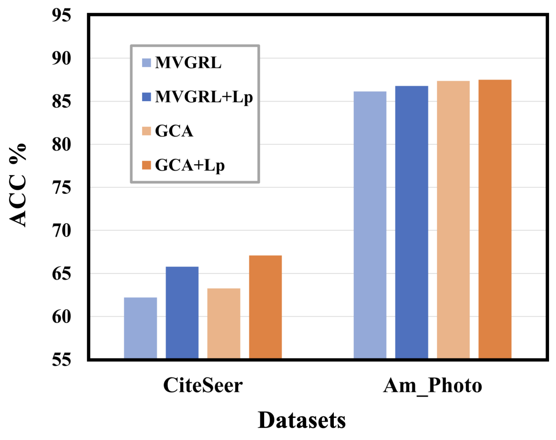
	Method	DIRECTED			UNDIRECTED	
		CORA-ML	CITESEER	AM-PHOTO	PUBMED	DBLP
SUPERVISED	GCN [88]	70.92 ± 0.39	63.00 ± 0.45	88.52 ± 0.47	78.78 ± 0.30	73.54 ± 0.77
	GAT [165]	72.22 ± 0.57	63.73 ± 0.57	88.36 ± 1.25	77.49 ± 0.47	76.08 ± 0.54
	APPNP [90]	70.31 ± 0.67	61.63 ± 0.63	87.43 ± 0.98	79.35 ± 0.48	77.92 ± 0.75
	MagNet [216]	76.32 ± 0.10	65.04 ± 0.47	86.80 ± 0.65	74.23 ± 0.46	69.73 ± 0.98
	DiGCN [160]	<u>77.03 ± 0.70</u>	64.60 ± 0.60	88.66 ± 0.51	76.79 ± 0.49	73.37 ± 0.72
UNSUPERVISED	DGI[166]	75.21 ± 1.29	64.58 ± 1.78	85.25 ± 0.59	74.11 ± 0.62	76.53 ± 1.24
	GMI[133]	76.59 ± 0.35	63.29 ± 0.70	81.12 ± 0.01	80.27 ± 0.16	76.66 ± 0.48
	MVGRL[57]	76.67 ± 0.12	62.22 ± 0.02	86.15 ± 0.21	79.98 ± 0.04	OOM
	GraphCL [207]	67.34 ± 0.12	57.84 ± 0.11	67.66 ± 0.05	75.29 ± 0.08	77.85 ± 0.22
	GRACE [228]	73.88 ± 0.25	61.20 ± 0.20	87.95 ± 0.32	79.54 ± 0.05	78.03 ± 0.09
	GCA [229]	76.32 ± 0.33	63.25 ± 0.10	87.35 ± 0.27	79.81 ± 0.61	77.83 ± 0.35
	Ours + No Curr	75.86 ± 0.09	66.99 ± 0.54	87.32 ± 0.14	79.57 ± 0.12	78.28 ± 0.05
	Ours + Random	76.52 ± 1.66	<u>67.15 ± 0.82</u>	<u>89.03 ± 0.46</u>	$\mathbf{80.75 \pm 0.10}$	<u>79.58 ± 0.14</u>
	Ours + Anti Curr	76.12 ± 1.04	66.83 ± 1.13	88.83 ± 0.73	80.22 ± 0.37	79.42 ± 0.15
	Ours + Curr	$\mathbf{77.53 \pm 0.14}$	$\mathbf{67.42 \pm 0.14}$	$\mathbf{89.41 \pm 0.11}$	<u>80.69 ± 0.08</u>	$\mathbf{79.70 \pm 0.13}$

Ablation study on curriculum learning. We validate the effectiveness of curriculum learning strategy and the results are shown in Table 5.7. We find that even with the contrastive views containing the maximum information (with the largest perturbation term), the effect of the model without curriculum learning is significantly lower than the model with curriculum learning. This means that features learned from fixed views is always incomplete. Besides, we also find that the order of curriculum learning is vital, the ACC of learning from easy-to-difficult > random order > difficult-to-easy. This substantially validates that scoring function do help to learn from multiply tasks.

Impact of different pacing functions. We study the three different pacing functions defined in Table 5.3 and the results in Figure 5.4(a) show that using different path functions can have an impact on the performance but not as much as changing the order of curriculum learning. Among them, the log pacing function performs best as it speeds up learning on easy tasks and stays on harder tasks for more epochs, thus helping the model to grasp the more subtle differences between contrastive views.



(a)



(b)

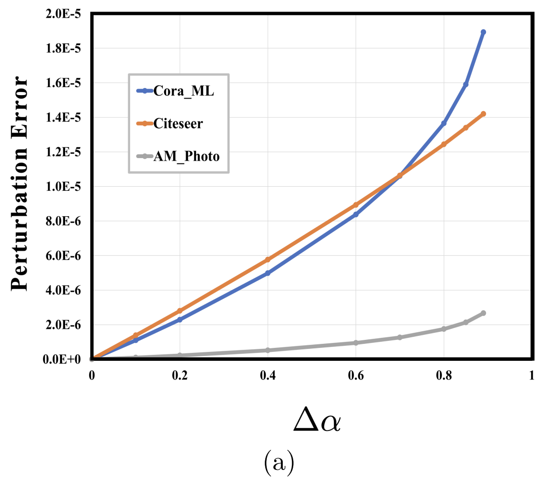
Figure 5.4: (a) node classification results with various pacing functions; (b) node classification results for the generalizability of Laplacian perturbation on two datasets, Lp means with Laplacian perturbation.

Table 5.8: Overall accuracy (%) with STD with various data augmentation methods. Our model uses curriculum learning, and use the log pacing function. The best results are highlighted with **bold**.

Methods	Data Augmentation Method	CiteSeer	Am-Photo	PubMed
GraphCL [207]	Edge perturbation	57.84 ± 0.11	67.66 ± 0.05	75.29 ± 0.09
	Node dropping	57.45 ± 0.12	66.69 ± 0.07	75.25 ± 0.08
	Subgraph sampling	57.59 ± 0.10	66.75 ± 0.07	74.75 ± 0.11
GRACE [228]	Random removing edges	61.20 ± 0.20	87.95 ± 0.32	79.54 ± 0.05
MVGRL [57]	Graph diffusion with heat kernel	61.22 ± 0.07	79.63 ± 0.31	78.54 ± 0.33
	Graph diffusion with PageRank kernel	62.22 ± 0.02	86.15 ± 0.21	79.98 ± 0.04
GCA [229]	Removing edges by degree score	63.25 ± 0.10	87.35 ± 0.27	79.81 ± 0.61
	Removing edges by PageRank score	62.21 ± 0.16	86.88 ± 0.37	78.92 ± 0.37
	Removing edges by eigenvalue score	63.12 ± 0.08	87.02 ± 0.56	79.70 ± 0.09
Ours	Random removing edges	64.97 ± 0.08	88.45 ± 0.01	79.66 ± 0.13
	Laplacian perturbation	$\mathbf{67.42 \pm 0.14}$	$\mathbf{89.41 \pm 0.11}$	$\mathbf{80.69 \pm 0.08}$

Effect of Laplacian perturbation and other data augmentation methods. Table 5.8 shows the effect of the different data augmentation methods on the five contrastive learning models. We our Laplacian perturbation works best on both directed and undirected graphs. But the difference with other methods is not significant on the undirected graph dataset PubMed. In addition to this, we also compare with random removing edges used in GRACE under our DiGCL framework. We find that the data augmentation approach without damaging the graph structure can achieve better performance.

Generalization of Laplacian perturbation. Here, we will add generalizability experiments that migrate Laplacian perturbation to MVGRL [57] and GCA [229]. We replace the topological data augmentation method in the original model with our Laplacian perturbation and follow all the configurations in the original model, including the parameters of the data augmentation, the number of contrastive views, the structure of the model, and the training parameters. We test the generalization performance of our Laplacian perturbation in the node classification task. The results are shown in Figure 5.4(b). We can clearly find that the performance of the model on directed graphs can be improved after using Laplacian perturbation, which indicates that our method can help existing contrastive learning models to learn structure information from complex networks.



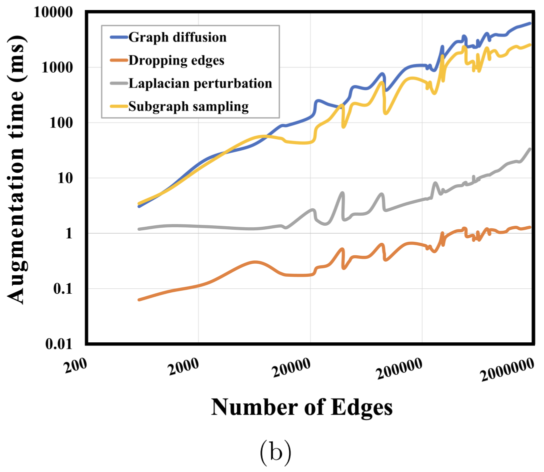
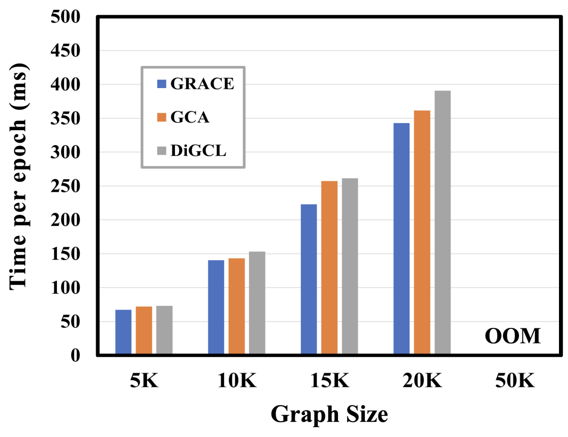


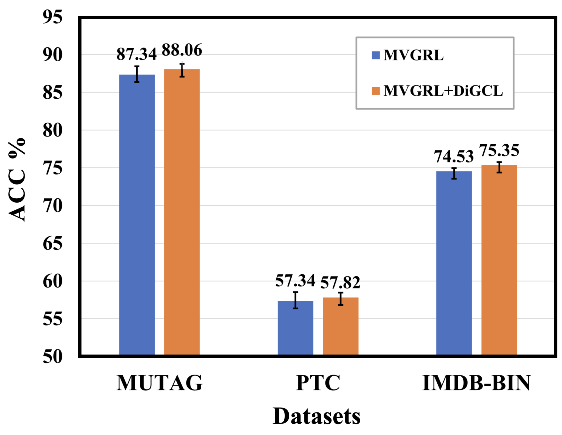
Figure 5.5: (a) perturbation error with various perturbation terms $\Delta\alpha$; (b) time of different data augmentation methods on random graphs with different size.

Monotonicity of the perturbation error in real-world datasets. To visualize the variation of the perturbation error with the perturbation term, we empirically show the variation of directed graph entropy in the three real-world directed graph datasets: Cora-ML, Citeseer and AM-Photo. We set $\alpha = 0.1$ and take $\Delta\alpha \in \{0, 0.1, 0.2, 0.4, 0.6, 0.7, 0.8, 0.85, 0.89\}$. The results are shown in Figure 5.5(a), and it is clearly shown that as the perturbation term increases, the degree of perturbation is elevated, and the perturbation error will increase.

Augmentation time at different graph scales. We construct 20 random directed graphs of different scales, with the number of edges ranging from 1K to 2M, then test four different data augmentation methods' mean running time in milliseconds (ms) for 5 times. Figure 5.5(b) summarizes the results and shows that our Laplacian perturbation is very competitive in terms of runtime among these methods. In contrast to the graph diffusion, which also require the computation of eigenvectors, we can reduce the time by a factor of one hundred with our fast algorithm proposed in Section 5.2.2. This makes it possible to dynamically perform Laplacian perturbation to generate contrastive views during training.



(a)



(b)

Figure 5.6: (a) running time per epoch of different models on random size graphs, OOM means out of memory; (b) generalization experiment of graph classification task based on MVGRL.

Running time with different graph size. To generate arbitrary graph size, we construct a simple random graph with N nodes and assign $10N$ directed edges uniformly at random. We compare with GRACE[228] and GCA [229] since these two models have the similar accuracy with ours. We record the running time of the models for each epoch at different graph sizes, and the results are shown in Figure 5.6(a). We can find that processing 20,000 nodes and 200,000 edges with 12 GB memory is the limit for these three models. Since our model requires a Laplacian perturbation operation, this will take more time than the data augmentation performed by GCA and GRACE.

Generalization to graph classification task. To validate the generalization ability of our DiGCL model, we migrate it to MVGRL [57] and test on three graph classification datasets: MUTAG [91], PTC [91], and IMDB-Bin [202]. The generalized MVGRL model is denoted by MVGRL+DiGCL and the results are shown in Figure 5.6(b). Using our model can improve the accuracy on all datasets, however, the improvement is not significant. We consider there are three main potential reasons for this: 1) the datasets are undirected graphs, and the effect of using Laplacian perturbation on them is not as obvious as in directed graphs; 2) the graph classification problem requires more graph-level contrastive information, while our model focuses on node-level; 3) our scoring function designed for node-level curriculum learning is not suitable for graph-level difficulty measure.

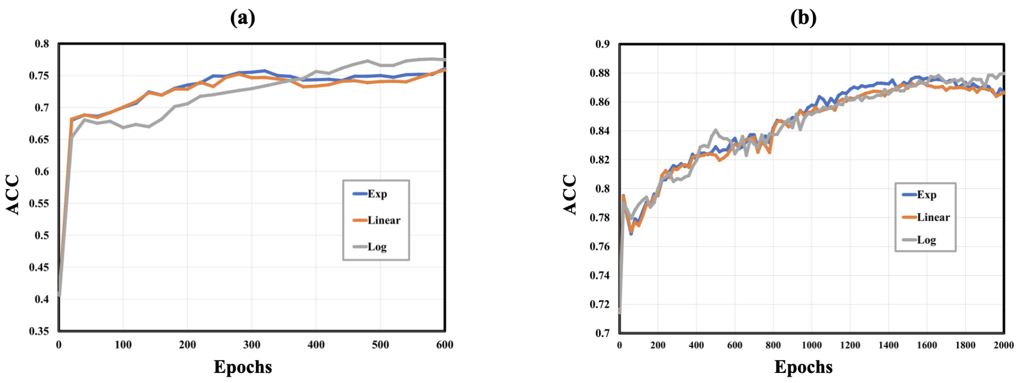
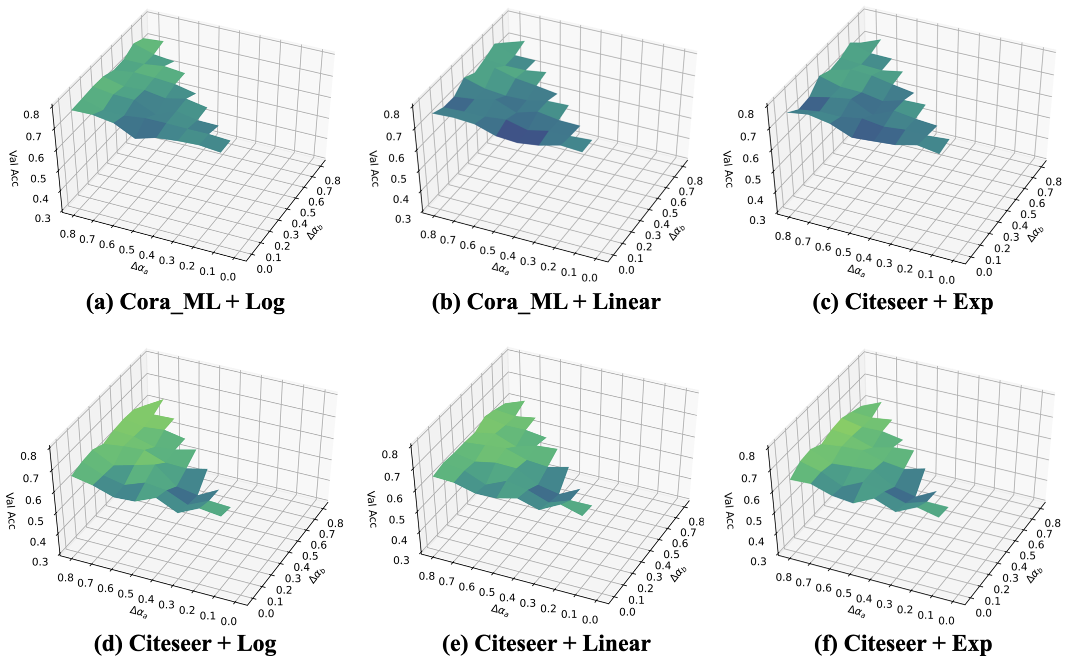


Figure 5.7: (a) performance of node classification task on Cora-ML with different pacing functions; (b) performance of node classification task on AM-Photo with different pacing functions.

Accuracy with epoch for different pacing functions. First, we give the results of the val accuracy changes with three different pacing functions in Cora-ML and AM-Photo in Figure 5.7(a) and 5.7(b) separately. We can find that different pacing functions perform differently at different training stages. Linear performs evenly throughout the training process; Exp improves faster at the beginning of training, but plateaus in the later stages; Log improves slowly at the beginning of training, but it continues to improve and achieves the best results at the end of training. The main reason is the log pacing function speeds up learning on easy tasks and stays on harder tasks for more epochs, helping the model to grasp the more subtle differences between contrastive views. This is the cause of its ability to consistently improve his performance in the later stages.



(a) Cora_ML + Log (b) Cora_ML + Linear (c) Citeseer + Exp

(d) Citeseer + Log (e) Citeseer + Linear (f) Citeseer + Exp

Figure 5.8: Validation accuracy of node classification task with different perturbation terms. The shade of the color represents the accuracy, with lighter shades indicating higher accuracy.

Sensitivity analysis for initial and ending difficulty. Recalling the analysis in Section 5.3.2, to obtain comprehensive contrastive information on the one hand, and to reduce the need for hyperparameters on the other hand, we set the initial perturbation term $\Delta\alpha_a$ to 0.8 and the ending perturbation term $\Delta\alpha_b$ to 0. In this experiment, we will explore the effect of different initial and ending difficulties on the accuracy of the model. We traverse the $\Delta\alpha_a, \Delta\alpha_b \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and make sure $\Delta\alpha_a \geqslant \Delta\alpha_b$ ($\Delta\alpha_a = \Delta\alpha_b$ is equivalent to fixed view contrastive learning). We use two datasets and three pacing functions in the experiment. The results are shown in Figure 5.8. We can clearly find that setting the perturbation terms as the boundary values allows the model to learn all views as much as possible, thus improving the performance. Also, comparing the results of **Log**, **Liner**, and **Exp**, we can find that using log as the pacing function can get more stable and accurate results.

From these experiments, we can draw a few empirical conclusions as follows.

- Log-based pacing function performs the best of the three pacing functions, but not too far from the other two pacing functions.
- The best results are obtained by setting the start and end points to be the boundary points of the Laplacian perturbation parameter space.
- The order in which the views are learned is crucial, with contrastive views working best from easy to difficult (Concluded from Table 5.7).

For the starting and ending difficulty scores, in accordance with the second conclusion, we consider that it is better to take the boundary values, which are effective and do not require parameter selection. For the type of pacing functions, according to the first and third conclusions, the different pacing functions have an impact on the results of the model but are not as important as the learning order. We believe that any pacing functions that satisfy the order of easy to difficult can be chosen.

5.5 Summary

In this chapter, we design a directed graph data augmentation scheme called Laplacian perturbation and theoretically investigate how it can provide contrastive information without changing the directed graph structure. Moreover, we present the DiGCL which utilizes Laplacian perturbation and curriculum learning to progressively learn from dynamic easy-to-difficult contrastive views. Finally, we use several tasks on various datasets to demonstrate the effectiveness and generalization ability of our proposed DiGCL. We empirically show that DiGCL can retain more structural features of directed graphs than other GCL models while providing adequate contrastive information. Extensive experiments show that our DiGCL outperforms the state-of-the-art approaches.

Chapter 6

Conclusion and Future Work

This chapter provides a thorough conclusion for the thesis, followed by discussions of the future works.

6.1 Conclusion

Graph-structured data is ubiquitous, ranging from social relationships to urban road networks. It records the connections between entities with an efficient data structure. How to make good use of the large-scale graph structure data that contains information is the direction that people are looking for. Deep learning is one type of learning method. By designing a deep neural network suitable for graph-structured data, we can efficiently collect node, edge, and graph-level features from graph data, so as to facilitate us to solve related real-world problems. This thesis is small while a concrete step towards this grand subject, deep learning on graph-structured data.

In this thesis, we focus on extending the application of deep graph neural networks to directed graphs. In the first part of this thesis (Chatper 1), we introduce deep learning and some real-world problems addressed using deep graph neural networks. At the same time, according to the issue that existing graph deep learning methods cannot be used in directed graphs, we propose some directions. for improvement and explain our motivations. Based on these directions where improvements can be made, we present the main objectives and contributions of this paper.

In Chapter 2, we provide a comprehensive overview of Graph Neural Networks. We begin with the overall design process of GNNs and summarize the tasks associated with the various modules. In addition, we provide a summary of the pros and cons of various strategies and address the flaws of existing work on directed graphs. Moreover, we provide an in-depth analysis of the work in the fields of Graph Convolutional Networks and Graph Contrastive Learning, which are closely related to later chapters, as well as some preliminaries.

In Chapter 3, we introduce Directed Graph Convolution Networks (DGCN), a novel graph neural network that can be applied to directed graphs. To enable spectral-based GCNs to generalize to directed graphs, we define first- and second-order proximity. It can keep graph's directed characteristics and enlarge the convolution operation's receptive field in order to extract and exploit nearby node features. In addition, we provide empirical evidence that this strategy increases the quantity and quality of the information obtained. Finally, we apply semi-supervised node classification tasks and extensive tests on numerous real-world datasets to demonstrate the efficacy and generalization capabilities of first- and second-order proximity, as well as the enhancements acquired by DGCN over other models.

In Chapter 4, we present Directed Graph Inception Convolutional Networks (DiGCN), which can be effectively learn directed graph representation. We theoretically extend spectral-based graph convolution to directed graph using the inherent connections between graph Laplacian and stationary distributions of PageRank. We further simplified it to optimizate propagation speed and memory usage. Besides, we define k^{th}-*order* proximity and design the directed graph inception networks to learn multi-scale features. This simple and scalable model can not only learn directed graph structure, but also get hidden information through k^{th}-order proximity relationship. Finally, we utilize a number of tasks on a range of real-world datasets to verify our model's performance and generalizability. The results show that our model outperforms several state-of-the-art methods.

In Chapter 5, we construct the Laplacian perturbation data augmentation strategy for directed graphs and study how it can give contrastive information without altering the directed graph's structure. Moreover, we present the Directed Graph Contrastive Learning (DiGCL) which utilizes Laplacian perturbation and curriculum learning to progressively learn from dynamic easy-to-difficult contrastive views. Finally, we perform a number of tasks across a range of datasets to showcase the efficacy and generalizability of our proposed method. Through extensive experimentation, we demonstrate that DiGCL is superior to competing GCL models in its ability to preserve critical structural aspects of directed graphs while still yielding useful contrasting information. Compared to the state-of-the-art methods, our DiGCL performs significantly better in extensive experiments.

By means of the presented studies, we have demonstrated the efficacies brought by deep learning on graph-structured data, especially on directed graph. In the future, we would like to transfer the experience that we learned to address directed graph problems from various perspectives.

6.2 Future Work

There are many potential directions to further explore on the topics studied in this thesis. We discuss some possible future work below.

Multi-level Tasks. As elaborated in Section 5.2.1 and 5.4.2, our approach is more suitable for obtaining node-level contrastive information. This limits us to use the model to solve some graph-level problems, such as pre-training on protein molecular, drug property prediction, etc. We will extend our approach to graph-level tasks in subsequent work. Besides, it is also worth exploring the direction of how to combine multi-scale information, such as combining node-level and graph-level information [110], to solve multi-level graph tasks.

Robustness. As a family of neural network based models, GNNs are also vulnerable to adversarial attacks. Compared to adversarial attacks on images or text which only focuses on features, attacks on graphs further consider the graph structural information [152]. Our models may be adversarial attacked by adding new nodes or deleting existing edges. For example, in a graph-based recommender system, our model may produce completely different recommendation results due to being attacked. Several efforts have been proposed to attack existing graph models [232, 27], while more robust models [224] have been presented to protect against these attacks. As a follow-up work we can explore the unique structural attack methods and defenses on directed graphs.

Large-scale Graphs. Real-world graphs are all very large in size, and due to the complex dependencies between nodes, the computational cost and memory space requirements increase exponentially as the number of GNN layers increases [115]. For example, Facebook's social network graph contains over 2 billion vertices and

1 trillion edges. Our model is not optimized for large-scale directed graphs, which may lead to insufficient memory or slow speed during training. How to efficiently sample and learn on large-scale directed graphs is also a pressing problem.

Dynamic Graphs. Mainstream research is limited to dealing with static graph data, while most of the real complex networks evolve in structure and properties over time [131]. Dynamic graph models add a temporal dimension to static graph models, enabling them to characterize both structural and temporal information of complex systems [148]. Dynamic graphs differ from static graphs in that the former have 1) dynamic structure, where connected edges and nodes disappear/appear over time, and 2) dynamic properties, where nodes and connected edge states change over time. In the subsequent work, we can migrate our static directed graph-based approach to dynamic graphs.

Publications during PhD Study

[1] **Z. Tong**, Y. Liang, C. Sun, X. Li, and Y. M. Chee, "Second-order directed graph convolutional network", *Working paper*, 2022.

[2] **Z. Tong**, Y. Liang, H. Ding, Y. Dai, X. Li, and C. Wang, "Directed graph contrastive learning", *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[3] **Z. Tong**, Y. Liang, C. Sun, X. Li, D. Rosenblum, and A. Lim, "Digraph inception convolutional networks", *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[4] **Z. Tong**, Y. Liang, C. Sun, D. S. Rosenblum, and A. Lim, "Directed graph convolutional network", *arXiv preprint arXiv:2004.13970*, 2020.

[5] Y. Dai, Y. Sun, J. Liu, **Z. Tong**, Y. Yang, and L.-Y. Duan, "Bridging the source-to-target gap for cross-domain person re-identification with intermediate domains", *arXiv preprint arXiv:2203.01682*, 2022.

[6] X. Li, H. Ding, **Z. Tong**, Y. Wu, and Y. M. Chee, "Primtive3d: 3d object dataset synthesis from randomly assembled primitives", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[7] Y. Dai, J. Liu, Y. Bai, **Z. Tong**, and L.-Y. Duan, "Dual-refinement: Joint label and feature refinement for unsupervised domain adaptive person re-identification", *IEEE Transactions on Image Processing*, vol. 30, pp. 7815–7829, 2021.

[8] X. Li, Z. Chen, Y. Zhao, **Z. Tong**, Y. Zhao, A. Lim, and J. T. Zhou, "Pointba: Towards backdoor attacks in 3d point cloud", in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2021, pp. 16 492–16 501.

[9] Y. Dai, J. Liu, Y. Sun, **Z. Tong**, C. Zhang, and L.-Y. Duan, “Idm: An intermediate domain module for domain adaptive person re-id”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 11 864–11 874.

[10] Y. Dai, X. Li, J. Liu, **Z. Tong**, and L.-Y. Duan, “Generalizable person re-identification with relevance-aware mixture of experts”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 16 145–16 154.

[11] X. Li, C. Li, **Z. Tong**, A. Lim, J. Yuan, Y. Wu, J. Tang, and R. Huang, “Campus3d: A photogrammetry point cloud benchmark for hierarchical understanding of outdoor scene”, in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 238–246.

[12] Y. Liang, K. Ouyang, H. Yan, Y. Wang, **Z. Tong**, and R. Zimmermann, “Modeling trajectories with neural ordinary differential equations”, in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, 2021, pp. 1498–1504.

[13] K. Ouyang, Y. Liang, Y. Liu, **Z. Tong**, S. Ruan, D. Rosenblum, and Y. Zheng, “Fine-grained urban flow inference”, *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Bibliography

[1] S. Abu-El-Haija, A. Kapoor, B. Perozzi, and J. Lee, "N-gcn: Multi-scale graph convolution for semi-supervised node classification", *arXiv preprint arXiv:1802.08888*, 2018.

[2] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network", in *2017 international conference on engineering and technology (ICET)*, Ieee, 2017, pp. 1–6.

[3] S. Back, J. Yoon, N. Tian, W. Zhong, K. Tran, and Z. W. Ulissi, "Convolutional neural network of atomic surface structures to predict binding energies for high-throughput screening of catalysts", *The journal of physical chemistry letters*, vol. 10, no. 15, pp. 4401–4408, 2019.

[4] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate", *arXiv preprint arXiv:1409.0473*, 2014.

[5] B. Bahmani, A. Chowdhury, and A. Goel, "Fast incremental and personalized pagerank", *arXiv preprint arXiv:1006.2880*, 2010.

[6] L. Bai, L. Yao, S. Kanhere, X. Wang, Q. Sheng, *et al.*, "Stg2seq: Spatial-temporal graph to sequence model for multi-step passenger demand forecasting", *arXiv preprint arXiv:1905.10069*, 2019.

[7] G. Barker and H. Schneider, "Algebraic perron-frobenius theory", *Linear Algebra and its Applications*, vol. 11, no. 3, pp. 219–233, 1975.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning", in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[9] S. Berrone, F. Della Santa, A. Mastropietro, S. Pieraccini, and F. Vaccarino, "Graph-informed neural networks for regressions on graph-structured data", *Mathematics*, vol. 10, no. 5, p. 786, 2022.

[10] A. Bojchevski and S. Günnemann, "Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking", *arXiv:1707.03815*, 2017.

[11] A. Bojchevski, J. Klicpera, B. Perozzi, M. Blais, A. Kapoor, M. Lukasik, and S. Günnemann, "Is pagerank all you need for scalable graph neural networks?" In *KDD, MLG Workshop*, 2019.

[12] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine", *Computer networks*, vol. 56, no. 18, pp. 3825–3833, 2012.

[13] M. Chatzianastasis, G. Dasoulas, G. Siolas, and M. Vazirgiannis, "Graph-based neural architecture search with operation embeddings", in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 393–402.

[14] C. Chen, W. Ye, Y. Zuo, C. Zheng, and S. P. Ong, "Graph networks as a universal machine learning framework for molecules and crystals", *Chemistry of Materials*, vol. 31, no. 9, pp. 3564–3572, 2019.

[15] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction", *arXiv preprint arXiv:1710.10568*, 2017.

[16] J. Chen, T. Ma, and C. Xiao, "Fastgcn: Fast learning with graph convolutional networks via importance sampling", *arXiv preprint arXiv:1801.10247*, 2018.

[17] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs", *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[18] T. Chen, W. Zhang, Q. Lu, K. Chen, Z. Zheng, and Y. Yu, "Svdfeature: A toolkit for feature-based collaborative filtering", *Journal of Machine Learning Research*, vol. 13, pp. 3619–3622, 2012.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations", *arXiv preprint arXiv:2002.05709*, 2020.

[20] Z. Chen, F. Chen, L. Zhang, T. Ji, K. Fu, L. Zhao, F. Chen, and C.-T. Lu, "Bridging the gap between spatial and spectral domains: A survey on graph neural networks", *arXiv preprint arXiv:2002.11867*, 2020.

[21] D. Cheng, Y. Tu, Z.-W. Ma, Z. Niu, and L. Zhang, "Risk assessment for networked-guarantee loans using high-order graph attention representation." In *IJCAI*, 2019, pp. 5822–5828.

[22] 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*, 2019, pp. 257–266.

[23] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches", *arXiv preprint arXiv:1409.1259*, 2014.

[24] F. Chung, "Laplacians and the cheeger inequality for directed graphs", *Annals of Combinatorics*, vol. 9, no. 1, pp. 1–19, 2005.

[25] F. R. Chung, "Spectral graph theory", American Mathematical Soc., 1997, vol. 92.

[26] M. Cucuringu, H. Li, H. Sun, and L. Zanetti, "Hermitian matrices for clustering directed graphs: Insights and applications", in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 983–992.

[27] H. Dai, H. Li, T. Tian, X. Huang, L. Wang, J. Zhu, and L. Song, "Adversarial attack on graph structured data", in *International conference on machine learning*, PMLR, 2018, pp. 1115–1124.

[28] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering", in *NIPS*, 2016, pp. 3844–3852.

[29] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings", in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] Y. Ding, L.-P. Tian, X. Lei, B. Liao, and F.-X. Wu, “Variational graph auto-encoders for mirna-disease association prediction”, *Methods*, vol. 192, pp. 25–34, 2021.

[31] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, “Brp-nas: Prediction-based nas using gcns”, *Advances in Neural Information Processing Systems*, vol. 33, pp. 10 480–10 490, 2020.

[32] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints”, *Advances in neural information processing systems*, vol. 28, 2015.

[33] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, “Benchmarking graph neural networks”, *arXiv preprint arXiv:2003.00982*, 2020.

[34] C. Engström and S. Silvestrov, “Pagerank for networks, graphs, and markov chains”, *Theory of Probability and Mathematical Statistics*, vol. 96, pp. 59–82, 2018.

[35] F. Errica, M. Podda, D. Bacciu, and A. Micheli, “A fair comparison of graph neural networks for graph classification”, *arXiv preprint arXiv:1912.09893*, 2019.

[36] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang, and D. Yin, “Graph neural networks for social recommendation”, in *The world wide web conference*, 2019, pp. 417–426.

[37] Q. Feng, E. Dueva, A. Cherkasov, and M. Ester, “Padme: A deep learning-based framework for drug-target interaction prediction”, *arXiv preprint arXiv:1807.09741*, 2018.

[38] M. Fey, “Just jump: Dynamic neighborhood aggregation in graph neural networks”, *arXiv preprint arXiv:1904.04849*, 2019.

[39] M. Fey, J. E. Lenssen, C. Morris, J. Masci, and N. M. Kriege, “Deep graph matching consensus”, *arXiv preprint arXiv:2001.09621*, 2020.

[40] M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch Geometric”, in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[41] H. Gao and S. Ji, “Graph u-nets”, in *international conference on machine learning*, PMLR, 2019, pp. 2083–2092.

[42] X. Geng, X. Wu, L. Zhang, Q. Yang, Y. Liu, and J. Ye, “Multi-modal graph interaction for multi-graph convolution network in urban spatiotemporal forecasting”, *arXiv preprint arXiv:1905.11395*, 2019.

[43] M. Ghorbani, M. S. Baghshah, and H. R. Rabiee, “Mgcn: Semi-supervised classification in multi-layer graphs with graph convolutional networks”, in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2019, pp. 208–211.

[44] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry”, in *International conference on machine learning*, PMLR, 2017, pp. 1263–1272.

[45] D. Gleich, “Hierarchical directed spectral graph partitioning”, *Information Networks*, 2006.

[46] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks”, in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[47] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks”, *arXiv preprint arXiv:1406.2661*, 2014.

[48] M. Granovetter, “The strength of weak ties: A network theory revisited”, *Sociological theory*, pp. 201–233, 1983.

[49] R. Grone and R. Merris, “The laplacian spectrum of a graph ii”, *SIAM Journal on discrete mathematics*, vol. 7, no. 2, pp. 221–229, 1994.

[50] G. H. Gu, J. Noh, S. Kim, S. Back, Z. Ulissi, and Y. Jung, “Practical deep-learning representation for fast heterogeneous catalyst screening”, *The Journal of Physical Chemistry Letters*, vol. 11, no. 9, pp. 3185–3191, 2020.

[51] H. Guo, R. Pasunuru, and M. Bansal, “Autosem: Automatic task selection and mixing in multi-task learning”, *arXiv preprint arXiv:1904.04153*, 2019.

[52] M. Guo, A. Haque, D.-A. Huang, S. Yeung, and L. Fei-Fei, “Dynamic task prioritization for multitask learning”, in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 270–287.

[53] Z. Guo, Y. Zhang, and W. Lu, “Attention guided graph convolutional networks for relation extraction”, *arXiv preprint arXiv:1906.07510*, 2019.

[54] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs”, in *NIPS*, 2017, pp. 1024–1034.

[55] D. K. Hammond, P. Vandergheynst, and R. Gribonval, “Wavelets on graphs via spectral graph theory”, *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[56] A. Hasanzadeh, E. Hajiramezanali, K. Narayanan, N. Duffield, M. Zhou, and X. Qian, “Semi-implicit graph variational auto-encoders”, *Advances in neural information processing systems*, vol. 32, 2019.

[57] K. Hassani and A. H. Khasahmadi, “Contrastive multi-view representation learning on graphs”, *arXiv preprint arXiv:2006.05582*, 2020.

[58] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, “Momentum contrast for unsupervised visual representation learning”, *arXiv preprint arXiv:1911.05722*, 2019.

[59] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask r-cnn”, in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[60] Y. He, G. Reinert, and M. Cucuringu, “Digrac: Digraph clustering based on flow imbalance”, *arXiv preprint arXiv:2106.05194*, 2021.

[61] Y. He, X. Zhang, J. Huang, M. Cucuringu, and G. Reinert, “Pytorch geometric signed directed: A survey and software on graph neural networks for signed and directed graphs”, *arXiv preprint arXiv:2202.10793*, 2022.

[62] S. Heindorf, Y. Scholten, H. Wachsmuth, A.-C. Ngonga Ngomo, and M. Potthast, “Causenet: Towards a causality graph extracted from the web”, in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3023–3030.

[63] M. Henaff, J. Bruna, and Y. LeCun, “Deep convolutional networks on graph-structured data”, *arXiv preprint arXiv:1506.05163*, 2015.

[64] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks”, *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[65] R. Hisano, “Semi-supervised graph embedding approach to dynamic link prediction”, in *International workshop on complex networks*, Springer, 2018, pp. 109–121.

[66] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, “Learning deep representations by mutual information estimation and maximization”, *arXiv preprint arXiv:1808.06670*, 2018.

[67] D. Hong, L. Gao, J. Yao, B. Zhang, A. Plaza, and J. Chanussot, “Graph convolutional networks for hyperspectral image classification”,, 2020.

[68] R. Horaud, “A short tutorial on graph laplacians, laplacian embedding, and spectral clustering”, *URl: http://csustan. csustan. edu/˜ tom/Lecture-Notes/Clustering/GraphLaplacian-tutorial. pdf*, 2009.

[69] Y. Hou, J. Zhang, J. Cheng, K. Ma, R. T. B. Ma, H. Chen, and M.-C. Yang, “Measuring and improving the use of graph information in graph neural networks”, in *ICLR*, 2020.

[70] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, “Open graph benchmark: Datasets for machine learning on graphs”, *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[71] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, “Strategies for pre-training graph neural networks”, *arXiv preprint arXiv:1905.12265*, 2019.

[72] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun, “Gpt-gnn: Generative pre-training of graph neural networks”, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1857–1867.

[73] Z. Hu, C. Fan, T. Chen, K.-W. Chang, and Y. Sun, "Pre-training graph neural networks for generic structural feature extraction", *arXiv preprint arXiv:1905.13728*, 2019.

[74] L. Huang, D. Ma, S. Li, X. Zhang, and H. Wang, "Text level graph neural network for text classification", *arXiv preprint arXiv:1910.02356*, 2019.

[75] W. Huang, T. Zhang, Y. Rong, and J. Huang, "Adaptive sampling towards fast graph representation learning", *Advances in neural information processing systems*, vol. 31, 2018.

[76] V. N. Ioannidis, A. G. Marques, and G. B. Giannakis, "Graph neural networks for predicting protein functions", in *2019 IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, IEEE, 2019, pp. 221–225.

[77] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon, "A survey on contrastive self-supervised learning", *Technologies*, vol. 9, no. 1, p. 2, 2020.

[78] G. Jeh and J. Widom, "Scaling personalized web search", in *Proceedings of the 12th international conference on World Wide Web*, 2003, pp. 271–279.

[79] Q. Ji, E. Bouri, R. Gupta, and D. Roubaud, "Network causality structures among bitcoin and other financial assets: A directed acyclic graph approach", *The Quarterly Review of Economics and Finance*, vol. 70, pp. 203–213, 2018.

[80] S. Ji, Y. Feng, R. Ji, X. Zhao, W. Tang, and Y. Gao, "Dual channel hypergraph collaborative filtering", in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 2020–2029.

[81] W. Jin, T. Derr, H. Liu, Y. Wang, S. Wang, Z. Liu, and J. Tang, "Self-supervised learning on graphs: Deep insights and new direction", *arXiv preprint arXiv:2006.10141*, 2020.

[82] W. Jin, T. Derr, Y. Wang, Y. Ma, Z. Liu, and J. Tang, "Node similarity preserving graph convolutional networks", in *Proceedings of the 14th ACM international conference on web search and data mining*, 2021, pp. 148–156.

[83] N. Jovanović, Z. Meng, L. Faber, and R. Wattenhofer, “Towards robust graph contrastive learning”, *arXiv preprint arXiv:2102.13085*, 2021.

[84] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing, “Rethinking knowledge graph propagation for zero-shot learning”, in *CVPR*, 2019, pp. 11 487–11 496.

[85] J. Kim, T. Kim, S. Kim, and C. D. Yoo, “Edge-labeling graph neural network for few-shot learning”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 11–20.

[86] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization”, *Third ICLR*, 2015.

[87] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, “Neural relational inference for interacting systems”, in *International Conference on Machine Learning*, PMLR, 2018, pp. 2688–2697.

[88] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks”, *arXiv preprint arXiv:1609.02907*, 2016.

[89] J. M. Kleinberg, “Authoritative sources in a hyperlinked environment”, *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.

[90] J. Klicpera, A. Bojchevski, and S. Günnemann, “Predict then propagate: Graph neural networks meet personalized pagerank”, *arXiv preprint arXiv:1810.05997*, 2018.

[91] N. Kriege and P. Mutzel, “Subgraph matching kernels for attributed graphs”, *arXiv preprint arXiv:1206.6483*, 2012.

[92] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, *Advances in neural information processing systems*, vol. 25, 2012.

[93] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning”, *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[94] J. Lee, I. Lee, and J. Kang, “Self-attention graph pooling”, in *International conference on machine learning*, PMLR, 2019, pp. 3734–3743.

[95] B. Li, W. Ye, Z. Sheng, R. Xie, X. Xi, and S. Zhang, "Graph enhanced dual attention network for document-level relation extraction", in *Proceedings of the 28th international conference on computational linguistics*, 2020, pp. 1551–1560.

[96] C. Li, J. Yan, F. Wei, W. Dong, Q. Liu, and H. Zha, "Self-paced multi-task learning", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[97] G. Li, M. Müller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" *arXiv preprint arXiv:1904.03751*, 2019.

[98] J. Li, Y. Rong, H. Cheng, H. Meng, W. Huang, and J. Huang, "Semi-supervised graph classification: A hierarchical graph perspective", in *The World Wide Web Conference*, 2019, pp. 972–982.

[99] J. Li, H. Ma, Z. Zhang, and M. Tomizuka, "Social-wagdat: Interaction-aware trajectory prediction via wasserstein graph double-attention network", *arXiv preprint arXiv:2002.06241*, 2020.

[100] Q. Li, X.-M. Wu, H. Liu, X. Zhang, and Z. Guan, "Label efficient semi-supervised learning via graph filtering", in *CVPR*, 2019, pp. 9582–9591.

[101] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks", in *AAAI*, 2018.

[102] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects", in *International conference on machine learning*, PMLR, 2019, pp. 3835–3845.

[103] R. Liao, Z. Zhao, R. Urtasun, and R. S. Zemel, "Lanczosnet: Multi-scale deep graph convolutional networks", *arXiv preprint arXiv:1901.01484*, 2019.

[104] Z.-H. Lin, S.-Y. Huang, and Y.-C. F. Wang, "Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis", in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1800–1809.

[105] C. Liu and Y. Li, "A parallel pagerank algorithm with power iteration acceleration", *International Journal of Grid and Distributed Computing*, vol. 8, no. 2, pp. 273–284, 2015.

[106] X. Liu, Y. Luo, S. Song, and J. Peng, "Pre-training of graph neural network for modeling effects of mutations on protein-protein binding affinity", *arXiv preprint arXiv:2008.12473*, 2020.

[107] X. Liu, X. You, X. Zhang, J. Wu, and P. Lv, "Tensor graph convolutional networks for text classification", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 2020, pp. 8409–8416.

[108] X. Liu, Y. Liang, Y. Zheng, B. Hooi, and R. Zimmermann, "Spatio-temporal graph contrastive learning", *arXiv preprint arXiv:2108.11873*, 2021.

[109] Y. Liu, M. Jin, S. Pan, C. Zhou, Y. Zheng, F. Xia, and P. Yu, "Graph self-supervised learning: A survey", *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[110] Y. Lu, X. Jiang, Y. Fang, and C. Shi, "Learning to pre-train graph neural networks", in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 4276–4284.

[111] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation", *arXiv preprint arXiv:1508.04025*, 2015.

[112] Y. Ma, S. Wang, C. C. Aggarwal, and J. Tang, "Graph convolutional networks with eigenpooling", in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 723–731.

[113] Y. Ma, J. Hao, Y. Yang, H. Li, J. Jin, and G. Chen, "Spectral-based graph convolutional network for directed graphs", *arXiv preprint arXiv:1907.08990*, 2019.

[114] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne", *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[115] V. Md, S. Misra, G. Ma, R. Mohanty, E. Georganas, A. Heinecke, D. Kalamkar, N. K. Ahmed, and S. Avancha, "Distgnn: Scalable distributed training for large-scale graph neural networks", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–14.

[116] A. Mohamed, K. Qian, M. Elhoseiny, and C. Claudel, “Social-stgcnn: A social spatio-temporal graph convolutional neural network for human trajectory prediction”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 14 424–14 432.

[117] C. B. Moler, “Experiments with MATLAB”, Society for Industrial and Applied Mathematics, 2011.

[118] F. Monti, K. Otness, and M. M. Bronstein, “Motifnet: A motif-based graph convolutional network for directed graphs”, in *IEEE DSW*, IEEE, 2018, pp. 225–228.

[119] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, “Weisfeiler and leman go neural: Higher-order graph neural networks”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 4602–4609.

[120] K. Murugesan and J. Carbonell, “Self-paced multitask learning with shared knowledge”, *arXiv preprint arXiv:1703.00977*, 2017.

[121] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU, “Query-driven active surveying for collective classification”, in *10th International Workshop on Mining and Learning with Graphs*, vol. 8, 2012.

[122] J. R. Norris and J. R. Norris, “Markov chains”, 2. Cambridge university press, 1998.

[123] A. v. d. Oord, Y. Li, and O. Vinyals, “Representation learning with contrastive predictive coding”, *arXiv preprint arXiv:1807.03748*, 2018.

[124] L. Page, S. Brin, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web.” Stanford InfoLab, Tech. Rep., 1999.

[125] A. Palizhati, W. Zhong, K. Tran, S. Back, and Z. W. Ulissi, “Toward predicting intermetallics surface properties with high-throughput dft and convolutional neural networks”, *Journal of chemical information and modeling*, vol. 59, no. 11, pp. 4742–4749, 2019.

[126] W. R. Palmer and T. Zheng, “Spectral clustering for directed networks”, in *International Conference on Complex Networks and Their Applications*, Springer, 2020, pp. 87–99.

[127] S. Pan, J. Wu, X. Zhu, C. Zhang, and Y. Wang, “Tri-party deep network representation”, *Network*, vol. 11, no. 9, p. 12, 2016.

[128] X. Pan and H.-B. Shen, “Inferring disease-associated micrornas using semi-supervised multi-label graph convolutional networks”, *Iscience*, vol. 20, pp. 265–277, 2019.

[129] A. Pareja, G. Domeniconi, J. Chen, T. Ma, T. Suzumura, H. Kanezashi, T. Kaler, and C. E. Leisersen, “Evolvegcn: Evolving graph convolutional networks for dynamic graphs”, *arXiv preprint arXiv:1902.10191*, 2019.

[130] S. Park, W. Lee, B. Choe, and S.-G. Lee, “A survey on personalized pagerank computation algorithms”, *IEEE Access*, vol. 7, pp. 163 049–163 062, 2019.

[131] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, *et al.*, “Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting”, *Information Sciences*, vol. 521, pp. 277–290, 2020.

[132] Z. Peng, Y. Dong, M. Luo, X.-M. Wu, and Q. Zheng, “Self-supervised graph representation learning via global context prediction”, *arXiv preprint arXiv:2003.01604*, 2020.

[133] Z. Peng, W. Huang, M. Luo, Q. Zheng, Y. Rong, T. Xu, and J. Huang, “Graph representation learning via graphical mutual information maximization”, in *Proceedings of The Web Conference 2020*, 2020, pp. 259–270.

[134] A. Pentina, V. Sharmanska, and C. H. Lampert, “Curriculum learning of multiple tasks”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5492–5500.

[135] C. Poignard, T. Pereira, and J. P. Pade, “Spectra of laplacian matrices of weighted graphs: Structural genericity properties”, *SIAM Journal on Applied Mathematics*, vol. 78, no. 1, pp. 372–394, 2018.

[136] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang, “Gcc: Graph contrastive coding for graph neural network pre-training”, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1150–1160.

[137] R. Qiu, H. Yin, Z. Huang, and T. Chen, “Gag: Global attributed graph neural network for streaming session-based recommendation”, in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 669–678.

[138] S. M. Reich, K. Subrahmanyam, and G. Espinoza, “Friending, iming, and hanging out face-to-face: Overlap in adolescents’ online and offline social networks.” *Developmental psychology*, vol. 48, no. 2, p. 356, 2012.

[139] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks”, *Advances in neural information processing systems*, vol. 28, 2015.

[140] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, and F. Monti, “Sign: Scalable inception graph neural networks”, *arXiv preprint arXiv:2004.11198*, 2020.

[141] N. Sarafianos, T. Giannakopoulos, C. Nikou, and I. A. Kakadiaris, “Curriculum learning of visual attribute clusters for multi-task classification”, *Pattern Recognition*, vol. 80, pp. 94–108, 2018.

[142] V. Satuluri and S. Parthasarathy, “Symmetrizations for clustering directed graphs”, in *Proceedings of the 14th International Conference on Extending Database Technology*, 2011, pp. 343–354.

[143] K. Schütt, P.-J. Kindermans, H. E. Sauceda Felix, S. Chmiela, A. Tkatchenko, and K.-R. Müller, “Schnet: A continuous-filter convolutional neural network for modeling quantum interactions”, *Advances in neural information processing systems*, vol. 30, 2017.

[144] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, “Collective classification in network data”, *AI magazine*, vol. 29, no. 3, pp. 93–93, 2008.

[145] O. Shchur, M. Mumme, A. Bojchevski, and S. Günnemann, “Pitfalls of graph neural network evaluation”, *arXiv preprint arXiv:1811.05868*, 2018.

[146] W. Shi and R. Rajkumar, “Point-gnn: Graph neural network for 3d object detection in a point cloud”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1711–1719.

[147] 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.

[148] J. Skarding, B. Gabrys, and K. Musial, “Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey”, *IEEE Access*, vol. 9, pp. 79 143–79 168, 2021.

[149] F.-Y. Sun, J. Hoffmann, V. Verma, and J. Tang, “Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization”, *arXiv preprint arXiv:1908.01000*, 2019.

[150] J. Sun, J. Zhang, Q. Li, X. Yi, Y. Liang, and Y. Zheng, “Predicting city-wide crowd flows in irregular regions using multi-view graph convolutional networks”, *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[151] K. Sun, Z. Lin, and Z. Zhu, “Multi-stage self-supervised learning for graph convolutional networks on graphs with few labeled nodes”, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5892–5899.

[152] L. Sun, Y. Dou, C. Yang, J. Wang, P. S. Yu, L. He, and B. Li, “Adversarial attack and defense on graph data: A survey”, *arXiv preprint arXiv:1812.10528*, 2018.

[153] Q. Sun, J. Li, H. Peng, J. Wu, Y. Ning, P. S. Yu, and L. He, “Sugar: Subgraph neural network with reinforcement pooling and self-supervised mutual information mechanism”, in *Proceedings of the Web Conference 2021*, 2021, pp. 2081–2091.

[154] Q. Sun, H. Peng, J. Li, S. Wang, X. Dong, L. Zhao, S. Y. Philip, and L. He, “Pairwise learning for name disambiguation in large-scale heterogeneous academic networks”, in *2020 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2020, pp. 511–520.

[155] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[156] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding", in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[157] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding", *arXiv preprint arXiv:1906.05849*, 2019.

[158] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning", *arXiv preprint arXiv:2005.10243*, 2020.

[159] **Z. Tong**, Y. Liang, H. Ding, Y. Dai, X. Li, and C. Wang, "Directed graph contrastive learning", *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[160] **Z. Tong**, Y. Liang, C. Sun, X. Li, D. Rosenblum, and A. Lim, "Digraph inception convolutional networks", *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[161] **Z. Tong**, Y. Liang, C. Sun, D. S. Rosenblum, and A. Lim, "Directed graph convolutional network", *arXiv preprint arXiv:2004.13970*, 2020.

[162] M. Tschannen, J. Djolonga, P. K. Rubenstein, S. Gelly, and M. Lucic, "On mutual information maximization for representation learning", in *International Conference on Learning Representations*, 2019.

[163] A. Tsitsulin, J. Palowitch, B. Perozzi, and E. Müller, "Graph clustering with graph neural networks", *arXiv preprint arXiv:2006.16904*, 2020.

[164] W. G. Underwood, A. Elliott, and M. Cucuringu, "Motif-based spectral clustering of weighted directed networks", *Applied Network Science*, vol. 5, no. 1, pp. 1–41, 2020.

[165] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks", *arXiv preprint arXiv:1710.10903*, 2017.

[166] P. Veličković, W. Fedus, W. L. Hamilton, P. Liò, Y. Bengio, and R. D. Hjelm, "Deep graph infomax", *arXiv preprint arXiv:1809.10341*, 2018.

[167] V. Verma, M. Qu, A. Lamb, Y. Bengio, J. Kannala, and J. Tang, “Graphmix: Regularized training of graph neural networks for semi-supervised learning”, *arXiv preprint arXiv:1909.11715*, 2019.

[168] U. Von Luxburg, “A tutorial on spectral clustering”, *Statistics and computing*, vol. 17, pp. 395–416, 2007.

[169] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, “A semi-supervised graph attentive network for financial fraud detection”, in *2019 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2019, pp. 598–607.

[170] H. Wang, Z. Wei, J. Gan, S. Wang, and Z. Huang, “Personalized pagerank to a target node, revisited”, in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 657–667.

[171] H. Wang, F. Zhang, M. Zhang, J. Leskovec, M. Zhao, W. Li, and Z. Wang, “Knowledge-aware graph neural networks with label smoothness regularization for recommender systems”, in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 968–977.

[172] L. Wang, Z.-H. You, Y.-M. Li, K. Zheng, and Y.-A. Huang, “Gcncda: A new method for predicting circrna-disease associations based on graph convolutional network algorithm”, *PLOS Computational Biology*, vol. 16, no. 5, e1007568, 2020.

[173] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma, Z. Huang, Q. Guo, H. Zhang, H. Lin, J. Zhao, J. Li, A. J. Smola, and Z. Zhang, “Deep graph library: Towards efficient and scalable deep learning on graphs”, *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. [Online]. Available: `https://arxiv.org/abs/1909.01315`.

[174] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, “End-to-end text recognition with convolutional neural networks”, in *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, IEEE, 2012, pp. 3304–3308.

[175] X. Wang, X. He, Y. Cao, M. Liu, and T.-S. Chua, "Kgat: Knowledge graph attention network for recommendation", in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 950–958.

[176] X. Wang and A. Gupta, "Videos as space-time region graphs", in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 399–417.

[177] X. Wang, Y. Ye, and A. Gupta, "Zero-shot recognition via semantic embeddings and knowledge graphs", in *CVPR*, 2018, pp. 6857–6866.

[178] Y. Wang, Y. Cai, Y. Liang, H. Ding, C. Wang, S. Bhatia, and B. Hooi, "Adaptive data augmentation on temporal graphs", in *Advances in Neural Information Processing Systems*, 2021.

[179] Y. Wang, S. Liu, M. Yoon, H. Lamba, W. Wang, C. Faloutsos, and B. Hooi, "Provably robust node classification via low-pass message passing", in *2020 IEEE International Conference on Data Mining (ICDM)*, IEEE, 2020, pp. 621–630.

[180] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Graphcrop: Subgraph cropping for graph classification", *arXiv preprint arXiv:2009.10564*, 2020.

[181] Y. Wang, W. Wang, Y. Liang, Y. Cai, and B. Hooi, "Mixup for node and graph classification", in *Proceedings of the Web Conference 2021*, 2021, pp. 3663–3674.

[182] Y. Wang, W. Wang, Y. Liang, Y. Cai, J. Liu, and B. Hooi, "Nodeaug: Semi-supervised node classification with data augmentation", in *KDD*, 2020, pp. 207–217.

[183] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds", *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[184] M. Weber, G. Domeniconi, J. Chen, D. K. I. Weidele, C. Bellei, T. Robinson, and C. E. Leiserson, "Anti-money laundering in bitcoin: Experimenting with graph convolutional networks for financial forensics", *arXiv preprint arXiv:1908.02591*, 2019.

[185] Wikipedia, “Laplacian matrix — Wikipedia, the free encyclopedia”, http://en.wikipedia.org/w/index.php?title=Laplacian%20matrix&oldid=1131319168, [Online; accessed 29-January-2023], 2023.

[186] D. P. Williamson, “Orie 6334 bridging continuous and discrete optimization”, https://people.orie.cornell.edu/dpw/orie6334/lecture7.pdf, [Online; accessed 29-January-2023], 2019.

[187] F. Wu, T. Zhang, A. H. d. Souza Jr, C. Fifty, T. Yu, and K. Q. Weinberger, “Simplifying graph convolutional networks”, *arXiv preprint arXiv:1902.07153*, 2019.

[188] L. Wu, P. Cui, J. Pei, and L. Zhao, “Graph Neural Networks: Foundations, Frontiers, and Applications”, Singapore: Springer Singapore, 2022, p. 725.

[189] M. Wu, C. Zhuang, M. Mosse, D. Yamins, and N. Goodman, “On mutual information in contrastive learning for visual representations”, *arXiv preprint arXiv:2005.13149*, 2020.

[190] S. Wu, F. Sun, W. Zhang, X. Xie, and B. Cui, “Graph neural networks in recommender systems: A survey”, *ACM Computing Surveys (CSUR)*, 2020.

[191] X. Wu, E. Dyer, and B. Neyshabur, “When do curricula work?” *arXiv preprint arXiv:2012.03107*, 2020.

[192] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks”, *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[193] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks”, *arXiv preprint arXiv:1901.00596*, 2019.

[194] F. Xiao, C.-M. Li, M. Luo, F. Manya, Z. Lü, and Y. Li, “A branching heuristic for sat solvers based on complete implication graphs”, *Science China Information Sciences*, vol. 62, no. 7, pp. 1–13, 2019.

[195] T. Xiao, X. Wang, A. A. Efros, and T. Darrell, “What should not be contrastive in contrastive learning”, *arXiv preprint arXiv:2008.05659*, 2020.

[196] G.-S. Xie, J. Liu, H. Xiong, and L. Shao, "Scale-aware graph neural network for few-shot semantic segmentation", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5475–5484.

[197] T. Xie and J. C. Grossman, "Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties", *Physical review letters*, vol. 120, no. 14, p. 145 301, 2018.

[198] Y. Xiong, Y. Zhang, X. Kong, H. Chen, and Y. Zhu, "Graphinception: Convolutional neural networks for collective classification in heterogeneous information networks", *IEEE TKDE*, 2019.

[199] H. Xu, C. Jiang, X. Liang, and Z. Li, "Spatial-aware graph relation network for large-scale object detection", in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[200] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[201] K. Xu, C. Li, Y. Tian, T. Sonobe, K.-i. Kawarabayashi, and S. Jegelka, "Representation learning on graphs with jumping knowledge networks", *arXiv preprint arXiv:1806.03536*, 2018.

[202] P. Yanardag and S. Vishwanathan, "Deep graph kernels", in *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, 2015, pp. 1365–1374.

[203] C. Yang, Z. An, and Y. Xu, "Multi-view contrastive learning for online knowledge distillation", in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2021, pp. 3750–3754.

[204] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification", in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 7370–7377.

[205] C. Ye, R. C. Wilson, C. H. Comin, L. d. F. Costa, and E. R. Hancock, "Approximate von neumann entropy for directed graphs", *Physical Review E*, vol. 89, no. 5, p. 052 804, 2014.

[206] 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 & data mining*, 2018, pp. 974–983.

[207] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen, "Graph contrastive learning with augmentations", *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[208] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting", *arXiv preprint arXiv:1709.04875*, 2017.

[209] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method", *arXiv preprint arXiv:1907.04931*, 2019.

[210] C. Zhang, J. James, and Y. Liu, "Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting", *IEEE Access*, vol. 7, pp. 166 246–166 256, 2019.

[211] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network", in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 793–803.

[212] J. Zhang, X. Shi, J. Xie, H. Ma, I. King, and D.-Y. Yeung, "Gaan: Gated attention networks for learning on large and spatiotemporal graphs", *arXiv preprint arXiv:1803.07294*, 2018.

[213] M. Zhang and Y. Chen, "Link prediction based on graph neural networks", *Advances in neural information processing systems*, vol. 31, 2018.

[214] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: A comprehensive review", *Computational Social Networks*, vol. 6, no. 1, p. 11, 2019.

[215] X.-M. Zhang, L. Liang, L. Liu, and M.-J. Tang, "Graph neural networks and their current applications in bioinformatics", *Frontiers in genetics*, vol. 12, 2021.

[216] X. Zhang, N. Brugnone, M. Perlmutter, and M. Hirn, “Magnet: A magnetic neural network for directed graphs”, *arXiv preprint arXiv:2102.11391*, 2021.

[217] Y. Zhang, X. Yu, Z. Cui, S. Wu, Z. Wen, and L. Wang, “Every document owns its structure: Inductive text classification via graph neural networks”, *arXiv preprint arXiv:2004.13826*, 2020.

[218] J. Zhao, Z. Zhou, Z. Guan, W. Zhao, W. Ning, G. Qiu, and X. He, “Intentgc: A scalable graph convolution framework fusing heterogeneous information for recommendation”, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2347–2357.

[219] J. Zhao, M. Mathieu, and Y. LeCun, “Energy-based generative adversarial network”, *arXiv preprint arXiv:1609.03126*, 2016.

[220] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, “Random erasing data augmentation.” In *AAAI*, 2020, pp. 13 001–13 008.

[221] D. Zhou, T. Hofmann, and B. Schölkopf, “Semi-supervised learning on directed graphs”, in *NIPS*, 2005, pp. 1633–1640.

[222] D. Zhou and B. Schölkopf, “A regularization framework for learning from graph data”, in *ICML 2004 Workshop on Statistical Relational Learning and Its Connections to Other Fields (SRL 2004)*, 2004, pp. 132–137.

[223] 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.

[224] 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*, 2019, pp. 1399–1407.

[225] X. Zhu and A. B. Goldberg, “Introduction to semi-supervised learning”, *Synthesis lectures on artificial intelligence and machine learning*, vol. 3, no. 1, pp. 1–130, 2009.

[226] X. J. Zhu, “Semi-supervised learning literature survey”,, 2005.

[227] Y. Zhu, Y. Xu, Q. Liu, and S. Wu, "An empirical study of graph contrastive learning", 2021. arXiv: `2109.01116 [cs.LG]`.

[228] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Deep graph contrastive representation learning", *arXiv preprint arXiv:2006.04131*, 2020.

[229] Y. Zhu, Y. Xu, F. Yu, Q. Liu, S. Wu, and L. Wang, "Graph contrastive learning with adaptive augmentation", *arXiv preprint arXiv:2010.14945*, 2020.

[230] Y. Zhu, Y. Xu, F. Yu, S. Wu, and L. Wang, "Cagnn: Cluster-aware graph neural networks for unsupervised graph representation learning", *arXiv preprint arXiv:2009.01674*, 2020.

[231] C. Zhuang and Q. Ma, "Dual graph convolutional networks for graph-based semi-supervised classification", in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 499–508.

[232] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data", in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 2847–2856.