UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Graph Convolutional Neural Networks with Negative Sampling

by Wei Duan

Thesis submitted in fulfilment of the requirements for the degree of

Research Master of Science in Computing Science

under the supervision of Junyu Xuan, Jie Lu

University of Technology Sydney
Faculty of Engineering and Information Technology

January 2022

Certificate of Original Authorship

I, Wei Duan declare that this thesis, is submitted in fulfilment of the requirements for the award of master, in the faculty of engineering and information at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signed: Production Note: Signature removed prior to publication.

Date: 16/12/2021

Abstract

Convolutional neural networks (CNNs) can learn potential features from large amounts of Euclidean data, such as text, images, which produce a satisfactory performance on pattern recognition and data mining. Besides Euclidean data, graphs, as non-Euclidean data, are powerful structures for modeling molecules, social networks, citation networks, traffic networks, etc. However, to lend learning power from the Euclidean space to a graph is not trivial. Learning with graphs requires an effective representation of their data structure. Graph Convolutional Neural Networks (GCNs) have been generally accepted to be an effective tool for node representations learning. An interesting way to understand GCNs is to think of them as a message-passing mechanism where each node updates its representation by accepting information from its neighbors (also known as positive samples). However, beyond these neighboring nodes, graphs have a large, dark, all-but-forgotten world in which we find the non-neighboring nodes (negative samples).

In this thesis, we consider that this great dark world holds a substantial amount of information that might be useful for representation learning. Most specifically, it can provide negative information about the node representations. The key is how to select the appropriate negative samples for each node and incorporate the negative information contained in these samples into the representation update. We first propose a generalized method for fusing negative samples to graph convolutional neural networks. We next illustrate that the process of selecting the appropriate negative samples is not trivial. We proposed a determinantal point process algorithm-based method for efficiently obtaining samples. Finally, we use diverse negative samples to boost GCNs which jointly consider the positive and negative information when passing messages. The findings of this study not only have the development of the theory about negative samples in GCNs but also provide new ideas to alleviate the over-smoothing problem.

Acknowledgements

I would firstly like to express my earnest thanks to my principal supervisor Dr. Junyu Xuan, and my co-supervisor Prof. Jie Lu. Their comprehensive guidance has covered all aspects of my master study, including research topic selection, research methodology experiments, academic writing skills and thesis writing, even sentence structure and formulas. Their step-by-step approach taught me how to do scientific research. Their academic rigor and respectful personalities have benefited my master's research and will be a treasure in my life. Without their excellent guidance and constant encouragement, this research would not have been completed a semester early. Once again, I would like to thank my two supervisors for guiding me against numerous difficulties during the special time of the epidemic.

I am grateful to all members of the Decision Systems and e-Service Intelligent (DeSI) Lab in the Australian Artificial Intelligence Institute (AAII) for their careful participation in my presentation and valuable comments on my research. I am especially grateful to Zihe Liu for staying in full communication with me, helping and supporting me with my coursework, papers, and thesis while we were stuck in China together by the epidemic. I would also like to thank Jemima for helping me to proofread my publications.

Last but not least, I would also like to thank my parents for their generous support of my study abroad, both financially and spiritually. I would also like to thank my friends for their support.

Contents

Declaration of Authorship **3**

Abstract **5**

Acknowledgements **7**

List of Figures **11**

List of Tables **13**

1 Introduction **1**
1.1 Background and motivation 1
1.2 Research questions 3
1.3 Research contributions 4
1.4 Research significance 4
1.5 Thesis structure 5
1.6 Publications 5

2 Literature review **7**
2.1 Brief history of graph neural networks 7
2.2 Analytic tasks of graph neural networks 9
2.3 Categories of graph neural networks 9
2.3.1 Recurrent graph neural networks (RecGNNs) 9
2.3.2 Graph convolutional neural networks (GCNs) 10
2.3.2.1 Spectral-based graph neural networks 10
2.3.2.2 Spatial-based graph neural networks 15
2.3.3 Graph autoencoders (GAEs) 17
2.3.4 Spatial-temporal graph neural networks (STGNNs) 17
2.4 Main application areas of graph neural networks 17
2.4.1 Recommender systems 18
2.4.2 Computer vision 18
2.4.3 Natural language processing 19
2.4.4 Physics 20

2.4.5 Chemistry and biology 20
2.4.6 Others 21
2.5 Negative sampling in graph neural networks 21
2.6 Summary 22

3 Graph convolutional neural networks with Monte-Carlo negative sampling 25
3.1 Introduction to negative sampling in Graph convolutional neural networks . 25
3.2 Graph convolutional neural networks with Monte-Carlo negative sampling . 26
3.2.1 Negative sampling 27
3.2.2 Graph convolution with negative sampling 28
3.3 Experiments 30
3.3.1 Datasets 30
3.3.2 Experimental setup 31
3.3.3 Baselines 31
3.3.4 Experimental results 31
3.4 Summary 34

4 Graph convolutional neural networks with DPP negative sampling 37
4.1 Introduction to determinant point process (DPP) 37
4.2 Graph convolutional neural networks with DPP negative sampling 39
4.2.1 DPP-based negative sampling 40
4.2.2 GCN boosted by diverse negative samples 43
4.3 Experiments and results analysis 45
4.3.1 Baselines 45
4.3.2 Setup 46
4.3.3 Metrics 46
4.3.4 Results analysis 46
4.4 Summary 48

5 Conclusion and Further Study 51
5.1 Conclusions 51
5.2 Further study 52

Appendices 53

Bibliography 53

List of Figures

1.1 Illustration of the motivation of this work, including the dark world (gray shadow), different semantic clusters (green, yellow, purple, blue nodes), positive samples (nodes 2, 3, 4), and selected diverse negative samples (nodes 6, 11, and 18) of a given node (node 1). 2

3.1 **Mechanism of the negative sampling graph convolution**. The central node is $v = 5$ and $f(\cdot)$ is graph convolution layer [1]. Node 4, 7 are directly linked with node 5 by real positive edges, thus positive sampling convolution is performed by $x_{pos} = f(4, 7, 5)$. Node 3, 8 are negative sampled using MCNS methods [2], which are based on Markov Chain Monte-Carlo methods and DFS, message passing to central node $v = 5$ along virtual imaginary edges, then negative sampling convolution is performed by $x_{neg} = f(3, 8)$. Given a certain negative rate β, we get negative sampling graph convolution result of this layer, i.e. $x^{'} = x_{pos} - \beta x_{neg}$ 26
3.2 Performance of the three models on the training set in the Cora dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.25$. 32
3.3 Performance of the three models on the training set in the Citeseer dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.25$. 33
3.4 Performance of the three models on the training set in the Pubmed dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.00$. 33
3.5 Performance of a different number of negative sampling nodes on Cora training set using NegGCNs with $\beta = 1.25$. We tested sampling 10%, 30%, 50%, 70%, and 90% of the total negative sampling points to calculate x_{neg}. . . . 34

4.1 The concept of DPP-based negative sampling. The target node is Node 1. Nodes 2, 3 and 4 are positive samples. Nodes 5-18 are the dark world of Node 1. The 4-length DFS path of Node 1 is $\{3, 5, 11, 13\}$, where $\{5, 11, 13\}$ are the central nodes on the path in the dark world. With their first-order neighbouring nodes, they form the candidate set of DPPs, i.e.$\{5, 6, 7, 11, 12, 13, 14, 18\}$. The selected negative samples from this set are 6, 11, and 18, which can be seen as virtual negative links to Node 1. 42

4.2 The Accuracy and MAD of five models on three datasets with a varying number of layers from 2 to 6. The x-axis denotes the layer number, and the mean and standard deviation of 10 runs are given for each model with each layer number. 47
4.3 The Accuracy and MAD of D2GCN with 5 layers on Citeseer datasets in the varying length of DFS and scale of negative rate. The mean and standard deviation of 10 runs are given for each setting. 47

List of Tables

3.1 Statistics of the dataset 30
3.2 Experimental results 32

Chapter 1

Introduction

1.1 Background and motivation

Convolutional neural networks (CNNs) [3] can learn potential features from large amounts of Euclidean data, such as text [4], images [5], music [6], and video [7], which produce a satisfactory performance on pattern recognition and data mining. Besides Euclidean data, graphs, as non-Euclidean data, are powerful structures for modeling molecules, social networks, citation networks, traffic networks, etc. [8]. However, the representation power of graphs is not a free lunch; it brings with its issues of incompatibility with some of the strongest and most popular deep learning algorithms, as these can often only handle regular data structures like vectors or arrays. Hence, to lend learning power to these great tools of representation, graph neural networks (GNNs) have emerged as a congruent deep learning architecture [9]. Such has been their success that, today, there are many different GNN variants, each adapted for a specific task – for instance, graph sequence neural networks [10], graph convolutional neural networks [1], and spatio-temporal graph convolutional networks [11].

Among all the varieties of GNNs, graph convolutional neural networks (GCN) [1] is a simple but representative and salient one, which introduces the concept of convolution to GNNs that means to share weights for nodes within a layer. An easy and intuitive way to understand GCNs is to think of them as a message passing mechanism [12], where

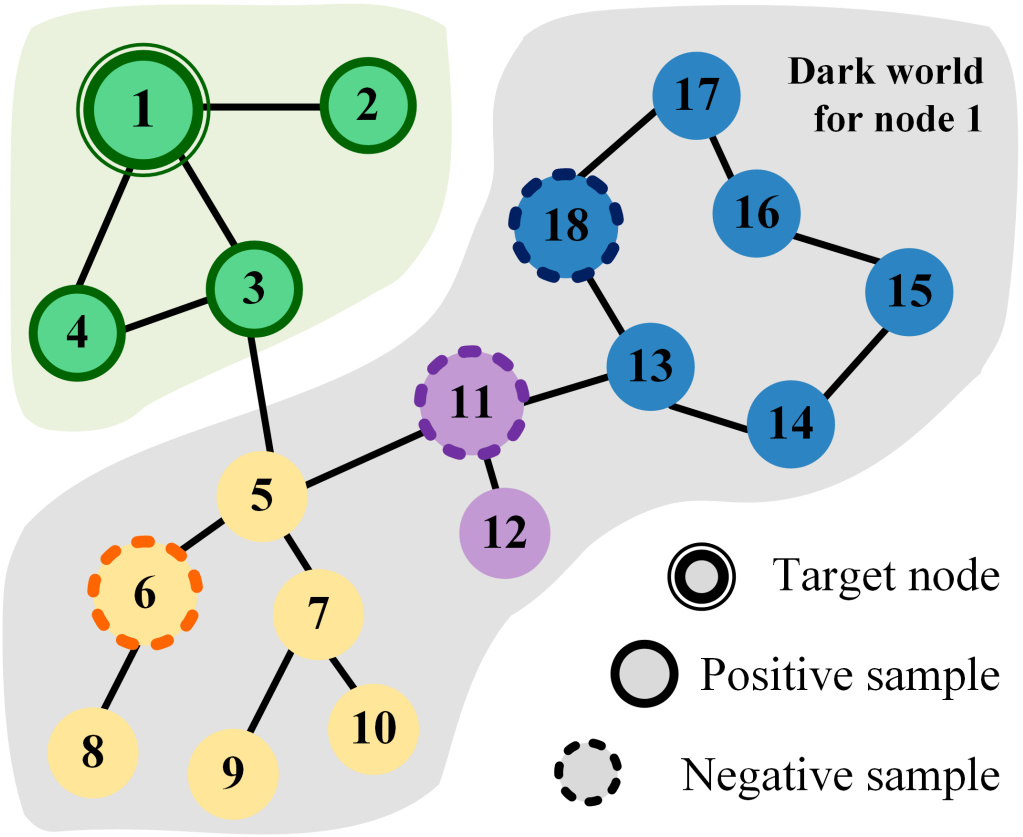

FIGURE 1.1: Illustration of the motivation of this work, including the dark world (gray shadow), different semantic clusters (green, yellow, purple, blue nodes), positive samples (nodes 2, 3, 4), and selected diverse negative samples (nodes 6, 11, and 18) of a given node (node 1).

each node accepts information from its neighboring nodes to update its representation. The basic idea is that the representations of nodes with edge between them should be positively correlated. Hence, the neighboring nodes are also named as positive samples[1]. This message-passing mechanism is highly effective in many scenarios, but it does lead to an annoying over-smoothing problem where the node representations become more and more similar as the number of layers of the graph neural network increases. This is hardly surprising when each node only updates its representation according to its neighbours.

Beyond these observed edges, there is a dark world that could provide diverse and useful information to the representation updates and help to overcome the over-smoothing problem at the same time, as illustrated in Figure 1.1. The reasoning is this: two nodes without edge between them should have different representations, so we can understand

[1]Hereafter we refer to neighbouring nodes and positive samples interchangeably.

this as a negative link. However, in contrast to the commonly used positive links, these negative links are rarely used in the existing various GCNs.

Negative sampling was first used in natural language processing to facilitate the training of word2vec [13] as a simpler form of noise contrast estimates [14]. In the GNNs domain, selecting appropriate negative samples in a graph is not trivial. The first is Kim and Oh [1], who propose the natural and easy approach of uniformly randomly selecting some negative samples from non-neighbouring nodes. However, as we all know, a large graph normally has the small-world property [15], which means the graph will tend to contain some clusters. Moreover, nodes in the same cluster will tend to have similar representations while nodes within different clusters will tend to have different representations, as illustrated in Figure 1.1. Hence, under uniform random selection, the large clusters will overwhelm the smaller ones, and the final converged node representations will be short on information contained in the small clusters. Of the other two methods, one is based on Monte Carlo chains [2] and the other on personalised PageRank [16], and neither covers diverse information as well. Further, the selected negative samples should not be redundant; each of them should not be overlapping and hold distinct information. Hence, as a definition, a good negative sample should *contribute negative information to the given node contrast to its positive samples and include as much information as possible to reflect the variety of the dark world.* Moreover, these three approaches only fuse negative sampling into the loss function to train the model rather than design a new graph convolution layer.

1.2 Research questions

This research will answer the following research questions:

- How to fuse negative samples into the graph convolution operation with a suitable method?
- How to perform negative sampling to obtain samples with different semantics while containing complete knowledge of the entire graph that can better reflect the real-world situation?

1.3 Research contributions

The main contributions of this study are concisely summarised as follows:

- The negative samples are first directly fused into the graph convolution operation (refer to Chapter 3).
- A new negative sampling method is proposed based on a Determinant Point Process(DPP) that selects diverse negative samples to better represent the dark information (refer to Chapter 4).
- Using the sampling method proposed in this thesis, fusing negative samples into GCNs can improve the quality of node representation and alleviate the over-smoothing problem (refer to Chapter 4).

1.4 Research significance

The theoretical and practical significance of this research is summarized as follows:

Theoretical significance: The findings of this study contribute to the graph convolutional neural network in the following two ways: presenting a description of good negative samples in GCNs, and enriching the theory of graph convolution operations. More specifically, a good negative sample should contribute negative information to the given node in contrast to its positive samples and include as much information as possible to reflect the variety of the dark world. Meanwhile, the proposed ideas on graph convolution operations incorporate both positive neighboring information and negative diverse information to better reflect the real world.

Practical significance: Given the important role of graph data in modern life, the results of this study contribute to the benefit of society. These findings can help solve prediction, classification, and other problems related to graph data. Technology companies can discover relationships between users in social networks to facilitate ad push. Pharmaceutical companies can learn the structure of networks between molecules to facilitate new drug

development. There is potential for many other such applications that could benefit from this study.

1.5 Thesis structure

To present our work in detail, the chapters are organized as follows:

- In Chapter 2, we briefly review the history of the development of graph convolutional networks, the classification and application of graph convolutional networks. In addition to this, we review several existing methods that use negative sampling in graph convolutional networks.
- In Chapter 3, we propose a Negative Samples-enhanced Graph Convolutional Neural Networks (NegGCNs), where the negatively sampled nodes are directly incorporated into the message passing mechanism and used to generate new node feature vectors. This is the basic idea of this thesis, and the key to achieve this goal is to find the appropriate negative samples. The first method we propose for negative sampling is based on Monte-Carlo chains.
- In Chapter 4, we propose a determinantal point process algorithm for getting samples with different semantics while containing complete knowledge of the whole graph. Meanwhile, we compare several existing negative sample sampling methods to test whether they can obtain satisfactory samples. Experimental evaluations show that our idea not only improves the overall performance of standard representation learning but also significantly alleviates over-smoothing problems.
- In Chapter 5, we summarise the findings of the thesis and point to directions for future work.

1.6 Publications

Below is a list of the refereed international journal and conference papers during my master research that has been published:

- **Wei Duan**, Junyu Xuan, Jie Lu, *Negative samples-enhanced graph convolutional neural networks*, International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Chengdu, China, November 26-28, 2021. (ERA rank: B)
- **Wei Duan**, Junyu Xuan, Maoying Qiao, Jie Lu, *Learning from the Dark: Boosting Graph Convolutional Neural Networks with Diverse Negative Samples*, In Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI). Virtual event, February 22-March 1, 2022. (ERA rank: A)

Chapter 2

Literature review

Work on pattern recognition and deep learning has recently encouraged the success of neural networks. Due to the computational strength of modern computers based on GPUs, convolutional neural networks (CNNs) [3] can learn potential features from substantial amounts of Euclidean data, such as text, images, music, videos. Thanks to this, CNNs can produce satisfactory performance in a number of tasks, such as target detection [17, 18], natural language processing [19, 20], image segmentation [21, 22], etc.

Graphs, as non-Euclidean data, are powerful structures for modeling specific kinds of data such as molecules, social networks, citation networks, traffic networks, etc. [8]. By the nature of non-Euclidean data, familiar properties such as common coordinate systems and vector space structures do not exist. It brings with it issues of incompatibility with some of the strongest and most popular deep learning algorithms. Hence, to lend learning power to these great tools of representation, graph neural networks (GNNs) have emerged as a congruent deep learning architecture.

2.1 Brief history of graph neural networks

In 1997, Sperduti et al. [23] showed that neural networks can display organized patterns and categorize acyclic graphs, who was the first person applying neural networks to them. The main ideas are to use the so-called "generalized recursive neuron", widespread use

of recurrent neuron architectures. In 2005, Gori et al. [24] first suggested the idea of graph neural networks. They showed that GNNs expand recursive neural networks, so they can process a wider class of graphs and use them to address node-focused issues. In 2009, the definition of GNN was further enriched by Scarselli et al. [25]. The GNN has been demonstrated as both an extension of recurrent neural and random walking networks that not only maintain their features but also are appropriate for graph and node-focused applications. These early studies are in the class of recurring graphical neural networks (RecNNs), which have been able to solve graphical interpretation problems by spreading neighboring information iteratively until a stable point has been reached. This is an extremely costly procedure that has recently been overcome by an increasing number of people. Li et al. [10] modified GNN in order to use gated recurring units and modern techniques of optimisation and then extended it to output sequences which achieved the most advanced performance with a program verification problem.

Many approaches have been created to redefine the concept of convolution graph data, motivated by the achievements of CNNs in computer vision, which have the capacity to extract and produce highly articulate representations of multi-scale localized spatial features. Graph Convolutional neural networks (GCNs) are classified into two types, approaches derived from the spectral domain and from the spatial domain. Bruna et al. [26] published one of the most influential studies on spectral-based GCNs and a graph convolution based on spectral graph theory was created for the first time. Although their paper was conceptually important, it had major computational flaws that prevented it from being a genuinely useful tool. Since then, a growing number of enhancements, extensions, and approximations of GCNs have been made to overcome these flaws.

In addition to RecNNs and GCNs, a wide variety of modifications of GNNs have been proposed for a specific task for instance, including graph auto-encoders (GAEs) [27] and spatial-temporal graph neural networks (STGNNs) [28].

2.2 Analytic tasks of graph neural networks

Node-level [1, 29–32]. GCNs can achieve classification prediction of nodes in an end-to-end manner. The dataset, in this case, generally consists of one or several large graphs, such as social networks, which generally have thousands of nodes. The graph convolution layer generates a vector formulation per node in the graph by associating the characteristic messages of its residents. After feature coalescence, the resultant output is nonlinearly transformed using an activation function.

Edge-level [33, 34]. Graph edges are often in possession of rich information like strengths, types, etc. Utilizing every two-node features in the network, GCNs are able to perform edge classification when there are edges between nodes and edge strength prediction when there are not.

Graph-level [35–37]. GCNs can classify every graph. The dataset, in this case, generally consists of a great number of small graphs, such as molecules, which generally have ten to one hundred nodes. Unlike the node-level, in order to obtain a compressed representation on the graph level, the graph-level GCNs usually requires a readout operation after the last layer of convolution operations.

2.3 Categories of graph neural networks

2.3.1 Recurrent graph neural networks (RecGNNs)

Recurrent graph neural networks (RecGNNs) are mostly the pioneer of GNNs. They repeatedly apply the same set of parameters on the nodes in the graph to extract high-level node representations. Scarselli et al. [25] proposed the RecGNNs model, which can directly process most of the practically useful types of graphs, e.g., acyclic, cyclic, directed, and undirected. This model updates the state of the nodes by repeatedly exchanging neighborhood information until reaching a stable equilibrium. The recursive update process for

the state of a node can be written as:

$$h_v^{(t)} = \sum_{u \in N(v)} f(x_v, e_{(v,u)}, x_u, h_u^{(t-1)}), \tag{2.1}$$

where $h_v^{(t)}$ is the hidden state of node v at t^{th} time, x_v, x_u are vectors of node feature,$e_{(v,u)}$ is edge feature , $f()$ is a parametric function, and $h_v^{(0)}$ is initialized randomly. In follow-up works, Graph Echo State Network [38] extends echo state networks to improve the training efficiency of RecGNNs. RecGNNs are conceptually important, and its idea of message passing inspired the later research on graph convolutional neural networks.

2.3.2 Graph convolutional neural networks (GCNs)

GCNs introduces the concept of convolution to GNNs, which means to share weights between nodes within a layer. Unlike RecGNNs, GCNs stack multiple graph convolution layers to extract higher node representations. GCNs play a central role in building up many other complex GNN models [9].

2.3.2.1 Spectral-based graph neural networks

For the spectral-based approaches, based on the work of Bruna et al. [26], Kipf & Welling [1] proposed GCNs, which is a localized first-order approximation of spectral graph convolutions as a generalized method for semi-supervised learning on graph-structured data. Their model acquires implicit representations, encodes the region graph structure and node attributes, and expands linearly in terms of the number of graph edges.

First, the spectral convolution on the graph can be defined as

$$g_\theta * x = U g_\theta U^\top x, \tag{2.2}$$

where $x \in \mathbb{R}^N$ is a signal, g_θ is a filter parameterized by $\theta \in \mathbb{R}^N$in the Fourier domain, U is the eigenvector matrix of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, with a diagonal matrix of its eigenvalues Λ and $U^\top x$ being the graph Fourier transform of x.To

reduce the computational cost in Eq.(2.2), $g_\theta(\Lambda)$ can be approximated as

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{\Lambda}), \tag{2.3}$$

where $T_k(x)$ is a truncated expansion in terms of Chebyshev polynomials up to K^{th} order, and $\tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$. λ_{max} is the largest eigenvalue of L. Put Eq.(2.3) into Eq.(2.2), we have

$$g_{\theta'} * x \approx \sum_{k=0}^{K} \theta'_k T_k(\tilde{L})x \tag{2.4}$$

with $\tilde{L} = \frac{2}{\lambda_{max}}L - I_N$.

Next, by using Chebyshev polynomials $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$, $K = 1$ was limited in Eq.(2.4) to ensure the existence of a linear function in the spectral space. To facilitate the training of neural networks, the authors make further approximations by Editor mode. By setting $\lambda_{max} \approx 2$, so that Eq.(2.4) becomes

$$g_{\theta'} * x \approx \theta'_0 x + \theta'_1 (L - I_N)x = \theta'_0 x - \theta'_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x. \tag{2.5}$$

Since the two parameters θ'_0 and θ'_1 can be trained together by the neural network, so set $\theta = \theta'_0 = -\theta'_1$ Eq.(2.5) can be written as

$$g_{\theta'} * x \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x. \tag{2.6}$$

When used in a deep neural network model, repeated application of this operator can trigger in value instabilities and exploding/vanishing gradients. In order to solve this problem, the following re-normalization method was proposed: $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow^{-\frac{1}{2}} \tilde{A}\tilde{D}^{-\frac{1}{2}}$ with $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$.

Finally, the definition of a layer in GCNs can be defined as:

$$X^{(k)} = \mathrm{ReLU}\Big(\hat{A}X^{(k-1)}W^{(k-1)}\Big), \tag{2.7}$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, $\tilde{A} = A + I_N$ and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. The matrix form can be written as the following vector form

$$x_i{}^{(k)} = \sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \left(\Theta^{(k)} \cdot x_j^{(k-1)}\right), \tag{2.8}$$

where $\deg(j)$ denotes the degree of node i and $x_i{}^{(k)}$ denotes the feature vector of the k^{th} iteration of node i and $\Theta^{(k)}$ is a trainable weight.

Their model acquires implicit representations, encodes the region graph structure and node attributes, and expands linearly in terms of the number of graph edges.

One special kind of Spectral-based graph neural network proposed recently is Bayesian-based graph neural networks. The Bayesian approach considers the ambiguity associated with the configuration of the underlying graph, which is frequently obtained from noisy data or erroneous modeling assumptions. The Bayesian approach seeks to catch GNNs model uncertainty by putting previous distributions over model parameters so that further changes are possible during the GNNs training course.

- **Bayesian graph convolutional neural networks using node copying**

Since graphs are often obtained from noisy data or erroneous modeling assumptions, bogus edges may exist or edges among very close nodes may be missed, resulting in a decrease in the efficiency of learning algorithms [30]. Several current methods, such as the graph focus attention [39] and the graph ensemble-based approach [40], address this concern in part. Nonetheless, none of these approaches is flexible enough to incorporate edges that might be absent from the observed graph. Pal et al. [30] present a novel generative model for graphs that is focused on copying nodes from one position to another and that copies existing nodes rather than adding new ones.

- **Bayesian Graph Neural Networks with Adaptive Connection Sampling**

GNN implementations in use today do not have uncertainty quantification of performance forecasts. Moreover, empirical findings reveal that, since GNNs have Laplacian smoothing, graph convolutions have an over-smoothing tendency of integrating representations of

adjacent nodes, such that as the number of GNN layers grows, all node representations aggregate to a fixed level, making them irrelevant to node features.

In 2020, Hasanzadeh et al. [31] proposed a coherent paradigm for adaptive relation sampling that summarizes available stochastic regularization approaches for training GNNs, not only alleviating its over smoothing and over-fitting tendencies, but also enabling learning under ambiguity in graph analysis tasks with GNNs. Their adaptive relation sampling can be trained in conjunction with the parameters of a GNNs model, which has been seen to be mathematically similar to training Bayesian GNNs with an effective approximation.

Dropout [41] is a method to prevent overfitting which is commonly used to train deep learning models. DropOut in a GNN layer eliminates output elements of the previously hidden layer at random using separate Bernoulli random draws with a steady performance rate at each training iteration. Based on Eq. 10, DropOut can be formulated as

$$X^{(k+1)} = \mathrm{ReLU}\big(\hat{A}(Z^{(k)} \odot X^{(k)})W^{(k)}\big) \tag{2.9}$$

where $\odot$ represents the element-wise product and $Z^{(k)}$ is a random binary matrix of the same dimensions as $X^{(k)}$, with components that are $Bernoulli(\pi)$ samples.

DropEdge [42] eliminates a specified amount of edges from the input graph at random during and training epoch, serving as both a data enhancer and a message forwarding decelerator. DropEdge in a GNN layer can be formulated as

$$X^{(k+1)} = \mathrm{ReLU}\big((\hat{A} \odot \hat{Z}^{(k)})X^{(k)}W^{(k)}\big) \tag{2.10}$$

where $\hat{A} \odot \hat{Z}^{(k)}$ is normalized $(A \odot Z^{(k)})$.

It is time and memory-consuming training for big and dense graphs when performing recursive neighborhood expansion. To overcome this difficulty, Chen et al. [32] suggested FastGCN with **Node Sampling** to relax the condition of test data availability at the same time which can be formulated as

$$X^{(k+1)} = \mathrm{ReLU}\big(\hat{A}\mathrm{diag}(z^{(k)})X^{(k)}W^{(k)}\big) \tag{2.11}$$

where $z^{(k)}$ is a random vector of elements derived from $Bernoulli(\pi)$.

Combining the work of the three aforementioned, Hasanzadeh et al. [31] used **Graph DropConnect (GDC)**, a general stochastic regularization technique in GNN, to calculate different random masks independently for every channel and edge, which can be considered as an approximation of Bayesian GNNs. The GNN layer with GDC can be formulated as

$$X^{(k+1)} = \mathrm{ReLU}\Big(\sum_{i=1}^{f_k}(\hat{A} \odot \hat{Z}_i^{(k)})X^{(k)}[:,i]W^{(k)}[i,:]\Big) \tag{2.12}$$

where $\hat{Z}_i^{(k)}$ is a sparse random matrix whose non-zero elements are randomly calculated by $Bernoulli(\pi)$.

GDC as Bayesian Approximation. The output feature space of GDC reference sampling can be translated to the parameter space, enabling it to be regarded as reasonable Bayesian extensions of GNNs. To begin with, Eq.(2.12) can be wiritten in a node-wise form of a GNN layer with GDC as:

$$\boldsymbol{x}^{(k+1)} = \mathrm{ReLU}\Big(\frac{1}{c_v}(\sum_{u\in\hat{N}(v)} \boldsymbol{z}_{vu}^{(k)} \odot \boldsymbol{x}_u^{(k)})W^{(k)}\Big) \tag{2.13}$$

where c_v is a node grade-derived constant and mask row vector $\boldsymbol{z}_{vu}^{(k)}$ is the vector of three-dimensional tensor $[Z_1^{(k)}, ..., Z_{f_k}^{(k)}]$ relation between nodes v and u. The constant c_v in the rest of this section is neglected for brevity and without lack of generality. Then Eq.(2.13) can be reorganized to transform the randomness from sampling to the parameter space as:

$$\begin{aligned}\boldsymbol{x}^{(k+1)} &= \mathrm{ReLU}\Big((\sum_{u\in\hat{N}(v)} \boldsymbol{x}_u^{(k)}\mathrm{diag}(\boldsymbol{z}_{vu}^{(k)}))W^{(k)}\Big) \\ &= \mathrm{ReLU}\Big(\sum_{u\in\hat{N}(v)} \boldsymbol{x}_u^{(k)}\big(\mathrm{diag}(\boldsymbol{z}_{vu}^{(k)})W^{(k)}\big)\Big)\end{aligned} \tag{2.14}$$

Define $\boldsymbol{W}_{vu}^{(k)} = \boldsymbol{z}_{vu}^{(k)}W^{(k)}$ to have:

$$\boldsymbol{x}^{(k+1)} = \mathrm{ReLU}\Big(\sum_{u\in\hat{N}(v)} \boldsymbol{x}_u^{(k)}\boldsymbol{W}_{vu}^{(k)}\Big). \tag{2.15}$$

$\boldsymbol{W}_{vu}^{(k)}$ is the weight of edges in the graph and the operation in Eq.(2.16) can be seen as learning suitable weights along edge $e = (u, v) \in E$ for each of message passing.

From a Bayesian perspective, learning the parameters of a neural network can be viewed as finding the posterior probability of a particular parameter given the node data X and adjacency matrix A of the graph, which can be formalized as finding $P(\omega \mid A, X)$. Following the variational interpretation in Gal et al. [43], due to the posterior probability is intractable, we define a new distribution $q(\omega)$, generally a Gaussian distribution, to go infinitely close to $P(\omega \mid A, X)$, which can be achieved by computing the Kullback–Leibler (KL) divergence $\mathrm{KL}(q_\theta(\omega)||p(\omega))$ of the two distributions. Then, the loss function can be written as:

$$\begin{aligned} Loss =& E_{q(\{\omega^k, Z^{(k)}\}_{k=1}^K)}\Big[\log P(Y_o \mid X, \{\omega^k, Z^k\}_{k=1}^K)\Big] \\ &- \sum_{k=1}^{K} \mathrm{KL}\Big(q(\omega^k, Z^k) \parallel p(\omega^k, Z^k)\Big) \end{aligned} \tag{2.16}$$

where Y_o is labeled nodes. The experimental findings of the authors indicate that GDC improves the efficiency of GNNs in semi-supervised classification tasks by reducing over-smoothing and overfitting.

2.3.2.2 Spatial-based graph neural networks

A representative work in a spatial way is GraphSAGE [29], which is a general framework for generating node embedding by sampling and aggregating features from the neighborhood of a node. The vector form of each layer of the graph convolution operation defined by GraphSAGE is as follows

$$\begin{aligned} x_{\mathcal{N}(v)}^{(k)} &= \mathrm{AGGREGATE}^{(k)}(\{x_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}) \\ x_v^{(k)} &= \mathrm{ReLU}\big(W^{(k)} \cdot \mathrm{CONCAT}(x_v^{(k-1)}, x_{\mathcal{N}(v)}^{(k)})\big) \end{aligned} \tag{2.17}$$

A significant difference between GCNs and GraphSAGE is that the previous layer representation of the node is concatenated with the aggregated neighborhood vector by the convolutional aggregator. In addition to the different concentration methods, GraphSAGE

has also made changes in the way information is aggregated. GCNs use the average aggregator, while GraphSAGE uses the max aggregator which was defined as

$$\text{AGGREGATE}_k = \max(\{\text{ReLU}(W x_{u_i}^k + b), \forall u_i \in \mathcal{N}(v)\}) \tag{2.18}$$

where max denotes the element-wise max operator. The model effectively captures various aspects of the neighborhood set after introducing the max-pooling operator to every computed function. However, the authors did not find a distinct difference between the maximum aggregator and the mean aggregator in the developmental test. Therefore, which factors affect the prediction effect of GNN remains to be further investigated.

In order to analyze the representational power of GNNs theoretically, Xu et al. [44] formally characterize how expressive different GNN variants are in learning to represent different graph structures based on the graph isomorphism test [45]. They proposed that since modern GNNs follow a neighborhood aggregation strategy, the network at the k^{th} layer can be summarized formally in two steps AGGREGATE and COMBINE:

$$\begin{aligned} a_v^{(k)} &= \text{AGGREGATE}^{(k)}(\{x_u^{(k-1)}, \forall u \in \mathcal{N}(v)\}) \\ x_v^k &= \text{COMBINE}^{(k)}\big(x_v^{(k-1)}, a_v^{(k)})\big), \end{aligned} \tag{2.19}$$

where $x_v^{(k)}$ is the feature vector of node v in k^{th} layer. In this way, the AGGREGATE and COMBINE in Eq. (2.7) can be defined as

$$x_v^{(k)} = \text{ReLU}\Big(W \cdot \text{MEAN}\Big\{x_u^{(k-1)}, \forall u \in \mathcal{N}(v) \cup \{v\}\Big\}\Big), \tag{2.20}$$

where W is a learnable weight matrix. AGGREGATE in GraphSAGE [29] can be defined as

$$a_v^{(k)} = \max\left(\left\{\text{ReLU}\left(W \cdot x_u^{(k-1)}\right), \forall u \in \mathcal{N}(v)\right\}\right), \tag{2.21}$$

where max denoted an element-wise max-pooling. In addition, they further proposed GINs [44]:

$$x_v^{(k)} = \text{MLP}^{(k)}\Big(\big(1 + \epsilon^k\big) \cdot x_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} x_u^{(k-1)}\Big), \tag{2.22}$$

which can distinguish different graph structures and capture dependencies between graph structures to achieve better classification results.

2.3.3 Graph autoencoders (GAEs)

Graph autoencoders (GAEs) encode nodes/graphs into a low-dimensional vector representation and reconstruct graph data from the encoded information for graph analytics. Cao et al. [46] adopted a random surfing model to capture graph structural information directly to generate a low-dimensional vector representation for each vertex by capturing the graph structural information . In order to let the latent representation match a prior distribution, Pan et al. [47] proposed two variants of adversarial approaches, adversarially regularized graph autoencoder and adversarially regularized variational graph autoencoder.

2.3.4 Spatial-temporal graph neural networks (STGNNs)

Spatial-temporal graph neural networks aim to learn hidden patterns from spatial-temporal graphs which consider spatial dependency and temporal dependency simultaneously. Many current methods incorporate graphical convolution with RNNs or CNNs to capture temporal dependence. For traffic speed forecasting, Li et al. [48] proposed to model the traffic flow as a diffusion process on a directed graph and introduce Diffusion Convolutional Recurrent Neural Network. For human action recognition, Yan et al. [49] proposed a novel model of dynamic skeletons called Spatial-Temporal Graph Convolutional Networks, which moves beyond the limitations of previous methods by automatically learning both the spatial and temporal patterns from data.

2.4 Main application areas of graph neural networks

GNNs have a vast range of uses since graph-structured data are omnipresent. In this section, we summarize the applications of GNNs in different fields.

2.4.1 Recommender systems

GNNs, which can naturally combine node knowledge and topology, have been shown to be effective in studying graphical data in recent years. Since data can be shown as user social graphics and user object graphs in recommending systems, these advantages of GNNs provide tremendous potential for furthering recommendations. Ying et al. [16] built and implemented a massive in-depth recommendation engine at Pinterest that incorporates efficient random wandering and graph convolution to produce embeddings of nodes (i.e., objects) that contain graph structure and node function information, in order to extend GNNs literally to recommender systems and scale to networks of billions of items and trillions of users.

Although GNNs offer great potential for advancing social recommender systems, Fan et al. [50] suggest that they still face problems such as user-item graphs encoding interactions and their associated opinions, social relationships having different advantages, and users involving two graphs. To address all three challenges simultaneously, they propose a principled approach to collectively capture interactions and perspectives in user project graphs and use the GraphRec framework, which models the strength of the two graphs and heterogeneity in a coherent way.

There is more research on GNN in recommender systems, such as Memory Augmented Graph Neural Networks [51], Revisiting Graph-based Collaborative Filtering [52], Inductive Matrix Completion Based on GNNs [53].

2.4.2 Computer vision

The first use of GNN in the Computer Vision field was by Monfardini et al. in 2006 [54]. They suggest a learning method for dealing with complex data resulting from image segmentation without relying on previous knowledge of the dataset. In recent years, GNN applications in computer vision include scene graph generation, action recognition, and classification of point clouds.

Scene Graph Generation Understanding a visual scene entails more than just identifying particular objects. The relationships between objects often provide a wealth of

semantic data about the scene. Xu et al. [55] used scene graphs to explicitly model objects and their relationships, a visual-based graphical structure of images, to generate such structured representations of scenes from input images. Li et al. [56] suggested a subgraph-based connectivity graph to concisely depict the scenario graph during inference to increase the efficiency of scenario graph creation.

Action Recognition The identification of human behavior in images helps to explain the video content of a computer. Jain et al. [57] propose a general, principled approach that successfully combines upper-level Spatio-temporal maps with sequence modeling of RNNs. Their work provides significant improvements on three different Spatio-temporal problems, including. (i) human motion modeling; (ii) human-object interaction; and (iii) driver mobility prediction. Yan et al. [49] suggested a new model called Spatial-Temporal Graph Convolutional Networks dynamic skeletons, which goes upon the shortcomings of past models by learning spatial and temporal features of data automatically which contributes not only to more verbal power but also to greater capacity to generalize.

Point Clouds Classification Point clouds provide a dynamic geometric representation for a myriad of applications in computer graphics, for example, point clouds can represent objects seen around you using LIDAR. Wang et al. [58] proposed a new neural network module called EdgeConv for advanced CNN-based tasks on point clouds, including classification and segmentation. Landrieu et al. [59] proposed a novel deep learning-based framework to efficiently capture the organization of 3D point clouds using the structure of a super-point graph to address the semantic segmentation challenge of massive point clouds with millions of points.

2.4.3 Natural language processing

The issue with the generation of Abstract Meaning Representation(AMR) to text is the recovery of a text with the same value as an AMR input. Song et al. [60] presented a neural graph-to-sequence model that employs a novel LSTM structure to encode graph-level semantics directly. Bastings et al. [61] present a straightforward and efficient method for integrating the syntactic structure into a neural attention-based encoder-decoder paradigm for machine translation. Beck et al. [62] proposed a novel model for encoding the total

structural information embedded in the graph, solving the parameter bombing problem that existed in earlier work. Their model outperforms a robust baseline in generating AMR graphs and grammar-based neural machine translation.

2.4.4 Physics

Our world can be succinctly described as a structured scene of objects and relationships. For example, a living room contains salient objects such as a TV, a sofa, and a coffee table. These objects are often related to each other by their underlying causes and semantics, such as location, function, and shape. People use knowledge of objects and their relationships to learn a variety of tasks. Raposo et al. [63] propose relational networks (RNs) - a general neural network architecture for object-relational reasoning that learns object relationships from scene description data. There is also a lot of work that has investigated the content related to relational reasoning [64, 65].

2.4.5 Chemistry and biology

In the field of biology and chemistry, the generation of corresponding eigenvectors based on molecular structures helps in their further study. Molecular fingerprints follow the route of encoding information and can capture a large amount of the local structural information that is present in a molecule [66]. Duvenaud et al. [67] in 2015 performed an end-to-end convolution operation on molecular graphs of arbitrary size and shape to implement a standard molecular feature extraction method based on molecular fingerprints. This approach demonstrates that data-driven features are easier to interpret and have better predictive performance in a variety of tasks.

The mainstay of cheminformatics and machine learning in drug discovery applications is molecular fingerprints representing structural details. Molecular graph convolution, described by Kearnes et al. [68] in 2016, is a machine learning architecture for learning from undirected graphs, especially for small molecules. Graph convolution allows greater use of the details in the graph structure by using a basic encoding of the molecular graph - atoms, bonds, lengths, and so on.

2.4.6 Others

The applications of GNNs are not exclusive to the above domains and tasks.

In traffic, Yao et al. [69] propose a Deep Multi-View Spatio-Temporal Network (DMVST-Net) framework to model Spatio-temporal relationships to enable accurate predictive models that help cities pre-allocate resources to meet travel demand and reduce the number of empty cabs on the streets.

In healthcare, to address the issues of insufficient data and the fact that the representation learned through deep learning methods should be consistent with medical knowledge, Choi et al. [70] presented the GRaph-based attention model (GRAM), that complements the digital health record by level information inherent to the medical ontology.

In brain networks, BrainNetCNN is a GNN paradigm that Kawahara et al. [71] proposed for predicting clinical neurodevelopmental outcomes from brain networks. In comparison to conventional image-based CNNs, our BrainNetCNN is made up of novel edge-to-edge, edge-to-node, and node-to-graph convolutional filters that take advantage of structural brain networks' topological local

2.5 Negative sampling in graph neural networks

Most of the above GNNs can be considered to use positive sampling when generating new node feature vectors because the neighbor aggregation on $u \in \mathcal{N}(v)$ leads to a high correlation between the central node and neighbors. Moreover, the real existing edges in the graph data can be assumed to be sampled from an underlying positive distribution $P(u, v)$. GCN [1], GraphSAGE [29] regularizes the output embedding by local smoothness, which can be seen as an implicit regularization of an underlying positive distribution $P(u, v)$.

Negative sampling was first used in natural language processing to simplify Noise Contrastive Estimation [14], which can facilitate the training of word2vec [13]. A study using negative sampling in GNNs is relatively rare. To our knowledge, only three studies

have explored procedures to this end. The first is Kim and Oh [72], who propose the natural and easy approach of uniformly randomly selecting some negative samples from non-neighboring nodes. However, as we all know, a large graph is normally has the small-world property [15] which means the graph will tend to contain some clusters. Moreover, nodes in the same cluster will tend to have similar representations while nodes within different clusters will tend to have different representations.Hence, under uniform random selection, the large clusters will overwhelm the smaller ones, and the final converged node representations will be short on information contained in the small clusters.

The second one is based on personalized PageRank [16]. Ying et al. proposed PinSage, a random-walk graph convolutional network, which utilizes negative sampling in their loss function as an approximation of the normalization factor of edge likelihood to make GCNs practical and scalable to web-scale graphs containing billions of objects. Apart from that,

The last is based on Monte Carlo chains. In order to bridge the gap between positive and negative sampling in graph representation learning, Yang et al. [2] systematically analyzed the role of negative sampling, theoretically demonstrated that negative sampling is as important as positive sampling. Unlike previous studies that used random negative sampling with a certain probability, Yang et al. [73] proposes an effective and scalable negative sampling strategy, called Markov chain Monte Carlo Negative Sampling, which leverages a special Metropolis-Hastings algorithm.

2.6 Summary

GCNs have been generally accepted to be an effective tool for node representations learning. An interesting way to understand GCNs is to think of them as a message-passing mechanism where each node updates its representation by accepting information from its neighbours (also known as positive samples). While beyond these neighbouring nodes, graphs have a large, dark, all-but-forgotten world in which can find the non-neighboring nodes (negative samples). This great dark world can provide negative information about the node representations. However, extremely few studies use negative sampling with a GNN. The only three works [2, 16, 72] that do use it do not satisfy our criteria of good

negative samples: good negative samples should include as much information of the dark world as possible and, at the same time, without much overlapping and redundant information. Moreover, the above approaches only use the results of negative sampling in the loss function, while the direct application of convolution operations remains unexplored.

Chapter 3

Graph convolutional neural networks with Monte-Carlo negative sampling

3.1 Introduction to negative sampling in Graph convolutional neural networks

A number of modifications to improve the performance of GNNs have been proposed in the literature, including but not limited to Gated Graph Sequence Networks [10], Graph Attention Networks [39], Graph Isomorphism Networks [44], GraphSGAN for semi-supervised learning on graphs with GANs [74]. Since it is commonly accepted that the neighboring nodes have similar behavior with each other, the message passing mechanism benefits from these positive neighboring nodes compared to the negative non-neighboring nodes. In addition to get highly correlated samples to give the model positive examples, some uncorrelated samples are also useful to give the model some counterexamples, thus promoting the model to better fit the real data. However, although the importance of the contribution from non-neighboring nodes (also known as negative samples) is apparent, negative sampling which denotes that how to find and use negative samples does not come into sight of GNNs [2]. Yang et al. [2] first systematically analyzed and discussed negative

sampling in graph representation learning and investigated the role of negative sampling from an objective and risk perspective. Unfortunately, they only fuse negative sampling into the loss function to train the model rather than design a new graph convolution layer.

To the best of our knowledge, there is no research to apply negative sampling directly to the convolution operation. In this section, we first propose the negative sampling method which is based on Monte-Carlo chains, then we present a Negative Samples-enhanced Graph Convolutional Neural Networks (NegGCNs), where the negatively sampled nodes are directly incorporated into the message passing mechanism and used to generate new node feature vectors. The experimental results show that our proposed network can achieve better results than the Graph Convolutional Networks (GCNs) [1] and GraphSAGE citehamilton2017inductive for node classification in the citation network. We further investigated the effects of negative sampling rate and the number of negatively sampled nodes on the classification accuracy of graph nodes.

3.2 Graph convolutional neural networks with Monte-Carlo negative sampling

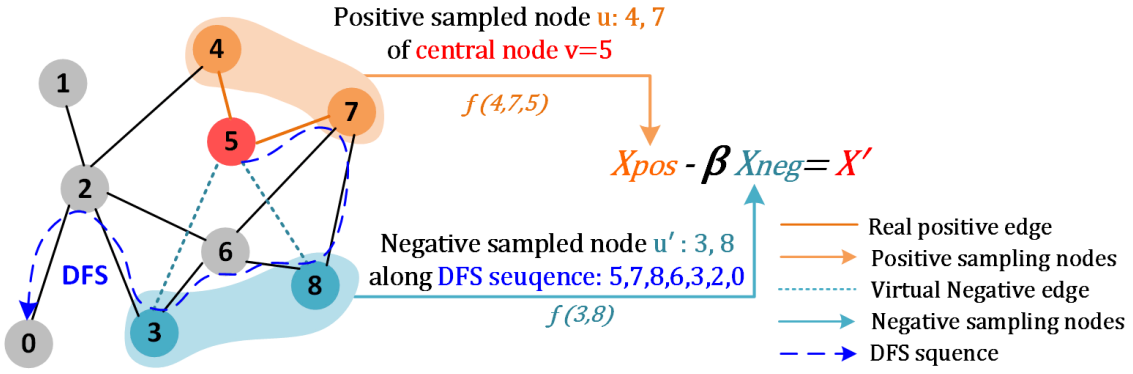

FIGURE 3.1: **Mechanism of the negative sampling graph convolution**. The central node is $v = 5$ and $f(\cdot)$ is graph convolution layer [1]. Node 4, 7 are directly linked with node 5 by real positive edges, thus positive sampling convolution is performed by $x_{pos} = f(4, 7, 5)$. Node 3, 8 are negative sampled using MCNS methods [2], which are based on Markov Chain Monte-Carlo methods and DFS, message passing to central node $v = 5$ along virtual imaginary edges, then negative sampling convolution is performed by $x_{neg} = f(3, 8)$. Given a certain negative rate β, we get negative sampling graph convolution result of this layer, i.e. $x^{'} = x_{pos} - \beta x_{neg}$

In this section, Figure 3.1 briefly illustrates the mechanism of the graph convolution with negative sampling. We first introduce how to perform negative sampling using Markov Chain Monte Carlo Negative Sampling (MCNS) method [2]. Next, we describe how to incorporate the results of negative sampling into the convolution operation to perform graph convolution with negative sampling.

3.2.1 Negative sampling

MCNS method [2] was adopted to perform Negative Sampling, which based on Markov chain Monte Carlo methods depth-first search (DFS).

We first describe **how to perform negative sampling with Markov chains**. An edge e, where $e = (u, v)$ and $u \in \mathcal{N}(v)$, in the graph data can be assumed to be sampled from a positive distribution $P_{pos}(u, v)$. Negative samples of the node v can be assumed to be sampled from a negative distribution $P_{neg}(u^{'}, v)$, where $u^{'} \in V$.

To obtain a sequence of negative samples from unnormalized distribution, the Metropolis-Hastings algorithm [73] is used for constructing a Markov chain $\{X(t)\}$, where $X(t) \sim \pi(x), t \to \infty$. The Metropolis-Hastings algorithm has the following steps:

1. Choose an arbitrary point X_0 to be the first sample and choose an arbitrary positive distribution $q(x \mid y)$ which is the transfer probability that the next value of x, given y.

2. For each iteration $t = 0, 1, 2, ..., T$:

   - The Markov chain state at the t^{th} moment is X_t, sampling $y \sim q(y|X(t))$
   - Generate a uniform random number $r \sim \text{Unifrom}[0, 1]$.
   - If $r < a\Big(X(t), y\Big) = \min\Big\{\frac{\tilde{\pi}(y)}{\tilde{\pi}(X(t))}\frac{q(X(t)|y)}{q(y|X(t))}, 1\Big\}$:
     $X(t+1) \leftarrow y$;
     else: $X(t+1) \leftarrow X(t)$.

In the following we explain **why DFS is used**. The core of MCNS [2] is the application of the Metropolis-Hastings algorithm for each node v with:

$$\tilde{\pi}(u \mid v) = (f(u) \cdot f(v)) \tag{3.1}$$

where $f(\cdot)$ denotes GCNs. According to Yang et al.'s theory [2], the negative sampling distribution should be positively but sub-linearly correlated to their positive sampling distribution. Thus, negative nodes are sampled from a self-contrast approximated distribution [2]:

$$P_{neg}(u^{'}, v) \propto P_{pos}(u, v)^{\alpha} \approx \frac{(f(u) \cdot f(v))^{\alpha}}{\sum_{u' \in V}(f(u') \cdot f(v))^{\alpha}}, \tag{3.2}$$

where $0 < \alpha < 1$. However, since the distribution of $X(t) \approx \pi(x)$ in a Markov chain only occurs when t is large, to improve the Metropolis-Hastings efficiency, the graph is traversed by DFS for each node and negative samples are generated from the last node of the Markov chain, as shown in Figure 3.1, which takes advantage of the fact that neighboring nodes in the graph have similar target negative sampling distributions $P_{neg}(u^{'}, v)$ and the Markov chain of one node is likely to remain valid for its neighbors.

3.2.2 Graph convolution with negative sampling

With the negatively sampled nodes, we describe how to incorporate these points directly into the graph convolution operation. The set of neighboring nodes of v can be rewritten in the form of positive sampling $\mathcal{N}(v) = \{u \in V | u \sim P_{pos}(u, v)\}$. In the same way, the set of sample points negatively correlated with node v using the MCNS method can be defined as $\mathcal{NS}(v) = \{u' \in V | u' \sim P_{neg}(u', v)\}$.

Recall that the Eq. (2.20) which denotes the k^{th} layer of a multi-layer GCNs [1] be written as:

$$x_v{}^{(k)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\deg(v)} \cdot \sqrt{\deg(u)}} \left(\Theta \cdot x_u^{(k-1)}\right) \tag{3.3}$$

where $\deg(v)$ denotes the degree of node v and $x_v{}^{(k)}$ denotes the feature vector of the k^{th} iteration of node v. To further analyze the above equation, positive sampling convolution in the graph means that for each central point $x_v{}^{(k-1)}$, all its neighboring nodes $u \in \mathcal{N}(v)$

will pass messages to it along the real existing edges $e = (u, v)$, aggregating into a new node ${x_v}^{(k)}$. In the same way, negative sampling convolution is that for each central point ${x_v}^{(k-1)}$ in the graph, find all nodes negatively associated with it $u^{'} \in \mathcal{NS}(v)$ and pass message to it along virtual imaginary edges $e^{'} = (u^{'}, v)$, subtracting all the incoming information to form a new node ${x_v}^{'(k)}$. Meanwhile, we also need to control the magnitude of the impact of negatively sampled information on the central node. Therefore, we set a hyperparameter β called the negative sampling rate. A graphical representation of the above process is illustrated in Figure 3.1.

Adopting the same ideas of GCNs [1], we propose the **Negative Samples-enhanced Graph Convolution(NegConv)**, which can be defined in k^{th} layer as:

$$\begin{aligned} {x_v}^{(k)} =& x_{pos}^{(k-1)} - \beta x_{neg}^{(k-1)} \\ =& \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{\deg(v)} \cdot \sqrt{\deg(u)}} \big(\Theta(x_u^{(k-1)})\big) \\ & - \beta \sum_{u^{'} \in \mathcal{NS}(v)} \frac{1}{\sqrt{\deg(v)} \cdot \sqrt{\deg(u^{'})}} \big(\Theta(x_{u^{'}}^{(k-1)})\big) \end{aligned} \tag{3.4}$$

where β is the negative sampling rate representing the weight of the negative sampling convolution.

Algorithm 1 demonstrates the complete procedure of graph convolution with negative sampling for a layer. We first calculate the DFS sequence for each node and initialize positive sampling distribution $\hat{p}_{pos}$, arbitrary positive distribution q, negative rate β, which as inputs of a NegConv layer. Then, we use graph convolution layer $f_{pos}(\cdot)$ [1] to perform positive sampling convolution to get x_{pos}. We next perform negative sampling to get $x^{'}$ for every input node x using MCNS [2]. At last, we use another GCNs layers $f_{neg}(\cdot)$ to perform negative sampling convolution to get x_{neg} and calculate the output of the current k^{th} convolution layer $x^k = x_{pos} - \beta x_{neg}$.

Algorithm 1: NegConv for a layer

Input: Output of the last layer x^{k-1}, DFS sequence $T = [v_0, ..., v_{|T|}]$, Arbitrary positive distribution q, Number of Negative Samples n, Negative Rate β, Graph Convolution Layer $f_{pos}(\cdot)$, $f_{neg}(\cdot)$.

for *each node* x_u^{k-1} *in last layer* x^{k-1} **do**
  Perform positive sampling convolution $x_{pos} = f_{pos}(x_u^{k-1})$;
  Initialize current negative node $u_0^{'}$ at random;
  for *each node* v *in DFS sequence* T **do**
    for $i = 1$ *to* n **do**
      Sample a node y from $q(y \mid u^{'})$;
      Generate a uniform random number $r \in [0, 1]$;
      if $r < \min\left\{\frac{(f_{pos}(v)\cdot f_{pos}(y))^{\alpha}}{(f_{pos}(v)\cdot f_{pos}(u^{'}))^{\alpha}} \frac{q(u^{'}|y)}{q(y|u^{'})}, 1\right\}$ **then**
        $u^{'} = y$
      end
    end
    Save negative sampling results $u^{'}$ as $x_{u^{'}}^{k-1}$;
  end
  Perform negative sampling convolution $x_{neg} = f_{neg}(x_{u^{'}}^{k-1})$;
  Calculate the result of this layer $x^k = x_{pos}^{k-1} - \beta x_{neg}^{k-1}$
end

TABLE 3.1: Statistics of the dataset

Dataset	Nodes	Edges	Classes
Cora	2,708	5,429	7
Citeseer	3,327	4,732	6
Pubmed	119,717	44,338	3

3.3 Experiments

We test the performance of our Negative sampling graph convolution network on a semi-supervised node classification using real-world citation graphs.

3.3.1 Datasets

The experimental setup and dataset splitting were strictly in accordance with Kipf & Welling [1]. Table 3.1 summarizes the dataset statistics, including Citeseer, Cora, and Pubmed [75]. The datasets include sparse bag-of-words feature vectors for each document as well as a list of document-to-document citation connections.

3.3.2 Experimental setup

We train a two-layer NegGCNs as described in Algorithm 1 and test the effect of different negative rates β on the accuracy of the prediction results. Using Adam Optimiser [76] with a learning rate of 0.01, the models are trained with 200 epochs for Cora and Citeseer and trained with 400 epochs for Pubmed.

To test the effect of different β values on the results, experiments were conducted with a gap of 0.25, from $\beta = 0.25$ to $\beta = 2.0$, when $\beta = 0$ NegGCNs become GCNs [1] exactly. The number of negatively sampled points is the same as the number of edges, which means that for each point v_i, the number of negatively sampled nodes is D_{ii}.

After completing the tests on different negative sampling rates, we further test the effect of the number of negative sampling nodes on the classification results on the Cora dataset of experimental results with the highest accuracy. We tested sampling 10%, 30%, 50%, 70%, and 90% of the total negative sampling points $\mathcal{NS}(v)$ and gave training accuracy curves. For example, if a node has 10 nodes connected to it when the sampling rate is 50%, we sample 5 nodes that are negatively correlated with the node.

All the experiments were conducted on a machine with Intel(R) Xeon(R) CPU @ 2.00GHz and NVIDIA Tesla T4 GPU and were implemented using PyTorch. [1]

3.3.3 Baselines

We compare against the methods as in GCNs[2] [1] and in GraphSAGE2[0] [29]. To ensure the consistency of the experiments, all the graph convolution models are two-layer and all data sets are processed in the same way.

3.3.4 Experimental results

Table 3.2 summarises experimental results. The experiments show that our proposed NegGCNs has improved prediction accuracy compared to GCN and GraphSAGE in the case

[1] The code is available at https://github.com/Wei9711/NegGCNs

[2] The two baseline methods are implemented using:
https://colab.research.google.com/drive/14OvFnAXggxB8vM4e8vSURUp1TaKnovzX?usp=sharing.

TABLE 3.2: Experimental results

Methods	**Cora**	**Citeseer**	**Pubmed**
GCNs [1]	0.8220	0.6870	0.7960
GraphSAGE [29]	0.8000	0.7020	0.7810
NegGCNs $\beta = 0.25$	0.8230	0.7090	0.7920
NegGCNs $\beta = 0.50$	0.8340	0.6940	0.7900
NegGCNs $\beta = 0.75$	0.8290	0.6950	0.7970
NegGCNs $\beta = 1.00$	0.8270	0.7100	**0.7990**
NegGCNs $\beta = 1.25$	**0.8350**	**0.7150**	0.7970
NegGCNs $\beta = 1.50$	0.8270	0.6890	0.7920
NegGCNs $\beta = 1.75$	0.8280	0.6600	0.7910
NegGCNs $\beta = 2.00$	0.8250	0.6580	0.7900

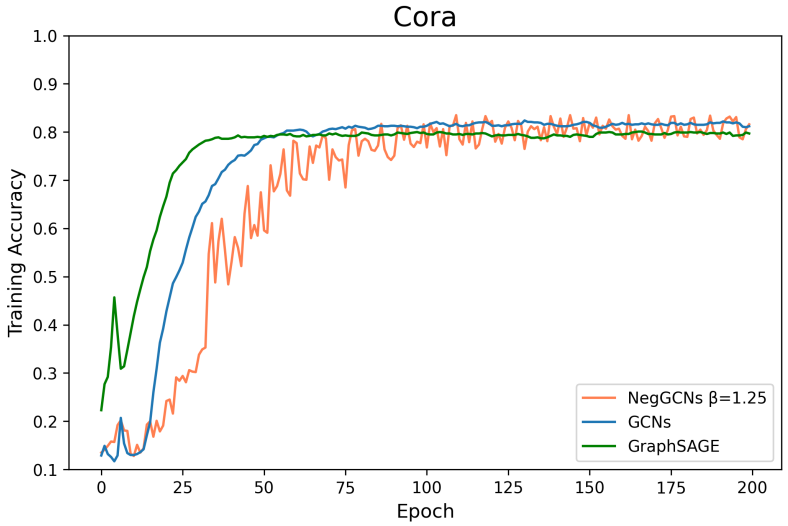

FIGURE 3.2: Performance of the three models on the training set in the Cora dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.25$.

of setting a specific negative rate β. When $\beta = 1.25$, the most significant improvement in node classification accuracy was achieved for both Cora and Citeseer datasets reaching a maximum value of 0.8350 and 0.7150, respectively. For the Pubmed dataset, the improvement is not significant compared to the other two datasets, compared with GCNs, only from 0.7960 to 0.7990 when $\beta = 1.00$. For Cora, the different β-values all improve the accuracy of node classification and both $\beta = 0.50$ and $\beta = 1.25$ bring the accuracy above 0.83. For Citeseer, when $\beta = 0.25, 1.00, 1.25$, the accuracy is over 0.70 in three cases and it starts to drop when β is greater than 1.25. For Pubmed, only when $\beta = 0.75$ and $\beta = 1.00$, the node classification accuracy exceeds baseline.

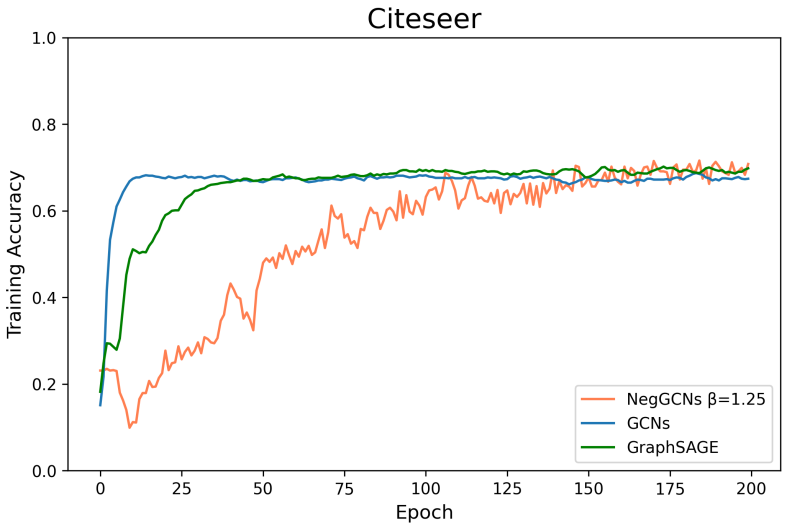

FIGURE 3.3: Performance of the three models on the training set in the Citeseer dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.25$.

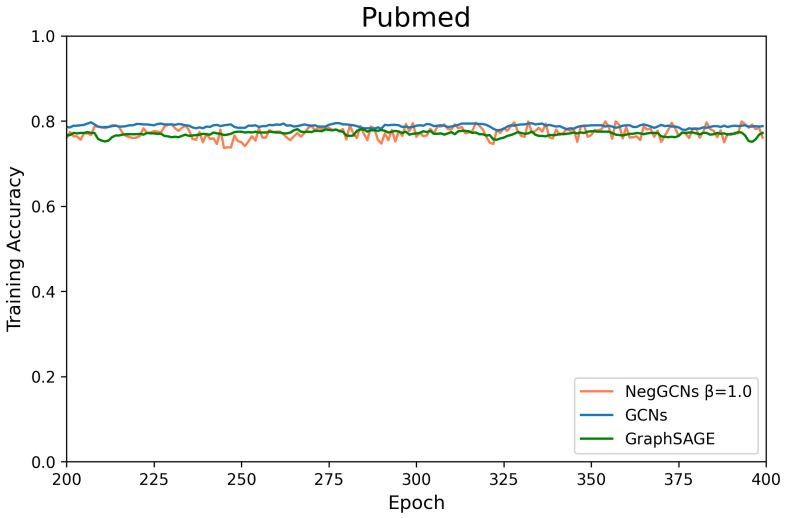

FIGURE 3.4: Performance of the three models on the training set in the Pubmed dataset. We choose the NegGCNs model with the highest accuracy for comparison, when $\beta = 1.00$.

Figure 3.2,3.3,3.4 show the performance of the three models on the training set in the Cora, Citeseer, Pubmed datasets. We choose the NegGCNs model with the highest accuracy for comparison. As can be seen from the three plots, our proposed method fluctuates more than the other two methods, thanks to this property when the training results tend to be stable, it is easier to obtain the maximum value of the three methods within a certain accuracy range. For the Cora dataset, the convergence speed of our proposed NegGCNs is close to the other two methods. When exceeding 100 Epochs, NegGCNs has achieved a

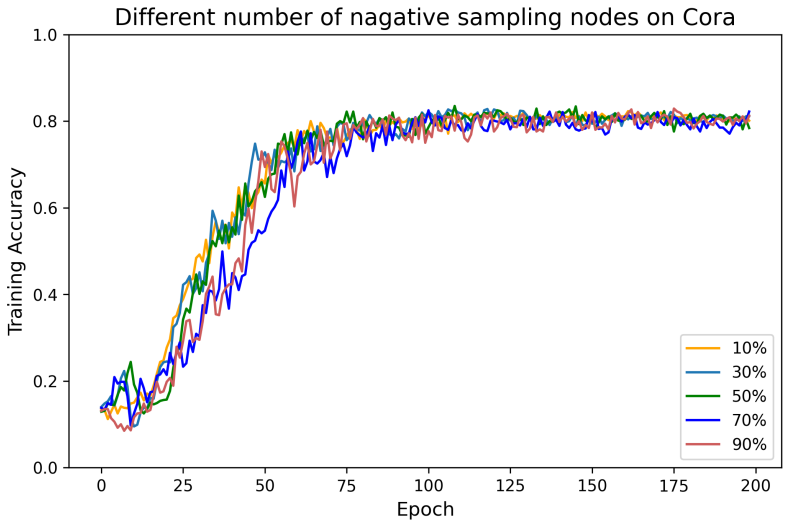

FIGURE 3.5: Performance of a different number of negative sampling nodes on Cora training set using NegGCNs with $\beta = 1.25$. We tested sampling 10%, 30%, 50%, 70%, and 90% of the total negative sampling points to calculate x_{neg}.

training accuracy comparable to the other two methods and starts to fluctuate in a small range. As the Epoch increases, there are more and more places where it can surpass the other two algorithms. For the Citeseer dataset, NegGCNs converge slower than the other two methods. It achieves similar training accuracy to the other two methods only after more than 150 Epochs but exceeds the other two methods after that at some points. From Figure4, on the Pubmed dataset, the results of all three methods were close after 200 epochs. After 300 epochs, some points of NegGCNs surpassed the other two methods.

Performance of different the number of negatively sampled nodes on Cora training set is illustrated in Figure 3.5. We can see that although the curves for various numbers of negative nodes are dissimilar at the beginning of the training, the difference between the five curves becomes smaller with the increase in the number of epochs. The number of negative nodes has no significant relationship with the training accuracy when other conditions are equal.

3.4 Summary

Many existing graph neural networks used message passing mechanisms for fusing neighborhood information to learn graph representation, so these methods can be viewed as

using positive sampling to obtain samples to generate new feature vectors. Study Using negative sampling in GNNs is relatively rare.

In this section, we have proposed Negative Samples-enhanced Graph Convolutional Neural Networks. Unlike previous graph neural networks that only use positive sampling, our NegGCNs incorporate the negative samples directly into the message passing mechanism to generate new node feature vectors. To the best of our knowledge, this is the first research to apply negative sampling directly to the convolution operation. Experiments on citation network datasets suggest that the proposed NegGCNs model can improve the accuracy of graph node classification with a specific negative rate compared to GCNs and GraphSAGE.

Since the MCNS sampling algorithm only obtains samples from DFS sequences, the candidate set is relatively limited, and thus the samples do not contain as much information as possible to reflect the variety of the dark world. To overcome this problem, we further conducted the next section of the study

Chapter 4

Graph convolutional neural networks with DPP negative sampling

4.1 Introduction to determinant point process (DPP)

Selecting appropriate negative samples in a graph is not trivial. In the previous chapter, we refer to the work of Yang et al. [2] for negative sampling using an approach based on Monte Carlo chains. However, this approach cannot cover diverse information and may have many redundant selected negative samples. Hence, as a definition, a good negative sample should *contribute negative information to the given node contrast to its positive samples and include as much information as possible to reflect the variety of the dark world.*

In this chapter, we propose a graph convolutional neural network boosted by diverse negative samples selected through a determinant point process (DPP). DPP is a special point process for defining a probability distribution on a set where a more diverse subset has a higher probability [77]. Given a ground set $\mathcal{Y} = \{1, 2, \ldots, |\mathcal{Y}|\}$, a determinantal point process $\mathcal{P}$ is a probability measure on all possible subsets of $\mathcal{Y}$ with size $2^{|\mathcal{Y}|}$. For

every $Y \subseteq \mathcal{Y}$, a DPP [77] defined via an $\mathbb{L}$-ensemble is formulated as

$$\mathcal{P}_{\mathbb{L}}(Y) = \frac{\det(\mathbb{L}_Y)}{\det(\mathbb{L}+I)} \tag{4.1}$$

where $\det(\cdot)$ denotes the determinant of a given matrix, $\mathbb{L}$ is a real and symmetric $|\mathcal{Y}| \times |\mathcal{Y}|$ matrix indexed by the elements of $\mathcal{Y}$, and $\det(\mathbb{L}+I)$ is a normalisation term that is constant once the ground dataset $\mathcal{Y}$ is fixed. Given a Gram decomposition $\mathbb{L}_Y = \bar{\mathbb{L}}^T\bar{\mathbb{L}}$, a determinantal operator can be interpreted geometrically as

$$\mathcal{P}_{\mathbb{L}}(Y) \propto \det(\mathbb{L}_Y) = \mathrm{vol}^2\left(\{\bar{\mathbb{L}}_i\}_{i\in Y}\right) \tag{4.2}$$

where the right-hand side is the squared volume of the parallelepiped spanned by the columns in $\bar{\mathbb{L}}$ corresponding to elements in Y. Intuitively, to get a parallelepiped of greater volume, the columns should be as repulsive as possible to each other. Hence, DPP assigns a higher probability to a subset of Y whose elements span a greater volume.

One important variant of DPP is k-DPP [78]. k-DPP measures only k-sized subsets of $\mathcal{Y}$ rather than all of them including an empty subset. It is formally defined as

$$\mathcal{P}_{\mathbb{L}}(Y) = \frac{\det(\mathbb{L}_Y)}{e_k^{|\mathcal{Y}|}} \tag{4.3}$$

with the cardinality of the subset Y being a fixed size k, i.e., $|Y| = k$. $e_k^{|\mathcal{Y}|}$ is the k^{th} elementary symmetric polynomial on eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{|\mathcal{Y}|}$ of $\mathbb{L}$, i.e., $e_k(\lambda_1, \lambda_2, \ldots, \lambda_{|\mathcal{Y}|})$. Both DPP and k-DPP can be used to sample a diverse subset, and both have been well studied in the machine learning area [78]. The popularity of DPP (and also of k-DPP) for modeling diversity is because of its great modelling power.

Our idea is to find diverse negative samples for a given node by firstly defining a DPP-based distribution on diverse subsets of all non-neighboring nodes of this node, and then outputting a diverse subset from this distribution. The number of subsets is limited because the samples are mutually exclusive. However, when applying this algorithm to large graphs, the computational cost can be as high as $O(N^3)$, where N is the number of non-neighboring nodes. Therefore, we propose a method based on a depth-first search (DFS) that collects diverse negative samples sequentially along the search path for a node.

The motivation is that a DFS path is able to go through all different clusters in the graph and, therefore, the local samples collected will form a good, diverse approximation of all the information in the dark world. Further, the computational cost is approximately $O(\overline{\text{pa}} \cdot \overline{\text{deg}}^3)$ where $\overline{\text{pa}} \ll N$ is the path length (normally smaller than the diameter of the graph) and $\overline{\text{deg}} \ll N$ is the average degree of the graph. As an example, consider a Cora graph [75] with 3,327 nodes, $\overline{\text{pa}} = 19$, and $\overline{\text{deg}} = 3.9$. Thus, $O(N^3) = 3.6e10 \gg 1,127 = O(\overline{\text{pa}} \cdot \overline{\text{deg}}^3)$.

In evaluating these ideas, we conducted empirical experiments on different tasks, including node classification and over-smoothing tests. The results show that our approach produces superior results.

Thus, the contributions of this chapter include:

- A new negative sampling method based on a DPP that selects diverse negative samples to better represent the dark information;
- A DFS-based negative sampling method that sequentially collects locally diverse negative samples along the DFS path to greatly improve computational efficiency;
- The negative samples are fused into the graph convolution, yielding a new GCN boosted by diverse negative samples (D2GCN) to improve the quality of node representations and alleviate the over-smoothing problem.

4.2 Graph convolutional neural networks with DPP negative sampling

This section first introduces a method for obtaining good negative samples for a given node, including two negative sampling algorithms. We then integrate the algorithms with a GCN to obtain a new graph convolutional neural network boosted by diverse negative samples (D2GCN).

Algorithm 2: Obtain a sample from k-DPP

Input: size k, a DPP distribution $\mathcal{P}_{\mathbb{L}}(Y)$ on a set with size V
Output: Y with size $|Y| = k$
Eigen-decompose $\mathbb{L}$ to obtain eigenvalues $\lambda_1, \ldots, \lambda_V$;
Iteratively calculate all elementary symmetric polynomials e_l^v for $l = 0, \ldots, k$ and $v = 0, \ldots, V$ following [78];
$Y \leftarrow \emptyset$;
$l \leftarrow k$;
for $v = V, \ldots, 1$ **do**
    if $l = 0$ **then**
        break;
    end
    if $\mu \sim U[0,1] < \lambda_v \frac{e_{l-1}^{v-1}}{e_l^v}$ **then**
        $Y \leftarrow \mathcal{Y} \cup \{v\}$;
        $l \leftarrow l - 1$;
    end
end
return Y

Algorithm 3: Diverse negative sampling

Input: A graph G
Output: $\overline{\mathcal{N}}(i)$ for all $i \in G$
Let $\overline{\mathcal{N}} = \mathbf{0}$;
for *each node* i **do**
    Compute $\mathbb{L}_{G \setminus i}$ using Eq. (4.4);
    Define a distribution on all possible subsets $\mathcal{P}_{\mathbb{L}}(Y_{G \setminus i})$ using Eq. (4.5);
    Obtain a sample of $\mathcal{P}_{\mathbb{L}}(Y_{G \setminus i})$ using Algorithm 2;
    Save nodes in the sample as $\overline{\mathcal{N}}(i)$;
end
return $\overline{\mathcal{N}}$

4.2.1 DPP-based negative sampling

Given a node, we believe that its good negative samples should include as much information about the dark world as possible and, at the same time, without much overlap and redundancy. The repulsive property of DPP inspires us to use it to select negative samples. However, applying DPP to different scenarios is not trivial; modifications are required to make it work. Hence, for a given node i in a graph G, we need to first define a $\mathbb{L}$-ensemble for our problem:

$$\mathbb{L}_{G_n \setminus i}(j, j') = \exp\left(-\cos\left(x_j, x_{j'}\right)\right), \tag{4.4}$$

where $G_n \backslash i$ denotes the node-set of G_n excluding node i, j and j' are two nodes within $G_n \backslash i$, and $\cos(\cdot, \cdot)$ is the cosine similarity between two node representations x. The node feature x is used here because, when defining the distance between nodes, nodes with similar information are expected to be further apart. x_j is expected to encode the information of node j in terms of both features and network structure. Feature information may dominate at the beginning but both types of information will be balanced after a number of training steps. The reason we chose cosine similarity is that the following prediction (label and link) is normally based on cosine similarity. We used only x to simplify the explanation. However, it would be straightforward to include more information here, like the network distance between nodes j and j', the similarity with node i, or the node degree of each node.

With this $\mathbb{L}$-ensemble, we can obtain a distribution on all possible subsets of all nodes in the graph except for i,

$$\mathcal{P}_{\mathbb{L}}(Y_{G_n \backslash i}) = \frac{\det(\mathbb{L}_{Y_{G_n \backslash i}})}{\det(\mathbb{L}_{G_n \backslash i} + I)} \tag{4.5}$$

where $Y_{G_n \backslash i}$ is a set of all subsets of $G_n \setminus i$ and $\det(\cdot)$ is the matrix determinant operator. Comparing other ordinary probability distributions with the same support, this distribution has a nice and unique property that the more diverse the subsets, the higher their probability values, the easier it is to obtain a diverse set from this distribution by sampling. To ensure the scale of negative samples is similar to the positive samples, we use k-DPP to fix the number of negative samples as $k = |\mathcal{N}_i| + 1$. Here, we use a sampling method based on eigendecomposition [78, 79]. Eq. (4.5) can be rewritten as the k-DPP distribution in terms of the corresponding DPP

$$\mathcal{P}^k_{\mathbb{L}}(Y_{G_n \backslash i}) = \frac{1}{e_k^{|G_n \backslash i|}} \det(\mathbb{L}_{G_n \backslash i} + I) \mathcal{P}_{\mathbb{L}}(Y_{G_n \backslash i}) \tag{4.6}$$

whenever $|Y_{G_n \backslash i}| = k$ and $e_k^{|G_n \backslash i|}$ denotes the k-th elementary symmetric polynomial. Following [78], Eq. (4.6) can be decomposed into elementary parts

$$\mathcal{P}^k_{\mathbb{L}}(Y_{G_n \backslash i}) = \frac{1}{e_k^{|G_n \backslash i|}} \sum_{|J|=k} \mathcal{P}^{V_J}(Y_{G_n \backslash i}) \prod_{m \in J} \lambda_m \tag{4.7}$$

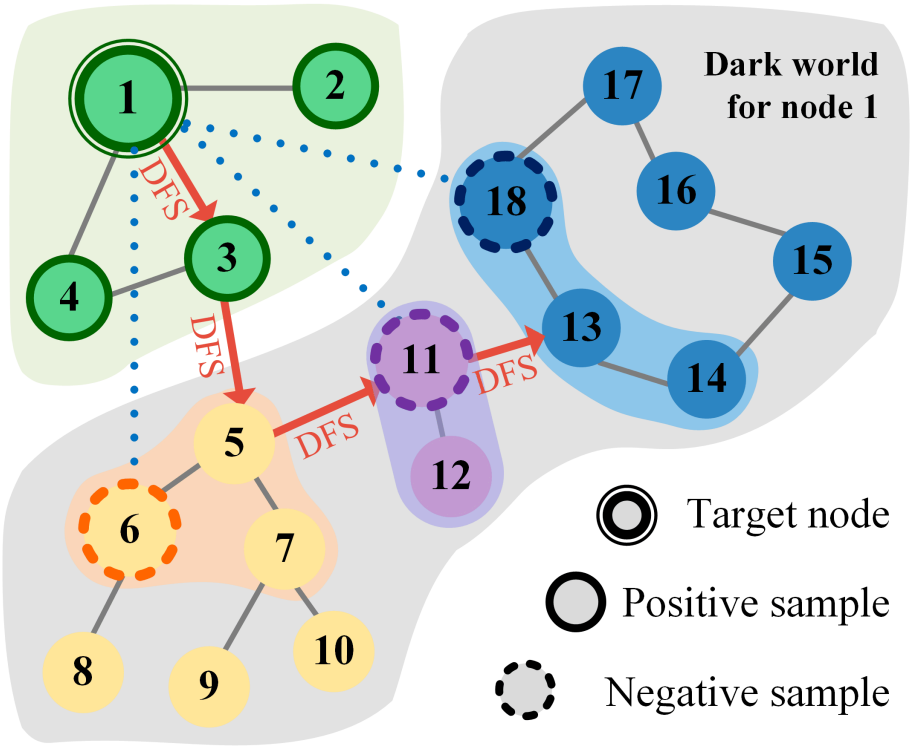

FIGURE 4.1: The concept of DPP-based negative sampling. The target node is Node 1. Nodes 2, 3 and 4 are positive samples. Nodes 5-18 are the dark world of Node 1. The 4-length DFS path of Node 1 is $\{3, 5, 11, 13\}$, where $\{5, 11, 13\}$ are the central nodes on the path in the dark world. With their first-order neighbouring nodes, they form the candidate set of DPPs, i.e.$\{5, 6, 7, 11, 12, 13, 14, 18\}$. The selected negative samples from this set are 6, 11, and 18, which can be seen as virtual negative links to Node 1.

where $V_{Y_{G_n \setminus i}}$ denotes the set $\{\boldsymbol{v}_m\}_{m \in Y_{G_n \setminus i}}$ and $\boldsymbol{v}_m$ and λ_m are the eigenvectors and eigenvalues of the $\mathbb{L}$-ensemble, respectively. Based on Eq. (4.7), for the self-contained reason, the complete process of sampling from k-DPP is given in Algorithm 2.

Note that compared with a sample, the mode of this distribution is a more rigorous output [80], but it is usually with an unbearable complexity, so we use a sample rather than a mode here. The experimental evaluation has shown that the sample has been able to achieve satisfactory results. Algorithm 3 shows a diverse negative sampling method based on DPP, but its computation cost is above the standard for a DPP on $G_n \setminus i$ is $O(|G_n \setminus i|^3)$ for each node. This totals an exorbitant $O(|G_n \setminus i|^4)$ for all nodes! Although Algorithm 3 is able to explore the whole dark world and find the best diverse negative samples, the large number of candidates makes it an impractical solution for even a moderately-sized graph. Hence, we propose the approximate heuristic method below.

Our belief is that the good negative samples are the ones with different semantics and complete knowledge of the whole graph, such as nodes 6, 11, and 18 in Figure 4.1. In this figure, we hope the selected negative samples could each belong to the different 'cluster' and all 'clusters' are represented by the negative samples. Hence, given a node i, we use a depth-first-search (DFS) method to build a fixed-length path from node i to the other nodes. We then collect the first-order neighbours of the nodes on this path to form a candidate set. Finally, we select diverse negative samples from this candidate set using the same DPP idea as outlined above. This DFS-based method can be considered as using a local diverse negative sample to approximate the global diverse negative samples. The reason is that DFS has the capability of breaking through the 'cluster' of the current node i to reach the semantics of other 'clusters'. So the first-order neighbours of nodes on the path are able to supply sufficient information from many 'clusters' but at a much smaller size. Although only first-order neighbours are collected here, collecting a higher order may further increase approximation performance; however, it would also increase the computational cost. In terms of path length, we found that performance is sufficiently good when the path length is set to around a quarter of the graph's diameter. The full procedure is summarised in the Algorithm 4.

Here, the overall compuational cost is $O(N \cdot |\overline{\text{pa}}| \cdot \overline{\text{deg}}^3)$ where $|\overline{\text{pa}}| \ll N$ is the path length (normally smaller than the diameter of the graph) and $\overline{\text{deg}} \ll N$ is the average degree of the graph. That is much smaller than $O(|G_n \setminus i|^4)$

4.2.2 GCN boosted by diverse negative samples

The classical GCN is only based on positive samples as shown in Eq.(3.3), which will inevitably lead to over-smoothing issues. With the negative samples from Algorithms 3 or 4, we propose the following new graph convolutional operation as

$$\begin{aligned} x_i{}^{(l)} = & \sum_{j \in \mathcal{N}_i \cup \{i\}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(j)}} \left(\Theta^{(l)} \cdot x_j^{(l-1)}\right) \\ & - \omega \sum_{\bar{j} \in \overline{\mathcal{N}_i}} \frac{1}{\sqrt{\deg(i)} \cdot \sqrt{\deg(\bar{j})}} \left(\Theta^{(l)} \cdot x_{\bar{j}^{(l-1)}}\right) \end{aligned} \tag{4.8}$$

Algorithm 4: DFS-based diverse negative sampling

Input: A graph G
Output: $\overline{\mathcal{N}}(i)$ for all $i \in G$
Let $\overline{\mathcal{N}} = \mathbf{0}$;
for *each node* i **do**
  Build a DFS path pa_i in $G \setminus i$;
  Let $C_i = [\,]$;
  for *each node* $j \in pa_i$ **do**
    Collect first-order neighbors $\mathcal{N}(j)$ of j;
    Expand $C_i = [C_i, \mathcal{N}(j)]$;
  end
  Compute $\mathbb{L}_{C_i}$ using Eq. (4.4);
  Define a distribution on all possible subsets $\mathcal{P}_{\mathbb{L}}(Y_{C_i})$ using Eq. (4.5);
  Obtain a sample of $\mathcal{P}_{\mathbb{L}}(Y_{C_i})$;
  Save nodes in the sample as $\overline{\mathcal{N}}(i)$;
end
return $\overline{\mathcal{N}}$

Algorithm 5: D2GCN

Input: A graph G
Output: $x_i^{(L)}$ for all $i \in G$
Build DFS path for all nodes;
for *each level* l **do**
  for *each node* i *in* l **do**
    Find negative samples for i using Algorithm 4;
    Update node representation $x_i{}^{(l)}$ using Eq. (4.8);
  end
end
return $x^{(L)}$

where $\overline{\mathcal{N}_i}$ is the negative samples of node i and ω is a hyper-parameter to balance the contribution of the negative samples. It is also interesting to consider that all these negative samples form a virtual graph with the same nodes as before but with negative links between the nodes. When using the message-passing framework for the mode learning, there are in fact two messages from each node: one is positive from neighbouring nodes and the other is negative from the negative samples. The positive messages push all the nodes with the same semantics to have similar representations, while the negative messages push all nodes with different semantics to have different representations. This strategy is similar to clustering, where samples within the same cluster are a small distance apart and large distances between samples indicate the sample is sitting in a different cluster. We believe

that these negative messages are precisely what is missing from GCNs and a significant element in their advancement.

The final GCN boosted by diverse negative samples (D2GCN) with L layers, is given in Algorithm 5. Excluding negative sampling, the computational cost is double that of the original GCN. Note that, although GCNs are used as the base model here, this idea can be easily applied to other GNNs as well.

4.3 Experiments and results analysis

The datasets we used are benchmark graph datasets in the literature: Citeseer, Cora, and Pubmed [75]. The datasets include sparse bag-of-words feature vectors for each document as well as a list of document-to-document citation connections. Datasets are downloaded from Pytorch geometric[1]. To better perform the experiments using DFS, we chose the maximum connected subgraph of each graph data. The datasets were split strictly in accordance with [1].

4.3.1 Baselines

GCN is the base model. We compare our sampling method with the only three negative sampling methods available to date, then put the selected samples into the convolution operation using Eq. (4.8). The first method is to select negative samples in a purely random way, named **RGCN** [72]. The second one is based on Monte Carlo chains, named **MCGCN** [2]. The last one is based on personalized PageRank, named **PGCN** [16]. To ensure the consistency and fairness of the experiments, all the graph convolution models had the same structure and were initialized and trained with the same methods. Note there is no other related negative sampling works in the field.

[1] https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

4.3.2 Setup

The experimental task was standard node classification. We set the length of DFS to 5 and the negative rate as a trainable parameter and trained all models with different numbers of layers in the range $\{2, \cdots, 6\}$ to test behavior at increasing depths. Each model was trained for 200 epochs on Cora and Citeseer and for 100 epochs with Pubmed, using an Adam optimiser with a learning rate of 0.01. Tests for each model at each depth with each dataset were conducted 10 times. Moreover, all experiments were conducted on an Intel(R) Xeon(R) CPU @ 2.00GHz and NVIDIA Tesla T4 GPU. The code was implemented in PyTorch[2].

4.3.3 Metrics

It was our goal to verify two capabilities of the proposed model: one is the ability to improve the prediction performance and the other is to alleviate the over-smoothing problem. Hence, we used the following two metrics:

- **Accuracy** is the cross-entropy loss for the node label prediction on test nodes (the larger is the better);
- **Mean Average Distance (MAD)** [81] reflects the smoothness of graph representation (the larger is the better):

$$\text{MAD} = \frac{\sum_i D_i}{\sum_i 1(D_i)}, \quad D_i = \frac{\sum_j D_{ij}}{\sum_j 1(D_{ij})} \tag{4.9}$$

  where $D_{ij} = 1 - \cos(x_i, x_j)$ is the cosine distance between the nodes i and j.

4.3.4 Results analysis

The results of five models with different numbers of layers on three datasets are shown in Figure 4.2. The performances of all five models on Cora and Citeseer were very similar in terms of both Accuracy and MAD at 2 and 3 layers. On Pubmed, our model was marginally

[2]The code is available at https://github.com/Wei9711/D2GCN.

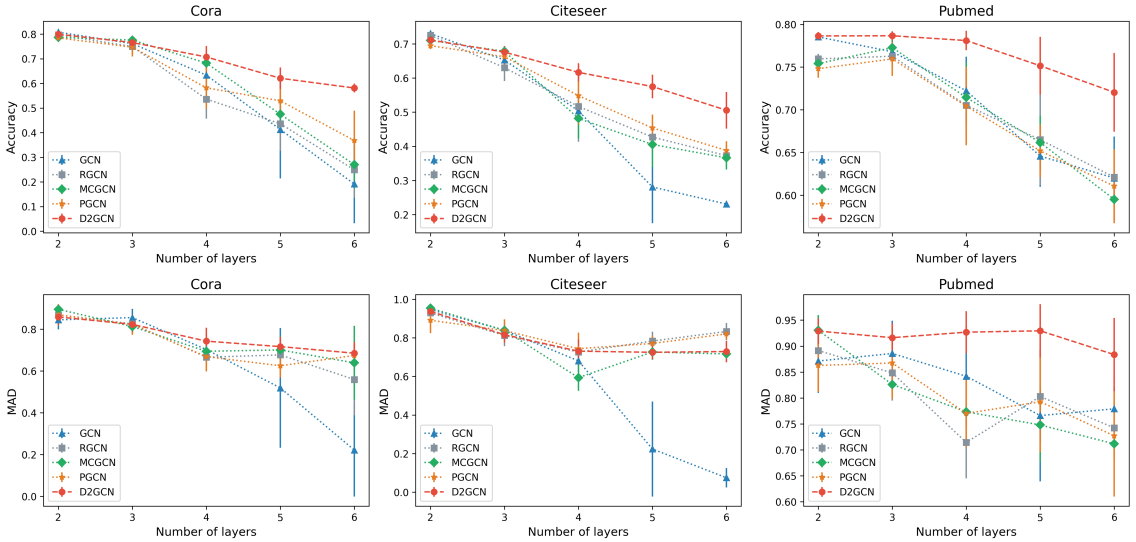

FIGURE 4.2: The Accuracy and MAD of five models on three datasets with a varying number of layers from 2 to 6. The x-axis denotes the layer number, and the mean and standard deviation of 10 runs are given for each model with each layer number.

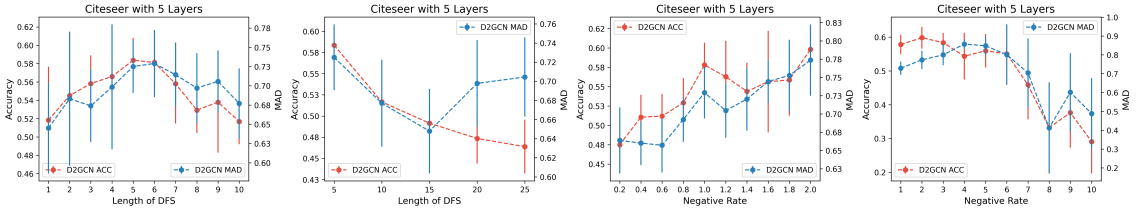

FIGURE 4.3: The Accuracy and MAD of D2GCN with 5 layers on Citeseer datasets in the varying length of DFS and scale of negative rate. The mean and standard deviation of 10 runs are given for each setting.

better than the others. When the depth increased from 3 to 6, both the Accuracy and MAD of all models decreased to some extent. This is a consistent observation with the literature that increasing the depth of the network leads to a performance drop due to over-smoothing. We also observed that the decreasing trends of RGCN, MCGCN, and PGCN on Accuracy were very similar, while PGCN is slightly better than the other two on both Cora and Citeseer datasets, especially at layers 5 and 6. As for MAD, their trends were similar on Pubmed, but on the other two datasets, GCN are much worse than the others. Moreover, although the MAD of RGCN, MCGCN and PGCN is close to our D2GCN on Cora and Citeseer, especially at layers 5 and 6 on Citeseer RGCN and PGCN even surpass our method, they are much less accurate than D2GCN. This observation suggests that adding negative samples to the convolution does reduce the over-smoothing to some

extent, but choosing the appropriate negative samples is not trivial. The additional effort needs to be given to the procedure for selecting negative samples.

As shown in Figure 4.2, our proposed model achieved consistently the best performance in terms of accuracy on all datasets. For MAD, it also performs outstandingly in particular on the Cora and Pubmed datasets. At first, we observed that the performance of our model also decreased along with the increasing depth. However, the rate at which performance decreased was much slower than for the other two models – and almost flat on Pubmed. Secondly, in terms of MAD, the performance of our model generally exceeds the others, especially on Pubmed by a large margin. These observations verify that our idea is able to significantly alleviate the over-smoothing problem and in turn improve the prediction accuracy. As a final observation, we found that the variance of our model was smaller than the other four. One possible reason is that the diverse negative samples may quickly change the current node representation rather than keep sticking around the initialization when only positive samples are used.

The sensitivity of hyper-parameters is analyzed and shown in Figure 4.3 where we trained 5-layer D2GCN on Citeseer dataset in the varying length of DFS and scale of negative rate. We observe that as negative rate or DFS-length goes up, the trends of both Accuracy and MAD are roughly the same, increasing first and then decreasing. For the length of DFS, the outstanding results are obtained when it is equal to 5 or 6. For the negative rate, it achieves the same performance as the trainable parameters, when it is equal to $\{1, 2, 3\}$.

4.4 Summary

In this chapter, we identified the importance of negative samples to GCNs and described the criteria for what constitutes a good negative sample. We introduced DPP to select meaningful negative samples from the dark world of the whole graph. To the best of our knowledge, we are the first to introduce DPP to GCNs for negative sampling and the first to fuse the negative samples into the graph convolution. We further presented a DFS-based heuristic approximation method to greatly reduce the computational cost. The experimental evaluations show that the proposed D2GCN consistently delivers better

performance than alternative methods. In addition to greater predictive accuracy, the method also helps to prevent over-smoothing. With this study, we identify that negative samples are important to graph neural networks and should be considered in future works. Note that the proposed idea can be applied to other graph neural networks apart from GCN.

Chapter 5

Conclusion and Further Study

This chapter concludes the entire thesis and provides some further research directions for this topic.

5.1 Conclusions

Graph neural networks have attracted much attention in the area of graph learning due to their powerful modeling ability. Many existing graph neural networks used message passing mechanisms for fusing neighborhood information to learn graph representation, so these methods can be viewed as using positive sampling to obtain samples to generate new feature vectors. The role of negative samples in graph convolutional neural networks is obvious yet ignored. Therefore, it is of great theoretical and practical importance to study how to find suitable negative samples in graphs and incorporate them into graph convolutional neural networks. The findings of this study are summarised as follows:

1. *The negative samples are first directly incorporated into the convolution operation.*

   Unlike previous graph neural networks that only use positive sampling, the negative samples are first directly incorporated into the message passing mechanism to generate new node feature vectors, by adopting the same ideas of GCNs [1]. To the best of our knowledge, this is the first research to apply negative sampling directly to the convolution operation.

2. *The DFS-based method can obtain samples with different semantics while containing complete knowledge of the entire graph that can better reflect the real-world situation.*

   We introduced DPP to select meaningful negative samples from the dark world of the whole graph. To the best of our knowledge, we are the first to introduce DPP to GCNs for negative sampling. We further presented a DFS-based heuristic approximation method to greatly reduce the computational cost. The experimental evaluations show that the proposed D2GCN consistently delivers better performance than alternative methods. In addition to greater predictive accuracy, the method also helps to prevent over-smoothing.

5.2 Further study

In future research, we will continue to investigate how to speed up the DPP sampling process in the algorithm. Another interesting follow-up work would be to investigate more effective aggregation of positive and negative samples as the current solution may lose some information when summing the samples together – especially when the samples are diverse.

Bibliography

[1] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations (ICLR), Toulon, France*, 2017.

[2] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining(KDD), Virtual Event, CA, USA*, pages 1666–1676, 2020.

[3] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[4] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence(AAAI), Austin, Texas, USA*, pages 2267–2273, 2015.

[5] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Trans. Neural Networks Learn. Syst.*, 30(11): 3212–3232, 2019.

[6] Keunwoo Choi, György Fazekas, Mark B. Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP), New Orleans, LA, USA,*, pages 2392–2396, 2017.

[7] Xin Chen, Jian Weng, Wei Lu, Jiaming Xu, and Jia-Si Weng. Deep manifold learning combined with convolutional neural networks for action recognition. *IEEE Trans. Neural Networks Learn. Syst.*, 29(9):3938–3952, 2018.

[8] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: laws, generators, and algorithms. *ACM Computing Surveys*, 38(1):1–69, 2006.

[9] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 32(1):4–24, 2021.

[10] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated graph sequence neural networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico*, 2016.

[11] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden*, pages 3634–3640, 2018.

[12] Floris Geerts, Filip Mazowiecki, and Guillermo A. Pérez. Let's agree to degree: comparing graph convolutional networks in the message-passing framework. In *Proceedings of the 38th International Conference on Machine Learning (ICML), Virtual Event*, pages 3640–3649, 2021.

[13] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems(NIPS), Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

[14] Andriy Mnih and Koray Kavukcuoglu. Learning word embeddings efficiently with noise-contrastive estimation. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems(NIPS), Lake Tahoe, Nevada, United States*, pages 2265–2273, 2013.

[15] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *Nature*, 393(6684):440–442, 1998.

[16] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining(KDD), London, UK*, pages 974–983, 2018.

[17] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Las Vegas, NV, USA*, pages 779–788, 2016.

[18] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems(NIPS), Montreal, Quebec, Canada*, pages 91–99, 2015.

[19] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: deep neural networks with multitask learning. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML), Helsinki, Finland*, volume 307, pages 160–167, 2008.

[20] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies(NAACL-HLT),New Orleans, Louisiana, USA*, pages 2227–2237, 2018.

[21] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Boston, MA, USA*, pages 3431–3440, 2015.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention 18th International Conference(MICCAI), Munich, Germany, October*, pages 234–241, 2015.

[23] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[24] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks(IJCNN)*, volume 2, pages 729–734, 2005.

[25] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[26] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *2nd International Conference on Learning Representations(ICLR), AB, Canada*, 2014.

[27] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[28] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. Graph wavenet for deep spatial-temporal graph modeling. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 1907–1913, 2019.

[29] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, United States*, pages 1024–1034, 2017.

[30] Soumyasundar Pal, Florence Regol, and Mark Coates. Bayesian graph convolutional neural networks using node copying. *arXiv preprint arXiv:1911.04965*, 2019.

[31] Arman Hasanzadeh, Ehsan Hajiramezanali, Shahin Boluki, Mingyuan Zhou, Nick Duffield, Krishna Narayanan, and Xiaoning Qian. Bayesian graph neural networks with adaptive connection sampling. In *Proceedings of the 37th International Conference on Machine Learning(ICML), Virtual Event*, volume 119, pages 4094–4104, 2020.

[32] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *6th International Conference on Learning Representations(ICLR), Vancouver, BC, Canada*, 2018.

[33] Liyu Gong and Qiang Cheng. Exploiting edge features for graph neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Long Beach, CA, USA*, pages 9211–9219, 2019.

[34] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D. Yoo. Edge-labeling graph neural network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Long Beach, CA, USA*, pages 11–20, 2019.

[35] Jia Li, Yu Rong, Hong Cheng, Helen Meng, Wen-bing Huang, and Junzhou Huang. Semi-supervised graph classification: A hierarchical graph perspective. In *The World Wide Web Conference(WWW), San Francisco, CA, USA*, pages 972–982, 2019.

[36] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence(AAAI),New Orleans, Louisiana, USA*, pages 3546–3553.

[37] Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. Path integral based convolution and pooling for graph neural networks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020(NIPS), Virtual Event*, 2020.

[38] Claudio Gallicchio and Alessio Micheli. Graph echo state networks. In *International Joint Conference on Neural Networks(IJCNN),Barcelona,Spain*, pages 1–8, 2010.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[40] Rushil Anirudh and Jayaraman J Thiagarajan. Bootstrapping graph convolutional neural networks for autism spectrum disorder classification. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3197–3201. IEEE, 2019.

[41] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[42] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *8th International Conference on Learning Representations(ICLR), Addis Ababa, Ethiopia*, 2020.

[43] Yarin Gal, Jiri Hron, and Alex Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems(NIPS), Long Beach, CA, USA*, pages 3581–3590, 2017.

[44] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *Proceedings of the 7th International Conference on Learning Representations (ICLR), New Orleans, LA, US*, 2019.

[45] Boris Weisfeiler and Andrei Leman. The reduction of a graph to canonical form and the algebra which appears therein. *NTI, Series*, 2(9):12–16, 1968.

[46] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence(AAAI),Phoenix, Arizona, USA*, pages 1145–1152, 2016.

[47] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence(IJCAI), Stockholm, Sweden*, pages 2609–2615, 2018.

[48] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: data-driven traffic forecasting. In *6th International Conference on Learning Representations(ICLR), Vancouver, BC, Canada*, 2018.

[49] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence(AAAI), New Orleans, Louisiana, USA*, pages 7444–7452, 2018.

[50] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference(WWW), San Francisco, CA, USA*, pages 417–426, 2019.

[51] Chen Ma, Liheng Ma, Yingxue Zhang, Jianing Sun, Xue Liu, and Mark Coates. Memory augmented graph neural networks for sequential recommendation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence(AAAI), New York, NY, USA, February 7-12, 2020*, pages 5045–5052, 2020.

[52] Lei Chen, Le Wu, Richang Hong, Kun Zhang, and Meng Wang. Revisiting graph based collaborative filtering: A linear residual graph convolutional network approach. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence(AAAI), New York, NY, USA, February 7-12, 2020*, pages 27–34, 2020.

[53] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. In *8th International Conference on Learning Representations(ICLR), Addis Ababa, Ethiopia*, 2020.

[54] Gabriele Monfardini, Vincenzo Di Massa, Franco Scarselli, and Marco Gori. Graph neural networks for object localization. *Frontiers in Artificial Intelligence and Applications*, 141:665, 2006.

[55] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Honolulu, HI, USA*, pages 3097–3106, 2017.

[56] Yikang Li, Wanli Ouyang, Bolei Zhou, Jianping Shi, Chao Zhang, and Xiaogang Wang. Factorizable net: an efficient subgraph-based framework for scene graph generation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 335–351, 2018.

[57] Ashesh Jain, Amir Roshan Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), Las Vegas, NV, USA*, pages 5308–5317, 2016.

[58] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[59] Loïc Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *2018 IEEE Conference on Computer Vision and Pattern Recognition(CVPR),Salt Lake City, UT, USA*, pages 4558–4567, 2018.

[60] Linfeng Song, Yue Zhang, Zhiguo Wang, and Daniel Gildea. A graph-to-sequence model for amr-to-text generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics(ACL), Melbourne, Australia*, pages 1616–1626, 2018.

[61] Jasmijn Bastings, Ivan Titov, Wilker Aziz, Diego Marcheggiani, and Khalil Sima'an. Graph convolutional encoders for syntax-aware neural machine translation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing(EMNLP), Copenhagen, Denmark*, pages 1957–1967, 2017.

[62] Daniel Beck, Gholamreza Haffari, and Trevor Cohn. Graph-to-sequence learning using gated graph neural networks. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics(ACL), Melbourne, Australia*, pages 273–283, 2018.

[63] David Raposo, Adam Santoro, David G. T. Barrett, Razvan Pascanu, Tim Lillicrap, and Peter W. Battaglia. Discovering objects and their relations from entangled scene representations. In *5th International Conference on Learning Representations(ICLR), Toulon, France*, 2017.

[64] Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems(NIPS), Long Beach, CA, USA*, pages 4967–4976, 2017.

[65] Thomas N. Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S. Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th*

International Conference on Machine Learning(ICML), Stockholmsmässan, Stockholm, Sweden, volume 80, pages 2693–2702.

[66] Robert C Glen, Andreas Bender, Catrin H Arnby, Lars Carlsson, Scott Boyer, and James Smith. Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to adme. *IDrugs*, 9(3):199, 2006.

[67] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems(NIPS), Montreal, Quebec, Canada*, pages 2224–2232, 2015.

[68] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[69] Huaxiu Yao, Fei Wu, Jintao Ke, Xianfeng Tang, Yitian Jia, Siyu Lu, Pinghua Gong, Jieping Ye, and Zhenhui Li. Deep multi-view spatial-temporal network for taxi demand prediction. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence(AAAI), New Orleans, Louisiana, USA*, pages 2588–2595, 2018.

[70] Edward Choi, Mohammad Taha Bahadori, Le Song, Walter F. Stewart, and Jimeng Sun. GRAM: graph-based attention model for healthcare representation learning. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD), Halifax, NS, Canada*, pages 787–795, 2017.

[71] Jeremy Kawahara, Colin J Brown, Steven P Miller, Brian G Booth, Vann Chau, Ruth E Grunau, Jill G Zwicker, and Ghassan Hamarneh. Brainnetcnn: Convolutional neural networks for brain networks; towards predicting neurodevelopment. *NeuroImage*, 146:1038–1049, 2017.

[72] Dongkwan Kim and Alice H. Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *9th International Conference on Learning Representations(ICLR), Virtual Event, Austria*, 2021.

[73] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[74] Ming Ding, Jie Tang, and Jie Zhang. Semi-supervised learning on graphs with generative adversarial nets. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management(CIKM), Torino, Italy*, pages 913–922, 2018.

[75] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[76] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations(ICLR), San Diego, CA, USA*, 2015.

[77] J. Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. *Zeros of gaussian analytic functions and determinantal point processes*, volume 51 of *University Lecture Series*. American Mathematical Society, 2009.

[78] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 5(2-3):123–286, 2012.

[79] J Ben Hough, Manjunath Krishnapur, Yuval Peres, and Bálint Virág. Determinantal processes and independence. *Probability Surveys*, 3:206–229, 2006.

[80] Jennifer Gillenwater, Alex Kulesza, and Ben Taskar. Near-optimal MAP inference for determinantal point processes. In *Proceedings of 26th Annual Conference on Neural Information Processing Systems (NIPS), Lake Tahoe, Nevada, United States*, pages 2744–2752, 2012.

[81] Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of The 34th AAAI Conference on Artificial Intelligence (AAAI), New York, NY, USA*, pages 3438–3445, 2020.