UTS UNIVERSITY OF TECHNOLOGY SYDNEY

Deep Graph Neural Networks for Unsupervised Graph Learning

by Chun Wang

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Prof. Chengqi Zhang, Dr. Guodong Long and Dr. Shirui Pan

University of Technology Sydney
Faculty of Engineering and Information Technology

December 2020

Certificate of Authorship/Originality

I, Chun Wang declare that this thesis, is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology 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.

2021/06/20

ABSTRACT

Deep Graph Neural Networks for Unsupervised Graph Learning

by

Chun Wang

Graphs are widely used to represent networked data, which contains complex relationships among individuals, and therefore cannot be well represented by traditional flat-table or vector format. Network applications, like social networks or citation networks, have been developing rapidly in recent years. Consequently, graph learning has also attracted much more attention.

Unsupervised graph learning is an important branch of the field since label information is usually not easily accessible. It is much more challenging as unsupervised graph learning aims to model the networked data without training supervision. Associated downstream tasks of unsupervised graph learning may include clustering, link prediction, visualization, etc., which are also very popular in modern graph analysing.

To perform unsupervised graph learning, previous methods may (1) consider only one aspect of the graph information; (2) use shallow approaches to capture simple or linear relationships among the data; (3) separate graph embedding learning and the downstream task as two steps and learn sub-optimal results since the learned embedding could not best fit the downstream method; (4) not able to manage corrupted data and perform robust learning, and (5) not able to make use of side information. These limitations have held back the development of unsupervised graph learning.

Therefore, we aim to address of the following challenges in our research: (1) How to characterize the individual properties of each node, and at the same time capture complex relationships in the graph; (2) How to learn deep and informative representation for graph data; (3) How to perform end-to-end learning for a certain graph data-based task; and (4) How to deal with different types of abnormal graph data information.

In this thesis, we aim to perform effective graph learning, with deep graph neural networks in an unsupervised manner. Firstly, we propose a special marginalized graph autoencoder, to integrate both node content and graph structure information into a unified framework. We add noise to the graph data, and employ a marginalized process for efficient computation. By further stacking multiple layers of such autoencoder, we learn deep and informative unsupervised graph embedding for graph clustering; Secondly, we combine graph autoencoder with a self-training model, to conduct a goal-directed training framework. In such a process, the clustering and embedding learning are performed simultaneously. Both of them can benefit from the other, thereby learn better graph embedding and clustering. Facing possible data corruption, especially structural corruption for graph data, we develop a dual-autoencoder interaction framework Cross-Graph, which takes advantage of the deep learning memorization effect that DNNs fit clean and easy data first. Two autoencoders filter out untrusted edges alternatively and learn robust embedding from graphs with redundant edges. Finally, to take advantage of possible side information in graph learning, we also propose a contrastive regularized graph autoencoder, that can improve the unsupervised graph learning ability using constraint information. All these frameworks are validated with unsupervised tasks like clustering in the experiments.

Dissertation directed by Dr. Guodong Long, Dr. Shirui Pan and Prof. Chengqi Zhang

Australian Artificial Intelligence Institute, Faculty of Engineering and Information Technology

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisors: Dr. Guodong Long, Dr. Shirui Pan, and Prof. Chengqi Zhang. They have continuously provided me support and guidance in the past four years. I could not go through my Ph.D. study without their patient guidance, immense knowledge and help on all sides.

I am also grateful to Dr. Bo Han and Dr. Fan Zhang, they are both friends and mentors who have also provided me a lot of guidance and help during my research. And also thanks to Dr. Jing Jiang, who has supported me along with my study.

I would also like to thank my school mates and research fellows, who had a positive influence on my Ph.D. study, including but not limited to: Ruiqi Hu, Tao Shen, Xiaolin Zhang, Zonghan Wu, Lu Liu, Han Zheng, Fengwen Chen, and Mengyao Li. They are the ones who share both my joyful and stressful times and help me from different aspects.

Finally, I would like to thank my family. No words could possibly express my gratitude for their endless love, encouragement and support.

Chun Wang

Sydney, Australia, 2020.

List of Publications

[1] Chun Wang, Shirui Pan, Guodong Long, Xingquan Zhu, Jing Jiang, “MGAE: Marginalized graph autoencoder for graph clustering”, CIKM 2017 (CORE rank A, citations: 106)

[2] Chun Wang, Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Chengqi Zhang, “Attributed graph clustering: A deep attentional embedding approach”, IJCAI 2019 (CORE rank A*, citations: 56)

[3] Chun Wang, Bo Han, Shirui Pan, Jing Jiang, Gang Niu, Guodong Long “Cross-Graph: Robust and Unsupervised Embedding for Attributed Graphs with Corrupted Structure”, ICDM 2020 (CORE rank A*)

[4] Chun Wang, Shirui Pan, Celina P Yu, Ruiqi Hu, Guodong Long, Chengqi Zhang, ”Deep Neighbor-aware Embedding for Node Clustering in Attributed Graphs”, Pattern Recognition (CORE rank A*, under the 3rd review process with minor revision decision)

[5] Chun Wang, Shirui Pan, Bo Han, Guodong Long, Chengqi Zhang, ”Constrained Node Clustering for Attributed Graphs with Regularized Autoencoder”, (It will be submitted to a CORE rank A journal in 2021)

Contents

List of Figures

Abbreviation

NMF - Non-negative Matrix Factorization

GCN - Graph Convolutional Network

GAT - Graph Attention Network

DNN: Deep Neural Networks

AE - Autoencoder

GAE - Graph (Convolutional) Autoencoder

DA - Denoising Autoencoder

SDA - Stacked Denoising Autoencoder

DEC - Deep Embedded Clustering

KL - Kullback-Leibler

2D: Two-dimensional

SGD: Stochastic Gradient Descent

NMI: Normalized Mutual Information

AE: Average Entropy

ARI: Adjusted Rand Index

Nomenclature and Notation

Capital letters denote matrices.

Lower-case alphabets denote column vectors.

$(.)^T$ denotes the transpose operation.

I_n is the identity matrix of dimension $n \times n$.

$\mathbb{R}$, $\mathbb{R}^+$ denote the field of real numbers, and the set of positive reals, respectively.

n is the number of nodes in the graph.

m is the number of individual attributes of each node.

k is the number of clusters for the node clustering task.

X is a $n \times m$ matrix representing the node attribute information, each row x_n is a m-dimensional vector representing the attribute values of node n.

A is a $n \times n$ adjacency matrix representing the graph structure information, in which $A_{i,j} = 1$ representing there is an edge between node i and j, and $A_{i,j} = 0$ otherwise.

$\|.\|_F^2$ represents the squared Frobenius norm.

$\tilde{.}$ represents the corrupted or approximated version.

$tr(.)$ represents the trace of the matrix.

$Z^{(l)}$ is the learned node embedding in the l-th neural network layer.

Chapter 1

Introduction

1.1 Background

Graphs are essential data formats that can handle complex structured data. With the rapid development of network applications in recent years, the data mining of various scenarios like social networks, citation networks, or protein-protein interaction networks, all demand high-quality network analysis. These facts have made graph learning (Zhu et al., 2020; Wang et al., 2020a; Wu et al., 2020b) practical and important.

Most of these networked data basically consist of two sides of information, the property of each object, and the relationships among the objects. Therefore, most recent graph learning works focus on attributed graphs for more comprehensive graph information, which contain both information from the graph structure and the node attributes.

Since label information is usually expensive and not easily accessible, unsupervised learning for attributed graph (Wu et al., 2020a), and its related tasks like node clustering and link prediction, has attracted much attention. It is especially challenging since it aims to characterize the individual properties of each node and capture the pairwise structure relationship between nodes in the networks, effectively model the complex graph information without supervision.

Graph clustering, or more precisely node clustering for attributed graph, is the most common task to evaluate the unsupervised learning effectiveness, and also a task we mostly focuses on. It aims to partition the nodes in the graph into k disjoint groups, so that: (1) nodes within the same cluster are close to each other while nodes in different clusters are distant in terms of graph structure; and (2) nodes within the same cluster are more likely to have similar attribute values.

To enable such unsupervised learning, a vast number of algorithms and theories have been developed, most of which can be considered as shallow methods that directly perform clustering, learn simple or linear representations from the given graph.

Early methods on graphs mainly focus on graph structure only. They either capture the betweenness of edges (Girvan and Newman, 2002), compute eigenvectors of the graph Laplacian (Newman, 2006a,b), or employ belief propagation (Hastings, 2006) to exploit the graph structure. Recently, overlapping community detection algorithms, like BigClam (Yang and Leskovec, 2013) and AgmFit (Yang and Leskovec, 2012), have also been developed. However, these algorithms are suboptimal because they only use one channel of information and ignore the other.

When considering integrating both node content and network information, early methods in (Cai et al., 2008; Gu and Zhou, 2009) apply a nonnegative matrix factorization (NMF) strategy to decompose node content matrix and use graph structure as regularization terms. Relational topic model methods (Chang and Blei, 2009; Sun et al., 2009) try to simultaneously model both the links and the contents for clustering. Zhou et al. add virtual attribute nodes and edges in a network and compute the similarity based on the augmented network (Zhou et al., 2009). By considering a graph as a dynamic system and modeling its structure as a consequence of interactions among nodes, Liu et al. proposed an algorithm from the view of content propagation and then modeled the interactions with influence propagation and random walk (Liu et al., 2015) .

However, all these methods, explicitly or implicitly, only capture the linear or shallow relationships between node content and network information, while better non-linear or deep representation learning techniques are not extensively explored.

To solve this problem, more recent studies have resorted to deep learning techniques to learn compact representation to exploit the rich information of both the content and structure data (Wu et al., 2019b). Based on the learned graph embedding, simple task-oriented algorithms such as k-means for clustering are applied. Autoencoder is a mainstream solution for this kind of embedding-based approach (Cao et al., 2016; Tian et al., 2014), as the autoencoder based hidden representation learning approach can be applied to purely unsupervised environments.

Nevertheless, all these embedding-based methods are two-step approaches. The drawback is that the learned embedding may not be the best fit for the subsequent graph clustering task, and the graph clustering task is not beneficial to the graph embedding learning. To achieve mutual benefit for these two steps, a goal-directed training framework is highly desirable.

Moreover, most of these embedding methods are based on the assumption that they have no difficulty accessing perfect graph-structure data. This assumption is too ideal to hold in real-world problems and may limit the efficacy of the learnt embedding. Also, these methods did not consider the situation if there could be some side information provided to help with the learning process.

1.2 Research Objectives

We highlight that the aims of this research are to perform effective unsupervised graph learning for tasks like clustering. Based on the above observations, we could summarize that previous graph learning methods are not efficient and effective enough, because they are not able to (1) learn from different aspects of the graph information; (2) take advantage from deep learning; (3) perform goal-directed graph learning; and (4) deal with unconventional data, including imperfect data and extra data. Therefore, to take a step forward in this area, we realize that we need to confront the following challenges:

1) How to integrate both graph structure and node content information for graph learning;
2) How to learn deep and informative representation;
3) How to design goal-directed framework for joint embedding learning and downstream tasks like clustering;
4) How to deal with different unconventional conditions like corrupted graph data;
5) How to make use of available side information.

Specifically, we conduct research studies to achieve our objectives, which could be separated into the following topics:

i. To address the challenge (1) and (2), we propose a marginalized graph autoencoder (MGAE) for graph clustering. Our algorithm takes the graph structure and content as input and learns a content and structure augmented autoencoder upon them, with the graph convolutional network (GCN) as a building block. To learn a better representation from the graph autoencoder, we further corrupt the content features with noise and propose to marginalize noise for efficient computation. By stacking multiple layers of graph autoencoder, our algorithm can further learn a deep representation for network nodes. Finally, the learned representation is refined and fed into the spectral clustering framework for the final clustering results.

ii. For challenge (3), we propose a Deep Neighbor-aware Embedded Graph Clustering framework (DNEGC) with two variants, namely DNEGC-Att (with graph attentional autoencoder) and DNEGC-Con (with graph convolutional autoencoder). To exploit the interrelationship of various-typed graph data, we develop a neighbor-aware graph autoencoder to learn latent representation, which integrates both content and structure information. The encoder progressively aggregates information from its neighbor via a convolutional style or an attentional mechanism, and multiple layers of encoders are stacked to build a deep architecture for embedding learning. The decoder on the other side, reconstructs the topological graph information and manipulates the latent graph representation. Furthermore, a carefully designed self-training module, which takes the "*confident*" clustering assignments as soft labels, is employed to guide the optimizing procedure. By forcing the current clustering distribution to approach a hypothetical better distribution, in contrast to the separated two-step embedding learning-based methods, this specialized clustering component makes it possible to simultaneously learn the embedding and perform clustering in a unified framework, thereby achieving better clustering performance.

iii. To deal with challenge (4), we propose a novel Cross-Graph framework, to learn robust graph embedding, strengthened against structural corruption. We use autoencoders to perform purely unsupervised learning. Our autoencoder-based method can learn effective embedding without access to, not only the label guidance, but also clean graph-structure data. We are inspired by the Co-training approach (Blum and Mitchell, 1998), and designed a dual graph interaction framework called Cross-Graph Learning. Based on the deep learning memorization effect that deep neural networks fit clean data first (Arpit et al., 2017; Zhang et al., 2016), we maintain two autoencoders and update them alternatively. In each iteration, since the trustworthy edges fit faster and will be closer to the ground-truth, the two autoencoders can evaluate the reliability of every graph edge with their reconstructed graph structure. Each autoencoder then updates its structure by slightly devaluing those distrusted edges. This updated graph structure is then passed to its peer-autoencoder, working as a provided "opinion" on how the real structure should present. The peer-autoencoder would take this updated structure as the input to the next iteration. Through the learning process, those redundant edges will be devalued faster, over and over again, and eventually filtered out. Since the two autoencoders have different embedding abilities, different types of corruption may be selected out, including some misjudgments. Meanwhile, benefit from our interactive process, these misjudgments caused by a single autoencoder, can be reduced by its peer one. This fact further strengthened the robustness of our model.

iv. To deal with the last challenge, we propose a constrained node clustering framework for attributed graphs which can improve the clustering performance using pair-wise constraint information. To explore the interaction between the graph structure and node content, we employ a graph convolution-based autoencoder, which learns node representation from the graph. A contrastive loss-based graph regularizer is further designed, to manipulate the embedding learning according to the prior pair-wise limitation. The learned embedding could therefore involve the constraint information. The pairs of nodes indicated to belong to the same community by the prior will learn similar embedding in the latent space, and further naturally assigned to the same cluster.

1.3 Thesis Organization

This thesis is organised as follows:

- *Chapter 2*: This chapter presents a survey of recent unsupervised graph learning models, and also researches that are related to our proposed frameworks.
- *Chapter 3*: In this chapter, we introduce our first work, "MGAE: Marginalized Graph Autoencoder for Graph Clustering", which designs a marginalized autoencoder to integrate graph structure and node content information for clustering-oriented graph embedding learning.
- *Chapter 4*: This chapter presents another proposed model for graph clustering: "Deep Neighbor-aware Embedding for Node Clustering in Attributed Graphs". It adopts a self-training module, to jointly learn graph embedding and clustering simultaneously, and perform goal-directed learning for graph clustering.
- *Chapter 5*: We further consider the data corruption problem in this chapter. We design a "Cross-Graph: Robust and Unsupervised Embedding for Attributed Graphs with Corrupted Structure" to deal with redundant edge problem.
- *Chapter 6*: We present "Constrained Graph Clustering with Contrastive Regularized Autoencoder" to make use of constraint information to improve the graph learning.
- *Chapter 7*: We conclude the thesis in this chapter with a summary of our research content and contribution.

Chapter 2

Literature Review

In the Literature Review chapter, we first briefly review the research field, from the general deep neural networks for graph data, to graph embedding models and especially those for unsupervised learning tasks like node clustering. Then, we introduce particular techniques with related papers that we employ in our proposed algorithms. Finally, we list those representative algorithms we use as baselines in our experiments.

2.1 Graph Learning Overview

2.1.1 Deep Neural Networks for Graphs

Deep learning has made remarkable achievements in many domains like voice recognition and image processing. Recently deep learning has also been generalized to graph structured data (Wu et al., 2019b).

The graph convolutional network, in particular, attracts wide attention in the community. Bruna et al. made the first attempt as we are aware of in (Bruna et al., 2013; Henaff et al., 2015). By using the recurrent Chebyshev polynomials, Defferrard et al. (Defferrard et al., 2016) further optimized the filtering scheme and avoided the expensive computation of the Laplacian eigenvectors. Graph convolutional networks (GCN) (Kipf and Welling, 2016a) further simplifies the filtering for only 1-step neighborhood nodes and, convolution is thereby considered as a multiplication of the Fourier-transform of a signal in the spectral domain. There are also several other recent works using convolution on graphs (Atwood and Towsley, 2016; Duvenaud et al., 2015), vary as different convolutional filtering schemes are used.

Graph attention can be considered a special kind of graph convolution which place more value on the most relevant parts. Graph attention networks (GATs) is presented for node classification of graph-structured data (Velickovic et al., 2017). It performs self-attention on the graph, computing the hidden representation of each graph node by attending over its neighbor nodes.

2.1.2 Graph Embedding Models

Without regard to more complex structure for certain tasks like heterogeneous graphs (Zhu et al., 2019b) or time series-oriented graphs (Wu et al., 2020c), based on the information available, graph embedding models can be separated into two categories: topological and attributed graph.

Topological embedding models are provided only with the graph structure information and subsequently focus on exploring and preserving this information. DeepWalk (Perozzi et al., 2014) uses random walk to generate context for graph nodes embedding learning; matrix factorization approaches like M-NMF (Wang et al., 2017b) and HOPE (Ou et al., 2016) learn graph representation from the transformation of the adjacency matrix; probabilistic models, like LINE (Tang et al., 2015) and node2vec (Grover and Leskovec, 2016), have also been developed.

Later, graph research focuses more on attributed graph embedding, since it has proven effective when node attributes are available. Many algorithms have been proposed to exploit and embed them simultaneously. TADW (Yang et al., 2015) uses a matrix factorization approach to model the content and graph structure interaction.

These models employ various approaches. However, these are generally old or traditional approaches that cannot compete with recent deep embedding methods.

Also, these models all consider their data real and clean and, as such, are vulnerable to corruptions.

2.1.3 Node Clustering in Graphs

Node clustering has been a long-standing research topic in the graph domain. Early methods have taken various shallow approaches to node clustering. Girvan and Newman used centrality indices to find community boundaries and detect social communities (Girvan and Newman, 2002). Hastings applied belief propagation to community detection and determined the most likely arrangement of communities (Hastings, 2006). Newman computed the eigenvectors of the graph Laplacian to perform clustering (Newman, 2006a,b).

To handle attributed graphs with both content and structure information, NMF-based methods (Cai et al., 2008; Gu and Zhou, 2009), probabilistic model (Cohn and Hofmann, 2001), relational topic models (Sun et al., 2009; Chang and Blei, 2009), and content propagation (Liu et al., 2015) have also been widely used.

The limitations of these methods are that (1) they only capture either parts of the network information or shallow relationships between the content and structure data, and (2) they are directly applied on original sparse graphs. As a result, these methods cannot effectively exploit the topological information or the interplay between the graph structure and the node content.

Benefiting from the development of deep learning, graph node clustering has progressed significantly in recent years. Many algorithms employ a deep architecture, adopting either sparse autoencoder (Tian et al., 2014; Hu et al., 2017) or denoising autoencoder (Cao et al., 2016) to exploit the deep structure information for clustering. For attributed graphs, graph convolution-based autoencoders are also developed (Kipf and Welling, 2016b), and are combined with marginalized process (Wang et al., 2017a), adversarial regularization (Pan et al., 2018, 2019), etc.

for node clustering, link prediction, and other unsupervised tasks. However, these methods are two-step methods, whereas the algorithm presented in this paper is a joint learning approach.

2.2 Techniques Employed in Our Frameworks

2.2.1 Autoencoder

Autoencoder has been a widely used tool in the deep learning area long before adopted to the graph domain, especially for unsupervised learning tasks such as clustering and anomaly detection (Zhou and Paffenroth, 2017). The autoencoder basically consists of an encoder mapping the input feature X to some hidden representation $h(X)$ and a decoder mapping it back to reconstruct the input feature. The parameters of the autoencoder can be learned by minimizing the reconstruction error.

Denoising autoencoders (DAs) corrupt the input features and then try to learn a hidden representation which best reconstructs the original input from its corrupted version also by minimizing the reconstruction error. Further, by regarding every hidden representation as the input feature of the next DA and stacking these hidden representations learned into a single matrix, the stacked denoising autoencoder (SDA) proposed by Vincent et al. could obtain a higher-level representation (Vincent et al., 2008).

A marginalized denoising autoendcoder (mSDA) was proposed in (Chen et al., 2012) inspired by SDA. It uses linear denoisers as the basic building blocks and marginalizes out the random feature corruption. It avoids iterative optimization and significantly simplifies parameter estimation while retaining a state-of-the-art classification performance.

Benefitting from the acceleration of mSDA, Shao et al. proposed a deep structure with a linear coder building block for graph clustering, which jointly learns the feature transform function and codings (Shao et al., 2015).

However, this series of autoencoder-based methods handles only one channel of data and therefore could not best suit attributed graph problems.

2.2.2 Deep Clustering Algorithms

Deep Embedded Clustering (DEC) is an autoencoder-based clustering technique for plain data (Xie et al., 2016). It employs a stacked denoising autoencoder learning approach. After obtaining the hidden representation of the autoencoder with the pre-train, rather than minimizing the reconstruction error through a decoder, the encoder pathway is fine-tuned by a defined Kullback-Leibler divergence clustering loss. Guo et al. considered that the defined clustering loss could corrupt the feature space, leading to non-representative features and a reduction in clustering performance. They improved the DEC algorithm by adding back the decoder and minimizing the reconstruction error as well as the clustering loss (Guo et al., 2017a).

There have since been many algorithms based on such deep clustering framework (Dizaji et al., 2017; Guo et al., 2017b). However, as far as we know, they are only designed for data with flat-table representation. For graph data, complex structure and content information need to be carefully exploited, and end-to-end clustering for graph data is still an open problem in this area.

2.2.3 Co-training based Methods

To learn from noisy data, a promising approach is to filter out some clean data for training. The Co-training strategy (Blum and Mitchell, 1998), though not focusing on this problem, has made it easy to solve by training two learning algorithms separately and has derived many methods based on this approach. MentorNet (Jiang et al., 2018) pre-trains an extra network to select clean instances and supervise the training of the StudentNet; Decoupling (Malach and Shalev-Shwartz, 2017) trains two networks simultaneously, and only uses those samples leading to different predictions in the two networks for updating; Co-teaching (Han et al., 2018) also maintains two networks, and each network selects small-loss instances in each mini-batch to guide its peer-network.

Unfortunately, all these methods are designed for one channel of plain data, like images, and not well-suited to the graph domain.

2.2.4 Outlier-Oriented Graph Models

Outlier has been a significant research topic for decades. For plain graph, Zhang et al. proposed an outlier edge detection method (Zhang et al., 2017) by comparing the actual and expected number of edges in an ego-network.

Specially for outliers in attributed graph data, most previous methods only try to detect outlier nodes. Radar (Li et al., 2017) defines three kinds of outlier nodes in the attributed graph, and detects them by analyzing the residuals of the attribute information; ALAD (Liu et al., 2017) exploits the attributed graph information by context extraction and a mini-batch SGD-based method is developed to accelerate its optimization.

Nevertheless, these methods only consider detecting isolated outlier, not robust learning on the corrupted graph, and they are not able to recover the graph from large-scale corruption. What is more, they investigate only outlier nodes, neglecting the possibility of edges being the outlier.

2.2.5 Contrastive Learning

Contrastive learning (Hadsell et al., 2006) aims to find similar and dissimilar things, and learn discriminative representations contrasting positive and negative samples. It is widely used in computer vision tasks to classify between different images, by contrasting different views of the images (He et al., 2020; Wu et al., 2018). Recently it has also been extended to learn graph representation (Qiu et al., 2020). The key to contrastive learning is a score function that measures the similar and dissimilar between features. In our proposed work, we are inspired by the loss function from InfoNCE (Oord et al., 2018).

2.2.6 Constrained Clustering

Pair-wise constraints have been long a piece of additional accessible information that can help clustering tasks. Constrained clustering has been widely used in clustering plain data (Zhu et al., 2015; Pathak et al., 2015; Ren et al., 2019).

Recently some methods have tried to help graph-based analyzing using constraint information. Eaton and Mansbach used a statistical physics model to combine external information into the community detection (Eaton and Mansbach, 2012). Ma et al. tried to encode the pair-wise constraints into the adjacency matrix and then factorized it for node embedding (Ma et al., 2010); Zhang et al. directly modified the graph structure information using the must-link and cannot-link priors (Zhang, 2013; Zhang et al., 2013); Yang et al. proposed a unified framework to integrate topological information with must-link priors (Yang et al., 2014).

Unfortunately, most of these methods are designed for plain data like images or structural-only network data. None, to the best of our knowledge, can be well-suited to the attributed graph problems.

2.3 Baseline Methods

We summarize all baselines we employ in our experiments for comparison with our proposed methods. For a particular method we propose, not all baselines are employed for comparison, since only some of them best suit the problem setting and can perform well.

Methods Using Structure or Content Only

- **K-means** is the base of many clustering methods. Many advanced clustering algorithms involve some kind of transformation of k-means clustering or use k-means on their embeddings. Here we run k-means on our original content data as a benchmark.

- **Spectral clustering** uses the eigenvalues of the similarity matrix to perform dimensionality reduction before clustering and is widely used in graph clustering.

- **Big-Clam** (Yang and Leskovec, 2013) is a non-negative matrix factorization approach for community detection which takes only the network structure into account.

- **DeepWalk** (Perozzi et al., 2014) is a structure-only representation learning method. It obtains random walks on graphs and then trains the representation through neural networks.

- **GraphEncoder** (Tian et al., 2014) employs deep learning into graph clustering by training a stacked sparse autoencoder and gets new representation for clustering.

- **DNGR** (Cao et al., 2016) is recent work which uses stacked denoising autoencoders and encodes each vertex into a low dimensional vector representation.

- **M-NMF** (Wang et al., 2017b) is a Nonnegative Matrix Factorization model targeted at community-preserved embedding.

Methods Using Both Structure and Content

- **Circles** (Leskovec and Mcauley, 2012) is an attributed graph clustering algorithm which represents overlapping hard-membership approaches for graph clustering.

- **RTM** (Chang and Blei, 2009) is a relational topic model capturing both structure and content information to learn the topic distributions of documents.

- **RMSC** (Xia et al., 2014), the robust multi-view spectral clustering method via low-rank and sparsity decomposition, tries to recover a shared low-rank transition probability matrix for clustering using a transition probability matrix from each view. We regard structure and content data as two views of information.

- **TADW** (Yang et al., 2015), text-associated DeepWalk. It re-interprets DeepWalk as a matrix factorization method and adds the text features of vertices into representation learning.

- **GAE & VGAE** (Kipf and Welling, 2016b) are representation learning algorithms. They combine the graph convolutional network with the (variational) autoencoder.

- **ARGA & ARVGA** (Pan et al., 2018) are graph convolutional autoencoder-based methods that manipulate GAE & VGAE learned embedding with an adversarial regularizer.

- **AGC** (Zhang et al., 2019) is an adaptive graph convolution method that exploits high-order graph convolution and captures global cluster structure.

Chapter 3

Learning Using Two-aspects Information
MGAE: Marginalized Graph Autoencoder for Graph Clustering

In this chapter, we introduce our first proposed algorithm MGAE, which aims to address the first and second challenge mentioned in the Introduction: to integrate both graph structure and node content information for graph learning, and learn deep informative graph representation.

3.1 Background

Network applications, such as social networks (Wang et al., 2019), citation networks, and protein interaction networks, have emerged increasingly and have attracted much attention in the last decade. Unlike traditional data which are represented as a flat-table or vector format, networked data are naturally represented as graphs for characterizing the individual properties of each node and capturing the pairwise structure relationship between nodes in the networks. The complexity of networked data has imposed many challenges on machine learning tasks, such as graph clustering. Given a graph (network) with node content and structure (link) information, graph clustering aims to partition the nodes in the graph into a number of disjoint groups. This has become one of the most important tasks in many applications, such as community detection (Fortunato, 2010), customer group segmentation (Kim et al., 2006), and functional group discovery in enterprise social networks (Hu et al., 2016). The major challenge of graph clustering is how to effectively utilize the information contained in the graph.

Shallow Representation for Graph Clustering: To enable graph clustering, a vast number of algorithms and theories have been developed, most of which can be considered as shallow methods that directly perform clustering or learn simple or linear representations from the given graph.

Early clustering methods on graphs mainly focus on graph structure only. They either capture the betweenness of edges (Girvan and Newman, 2002), compute eigenvectors of the graph Laplacian (Newman, 2006a,b), or employ belief propagation (Hastings, 2006) to exploit the graph structure. Recently, overlapping community detection algorithms, like GDPSO (Cai et al., 2015), BigClam (Yang and Leskovec, 2013) and AgmFit (Yang and Leskovec, 2012), have also been developed; however, these algorithms are suboptimal because they only use one channel of information and ignore the other.

When considering integrating both node content and network information, early methods in (Cai et al., 2008; Gu and Zhou, 2009) apply a nonnegative matrix factorization (NMF) strategy to decompose node content matrix and use graph structure as regularization terms. Relational topic model methods (Chang and Blei, 2009; Sun et al., 2009) try to simultaneously model both the links and the contents for clustering. Zhou et al. add virtual attribute nodes and edges in a network and compute the similarity based on the augmented network (Zhou et al., 2009). By considering a graph as a dynamic system and modeling its structure as a consequence of interactions among nodes, Liu et al. proposed an algorithm from the view of content propagation and then modeled the interactions with influence propagation and random walk (Liu et al., 2015). However, all these methods, explicitly or implicitly, only capture the linear or shallow relationships between node content and network information, while better non-linear or deep representation learning techniques were not extensively explored.

Deep Representation for Graph Clustering: Deep learning sheds light on modeling nonlinear or complex relationships, which has been successfully applied in many domains, such as speech recognition (Dahl et al., 2012), computer vision (Lawrence et al., 1997), and network representations (Pan et al., 2016). Of the deep learning methods, the **autoencoder** is the most commonly used approach for situations such as clustering where label information is unavailable, as the autoencoder based representation learning approach can be applied to purely unsupervised learning. There are indeed several existing autoencoder based deep methods for graph clustering (Tian et al., 2014; Cao et al., 2016). By showing that autoencoders and spectral clustering have the same optimization objectives, Tian et al. proposed to learn a non-linear mapping from the original graph before applying the K-means algorithm (Tian et al., 2014). By using a random surfing model to capture graph structural information, Cao et al. proposed a deep graph representation model for clustering (Cao et al., 2016). However, these approaches can only handle one kind of information (structure), and the underlying architecture cannot handle the complex structure and content information as a whole.

Motivated by these observations, we address the following challenges in dealing with graph clustering in the deep learning (*e.g.*, autoencoder) framework.

- **Content and Structure Integration:** Graph data have rich and complex information where content and structure information are inter-dependent. How to effectively integrate both structure and content information in a unified framework, and also analyze the interplay between content and structure?
- **Deep Representation for Graph Clustering:** Deep and nonlinear representation have achieved impressive results in many supervised learning tasks. How to learn an informative representation on graph data for the task of clustering?

To address the first challenge, in this paper, we propose a content and structure augmented autoencoder for graph clustering. Instead of learning a fully connected layer from the content or/and structure information, we develop a convolutional network as our building block in the autoencoder architecture. Our convolutional network combines both structure and content information, and performs the *convolution* operation in the spectral domain, which is motivated by the most recently developed graph convolutional network (GCN) (Kipf and Welling, 2016a). In GCN, the convolution is considered to be multiplication of the Fourier-transform of a signal, and it has proved very effective in classification tasks in graphs. In this paper, we further extend it into graph clustering, a purely unsupervised task in data mining.

When content and structure information are integrated, the interplay between them plays an important role in learning rich representation for graph clustering. We come up with the idea that the disturbance caused by random noise in training provides a more effective representation. In existing representation learning, the setting is rather *static*, where the structure and/or content are given and directly fed into the algorithms (Yang et al., 2015; Tian et al., 2014; Cao et al., 2016). We argue that such a simple solution can only provide a simple integration of information from a structure and content perspective, but cannot effectively exploit the interplay between them, and hence may result in suboptimal performance in representation and clustering. In our paper, we propose a *marginalization* process, *marginalized graph autoencoder*, which introduces some sort of "dynamics" by respectively adding random noises many times to the content information. The effectiveness of marginalization is illustrated in Fig. 3.1. Our marginalized autoencoder provides a number of advantages for graph clustering: (1) the marginalized process enables the interplay between content and structure, which is sufficiently exploited, resulting in better results; (2) by adding random corruptions (masking some feature values as 0) into the graph content information multiple times, the dataset is considerably

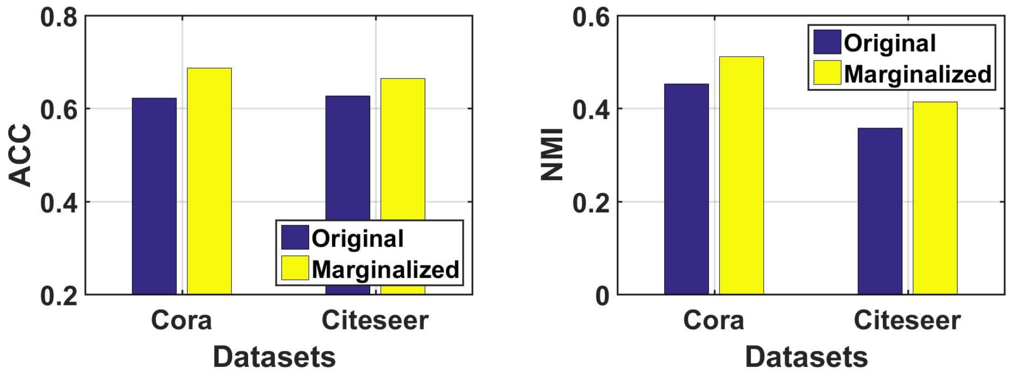


Figure 3.1 : The effectiveness of using marginalization for graph clustering. Marginalization introduces a small amount of disturbance to the node content, resulting in a dynamic environment for node content and structures to interact. Because the optimization process is well informed in relation to data disturbance, marginalization will cancel out the disturbance and the underlying graph autoencoders can learn optimized outcomes. Results are based on the accuracy (ACC) and normalized mutual information (NMI) of the spectral clustering before and after marginalization.

enlarged, which enables our algorithm to be trained with a larger dataset; (3) compared to traditional autoencoders (where dropout techniques are employed), which requires iteratively feeding the data to learn the neural networks in multiple epochs, the marginalized process enables the derivation of a closed form solution for our autoencoder, providing a much more efficient solution.

For the second challenge, we stack multiple layers of graph autoencoder to build a deep architecture for learning effective representation. Each graph autoencoder is trained sequentially with corrupted content information, and optimization is performed to minimize the reconstruction error between the encoded content and clean content information without corruption. After obtaining the representation of each node, we feed it into a spectral clustering algorithm to get the graph clustering results. As the building block in the autoencoder of our algorithm is the spectral convolutional network algorithm which performs in the spectral domain, employing spectral clustering on the representation turns out to be a good solution for graph clustering.

To summarize, in this paper, we propose a marginalized graph autoencoder (MGAE) for graph clustering. Our algorithm takes the graph structure and content as input and learns a content and structure augmented autoencoder upon them, with the graph convolutional network (GCN) as a building block. To learn a better representation from the graph autoencoder, we further corrupt the content features with noise and propose to marginalize noise for efficient computation. By stacking multiple layers of graph autoencoder, our algorithm can further learn a deep representation for network nodes. Finally, the learned representation is refined and fed into the spectral clustering framework for the final clustering results. Experimental results on real-life graph datasets validate our designs.

Our contributions can be summarized as follows:

- We propose a graph autoencoder algorithm to effectively integrate both structure and content information in a deep learning framework. This approach essentially advances the deep learning research to graph clustering with node attributes.

- We propose a marginalization process to corrupt content features of graphs in our deep learning framework, which enables us to (1) exploit the interplay between content and structure information; (2) learn on a larger dataset; and (3) obtain a closed form solution in an autoencoder framework.

- While convolutional networks are mainly used in classification tasks, we take this a step further by using graph convolutional networks for learning graph representation for clustering.

- We conduct extensive experiments and compare to 12 algorithms in total. The results demonstrate that our algorithm significantly outperforms all state-of-the-art methods on three benchmark datasets.

3.2 Problem Definition

We consider graphs with node content in the paper. A graph is represented as $G = (V, E, X)$, where $V = \{v_i\}_{i=1,\cdots,n}$ consists of a set of vertices, $E = \{e_{ij}\}$ is a set of edges, and $X = \{x_1; \dots; x_n\}$ is a set of attribute values. $x_i \in \mathbb{R}^m$ is a real-value attribute vector associated with vertex v_i. Formally, the graph can be represented by two types of information, the content information $X \in \mathbb{R}^{n\times m}$ and the structure information $A \in \mathbb{R}^{n\times n}$, where A is an adjacency matrix of G and $A_{i,j} = 1$ if $e_{i,j} \in E$ otherwise 0 (we consider only attributed graph with undirected and unweighted edges in our setting).

Given a graph G, graph clustering **aims** to partition the nodes in G into k disjoint groups $\{G_1, G_2, \cdots, G_k\}$, so that: (1) vertices within the same cluster are close to each other while vertices in different clusters are distant in terms of graph structure; and (2) vertices within the same cluster are more likely to have similar attribute values.

3.3 Proposed Method

Graphs have rich information in terms of node content and structure interaction. To fully exploit this information, we propose an effective content and structure augmented autoencoder for graph clustering. Instead of learning a fully connected layer from the content and structure information, in this paper, we develop a convolution network as our building block in our neural network architecture. The graph convolutional network is inspired by the most recently developed graph convolution network (Kipf and Welling, 2016a) for classification tasks for graphs, which learns a *convolution* on the structure information with node content in the spectral domain. We extend it to a purely unsupervised clustering task. Then we reconstruct the corrupted node content features with a marginalized autoencoder. A stacked

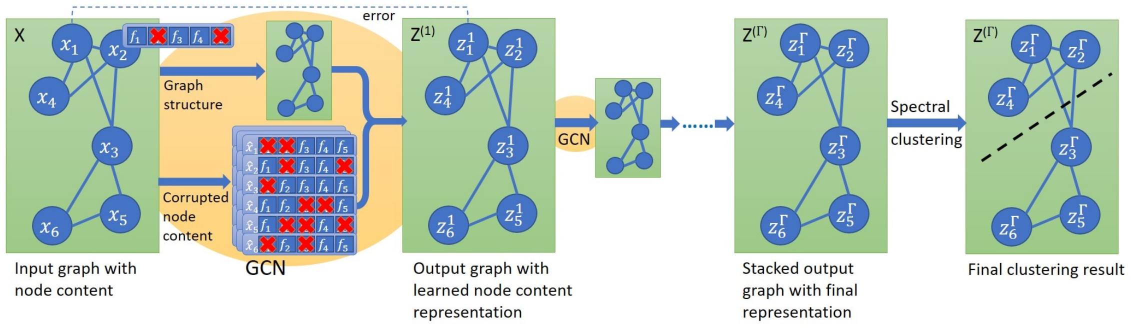

error
X
Graph structure
Corrupted node content
GCN
Input graph with node content
Z(1)
Output graph with learned node content representation
GCN
Z(Γ)
Stacked output graph with final representation
Spectral clustering
Final clustering result

Figure 3.2 : Conceptual framework of Marginalized Graph Autoencoder (MGAE) for graph clustering. Given a graph $G = (V, E, X)$, MGAE firstly learns a graph convolutional network (GCN) by using a mapping function $f(\widetilde{X}, A)$ based on the adjacency matrix A and corrupted node content $\widetilde{X}$. By minimizing the error between the output of GCN $f(\widetilde{X}, A)$ and X, we will get a latent representation $Z^{(1)}$. By stacking multiple GCNs and performing layer-wise training, our algorithm can learn a deep representation $Z^{(\Gamma)}$. Finally, a spectral clustering algorithm is performed on the refined representation $Z^{(\Gamma)}$ of the last layer.

architecture is further employed for learning a deep representation. Finally a successful clustering algorithm, spectral clustering, is used to obtain the final clustering results. Our framework is illustrated in Fig. 3.2.

The first component of our method is the construction of an autoencoder. We discuss the graph convolutional network of our autoencoder in Section 3.3.1, and then propose our marginalized graph autoencoder in Section 3.3.2.

3.3.1 Graph Convolutional Network

Graph convolutional networks (GCNs) define the concept of *convolution* from the spectral domain (Kipf and Welling, 2016a). Given the adjacency matrix A and content matrix X of graph G, a GCN aims to learn a layer-wise transformation by a spectral convolution function $f(Z^{(l)}, A)$, i.e.,:

$$Z^{(l+1)} = f(Z^{(l)}, A), \tag{3.1}$$

Here, $Z^l \in \mathbb{R}^{n \times m}$ (n nodes and m features) is the input for convolution, and $Z^{(l+1)}$ is the output after convolution. We have $Z^0 = X$ for our problem. If $f(Z^{(l)}, A)$ is well defined, one can build arbitrary deep convolutional neural networks efficiently.

Spectral convolution on a single feature: Consider each feature $s \in \mathbb{R}^n$ over all the nodes of the graph as a signal. The spectral convolution function $f(s, A)$ on a graph is defined as the multiplication of $s \in \mathbb{R}^n$ with a filter $g_\theta = \text{diag}(\theta)$ (parameterized by $\theta \in \mathbb{R}^n$) in the Fourier domain, such as:

$$g_\theta \star s = U g_\theta U^T s, \tag{3.2}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} = U \Lambda U^T$, with $D_{ii} = \sum_j A_{ij}$, I_N is the identity matrix, and Λ is a diagonal matrix in which the diagonal elements are the eigenvalues of L. g_θ can be considered as a function of the eigenvalues, i.e., $g_\theta(\Lambda)$.

It is expensive to compute the eigen-decomposition of L for large graphs. $g(\Lambda)$ can be approximated in terms of Chebyshev polynomials (Hammond et al., 2011):

$$g_\theta(\Lambda) \approx \sum_{\gamma=0}^{\Gamma} \theta_\gamma T_\gamma(\widetilde{\Lambda}), \tag{3.3}$$

where $\widetilde{\Lambda} = \frac{2}{\lambda_{max}} \Lambda - I_N$. λ_{max} is the largest eigenvalue of L. θ is the Chebyshev coefficients, $T_0(a) = 1$ and $T_1(a) = a$. By further limiting the layer wise convolution operation to $\Gamma = 1$ and approximate $\lambda_{max} \approx 2$, we could get a linear function on the graph Laplacian spectrum.

$$g_\theta \star s \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}) s, \tag{3.4}$$

where θ is the shared filter parameter over the whole graph and $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ can be approximated by $\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$ with $\widetilde{A} = A + I_N$ and $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$.

Spectral Convolution on multiple features: When considering multiple features (signals), i.e., $Z^l \in \mathbb{R}^{n \times m}$, the spectral convolution Eq. (3.4) can be generalized as:

$$H = g_W \star Z^{(l)} = \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} Z^{(l)} W, \tag{3.5}$$

where $W \in \mathbb{R}^{m \times m}$ is a matrix of filter parameters, and $H \in \mathbb{R}^{n \times m}$ is the convolved signal matrix, which can be computed efficiently in $\mathcal{O}(|E|d^2)$. Formally, then we have our layer-wise propagation rule for GCN,

$$f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} Z^{(l)} W^{(l)}). \tag{3.6}$$

Here, σ is a activation function such as $\mathrm{Relu}(t) = \max(0, t)$ or $\mathrm{sigmoid}(t) = \frac{1}{1+e^{-t}}$.

3.3.2 Marginalized Graph Autoencoder (MGAE)

With knowledge of the graph convolutional network, we now present our novel Marginalized Graph Autoencoder (MGAE) approach, which learns a hidden representation for each node in a network.

Content and Structure Augmented Autoencoder: Our model is built on a single-layer autoencoder. Different from the two-level encoder and decoder, it reconstructs the input $X = \{\boldsymbol{x_1}; \ldots; \boldsymbol{x_n}\} \in R^{n \times m}$ by using a single mapping function $f()$, that minimizes the squared reconstruction loss:

$$\|X - f(X)\|^2. \tag{3.7}$$

$f(X)$ is traditionally represented as $f(X) = \sigma(WX)$. By using graph convolution networks $f(X, A)$ in Eq. (3.6) instead of $f(X)$, our loss function becomes:

$$\|X - \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} X W\|^2 + \lambda \|W\|_F^2. \tag{3.8}$$

Here we use the linear activation function. $W \in R^{m \times m}$ is our parameter matrix. $\|W\|_F^2$ is a regularization term with coefficient λ being a tradeoff. Our idea is to learn a graph convolutional network from a given graph G, and minimize the reconstruction error of the autoencoder.

Marginalized Graph Autoencoder: Our autoencoder (Eq. (3.8)) provides an effective way to integrate both content and structure information. However, it cannot further exploit the interplay between content and structure information. To solve this problem, we propose a *marginalization* process for our graph autoencoder by randomly introducing some randomness into the content features (as shown in Fig. 3.2). Suppose $\widetilde{X} = \{\widetilde{\boldsymbol{x_1}}; \ldots; \widetilde{\boldsymbol{x_n}}\}$ is the corrupted version of the original input X. We can get the corrupted sample $\widetilde{\boldsymbol{x_i}}$ by randomly removing some features (setting them to 0) from $\boldsymbol{x_i}$.

Furthermore, to train the autoencoder, we need to pass the data multiple times. To this end, we generate corrupted X multiple times as input. Let us suppose we repeat it for m times as $[\widetilde{X}_1, \cdots, \widetilde{X}_m]$, then our final objective function becomes:

$$\frac{1}{m}\sum_{i=1}^{m} \|X - \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}\widetilde{X_i}W\|^2 + \lambda\|W\|_F^2. \tag{3.9}$$

If we further define $\hat{A} = \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$, our objective function becomes:

$$J = \mathrm{Tr}[(X - \hat{A}\bar{X}W)^T(X - \hat{A}\bar{X}W)] + \lambda\|W\|_F^2,$$

where $\mathrm{Tr}(\cdot)$ is the trace of a matrix. $\bar{X} = \frac{1}{m}\sum_{i=1}^{m}\widetilde{X_i}$. We can then get the solution for W:

$$L(W) = tr[X^TX - W^T\bar{X}^T\hat{A}^TX - X^T\hat{A}\bar{X}W$$
$$+ W^T\bar{X}^T\hat{A}^T\hat{A}\bar{X}W] + \lambda\|W\|_F^2,$$

$$\frac{\partial L}{\partial W} = -\bar{X}^T\hat{A}^TX - \bar{X}^T\hat{A}^TX + 2\bar{X}^T\hat{A}^T\hat{A}\bar{X}W + 2\lambda W = 0,$$

$$2(\bar{X}^T\hat{A}^T\hat{A}\bar{X} + \lambda)W = 2\bar{X}^T\hat{A}^TX,$$

$$W = \bar{X}^T \hat{A}^T X(\bar{X}^T \hat{A}^T \hat{A}\bar{X} + \lambda)^{-1}.$$

$$W = PQ^{-1} \text{ with } P = \bar{X}^T \hat{A}^T X \text{ and } Q = \bar{X}^T \hat{A}^T \hat{A}\bar{X} + \lambda.$$

Let us define $Y = \bar{X}^T \hat{A}^T$ for convenience, $Y = \{\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}\}$ and $\boldsymbol{y_i} \in R^d$, then P and Q can be expressed as

$$P = YX \text{ and } Q = YY^T + \lambda.$$

We are interested in the limit case where $m \to \infty$, so that we have enough samples to smooth out the corruption. In such a condition, matrices P and Q converge to their expected value by the weak law of large numbers and our W can be expressed as

$$W = E[P](E[Q])^{-1}. \tag{3.10}$$

For $E[Q]$, we have

$$E[Q] = \sum_{i=1}^{n} E[\boldsymbol{y_i}\boldsymbol{y_i}^T] + \lambda, \tag{3.11}$$

Let us assume that each feature is corrupted with a probability p, then the diagonal entries have a $(1 - p)$ probability of surviving the corruption while for the other entries, the probability is $(1 - p)^2$ as they have to survive two features at the same time. Therefore, we can form a corruption probability vector $\boldsymbol{u} = [1-p, \ldots, 1-p, 1]$ for each feature (the last item should be 1 as the constant feature is never corrupted), and then the expectation of the matrix Q is obtained as

$$E[Q]_{i,j} = \begin{cases} S_q\boldsymbol{u} + \lambda, \ i = j \\ S_q\boldsymbol{u}^2 + \lambda, \ i \neq j \end{cases},$$

where $S_q = X^T \hat{A}^T \hat{A} X$ is the uncorrupted version of YY^T.

Similarly, we have $E[p]_{i,j} = S_p\boldsymbol{u}$, where $S_p = X^T\hat{A}^TX$. Then we can directly obtain the weight for W according to Eq. (3.10) above.

Algorithm 1 Marginalized Graph Autoencoder

```
Require:
    $X$: the attribute matrix of the graph;
    $A$: Adjacency matrix of the graph;
    $p$: the corruption probability;
Ensure:
    $W$: the hidden representation in the autoencoder;
    $Z$: the reconstructed representation of the input X;
    $\widetilde{A} \leftarrow A + I_N$;    $\widetilde{D}_{ii} \leftarrow \sum_j \widetilde{A}_{ij}$;
    $\hat{A} \leftarrow \widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$;
    $\boldsymbol{u} \leftarrow [1-p, \ldots, 1-p, 1]$;
    $S_q \leftarrow X^T\hat{A}^T\hat{A}X$;
    $S_p \leftarrow X^T\hat{A}^TX$;
    $E[Q]_{i,j} \leftarrow \begin{cases} S_q\boldsymbol{u} + \lambda,\ i = j \\ S_q\boldsymbol{u}^2 + \lambda,\ i \neq j \end{cases}$;
    $E[P]_{i,j} \leftarrow S_p\boldsymbol{u}$;
    $W \leftarrow E[P]E[Q]^{-1}$;
    $Z \leftarrow \hat{A}XW$;
```

And the final graph representation Z is given as follows:

$$Z = \hat{A}XW. \tag{3.12}$$

The proposed marginalized graph convolutional autoencoder algorithm is given in Algorithm 1.

Stacked Graph Convolutional Autoencoder: Our model also has the capability of stacking multiple layers of autoencoders to create a deep learning architecture. We feed the output of the $(l-1)^{th}$ layer $Z^{(l-1)}$ as the input of the l^{th} layer. On the other hand, according to the rule $Z^l = \hat{A}X^lW^l$, each hidden representation W^l is learned to reconstruct Z^l from Z^{l-1} which is regarded as the corrupted form of Z^l. Finally, we regard the output of the last layer as the representation of the graph. The experiment results show that it improves the performance.

Attractive Properties: Our algorithm has a number of attractive properties:

i. **Interplay Exploitation.** Similar to the denoising autoencoder, the randomness injected into the content information allows us to better exploit the deep interplay between content and structure, which will help improve the performance.

ii. **Larger dataset.** The marginalization process enables our algorithm to be trained on a larger dataset due to the assumption that we repeat the corruption m times ($m \to \infty$).

iii. **High Efficiency**. Unlike traditional gradient descent based optimization algorithms that require large iterations to obtain convergence, benefit from our single-layer autoencoder and marginalized process, we can get a global optimal solution for the weight matrix.

3.3.3 Graph Clustering Algorithm

We have so far thoroughly described our autoencoder and gained the output Z^Γ, which can be regarded as our learned representation for the graph. As the new representation is ensured, we only need to run the clustering method. Here we use the spectral clustering algorithm in our work because the graph convolutional network is actually performed in the spectral domain, which makes the spectral clustering algorithm an ideal choice.

Algorithm 2 Clustering with MGAE Algorithm

```
Require:
  Graph $G$ with $n$ nodes, each node with $d$-dimension attribute value; Constructed attribute matrix $X \in R^{n\times d}$ and adjacency matrix $A \in R^{n\times n}$ of $G$; Number of clusters $k$; Corruption probability $p$; Stacked autoencoder layers number $\Gamma$; $Z^{(l)} \in R^{n\times d}$ is the output of layer $l$ except $Z^{(0)} = X$ is the input to the first layer.
Ensure:
  Final clustering results.
  for $l = 1$ to $\Gamma$ do
    1. Construct a single layer denoising autoencoder with input data $Z^{(l-1)}$;
    2. Learn the autoencoder output representation $Z^{(l)}$ according to Algorithm 1;
  end for
  $Z_0 \Leftarrow Z^{(\Gamma)}$;
  $Z_1 \Leftarrow Z_0 Z_0^T$;
  $Z_2 \Leftarrow \frac{1}{2}(|Z_1| + |Z_1^T|)$;
  Run spectral clustering on $Z_2$.
```

Table 3.1 : Experimental Results on **Cora** Dataset

	Information	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	Content	0.4922	0.3210	0.3680	0.3685	0.3693	1.7979	0.2296
Spectral	Structure	0.3672	0.1267	0.3180	0.1926	**0.9144**	2.4408	0.0311
Big-Clam	Structure	0.2718	0.0073	0.2812	0.1797	0.6452	2.6287	0.0011
GraphEncoder	Structure	0.3249	0.1093	0.2981	0.1817	0.8330	2.4598	0.0055
DeepWalk	Structure	0.4840	0.3270	0.3917	0.3612	0.4348	1.8140	0.2427
DNGR	Structure	0.4191	0.3184	0.3401	0.2660	0.4798	1.8816	0.1422
Circles	Both	0.6067	0.4042	0.4691	0.5010	0.4410	1.5563	0.3620
RTM	Both	0.4396	0.2301	0.3067	0.3319	0.2851	2.0208	0.1691
RMSC	Both	0.4066	0.2551	0.3305	0.2265	0.6410	2.1097	0.0895
TADW	Both	0.5603	0.4411	0.4805	0.3963	0.6289	1.6052	0.3320
VGAE	Both	0.5020	0.3292	0.3784	0.4087	0.3523	1.7541	0.2547
MGAE	Both	**0.6806**	**0.4892**	**0.5312**	**0.5648**	0.5016	**1.3315**	**0.4361**

In order to run spectral clustering, we need to refine our representation. We simply apply a linear kernel function $Z_1 = Z_0 Z_0^T$ to learn the pairwise relationship for the graph nodes. Then, similar to the multi-view representation learning algorithm clustering problems (Xia et al., 2014; Wang et al., 2016), we calculate $Z_2 = \frac{1}{2}(|Z_1| + |Z_1^T|)$ to make sure our representation is symmetric and nonnegative. Finally we run the spectral clustering procedure on Z_2 to obtain clusters results, and the whole clustering algorithm is summarized in Algorithm 2.

3.4 Experiments

3.4.1 Benchmark Datasets

Three benchmark datasets are used in our experiments.

Cora: A citation network with 2708 nodes and 5294 links between them, in which the nodes correspond to publications described by binary vectors of 1433 dimensions and classified into 7 classes.

Citeseer: A citation network consisting of 3312 publications labeled into 6 subfields. Each publication is described by a binary vector of 3703 dimensions and there are 4732 links between them.

Wiki: A network with 2405 documents and 17981 links. These documents have 4973-dimension vectors representing them and are divided into 19 classes.

Table 3.2 : Experimental Results on **Citeseer** Dataset

	Information	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	Content	0.5401	0.3054	0.4087	0.4052	0.4128	1.7543	0.2786
Spectral	Structure	0.2389	0.0557	0.2990	0.1786	0.9169	2.4451	0.0100
Big-Clam	Structure	0.2500	0.0357	0.2881	0.1817	0.6954	2.4614	0.0071
GraphEncoder	Structure	0.2252	0.0330	0.3007	0.1786	**0.9492**	2.4785	0.0100
DeepWalk	Structure	0.3365	0.0878	0.2699	0.2481	0.2998	2.3079	0.0922
DNGR	Structure	0.3259	0.1802	0.2997	0.1996	0.6093	2.1675	0.0429
Circles	Both	0.5716	0.3007	0.4238	0.4089	0.4399	1.7751	0.2930
RTM	Both	0.4509	0.2393	0.3421	0.3492	0.3353	1.9154	0.2026
RMSC	Both	0.2950	0.1387	0.3200	0.2037	0.8051	2.2770	0.0488
TADW	Both	0.4548	0.2914	0.4140	0.3119	0.6407	1.9160	0.2281
VGAE	Both	0.4670	0.2605	0.3452	0.3505	0.3403	1.8630	0.2056
MGAE	Both	**0.6691**	**0.4158**	**0.5257**	**0.5362**	0.5156	**1.4671**	**0.4250**

3.4.2 Baseline Methods

Twelve algorithms in total are compared in the experiments. As previously mentioned, advanced graph clustering algorithms differ as some use only network structure or node attributes, while others combine both. We take both classes of methods into consideration and compare our algorithms with the baselines as shown in the tables. Detailed description of these baselines could be found in Chapter 2.

For the representation learning based algorithms, such as DeepWalk, DNGR and TADW, we first get the representations from these algorithms, and then apply the K-means algorithm on the representations respectively, the best results are reported in the paper.

Table 3.3 : Experimental Results on **Wiki** Dataset

	Information	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	Content	0.4172	0.4402	0.2628	0.2108	0.4488	2.1241	0.1507
Spectral	Structure	0.2204	0.1817	0.1757	0.1055	0.5243	3.0891	0.0146
Big-Clam	Structure	0.1563	0.0900	0.1638	0.0946	0.6117	3.3690	0.0070
GraphEncoder	Structure	0.2067	0.1207	0.1717	0.1006	0.5936	3.2770	0.0049
DeepWalk	Structure	0.3846	0.3238	0.2574	0.2418	0.2779	2.4514	0.1703
DNGR	Structure	0.3758	0.3585	0.2538	0.2773	0.2353	2.2605	0.1797
Circles	Both	0.4241	0.4180	0.3035	0.3662	0.2592	2.0038	0.2420
RTM	Both	0.4364	0.4495	0.2481	0.1920	0.3539	2.0458	0.1384
RMSC	Both	0.3976	0.4150	0.2344	0.1672	0.3975	2.2201	0.1116
TADW	Both	0.3096	0.2713	0.2068	0.1203	**0.7538**	2.9080	0.0454
VGAE	Both	0.4509	0.4676	0.3278	0.3687	0.2957	1.8510	0.2634
MGAE	Both	**0.5293**	**0.5104**	**0.4294**	**0.5178**	0.3671	**1.6676**	**0.3787**

3.4.3 Evaluation Metrics & Parameter Settings

Evaluation Metrics: We use seven quality metrics (Xia et al., 2014) to measure the clustering result, namely Accuracy (ACC), Normalized Mutual Information (NMI), F-score (F), Precision (P), Recall (R), Average Entropy (AE) and Adjusted

Rand Index (ARI). A better clustering result should lead to a lower value of average entropy and higher values for the other metrics.

Parameter Settings: we set the corruption level p to 0.4, the number of layers to 3, and λ is fixed to 10^{-5} for our algorithm. For the other algorithms, we carefully select the parameters for each algorithm following the procedures in the original papers. For instance, in the Circle method, we set the regularization parameter $\lambda_\epsilon \in \{0, 1, 10, 100\}$, and choose the best values as the final results; in TADW, we select the dimension $k = 80$ and the harmonic factor $\lambda = 0.2$ for Cora and Citeseer but $k = 100, 200$ and $\lambda = 0.2$ for Wiki; for DNGR, we stack three layers for the autoencoder with 512 and 256 nodes in the hidden layers, etc.. For a fair comparison, we run each algorithm 50 times on each dataset and report the average results.

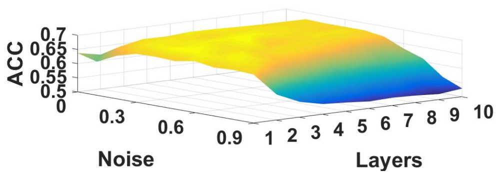
3.4.4 Experiment Results

We first compare MGAE with 11 baseline methods on graph clustering. After this, we perform a detailed analysis on the marginalization process, deep stacked architecture, time, and network visualization.

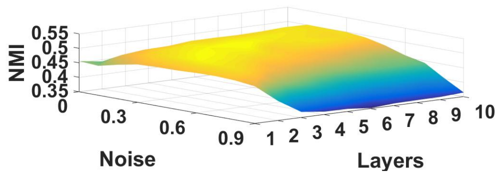
Clustering Performance Comparison:

Our experiment results on the three datasets are respectively summarized in Tables 3.1, 3.2, and 3.3, where the bold values in the text indicate the best results. It is obvious that our method outperforms all the baselines across different evaluation metrics except for Recall. In particular on the Cora data, our method's performance represents a relative increase of 12.18%, 10.90%, 10.55% and 12.73% w.r.t. accuracy, NMI, F-score and precision compared to the best baseline result.

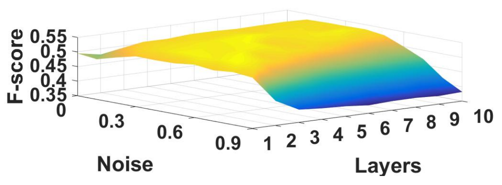
One side vs both side information: From the comparison, it is shown that methods using both structure and content information perform better than those using only one side of information in general. For instance, in the Cora dataset,



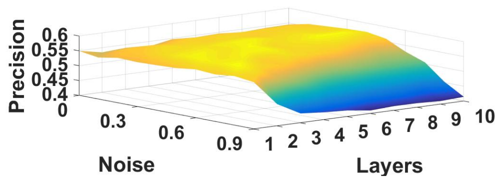
(a) ACC on Cora



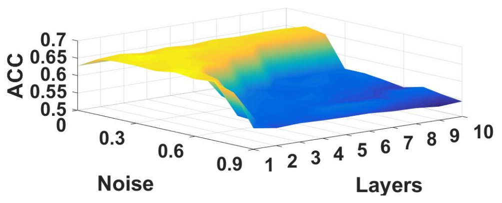
(b) NMI on Cora



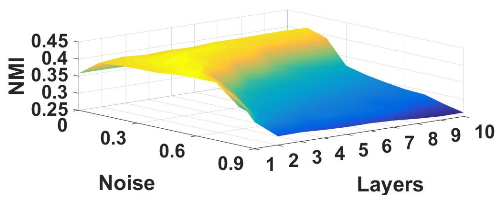
(c) F-score on Cora



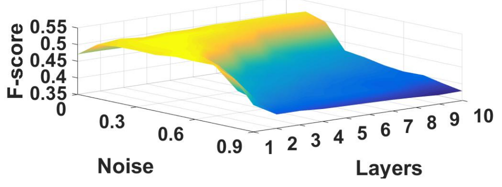
(d) Precision on Cora



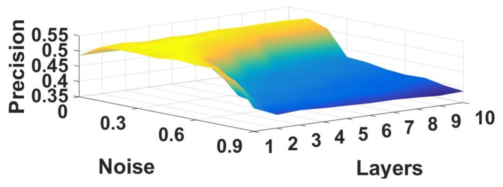
(e) ACC on Citeseer



(f) NMI on Citeseer



(g) F-score on Citeseer



(h) Precision on Citeseer

Figure 3.3 : Parameters study on noise and number of layers.

Circles and TADW algorithms significantly outperform the k-means, Spectral, and Big-Clam algorithms. This manifestation demonstrates that node content contains useful information for graph clustering.

Deep learning models: The results show that GraphEncoder and DNGR algorithms, both of which employ deep autoencoder architectures for graph clustering, are not necessarily an improvement over the other algorithms. This is because they only exploit the structure information and completely ignore the content information in the networks. In contrast, our MGAE algorithm achieves superior performance across all datasets because (1) we employ a graph convolutional network that effectively integrates both structure and content information in the spectral domain; (2) we use a deep marginalized architecture to learn a more informative representation, which results in better clustering results.

It is worth noting that our algorithm outperforms the VGAE algorithm, which is based on the variational autoencoder and convolutional network for graphs. This is because the marginalization process enables our algorithm to learn on much larger dataset ($m \to \infty$) and it can better exploit the interplay between content and structure information.

Marginalization & Deep Stacking Analysis

As the key innovation of the MGAE, the disturbance of the node content, through random noise, allows network structures and nodes to interact in a dynamic setting, so the graph autoencoder can learn effective feature representation. In this subsection, we study the impact of marginalization on MGAE performance by varying the disturbance noise level p and the number of stacked layers Γ, and report the performance of MGAE *w.r.t.* different performance metrics in Fig. 3.3.

Effectiveness of Marginalization: The results in Fig. 3.3 confirm that adding a certain level of disturbance noise (corruption) indeed helps improve the clustering performance. Overall, compared to noise-free settings, the best performance is likely to be achieved with a noise disturbance level between 0.3 to 0.5. The disturbance noise resembles a dynamic factor in our framework. Adding a small amount of noise to disturb the data is one way to generate different copies of data with minor variances. This helps to deliver a dynamic setting, allowing network node content and structures to interact. Because our marginalization process has sufficient knowledge of the disturbance noise, the corrupted data are canceled out through the optimization process and the graph autoencoder can leverage the dynamic data settings to learn better representation.

Effectiveness of Deep Stacking: Fig. 3.3 also show that when increasing the number of stacked layers Γ from 1 to 3, the performance of MGAE, including ACC, NMI, F-score and Precision, also increases accordingly. This validates that using a stacked architecture instead of a single-layer architecture can improve the clustering performance. However, when we continuously increase the number of layers Γ (from 6-9), the performance of MGAE reduces sharply in terms of all the evaluation metrics, especially on the Citeseer dataset. This is because a more complex architecture is more difficult to train and is subject to the risk of the information loss.

Time Consumption Analysis:

We also depict the training time for different methods in Fig. 3.4. We run all these methods on the same hardware and the time result is plotted in log scale. It can be observed that: 1) fundamental clustering methods such as k-means and spectral clustering are quite fast, whereas the recent developed methods are all slower as they have a much more complicated training procedure; 2) RMSC is the slowest of the observed methods because learning transition probability matrix via low-rank and sparse decomposition is time-consuming; 3) VGAE is a similar method to our MGAE, using a GCN-based autoencoder for attributed graph learning, however it is much slower, as it is based on a traditional kind of autoencoder and needs to iteratively train the graph convolutional network for optimization; 4) our MGAE is quite efficient compared to the other clustering methods, benefitting from its closed-form solution of the eigen-decomposition computing and avoidance of iterative optimization.

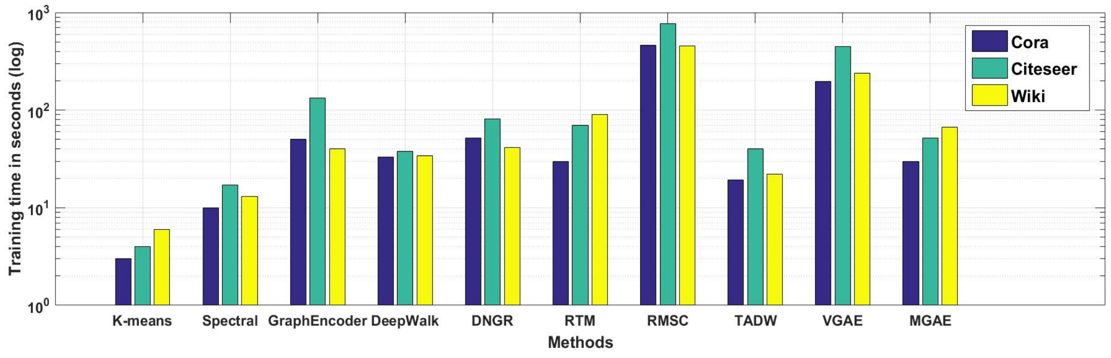


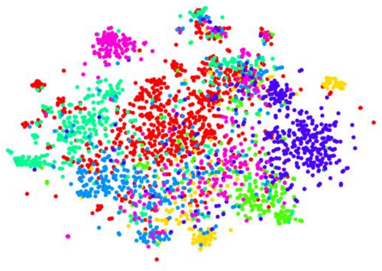
Figure 3.4 : Runtime comparisons of different methods.

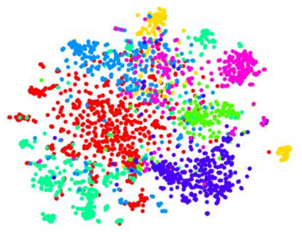
Network Visualization:

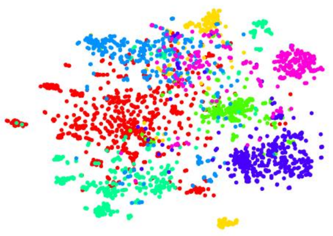
To intuitively show the quality of our learned representation, we follow (Tang et al., 2015) to learn low-dimensional representations for each node, and map Cora and Citeseer into 2D space in Fig. 3.5.

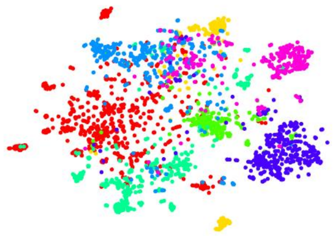
For both datasets, in order to show the need for our deep structure, we list and compare the visualization using representations learned from each stacked layer. We also show the results obtained by using the original content data representation. Similar to preprocessing for spectral clustering, we regard these representations as Z_0 in $Algorithm2$ and calculate Z_2 for visualization training.

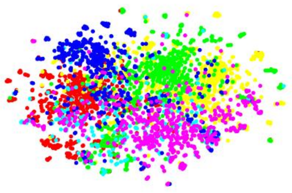
We can see from Fig. 3.5 that the visualization by the original representation is highly overlapping. The results obtained by our method are more clear with less overlapping and each node is better gathered to its own group. Moreover, as we stack our MGAE training layers from 1 to 3, the result becomes increasingly better as each group of nodes gradually gets away from each other.

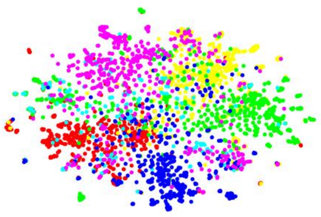

(a) Cora - before training

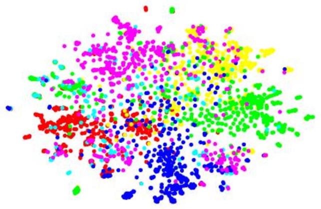

(b) Cora - 1 layer


(c) Cora - 2 layers


(d) Cora - 3 layers


(e) Citeseer - before training


(f) Citeseer - 1 layer


(g) Citeseer - 2 layers

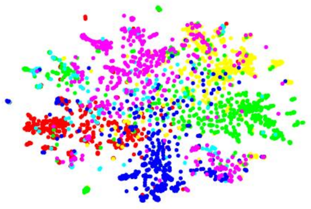

(h) Citeseer - 3 layers

Figure 3.5 : 2D visualization on representations learned from MGAE of various layers.

Chapter 4

Learning with Goal-directed Framework
Deep Neighbor-aware Embedding for Node Clustering in Attributed Graphs

We design another framework for node clustering here in this chapter. Aiming at the third challenge described in the Introduction, the proposed DNEGC is a goal-directed framework that can joint learn graph embedding and clustering simultaneously.

4.1 Background

The development of networked applications has resulted in an overwhelming number of scenarios in which data is naturally represented in graph format rather than flat-table or vector format. Attributed graph-based representation characterizes individual properties through node attributes, and at the same time captures the pairwise relationship through the graph structure. Many real-world tasks, such as the analysis of citation networks, social networks, protein-protein interaction and knowledge graphs (Ji et al., 2020), all rely on graph-data analytics skills. However, the complexity of graph structure has imposed significant challenges on these graph-related learning tasks, including clustering, which is one of the most popular topics.

Graph clustering aims to partition the nodes in the graph into disjoint groups (Bojchevski and Günnemann, 2018; Chen and Wu, 2017; Guo et al., 2018). Typical applications include community detection (Reihanian et al., 2018; Xie et al., 2018; Li et al., 2018a), group segmentation (Kim et al., 2006), and functional group discovery in enterprise social networks (Hu et al., 2016). Further for attributed graphs, a key problem is how to capture the structural relationship between nodes and exploit the node content information.

To solve this problem, more recent studies have resorted to deep learning techniques to learn compact representation or embedding to exploit the rich information of the graph data (Pan et al., 2016; Shen et al., 2018; Gao et al., 2018). Based on the learned graph embedding, simple clustering algorithms such as k-means are applied to obtain the clustering result. Autoencoder is a mainstream solution for this kind of embedding-based approach (Cao et al., 2016; Tian et al., 2014), as the autoencoder based hidden representation learning approach can be applied to purely unsupervised environments. Many autoencoder based graph clustering algorithms already exist: Tian et al. considered the similarity of autoencoder and spectral clustering and learned a latent representation for clustering through sparse autoencoder(Tian et al., 2014). Cao, Lu, and Xu proposed a deep graph representation model for clustering by capturing structure information through random surfing (Cao et al., 2016). The recently developed GAE and VGAE (Kipf and Welling, 2016b) based on graph convolutional network (GCN) can also be adopted for graph clustering analysis.

Nevertheless, all these embedding-based methods separate the embedding learning and clustering as two steps. The drawback is that the learned embedding may not be the best fit for the subsequent graph clustering task, and the graph clustering task is not beneficial to the graph embedding learning. To achieve mutual benefit for these two steps, a goal-directed training framework is highly desirable. However, traditional goal-directed training models are mostly applied to the classification task. Fewer studies on goal-directed graph clustering exist, to the best of our knowledge.

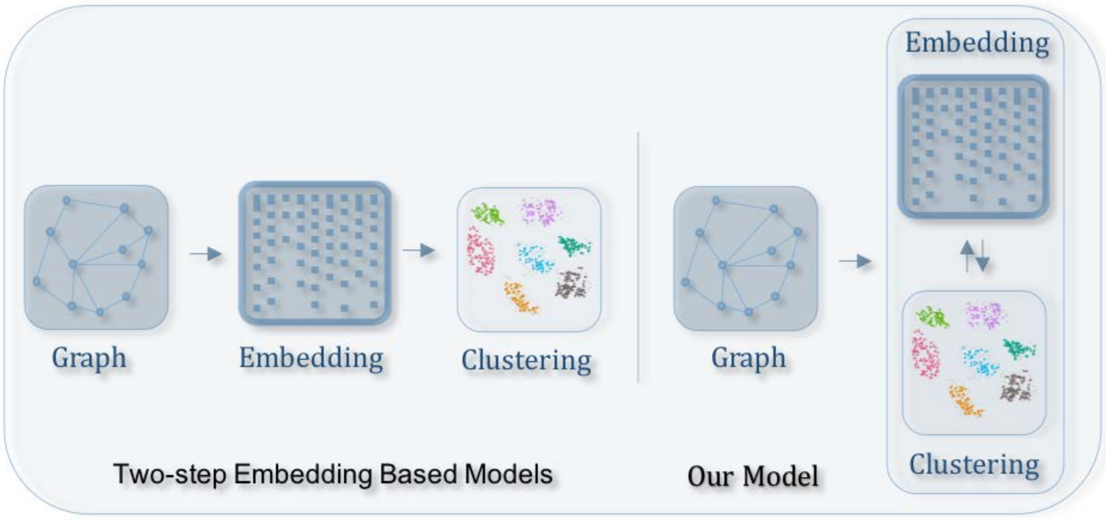


Figure 4.1 : The difference between two-step embedding learning models and our model.

Our Approach Motivated by the above observations, we propose a *Deep Neighbor-aware Embedded Graph Clustering* framework (DNEGC) with two variants, namely DNEGC-Att (with graph attentional autoencoder) and DNEGC-Con (with graph convolutional autoencoder) in this paper. To exploit the interrelationship of various-typed graph data, we develop a neighbor-aware graph autoencoder to learn latent representation, which integrates both content and structure information. The encoder progressively aggregates information from its neighbor via a *convolutional* style or an *attentional* mechanism, and multiple layers of encoders are stacked to build a deep architecture for embedding learning. The decoder on the other side, reconstruct the topological graph information and manipulates the latent graph representation. Furthermore, a carefully designed self-training module, which takes the "*confident*" clustering assignments as soft labels, is employed to guide the optimizing procedure. By forcing the current clustering distribution approaching a hypothetical better distribution, in contrast to the separated two-step embedding learning-based methods (shown in Fig. 4.1), this specialized clustering component makes it possible to simultaneously learn the embedding and perform clustering in a unified framework, thereby achieving better clustering performance. Our contributions can be summarized as follows:

- We introduce a neighbor-aware framework, by developing the first graph attention-based autoencoder, as well as a graph convolution-based autoencoder, to effectively integrate both the structure and content information for attributed graph representation learning.
- We propose a new end-to-end deep learning framework for graph clustering. The framework jointly optimizes the embedding learning and graph clustering, to the mutual benefit of both components.
- The experimental results show that our algorithm outperforms state-of-the-art graph clustering methods.

4.2 Problem Definition and Overall Framework

We consider clustering task on attributed graphs in this paper. An attributed graph is represented as $G = (V, E, X)$, where $V = \{v_i\}_{i=1,\cdots,n}$ consists of a set of nodes, $E = \{e_{ij}\}$ is a set of edges between these nodes. The topological structure of graph G can be represented by an adjacency matrix A, where $A_{ij} = 1$ if $(v_i, v_j) \in E$; otherwise $A_{ij} = 0$. $X = \{x_1; \ldots; x_n\}$ are the attribute values where $x_i \in \mathbb{R}^m$ is a real-value attribute vector associated with vertex v_i.

Given the graph G and cluster number k, graph clustering aims to partition the nodes in G into k disjoint groups $\{G_1, G_2, \cdots, G_k\}$, so that nodes within the same cluster are generally: (1) close to each other in terms of graph structure while distant otherwise; and (2) more likely to have similar attribute values.

4.2.1 Overall Framework

In this paper, we construct a graph-based neighbor-aware network to solve this problem. Our framework is shown in Fig. 4.2 and consists of two parts: a graph autoencoder and a self-training clustering module.

- **Graph Autoencoder:** Our neighbor-aware autoencoder takes the attribute values and graph structure as input, and learns the latent representation by minimizing the reconstruction loss.
- **Self-training Clustering:** The self-training module performs clustering based on the learned representation, and in return, manipulates the latent representation according to the current clustering result.

We jointly learn the graph embedding and perform clustering in an end-to-end manner, so that each component benefits the other.

4.3 Proposed Method

In this section, we present our proposed Deep Neighbor-aware Embedded Graph Clustering (`DNEGC`). We will first develop a graph autoencoder which effectively integrates both structure and content information to learn a latent representation. Based on the representation, a self-training module is proposed to guide the clustering algorithm towards better performance.

4.3.1 Graph Autoencoder

The graph autoencoder aims to learn a low-dimension embedding of the graph G based on both the node content and the graph structure. The basic idea is to progressively aggregate neighbor information to learn a more informative representation in a deep neural network architecture. To this end, we develop two variants,

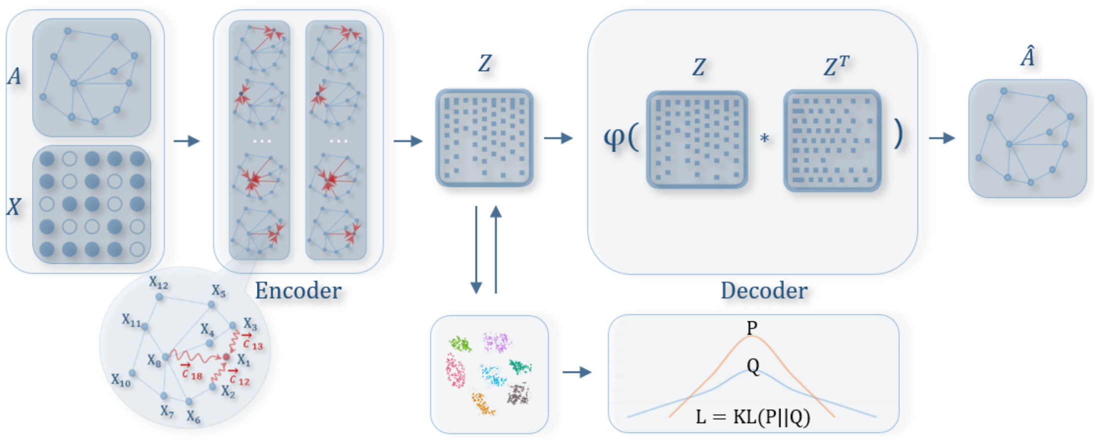


Figure 4.2 : The conceptual framework of Deep Neighbor-aware Embedded Graph Clustering (DNEGC). Given a graph $G = (V, E, X)$, DNEGC learns a hidden representation Z through a graph autoencoder, and manipulates it with a self-training clustering module, which is optimized together with the autoencoder and perform clustering during training. The two variants share similar framework and differ as their autoencoder encode the inputs through different strategy.

namely graph attentional encoder and graph convolutional encoder. They differ as they employ an *attentional* mechanism or a *convolutional* style respectively in their encoding strategies.

Graph Attentional Encoder

To represent both graph structure A and node content X in a unified framework, we develop a variant of the graph attention network (Velickovic et al., 2017) as a graph encoder for DNEGC-Att. The idea is to learn hidden representations of each node by attending over its neighbors, to combine the attribute values with the graph structure in the latent representation. The most straightforward strategy to attend the neighbors of a node is to integrate its representation equally with all its neighbors. However, in order to measure the importance of various neighbors, different weights are given to the neighbor representations in our layer-wise graph attention strategy:

$$z_i^{(l+1)} = \sigma(\sum_{j \in N_i} \alpha_{ij} W^{(l)} z_j^{(l)}). \tag{4.1}$$

Here for layer l, $z_i^{(l+1)}$ denotes the output representation of node i, and N_i denotes the neighbors of i. α_{ij} is the attention coefficient that indicates the importance of neighbor node j to node i, $W^l \in \mathbb{R}^{m_2 \times m_1}$ is the parameter matrix for our autoencoder to learn, with m_1 and m_2 being the input and output dimension of the layer respectively, and σ is a nonlinerity function. To calculate the attention coefficient α_{ij}, we measure the importance of neighbor node j from both the aspects of the attribute value and the topological distance.

From the perspective of attribute values, the attention coefficient α_{ij} can be represented as a single-layer feedforward neural network on the concatenation of x_i and x_j with weight vector $a \in \mathbb{R}^{2m_2}$:

$$c_{ij} = a^T [Wx_i || Wx_j]. \tag{4.2}$$

Topologically, neighbor nodes contribute to the representation of a target node. GAT considers only the 1-hop neighboring nodes (first-order) for graph attention (Velickovic et al., 2017). As graphs have complex structure relationships, we propose to exploit high-order neighbors in our encoder. We obtain a proximity matrix by considering t-order neighbor nodes in the graph:

$$M = (B + B^2 + \cdots + B^t)/t, \tag{4.3}$$

here B is the transition matrix where $B_{ij} = 1/d_i$ if $e_{ij} \in E$ and $B_{ij} = 0$ otherwise. d_i is the degree of node i. Therefore M_{ij} denotes the topological relevance of node j to node i up to t orders. In this case, N_i means the neighboring nodes of i in M. i.e., j is a neighbor of i if $M_{ij} > 0$.

The attention coefficients are usually normalized across all neighborhoods $j \in N_i$

with a softmax function to make them easily comparable across nodes:

$$\alpha_{ij} = \mathrm{softmax}_j(c_{ij}) = \frac{\exp(c_{ij})}{\sum_{r \in N_i} \exp(c_{ir})}. \tag{4.4}$$

Adding the topological weights M and an activation function δ (here LeakyReLU is used), the coefficients can be expressed as:

$$\alpha_{ij} = \frac{\exp(\delta(M_{ij}(a^T[Wx_i||Wx_j])))}{\sum_{r \in N_i} \exp(\delta(M_{ir}(a^T[Wx_i||Wx_r])))}. \tag{4.5}$$

We have $x_i = z_i^{(0)}$ as the input for our problem, and stack two graph attention layers:

$$z_i^{(1)} = \sigma(\sum_{j \in N_i} \alpha_{ij} W^{(0)} x_j), \tag{4.6}$$

$$z_i^{(2)} = \sigma(\sum_{j \in N_i} \alpha_{ij} W^{(1)} z_j^{(1)}), \tag{4.7}$$

in this way, our encoder encodes both the graph structure and the node attributes into a hidden representation, i.e., we will have $z_i = z_i^{(2)}$.

Graph Convolutional Encoder

On the other hand for DNEGC-Con, the encoder is defined as a variant of convolutional network from graph data. It extends the operation of *convolution* to graph data in a spectral domain and was formerly used in semi-supervised classification tasks (Kipf and Welling, 2016a). Our graph convolutional encoder aims to learn a layer-wise transformation combining both the adjacency matrix A representing the graph structure and the feature matrix X by a spectral convolution function $f(z_i^{(l)}, A)$:

$$z_i^{(l+1)} = f(z_i^{(l)}, A). \tag{4.8}$$

Here, $z_i^l \in \mathbb{R}^m$ (m features) is the input for convolution and $z_i^{(l+1)}$ is the convolution output. We view the convolution part from the graph level, therefore define $Z^l \in \mathbb{R}^{n \times m}$ and $Z^l = \prod_{i=1}^n z_i^l$ for later use.

We first consider each feature of the graph $s \in \mathbb{R}^n$ as a signal, the convolution function can be defined as the multiplication of the signal with a filter such as:

$$g_\theta \star s = U g_\theta U^T s, \tag{4.9}$$

where g_θ is a filter parameterized by $\theta \in R^n$, U is the eigenvectors of the normalized graph Laplacian $L = U\Lambda U^T = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$, with $D_{ii} = \sum_j A_{ij}$, and I_N the identity matrix, Λ represents a diagonal matrix where the diagonal elements are the eigenvalues of L. We can consider g_θ to be a function of the eigenvalues $g_\theta(\Lambda)$, and $U^T s$ be the graph Fourier transform of s.

Computing the eigen-decomposition of L could be expensive for large graphs. Hence, Hammond et al. suggested $g(\Lambda)$ to be approximated in terms of Chebyshev polynomials (Hammond et al., 2009):

$$g_\theta(\Lambda) \approx \sum_{y=0}^{Y} \theta_y T_y(\widetilde{\Lambda}), \tag{4.10}$$

where $\widetilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I_N$. λ_{max} is the largest eigenvalue of L. θ is the Chebyshev coefficients, $T_0(a) = 1$ and $T_1(a) = a$. By further constraint $Y = 1$ and approximate $\lambda_{max} \approx 2$, a linear function on the graph Laplacian spectrum is obtained:

$$g_\theta \star s \approx \theta(I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})s, \tag{4.11}$$

where θ is the shared filter over the whole graph and $I_N + D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ could be approximated by $\widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}}$ with $\widetilde{A} = A + I_N$ and $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$.

To extend this function to the graph level, or in other words for multiple features $Z^l \in \mathbb{R}^{n \times m}$, the convolution function could be adjusted as:

$$H = g_W \star Z^{(l)} = \widetilde{D}^{-\frac{1}{2}} \widetilde{A} \widetilde{D}^{-\frac{1}{2}} Z^{(l)} W, \tag{4.12}$$

where $H \in \mathbb{R}^{n\times m}$ is the convolved signal matrix, and W is a matrix of filter parameters replacing θ. Then the layer-wise propagation of the GCN is:

$$f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}), \tag{4.13}$$

with σ being an activation function such as $\text{Relu}(t) = \max(0, t)$ or $\text{sigmoid}(t) = \frac{1}{1+e^{-t}}$. This convolution propagation function can be computed efficiently in $\mathcal{O}(|E|m^2)$.

We adopt this convolution propagation function and construct a two-layer encoder for our autoencoder:

$$Z^{(1)} = f_{Relu}(X, A|W^{(0)}); \tag{4.14}$$

$$Z^{(2)} = f_{linears}(Z^{(1)}, A|W^{(1)}). \tag{4.15}$$

Our encoder encodes both node content and graph structure into a unified hidden representation $Z = Z^{(2)}$.

Inner Product Decoder

There are various kinds of decoders, which reconstruct either the graph structure, the attribute value, or both. In our method, we choose to reconstruct the graph structure, as our algorithm will be more flexible and will thus fit situations in which no content information is available. We use a simple inner product decoder which predicts whether there is a link between two nodes. The reconstructed link prediction layer is trained based on the hidden graph representation:

$$\hat{A}_{ij} = \text{sigmoid}({z_i}^\top z_j), \tag{4.16}$$

where $\hat{A}$ is the reconstructed structure matrix of the graph.

Reconstruction Loss

We minimize the reconstruction error by measuring the difference between A and $\hat{A}$:

$$L_r = \sum_{i=1}^{n} loss(A_{ij}, \hat{A}_{ij}). \tag{4.17}$$

In our paper, the binary cross-entropy loss function is used as the reconstruction loss. By optimizing the autoencoder reconstruction, we can learn the encoder parameter $W^{(0)}$ and $W^{(1)}$, and thereupon the optimized latent embedding Z.

4.3.2 Self-optimizing Embedding

One of the main challenges for graph clustering methods is the nonexistence of label guidance. The graph clustering task is naturally unsupervised and feedback during training as to whether the learned embedding is well optimized cannot, therefore, be obtained. To confront this challenge, we develop a self-optimizing embedding algorithm as a solution.

Apart from optimizing the reconstruction error, we input our hidden embedding into a self-optimizing clustering module which optimizes the following objective:

$$L_c = KL(P||Q) = \sum_i \sum_u p_{iu} log \frac{p_{iu}}{q_{iu}}, \tag{4.18}$$

Where q_{iu} measures the similarity between node embedding z_i and cluster center embedding μ_u. We measure it with a Student's t-distribution so that it could handle different scaled clusters and is computationally convenient (Maaten and Hinton, 2008):

$$q_{iu} = \frac{(1 + ||z_i - \mu_u||^2)^{-1}}{\sum_k (1 + ||z_i - \mu_k||^2)^{-1}}, \tag{4.19}$$

it can be seen as a soft clustering assignment distribution of each node with the current embedding. On the other hand, p_{iu} is the target distribution defined as:

$$p_{iu} = \frac{q_{iu}^2 / \sum_i q_{iu}}{\sum_k (q_{ik}^2 / \sum_i q_{ik})}. \tag{4.20}$$

Soft assignments with high probability (nodes close to the cluster center) are considered to be trustworthy in Q. So the target distribution P raises Q to the second power to emphasize the role of those "confident assignments". The minimizing of the KL distance then force the current distribution Q to approach the target distribution P, so as to set these "confident assignments" as soft labels to supervise Q's embedding learning.

To this end, we first train the autoencoder without the self-optimize clustering part to obtain a meaningful embedding z as described in Eq.(4.7) and Eq.(4.15). Self-optimizing clustering is then performed to improve this embedding. To obtain the soft clustering assignment distributions of all the nodes Q through Eq.(4.19), the k-means clustering is performed just once on the embedding z before training with the self-optimize clustering part, to obtain the initial cluster centers μ.

It is worth mentioning that in the following iterative training, the k-means clustering is never used again, and the cluster centers μ are updated using Stochastic Gradient Descent (SGD) based on the gradients of the clustering loss L_c with respect to μ:

$$\mu_u = \mu_u - \varphi \frac{\partial L_c}{\partial \mu_u}, \tag{4.21}$$

where φ is the step size. Similarly, $\partial L_c / \partial z_i$ is also computed and passed down to update the parameter matrix W in the encoder together with the gradient from the reconstruction loss of the autoencoder, so as to benefit the embedding learning.

We calculate the target distribution P according to Eq.(4.20), and the clustering loss L_c according to Eq.(4.18).

The target distribution P works as "ground-truth labels" in the training procedure, but also depends on the current soft assignment Q which updates at every iteration. It would be hazardous to update P at every iteration with Q as the constant change of target would obstruct learning and convergence. To avoid instability in the self-optimizing process, we update P every T iterations. As detailed choice of T will not affect clustering performance, we simply set it to 5 in our experiments.

In summary, we minimize the clustering loss to help the autoencoder manipulate the embedding space using the embedding's own characteristics and scatter embedding points to obtain better clustering performance.

4.3.3 Joint Embedding and Clustering Optimization

We jointly optimize the autoencoder embedding and clustering learning, and define our total objective function as:

$$L = L_r + \gamma L_c, \tag{4.22}$$

where L_r and L_c are the reconstruction loss and clustering loss respectively, $\gamma \geq 0$ is a coefficient that controls the balance in between. It can be optimized by directly back-propagate the gradient from both L_r and L_c to update W, or utilize the unrolled optimization strategy (Schmidt and Roth, 2014; Diamond et al., 2017; Liang et al., 2019). It is worth mentioning that we could gain our clustering result directly from the last optimized Q, and the label estimated for node v_i could be obtained as:

$$r_i = \arg\max_u q_{iu}, \tag{4.23}$$

which is the most likely assignment from the last soft assignment distribution Q.

Our method is summarized in Algorithm 3. Our algorithm has the following advantages:

- **Interplay Exploitation.** The graph neural network-based autoencoder efficiently exploits the interplay between both the graph structure and the node content information.

Table 4.1 : Benchmark Graph Datasets

Dataset	Nodes	Features	Clusters	Links	Content Words
Cora	2,708	1,433	7	5,429	3,880,564
Citeseer	3,327	3,703	6	4,732	12,274,336
Pubmed	19,717	500	3	44,338	9,858,500

- **Clustering Specialized Embedding.** The proposed self-training clustering component manipulates the attributed graph embedding to improve the clustering performance.
- **End-to-end Learning.** The framework jointly optimizes the two parts of the loss functions, learns the embedding and performs clustering in an end-to-end manner.

4.4 Experiments

4.4.1 Benchmark Datasets

We use three benchmark datasets in our experiments, which are widely used in assessment of attributed graph-based algorithms (Velickovic et al., 2017; Kipf and Welling, 2016a), summarized in Table 4.1. All these datasets consist of scientific publications as nodes, citation relationships as edges and unique words in the documents as features. Publications in these datasets are labeled as they could be assigned to different sub-fields.

4.4.2 Baseline Methods

We compared a total of 13 algorithms with our method in our experiments. The graph clustering algorithms include approaches that use only node attributes or

Algorithm 3 Unsupervised Deep Neighbor-aware Embedded Graph Clustering

```
Require:
  Graph $G$ with $n$ nodes, each node with $m$-dimension attribute value;
  Number of clusters $k$;
  Number of iterations $Iter$;
  Target distribution update interval $T$;
  Clustering Coefficient $\gamma$.
Ensure:
  Final clustering results.

  Update the autoencoder variable $W$ by minimizing Eq.(4.17) to get the autoencoder hidden embedding $Z$;
  Compute the initial cluster centers $\mu$ based on $Z$;
  for $l = 0$ to $Iter - 1$ do
    Calculate soft assignment distribution $Q$ with $Z$ and $\mu$ according to Eq.(4.19);
    if $l\%T == 0$ then
      Calculate target distribution $P$ with $Q$ by Eq.(4.20);
    end if
    Calculate reconstruction loss $L_r$ according to Eq.(4.17)
    Calculate clustering loss $L_c$ according to Eq.(4.18);
    Update the variable $W$ and thereupon the embedding $Z$ by minimizing Eq.(4.22);
  end for
  Get the clustering results with final $Q$ by Eq.(4.23)
```

Table 4.2 : Algorithm Comparison

	Content	Structure	Self-training	GCN encoder	GAT encoder	Recover A
K-means	★					
Spectral		★				
GraphEncoder		★				★
DeepWalk		★				★
DNGR		★				★
M-NMF		★				
RTM	★	★				
RMSC	★	★				
TADW	★	★				
GAE&VGAE	★	★		★		★
ARGA&ARVGA	★	★		★		★
AGC	★	★				
DNEGC-Att	★	★	★		★	★
DNEGC-Con	★	★	★	★		★

network structure information, and also approaches that combine both. Deep representation learning-based graph clustering algorithms were also compared. These algorithms are summarized in Table 4.2. Detailed description of these algorithms could be found in Chapter 2.

For representation learning algorithms such as DeepWalk, TADW and DNGR which do not specify the clustering algorithm, we first learned the representation from these algorithms, and then applied the k-means algorithm on their respective representations, but for algorithms like RMSC which require spectral clustering or an alternative algorithm, we followed their preference and used the specified algorithms. The best results we got are reported in this paper.

4.4.3 Evaluation Metrics & Parameter Settings

Evaluation Metrics: We follow (Xia et al., 2014) and use seven metrics to evaluate the clustering result namely Accuracy (ACC), Normalized Mutual Information (NMI), F-score (F), Precision (P), Recall (R), Average Entropy (AE) and Adjusted Rand Index (ARI). A better clustering result should lead to a lower value of average entropy and higher values for all the other metrics.

- **ACC** is the average performance of label matching clustering results and can be represented as $\sum_i (y_i == f(l_i))/n$, where f is the mapping function which maps category labels to cluster labels.
- **NMI** measures the mutual information entropy between the resulting cluster labels and ground truth labels followed by a normalization operation.
- **F-score** is the harmonic mean value of *Precision* and *Recall*;
- **Precision** is the fraction of correctly clustered nodes among the retrieved nodes;
- **Recall** is the fraction of correctly clustered nodes that have been retrieved over the total number of relevant nodes;
- **Average Entropy** $= \sum_{i=1}^{k} \frac{m_i}{m} e_i$, where k is the cluster number and m is the number of nodes, and $e_i = -\sum_{j=1}^{k} \frac{m_{ij}}{m_i} log_2 \frac{m_{ij}}{m_i}$, with m_i representing the number of nodes in cluster i and m_{ij} representing the number of nodes in cluster i and labeled j.
- **ARI** is the adjusted rand index (RI) that guarantees a value close to 0, where RI measures the percentage of correct clustering decisions;

Parameter Settings: For the baseline algorithms, we carefully select the parameters for each algorithm, following the procedures in the original papers. In

TADW, for instance, we set the dimension of the factorized matrix to 80, the dimension of the text feature to 200 and the regularization parameter to 0.2; For the DNGR algorithm, we build a three-layers denoising autoencoder with the number of nodes set as 512 and 256 in the hidden layers; For the RMSC algorithm, we regard graph structure and node content as two different views of the data and construct a Gaussian kernel on them. We run the k-means algorithm 50 times for all embedding learning methods for fair comparison.

For our method, we set the clustering coefficient γ to 10 for DNEGC-Att and 1 for DNEGC-con. For the variant DNEGC-Att with attentional encoder, we consider second-order neighbors and set $M = (B + B^2)/2$. The encoder is constructed with a 256-neuron hidden layer and a 16-neuron embedding layer for all datasets. For DNEGC-Con with convolutional encoder, a 32-neuron hidden layer and a 16-neuron embedding layer is used instead.

4.4.4 Experiment Results

We compare our DNEGC with baselines mentioned above on graph clustering first. Then we perform detailed analysis on coefficients in the model.

Clustering Performance Comparison

The experiment results on the three benchmark datasets are summarized in Tables 4.3, 4.4 and 4.5. C, S, and C&S indicate if the algorithm uses only content, structure, or both content and structure information, respectively. We can see that our methods clearly outperform most of the baselines across most of the evaluation metrics. AGC is able to outperform our method on the Pubmed dataset, may because Pubmed is a large and simple dataset which adverse to our deep architecture.

One Side v.s Both Side of Information: We can easily observe from these results that methods using both the structure and content information of the graph

Table 4.3 : Experimental Results on **Cora** Dataset

	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	C	0.500	0.317	0.376	0.376	0.376	1.810	0.239
Spectral	S	0.398	0.297	0.332	0.312	0.355	1.871	0.174
GraphEncoder	S	0.301	0.059	0.230	0.214	0.253	2.496	0.046
DeepWalk	S	0.529	0.384	0.435	0.392	0.504	1.681	0.291
DNGR	S	0.419	0.318	0.340	0.266	0.480	1.882	0.142
M-NMF	S	0.423	0.256	0.320	0.304	0.342	1.977	0.162
RTM	C&S	0.440	0.230	0.307	0.332	0.285	2.021	0.169
RMSC	C&S	0.466	0.320	0.347	0.345	0.352	1.808	0.203
TADW	C&S	0.536	0.366	0.401	0.342	0.492	1.749	0.240
GAE	C&S	0.530	0.397	0.415	0.431	0.401	1.583	0.293
VGAE	C&S	0.592	0.408	0.456	0.489	0.429	1.545	0.347
ARGA	C&S	0.669	0.489	0.666	0.680	0.686	1.322	0.422
ARVGA	C&S	0.581	0.426	0.560	0.562	0.588	1.492	0.329
AGC	C&S	0.689	0.522	0.656	0.672	0.675	1.273	0.448
DNEGC-Att	C&S	**0.704**	**0.528**	**0.682**	**0.704**	**0.706**	**1.229**	**0.496**
DNEGC-Con	C&S	0.683	0.512	0.659	0.665	0.689	1.269	0.477

generally perform better than those using only one side of information. In the Cora dataset, for example, TADW, GAE, VGAE, AGC and our method outperform all the baselines using one side of information. This observation demonstrates that both the graph structure and node content contain useful information for graph clustering, and illustrates the significance of capturing the interplay between two-sides information.

Deep Learning Models: The results of most of the deep learning models are sat-

Table 4.4 : Experimental Results on **Citeseer** Dataset

	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	C	0.544	0.312	0.413	0.411	0.416	1.738	0.285
Spectral	S	0.308	0.090	0.257	0.241	0.276	2.300	0.082
GraphEncoder	S	0.293	0.057	0.213	0.215	0.211	2.380	0.043
DeepWalk	S	0.390	0.131	0.305	0.282	0.336	2.201	0.137
DNGR	S	0.326	0.180	0.300	0.200	0.609	2.168	0.043
M-NMF	S	0.336	0.099	0.255	0.228	0.291	2.288	0.070
RTM	C&S	0.451	0.239	0.342	0.349	0.335	1.915	0.203
RMSC	C&S	0.516	0.308	0.404	0.383	0.430	1.767	0.266
TADW	C&S	0.529	0.320	0.436	0.376	0.532	1.781	0.286
GAE	C&S	0.380	0.174	0.297	0.291	0.304	2.093	0.141
VGAE	C&S	0.392	0.163	0.278	0.251	0.315	2.131	0.101
ARGA	C&S	0.559	0.289	0.544	0.578	0.539	1.795	0.257
ARVGA	C&S	0.598	0.323	0.570	0.583	0.566	1.703	0.322
AGC	C&S	0.672	0.414	0.627	0.635	0.631	1.500	0.420
DNEGC-Att	C&S	0.672	0.397	0.636	0.639	0.640	1.521	0.410
DNEGC-Con	C&S	**0.692**	**0.426**	**0.639**	**0.640**	**0.644**	**1.456**	**0.449**

isfactory since they generally perform better than those shallow ones. The GraphEncoder and DNGR algorithm are not necessarily an improvement over the other algorithms, although they both employ deep autoencoder for representation learning. This observation may result from their neglect at the node content information.

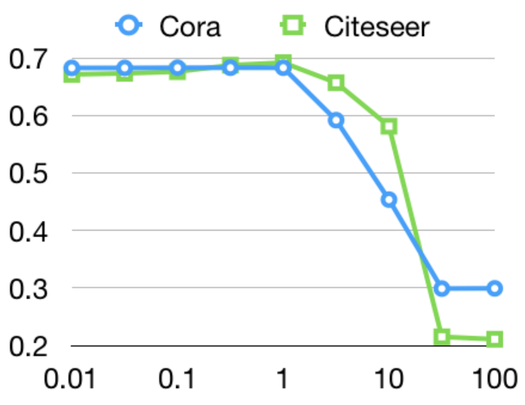
Superiority of DNEGC: It is worth mentioning that our algorithms, both DNEGC-Con and DNEGC-Att, significantly outperform GAE and VGAE. On the Cora dataset for example, our method DNEGC-Att represents a relative increase

Table 4.5 : Experimental Results on **Pubmed** Dataset

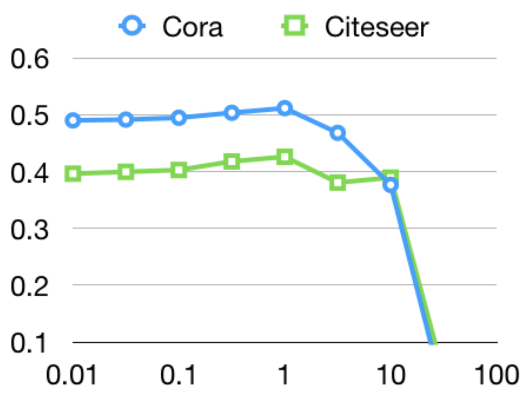
	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	AE(↓)	ARI(↑)
K-means	C	0.580	0.278	0.544	0.488	0.621	1.133	0.246
Spectral	S	0.496	0.147	0.471	0.407	0.561	1.323	0.098
GraphEncoder	S	0.531	0.210	0.506	0.456	0.569	1.231	0.184
DeepWalk	S	0.663	0.256	0.539	0.532	0.555	1.142	0.272
DNGR	S	0.468	0.153	0.445	0.387	0.523	1.314	0.059
M-NMF	S	0.470	0.084	0.443	0.391	0.529	1.411	0.058
RTM	C&S	0.575	0.194	0.444	0.456	0.433	1.230	0.149
RMSC	C&S	0.629	0.273	0.521	0.511	0.532	1.116	0.247
TADW	C&S	0.565	0.224	0.481	0.465	0.500	1.196	0.177
GAE	C&S	0.632	0.249	0.511	0.518	0.505	1.146	0.246
VGAE	C&S	0.619	0.216	0.478	0.492	0.464	1.194	0.201
ARGA	C&S	0.632	0.235	0.636	0.636	0.669	1.167	0.221
ARVGA	C&S	0.390	0.004	0.311	0.335	0.342	1.525	0.002
AGC	C&S	**0.679**	**0.306**	**0.688**	**0.733**	0.695	**1.082**	**0.311**
DNEGC-Att	C&S	0.671	0.266	0.659	0.677	0.687	1.122	0.278
DNEGC-Con	C&S	0.677	0.275	0.675	0.675	**0.699**	1.105	0.278

of 18.97% and 29.49% w.r.t. accuracy and NMI against VGAE, and the increase is even greater on the Citeseer dataset. The reasons for this are that (1) we employ a graph convolutional/attentional network that effectively integrates both content and structure information of the graph; (2) we use a deep architecture to learn the representation, which captures more underlying information; (3) Our self-training clustering component is specialized and powerful in improving the clustering efficiency.

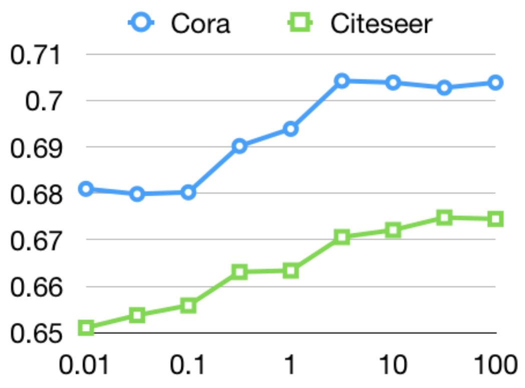
DNEGC-Att v.s. DNEGC-Con: The results show that DNEGC-Att outperforms DNEGC-Con on Cora dataset, while DNEGC-Con outperforms DNEGC-Att on Citeseer and Pubmed datasets. But their performance is very close to each other. This is because both of them are clustering-directly approaches. While there may be some difference on the learned embedding, the embedding will be regularized via the clustering objective, and finally they achieve very similar results for the clustering task.



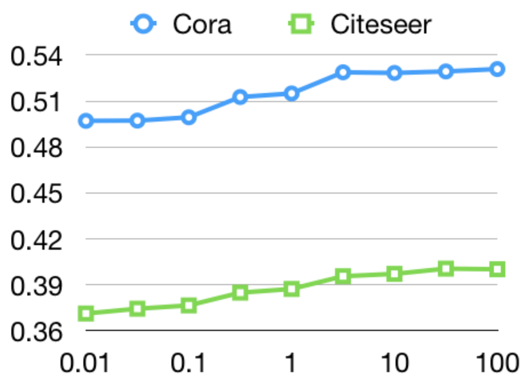
(a) ACC of DNEGC-Con



(b) NMI of DNEGC-Con



(c) ACC of DNEGC-Att



(d) NMI of DNEGC-Att

Figure 4.3 : Parameters study on clustering coefficient γ.

Parameter Study

We also investigated the sensitivity of the parameters for our algorithm.

Clustering Coefficient γ: We vary the clustering coefficient γ to study the effect of the self-training clustering component. The results are shown in Fig. 4.3.

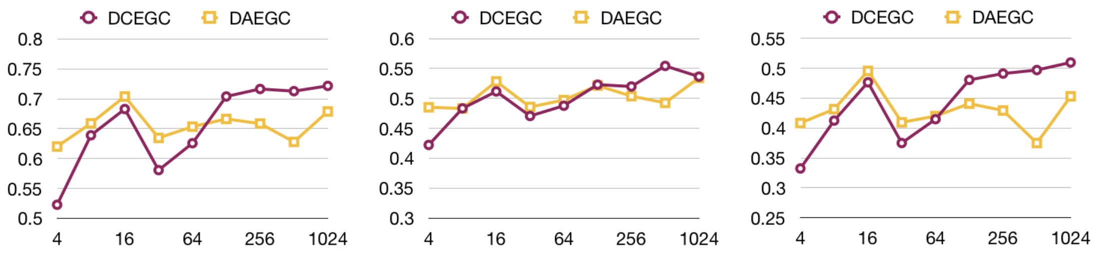
We could find that experiment on the Cora and Citeseer datasets show similar trends. For DNEGC-Con, we observe the best performance with γ around 1. Before γ is increased to that peak, the clustering performance measured by ACC and NMI steadily rise; As we keep adding γ up after that, the performance plummeted as a whole. However, for DNEGC-Att, the result keeps good as γ rises.

It shows that our self-training clustering component does work and improve the clustering result. However, a too large value of γ, which means excessively emphasis on the clustering loss, may distort the latent feature space since its trained on estimated targets and could lead to abnormal clustering result. DNEGC-Att avoids such plummeting may because the embedding learned with weighted neighbor features are more robust and effective, leading to more accurate initial targets, and make the self-training more stable.

Embedding Size: We also vary the dimension of embedding from 8 neurons to 1024 and report the clustering results on the Cora dataset in Fig. 4.4.

The results show that when increasing the dimension of embedding from 4-neuron to 16-neuron, the performance on clustering steadily rises; if we further increase the dimension, the performance of DNEGC-Con fluctuates but still have an overall tendency of rising, but the performance of DNEGC-Att does not improve as well, since the 8-neuron or 16-neuron embedding is already sufficient with its more efficient attention strategy as argued above. It is worth mentioning that we set the embedding size to 16 to obtain a stable and efficient model, but it could get markedly better performance when the embedding size is continuously enlarged, to for example, 128-neuron, 256-neuron or 1024-neuron.

Number of Layers: To show the effectiveness of deep architecture, we stack differ-



(a) ACC on Cora (b) NMI on Cora (c) ARI on Cora

Figure 4.4 : Parameters study on embedding size.

ent numbers of layers to observe the alteration of the performance on DNEGC-Con. For the autoencoder with only one hidden layer, we encoder the input feature directly into 16-neuron embedding; for the one with two layers, we add a 32-neuron layer in between and construct a d-32-16 encoder like the one we adopted, where d is the input layer dimension; for more layers, we construct d-64-32-16, d-128-64-32-16, etc. encoders with each newly added hidden layer doubling the dimension of its embedding. The performance of all these models on the Cora and Citeseer dataset are reported in Fig. 4.5.

We could observe that, when we stack 2 encoder layers to the model, the performance significantly improve compared with the model with only 1 hidden layer. The performance of the model with 3 stacked hidden layers is also satisfactory. These observations demonstrate that using a stacked architecture instead of a single-layer one can improve the model performance. However, as we continuously add more layers to the model, the performance reduces sharply in terms of all the observations. This is because stacking too many layers will increase the complexity of the architecture, raise the possibility of information loss and enhance the difficulty to the training process.

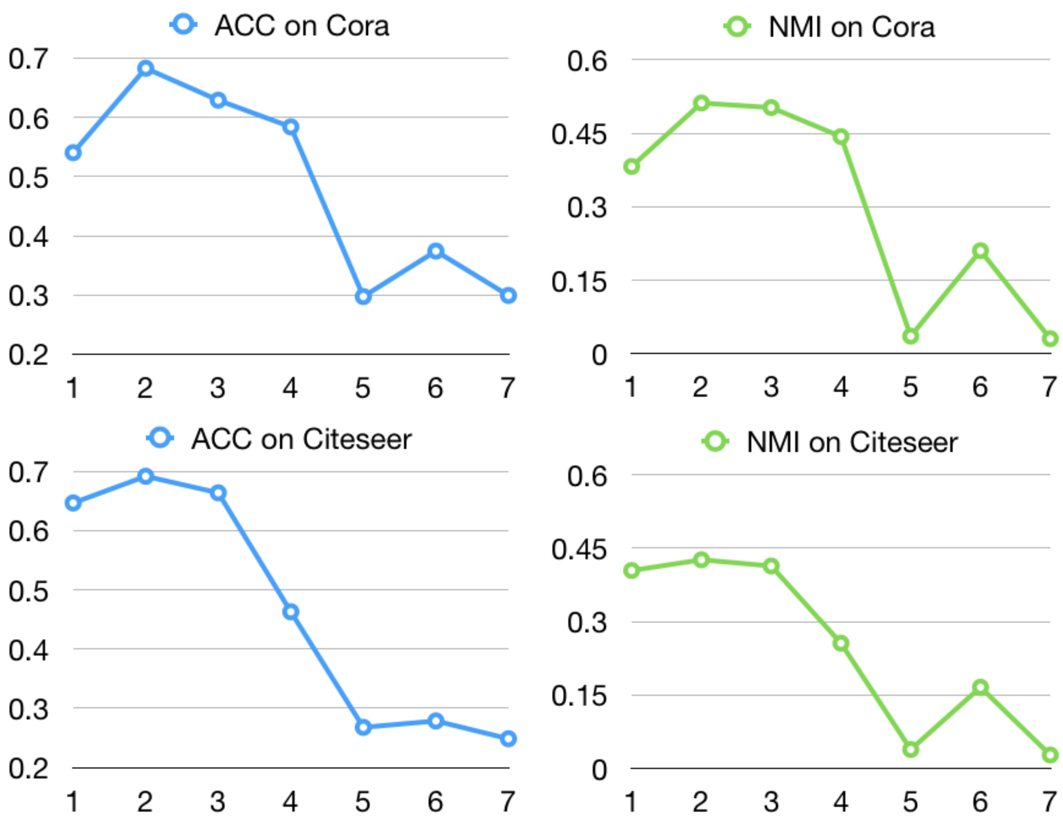
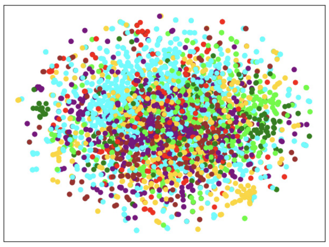
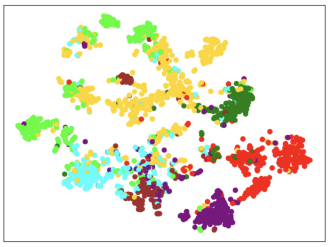
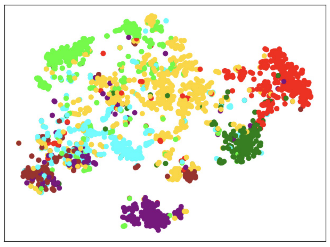
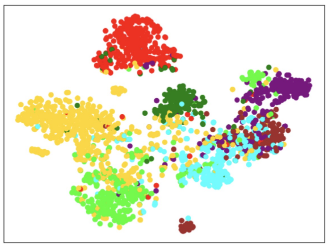


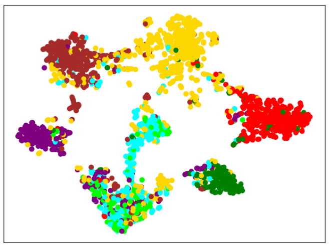
Figure 4.5 : Parameters study on number of layers.

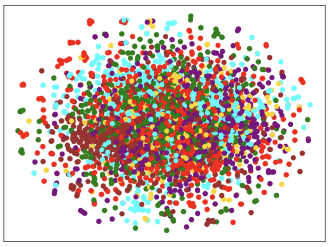

(a) GraphEncoder on Cora

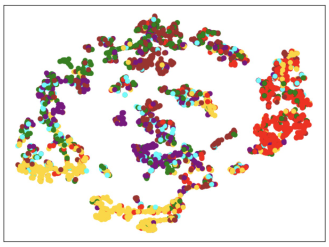

(b) GAE on Cora

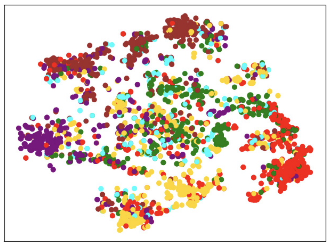

(c) VGAE on Cora

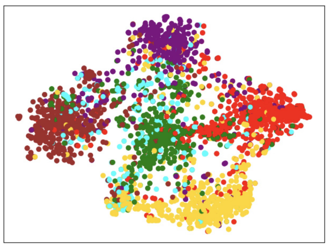

(d) DNEGC-Con on Cora


(e) DNEGC-Att on Cora


(f) GraphEncoder on Citeseer


(g) GAE on Citeseer


(h) VGAE on Citeseer


(i) DNEGC-Con on Citeseer

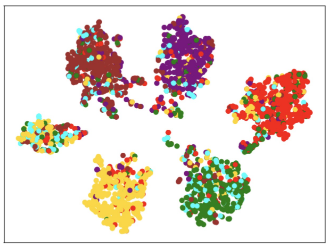

(j) DNEGC-Att on Citeseer

Figure 4.6 : 2D visualization of various methods using the t-SNE algorithm on the Cora and Citeseer dataset.

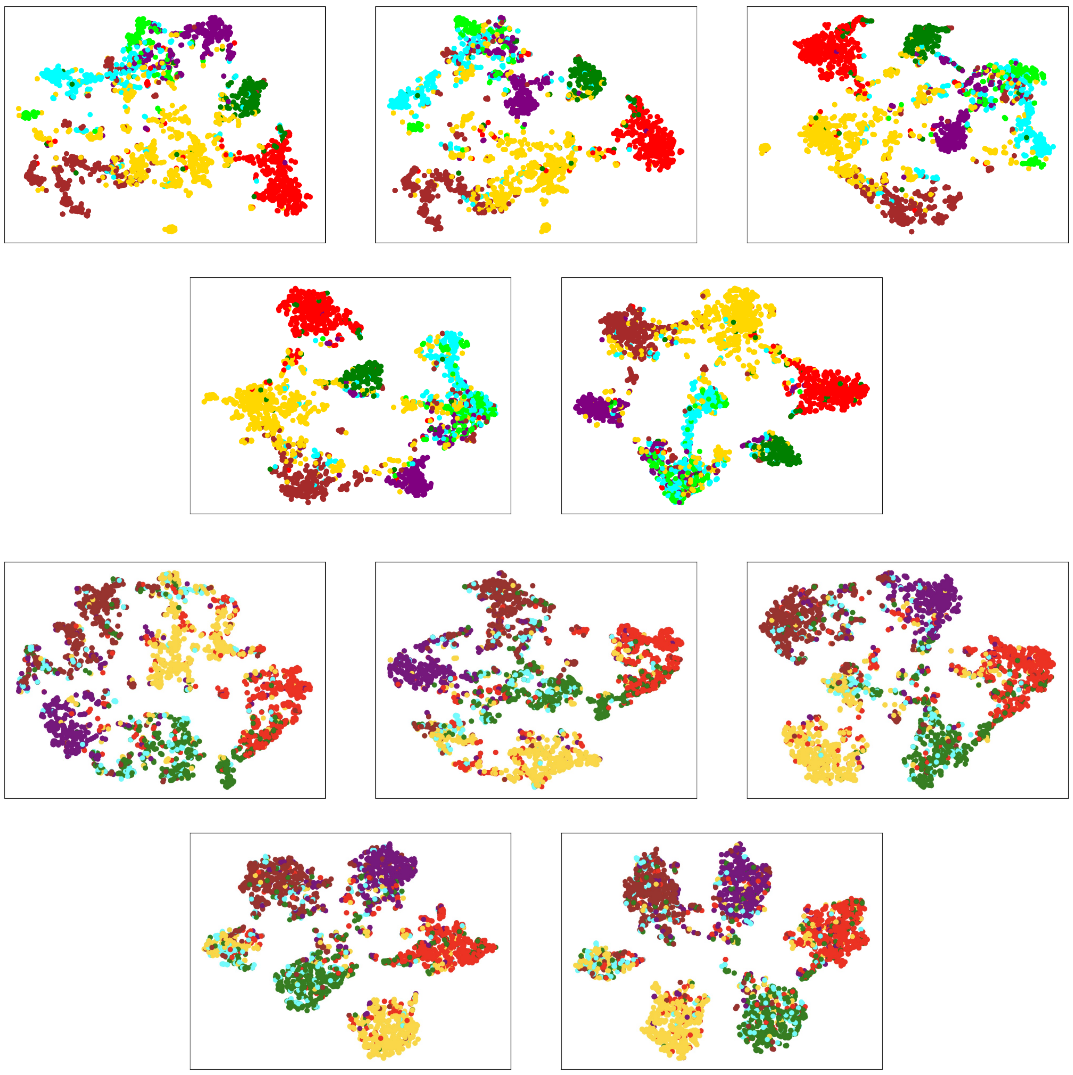


Figure 4.7 : 2D visualization of the DNEGC-Att algorithm using the t-SNE algorithm on the Cora and Citeseer dataset during training (the top line for the Cora dataset, and the bottom line for the Citeseer dataset). The first visualization of each line illustrates the embedding training with the graph autoencoder only, followed by visualizations showing subsequent equal epochs in which the self-training component is included, till the last one being the final embedding visualization.

Network Visualization

We visualize the Cora and Citeseer datasets in two-dimensional space by applying the t-SNE algorithm (Van Der Maaten, 2014) on the learned embedding. The results in Fig. 4.6 show that we obtain outstanding embedding as well as clearer clustering results, compared with the baseline methods, benefit from our self-clustering components which contribute to both clustering and embedding learning.

We also visualize the variation of the embedding on the Cora and Citeseer datasets during training as shown in Fig. 4.7. We can observe that, after training with our graph attentional autoencoder, the embedding is already meaningful. However by applying self-training clustering, the embedding becomes more evident as our training progresses, with less overlapping and each group of nodes gradually gathered together.

Chapter 5

Learning Corrupted Graph Data

Cross-Graph: Robust and Unsupervised Embedding for Attributed Graphs with Corrupted Structure

In this chapter, we extend our research to imperfect graph data settings. To confront the fourth challenge of dealing with corrupted graph data, we propose a novel Cross-Graph framework, to learn robust graph embedding for unsupervised tasks against structural corruption.

5.1 Background

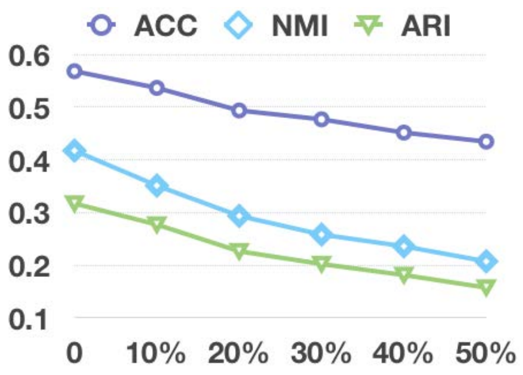
Graphs have attracted much more attention in recent years with the development of networked applications, such as social networks, citation networks, wireless networks (Wang et al., 2020b) and protein-protein interaction networks (Wu et al., 2019b). Unlike traditional data format, graphs are adept at characterizing individual properties as well as capturing the pairwise relationships between the individuals in the networks. The complexity of graph information has made graph analyzing significant, yet challenging.

For the past few years, graph embedding has evolved as a general solution to various graph analyzing tasks (Cai et al., 2018; Zhang et al., 2018). Its main strength is to preserve and combine different sides of graph information into a unified low-dimensional feature space. Based on the learnt embedding, classical task-oriented methods could be applied to handle various tasks such as classification (Kipf and Welling, 2017), clustering (Wang et al., 2017a) or link prediction (Wang et al., 2017c), which would otherwise be complicated for graph data.

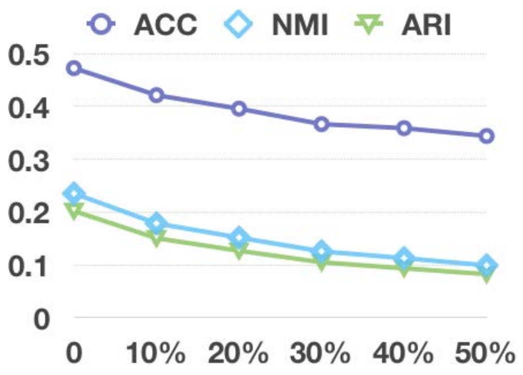
Recent graph embedding methods lean towards embedding attributed graphs for more comprehensive graph information. Attributed graph information consists of node content and edge connections between the nodes. Corruption, like noise or outliers, can occur in both aspects and affect the analysis of graph data. However, previous embedding works have not pay enough attention to the corruption in attributed graphs. A few works tried to detect isolated outlier nodes out of the graph (Li et al., 2017; Liu et al., 2017), but they were not able to recover the graphs' property from systematic corruption. So, they are not able to benefit the other graph analyzing tasks applied to the same graph. On the other hand, little previous works question the structure information in the provided graph data.

Edges could be citations among academic papers or friend relationships in a social network, etc.. They only represent some certain kinds of relationship between the two nodes. In other words, the disconnection of two nodes is commonplace in graphs and could always make some sense. Furthermore, current existing studies are able to help against one aspect of structural corruption, namely missing edges. This problem can be solved quite well by various link-prediction methods (Al Hasan and Zaki, 2011). Our intention, in this paper, is to focus on the other aspect of structural corruption, namely spurious edges. We therefore define the structural corruption in this paper as spurious edge connections, rather than the missing ones.

Broadly speaking, that spurious edges improperly connect two nodes is ubiquitous and considerably more problematical. These edges can be generated by malicious nodes, like robot account, to influence its neighbors or hide itself, or might be created unintentionally by the users or systems (Zhang et al., 2017). Adversarial poisoning attacks on graph data also tend to add edges to bring noise to the learnt node embedding (Zügner et al., 2018; Zhu et al., 2019a; Wu et al., 2019a). Furthermore, with the rapid development of graph learning, many CV or NLP-oriented methods also employ graph neural networks. They construct graphs from their plain



(a) Cora dataset



(b) Citeseer dataset

Figure 5.1 : We randomly add spurious edges to the Cora and Citeseer graph structure and then run Graph Autoencoder (GAE) on them for 10 times to record the average clustering performance evaluated by 3 clustering metrics. The X-axis shows the number of added edges represented as percentage to the number of original edges. It shows that spurious edges can easily ruin the performance of the graph.

data with a graph kernel. Such constructed graphs are always dense with a high percentage of redundancies. Even in a hypothetical "clean" graph, some of the edges could be regarded as partly abnormal for a given task. For instance, in citation networks, it is common for academic papers to cite weakly-related papers, which will appear the same as those key citations in the graph; in social networks, a parent-child connection may be considered abnormal when classifying users according to their hobbies, since they are connected due to family relationship and are unlikely to have similar hobbies.

Though many effective graph embedding methods have been proposed in recent years, most of these embedding methods are based on the assumption that they have no difficulty accessing perfect graph-structure data. This assumption is too ideal to hold in real-world problems and may limit the efficacy of the learnt embedding. As illustrated in Fig. 5.1, the performance of the embedding can be ruined easily, simply by adding random spurious edges to the graph. In a word, a graph embedding method robust against structural corruption is highly desirable.

To deal with this problem, we propose a novel Cross-Graph framework, to learn robust graph embedding, strengthened against structural corruption. Since label information is not easily accessible either, we decide to use autoencoders to perform unsupervised learning. Our autoencoder-based method can learn effective embedding without access to, not only the label guidance, but also clean graph-structure data. We are inspired by the Co-training approach (Blum and Mitchell, 1998), and designed a dual graph interaction framework called Cross-Graph Learning. Based on the deep learning memorization effect that deep neural networks fit clean data first (Arpit et al., 2017; Zhang et al., 2016), we maintain two autoencoders and update them alternatively. In each iteration, since the trustworthy edges fit faster and will be closer to the ground-truth, the two autoencoders can evaluate the reliability of every graph edge with their reconstructed graph structure. Each autoencoder then updates its structure by slightly devaluing those distrusted edges. This updated graph structure is then passed to its peer-autoencoder, working as a provided "opinion" on how the real structure should present. The peer-autoencoder would take this updated structure as the input to the next iteration. Through the learning process, those spurious edges will be devalued faster, over and over again, and eventually filtered out. Since the two autoencoders have different embedding abilities, different types of corruptions may be detected, including some misjudgments. Meanwhile, benefit from our interactive process, these misjudgments caused by a single autoencoder, can be reduced by its peer. This fact further strengthens the robustness of our model.

Our contribution can be summarized as follows:

- To the best of our knowledge, we are the first to discuss the influence of structural corruption, especially spurious edges, on attributed graph embedding. We show that corrupted graph structure could ruin the performance of the graph embedding learned on the basis of it.

- We propose a Cross-Graph strategy to learn graph embedding based on the graph structure and node content, which is unsupervised and robust against structural corruption.
- We conduct extensive experiments, compare our model with novel unsupervised graph embedding baselines on various kinds of structure-corrupted graphs. The results show that our Cross-Graph strategy significantly improves the performance when confronting structural corruption.

5.2 Problem Definition

We consider unsupervised graph embedding on attributed graphs in this paper. A graph is represented as $G = (V, E, X)$, in which $V = \{v_i\}_{i=1,\cdots,n}$ consists of a set of nodes, $E = \{e_{ij}\}$ is a set of edges between nodes. The topological structure of graph G can be simplified as an adjacency matrix A, where $A_{i,j} = 1$ if $(v_i, v_j) \in E$; otherwise $A_{i,j} = 0$. $X = \{x_1; \ldots; x_n\}$ are the attribute values where $x_i \in R^m$ is a m-dimension real-value attribute vector associated with vertex v_i.

In our setting, the graph structure is a corrupted one with spurious edges. In other words, the adjacency matrix $A \neq A_{clean}$, where it has many more extra 1 values instead of 0 in the corresponding positions of the spurious edges.

Our purpose is to learn latent representations $Z \in \mathbb{R}^{n \times d}$ to map the nodes $v_i \in V$ to a low-dimensional space, in which $z_i^\top$ mapping v_i is the i-th row of the matrix Z. For previous graph embedding methods, learning with the corrupted structure A would result in much worse embedding result compared with the one learned from A_{clean}, while we aim to keep up its performance against the structural corruption.

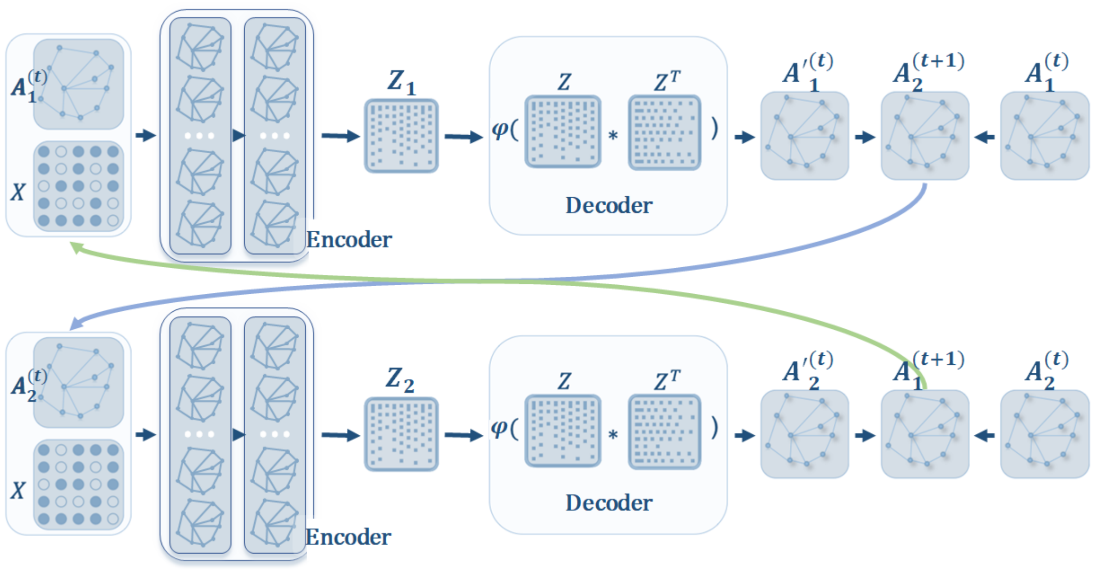


Figure 5.2 : Conceptual framework of Cross-Graph Autoencoder. Given a graph G with graph structure A and node content X, we maintain two autoencoders. Each autoencoder encodes A and X into a latent embedding Z, and then a decoder tries to reconstruct the structure A from Z and obtains A'. We regard A' partly as a reliability score of the edges in A and manipulate A according to it. Two updated A's are thereby formed and passed to the peer-autoencoder as the input for the next iteration.

5.3 Proposed Method

We present our proposed Cross-Graph framework in this section. After briefly introducing a basic graph autoencoder, we propose the Cross-Graph framework for robust learning. The mechanism and some analysis of our model are also presented.

5.3.1 Graph Autoencoder

The graph autoencoder aims to learn low-dimensional embedding of each node based on the graph data $G = (V, E, X)$. The main idea is to integrate both graph structure A and node content X through an encoder into a latent embedding Z, and reconstruct the graph structure A through a decoder to optimize Z.

Graph Convolutional Encoder: One of the most basic and popular kind of graph encoder is developed as a variant of the graph convolutional network (GCN)

(Kipf and Welling, 2017). It encodes both graph structure A and node content X into a hidden representation by extending the operation of convolution to the graph domain, and performs a layer-wise transformation by a spectral convolution function $f(Z^{(l)}, A)$:

$$Z^{(l+1)} = f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}). \tag{5.1}$$

Here, σ is a nonlinerity function. $Z^{(l)}$ is the input of the l-th layer transformation function, and $Z^{(l+1)}$ is the corresponding output. $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ is an approximation of the spectral convolution transformation $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, with $\widetilde{A} = A + I$, $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$, and I being the identity matrix.

The standard graph encoder $Enc(X, A)$ takes the node content X as the input to the first layer $Z^{(1)}$, and stacks two graph convolution layers to obtain a hidden representation Z:

$$Z = Enc(X, A) = f(f(X, A), A). \tag{5.2}$$

Inner Product Decoder: The decoder reconstructs the graph data from hidden representation. Previous works have tried to reconstruct different sides of the graph information, and reconstructing the graph structure A has been validated as the best solution, since it also can be more flexible and fit situations in which no content information is available. The inner product decoder is commonly used here to predict whether there is a link between two nodes. The reconstructed link prediction layer $Dec(Z)$ is trained based on the hidden graph representation:

$$A' = Dec(Z) = \text{sigmoid}(Z^{\top}Z), \tag{5.3}$$

where A' is the reconstructed structure matrix of the input graph.

Reconstruction Loss: The graph autoencoder learns to optimize the latent representation Z by minimizing the reconstruction error measuring the binary cross-entropy loss between the original structure A and the reconstructed A':

$$\mathcal{L} = \text{cross-entropy}(A, A'). \tag{5.4}$$

5.3.2 Cross-Graph Learning Framework

Graph autoencoders have superior ability embedding attributed graphs, but they lack protection against graph corruptions. Specially facing structural corruption, adding spurious edges to the graph substantially weakens the performance of the graph autoencoders (as shown in Fig. 5.1). To confront this challenge, we develop a Cross-Graph learning framework.

Our framework trains two graph autoencoders simultaneously (Fig. 5.2). In each epoch t, the two autoencoders take their graph structure $A_1^{(t)}$ and $A_2^{(t)}$ (which are both A in the first epoch) as the input, and obtain their reconstructed structure ${A'_1}^{(t)}$ and ${A'_2}^{(t)}$:

$$A'^{(t)}_1 = Dec_1(Enc_1(X, A_1^{(t)})), \tag{5.5}$$

$$A'^{(t)}_2 = Dec_2(Enc_2(X, A_2^{(t)})), \tag{5.6}$$

$$A_1^{(1)} = A_2^{(1)} = A. \tag{5.7}$$

Then, new structure matrix $A_1^{(t+1)}$ and $A_2^{(t+1)}$ are constructed, as a compromise between the input graph structure $A_1^{(t)}\&A_2^{(t)}$ and the reconstructed $A'^{(t)}_1\&A'^{(t)}_2$:

$$A_1^{(t+1)} = (1-\gamma)A_2^{(t)} + \gamma A'^{(t)}_2 \cdot A_2^{(t)}, \tag{5.8}$$

$$A_2^{(t+1)} = (1-\gamma)A_1^{(t)} + \gamma A'^{(t)}_1 \cdot A_1^{(t)}, \tag{5.9}$$

where γ is a coefficient control the balance between the two structure matrices, which we can adjust with the valid data. We dot product the $A'^{(t)}$ part with $A^{(t)}$ to make sure it learns the edge values taking the current structure as the frame of reference. Please notice that the new structure matrices are formed based on the information from the peer-autoencoder. In the following epoch $t+1$, the autoencoders use the updated graph structure ${A_1}^{(t+1)}$ and ${A_2}^{(t+1)}$ from the previous epoch t as their input. The randomly initialized weights of the autoencoder can prevent the two autoencoders from being too correlated.

5.3.3 Algorithm Description and Deeper Insights

The algorithm is summarized in Algorithm 4. In each iteration, we obtain the reconstruction from each autoencoder. The input structure compromise with it and an updated structure is thereby formed, which is further provided to the peer-autoencoder as the input structure of the next iteration. We use the hidden representation in the last iteration as our final embedding results. Fig. 5.3 gives an illustrated example on our procedure to update the graph structure to handle corrupted graphs.

Our Cross-Graph learning framework is simple but effective, which keeps up the performance of graph autoencoders against spurious edges. In the following, we will provide deep insights into our algorithm through answering two key questions.

Question 1: Why can manipulating the graph structure A towards the reconstructed structure A' help improve the robustness against structural corruption?

To answer question 1, we need to make it clear in advance what the elements in A and A' represent. A is the adjacency matrix of the graph, in which all elements are 0 or 1. $A_{jk} = 1$ shows there is an edge connecting node j and k in the graph, otherwise the two nodes are disconnected. On the other hand, A' is the reconstructed structure matrix, since it is constrained by the sigmoid function, its elements are all valued between 0 and 1. Therefore, A'_{jk} can be regarded as a predicted probability

Algorithm 4 Cross-Graph Autoencoder

```
Require:
  $X$: The feature matrix of the graph;
  $A$: The adjacency matrix of the graph;
  $T$: The number of iterations;
Ensure:
  $Z1$ & $Z2$: the learnt embeddings of the autoencoders;

  Construct the two autoencoders.
  for $t = 1$ to $T$ do
    Calculate $A'^{(t)}_1$ and $A'^{(t)}_2$ according to Eq.(5.5), Eq.(5.6) and Eq.(5.7);
    Construct $A^{(t+1)}_1$ and $A^{(t+1)}_2$ according to Eq.(5.8) and Eq.(5.9) for the next iteration to use.
  end for
  Calculate $Z_1$ & $Z_2$ according to Eq.(5.2) with $A^{(T)}_1$ and $A^{(T)}_2$ being the input $A$ respectively.
```

whether node j and k are connected with an edge from the autoencoder.

Furthermore, we do not fully trust the connections shown in the input graph structure. So, we take the reconstructed structure partly as a reference, to construct a new adjacency matrix for the training of the next iteration. The coefficient γ can be seen as the degree how the structure is trusted. So here comes a further question: Why the reconstructed structure can be trusted to judge if an edge is an outlier or not?

When the optimization starts, the autoencoder tries to reconstruct the input structure. Deep Neural Networks (DNNs), including graph autoencoders, have a strong ability fit with noisy data. However, according to a recent discovery of the

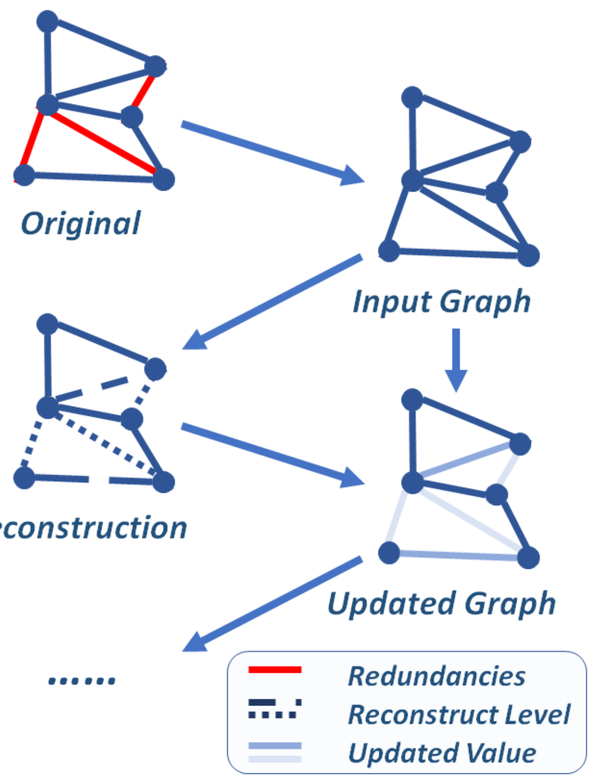


Figure 5.3 : The Cross-Graph Working Mechanism. The reconstruction level (the reconstruction's similarity to the input graph) could show its opinion of the possibility of each edge being spurious, represented by the sparsity degree of the dotted line. So, based on it and the input graph, our Cross-Graph constructs a new updated graph, which devalue those suspected edges. The edge value is positively correlated with the reconstruction level, represented by the gradation of the edge color. This updated graph is provided to the peer-autoencoder for the next iteration update.

DNN memorization mechanism, the network will better fit those easy and clean instances first (Arpit et al., 2017; Zhang et al., 2016). Thus, when the reconstructed A' tries to fit the input A, those elements trying to fit the real 1s in A will converge to 1 faster, while those trying to fit the corruptions will converge relatively slower. Therefore, when A compromises with the reconstructed A', those suspect edges in A will be given a balanced value instead of 1 to represent the reliability of the edge and supervise the subsequent training.

Question 2: Why do we need two autoencoders and that they update their parameters learning each other?

As for question 2, maintaining two autoencoders seems unnecessary, since a single autoencoder can work according to the above analysis. Indeed, a single autoencoder already has the ability to distinguish and filter out corruptions following our process. However, the performance of this will be unstable, because the reconstruction in the first few iterations are very random, highly depending on the initial parameter state.

Therefore, we further use two autoencoders, to reduce the impact of individual bias. Both autoencoders have an independent ability to learn and filter out corruptions. We exchange their reconstruction results so one might get advice from the other. Thereby, different corruptions can be selected out and our model is less possible to misjudge the corruptions because of a particular initial state.

Incorporate with Novel Algorithms other than GAE It is worth mentioning that, our Cross-Graph learning framework can work on, not only the basic graph convolutional autoencoder, but also other graph embedding frameworks, as long as they optimize their embedding by minimizing the difference between the original graph structure and the algorithm constructed structure.

5.3.4 Time Complexity Analysis

Our model is computationally efficient. The time complexity of a single graph convolutional layer should be $\mathcal{O}(ud_a + nd_1d_b)$ (Chiang et al., 2019), where u is the total number of edges, n is the number of nodes, d_a and d_b are the input and output dimension of the layer. GAE uses a two-layer convolutional encoder, which takes $\mathcal{O}(um + nmd' + ud' + nd'd)$, where d' is the dimension of the hidden layer, m is the input feature dimension and d is the latent representation dimension as defined in the problem definition. The decoder is an inner-product of the latent representation and takes $\mathcal{O}(dn^2)$. So, the GAE takes about $\mathcal{O}(um + nmd' + ud' + nd'd + dn^2)$. Our Cross-Graph maintains two autoencoders, and the construction of the updated graph structure basically takes $\mathcal{O}(n^2)$. So, the time complexity of the Cross-Graph incorporated GAE can be expressed as $\mathcal{O}(2(um + nmd' + ud' + nd'd + dn^2 + n^2))$, which is on par with the original GAE.

5.4 Experiments

Since we focus on unsupervised learning as explained before, we evaluate our graph embedding mainly with three unsupervised graph analytic tasks: node clustering, link prediction and network visualization. We conduct experiments for various algorithms on both corrupted and uncorrupted datasets to analyze the effect of our model.

5.4.1 Datasets

We use three graph datasets widely-used in attributed graph learning evaluation, summarized in Table 5.1. Cora and Citeseer are citation networks where nodes are publications categorized by the research sub-fields and edges denote the citation relationships. Wiki is a web-page network containing web-page documents, and the edges represent there are web-links between the two pages.

To construct the corrupted version of these datasets, we randomly add a certain percentage of edges to the network structure. We demonstrate in the parameter study that, our Cross-Graph can strengthen the performance of the baselines under any ratio of corruption, and with more edges added, the promotion become more apparent. So, we simply add 40%, 50% and 20% redundancy edges to the Cora, Citeseer and Wiki datasets respectively as default corruption to make a clear and comprehensive report in our experiments. For example, for the Cora dataset, originally with $5,278$ edges, we add 40%, that is $2,111$ spurious edges to the graph structure to determine our corrupted Cora dataset used in the experiments. All the experiments with corrupted data use this corruption ratio unless otherwise stated.

5.4.2 Baselines

We mainly focus on comparing the performance of different graph autoencoders with their performance when incorporating our Cross-Graph framework. So, we choose two representative graph autoencoders as our main baselines as below:

- **GAE** (Kipf and Welling, 2016b) leverages both topological and content information from the graph data with a graph convolution operation and learns unsupervised graph embedding.
- **ARGA** (Pan et al., 2018) is a graph convolutional autoencoder-based method that manipulates the learnt embedding with an adversarial regularizer.

For the clustering task, we also include some representative clustering algorithms: Spectral Clustering, Deepwalk and DNGR for comparison on the corrupted data, detailed description of these methods could be found in chapter 2.

5.4.3 Node Clustering on Corrupted Data

Evaluation Metrics

We follow Xia et al. (Xia et al., 2014), and employ six metrics to validate our clustering performance, namely accuracy (ACC), normalized mutual information (NMI), F-score, precision, recall and average rand index(ARI). A better clustering scheme should lead to higher scores for all these six metrics.

Experimental Setup

For the baseline algorithms, we keep the settings used in the original papers as far as possible and select parameters carefully following the procedures in the original papers. For example, for the GAE algorithm, we construct the encoders with a 32-neuron hidden layer and a 16-neuron embedding layer and train 200 epochs to

Dataset	Nodes	Features	Clusters	Links
Cora	2,708	1,433	7	5,278
Citeseer	3,327	3,703	6	4,552
Wiki	2,405	4,973	17	11,596

Table 5.1 : Benchmark Graph Datasets

Table 5.2 : Clustering Results on Corrupted **Cora** Dataset

	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
Spectral Clustering	33.55%	15.81%	26.81%	26.89%	26.80%	10.86%
DeepWalk	41.55%	20.48%	30.56%	30.97%	30.28%	15.57%
DNGR	38.33%	17.47%	26.40%	28.19%	24.84%	11.56%
GAE	45.19%	23.58%	11.73%	12.34%	12.41%	18.13%
GAE+Cross-Graph	51.05%	30.07%	18.00%	19.05%	18.74%	24.40%
ARGA	59.53%	37.92%	58.73%	59.80%	61.39%	32.98%
ARGA+Cross-Graph	64.25%	40.28%	61.63%	62.13%	63.73%	38.64%

Table 5.3 : Clustering Results on Corrupted **Citeseer** Dataset

	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
Spectral Clustering	27.64%	5.48%	23.98%	20.90%	28.56%	4.33%
DeepWalk	31.26%	5.91%	23.20%	22.35%	24.19%	5.69%
DNGR	21.23%	0.31%	30.29%	17.86%	99.71%	0.01%
GAE	34.42%	9.91%	16.12%	16.79%	16.63%	8.24%
GAE+Cross-Graph	37.33%	12.35%	16.64%	17.23%	17.01%	10.29%
ARGA	39.39%	13.87%	38.27%	41.64%	38.24%	11.58%
ARGA+Cross-Graph	42.82%	15.05%	41.81%	44.57%	41.40%	13.04%

get the result. On the other hand for ARGA, we use a 32-neuron embedding layer instead and train 100 epochs, in accordance with the original paper.

For our algorithm, we incorporate our Cross-Graph with two representative autoencoders for graph learning: Graph Convolutional Autoencoder (GAE) and Adversarially Regularized Graph Autoencoder (ARGA). For a fair comparison, the autoencoder part in our algorithm uses the exact same settings as the original method. For the Cross-Graph part, we set the coefficient γ to 0.02 in accordance with the parameter study result.

We learn the graph embedding for each method, and then perform k-means clustering on the embedding to get the results. We run the experiments under each setting 10 times to get an average score for report and comparison.

Experimental Results

The clustering results of various algorithms on the corrupted Cora, Citeseer and Wiki datasets are reported in Tables 5.2, 5.3 and 5.4 respectively.

It can be observed from the result that all the algorithms suffer from the corruption of the data. For instance, the DNGR algorithm completely loses its clustering ability on the corrupted Citeseer and Wiki datasets. Our Cross-Graph framework can improve significantly the performance of both GAE and ARGA when they are used to learn clustering from the corrupted datasets. Take the experiments on the corrupted Cora dataset as an example. Incorporating it with our Cross-Graph has increased the performance of GAE from 13.0% in terms of accuracy to 54.4% in terms of precision, and has also increased the performance of ARGA from 3.8% in terms of recall to 17.2% in terms of average rand index. Both comparisons demonstrate the ability of our Cross-Graph to increase robustness against edge corruption.

Detailed Analysis. We further monitor the value of both the spurious and original edges respectively in the Cora and Citeseer datasets along with the optimization of both GAE and ARGA. The change of averaged values is presented in Fig. 5.4.

The results show a clear trend. Both the original and redundancy edges are valued universally as 1 at the beginning of the training, because they have no difference in the input graph. However, our Cross-Graph process has a strong ability to filter out the redundancy edges, based on the mechanism explained in Section 4. After 200 epochs of training for the GAE-based algorithm and 100 epochs for the ARGA-based counterpart (the default epochs for these algorithms), the original and redundancy edges show a clear gap. Taking the GAE-based Cross-Graph as an example, the original edges keep a relatively high value, the average is 0.543 on the Cora dataset and 0.607 on the Citeseer dataset; while the average value of redundancy edges has decreased on these datasets to 0.322 and 0.399 respectively, much closer to 0, which means there is no edge in between. The difference between ARGA and ARGA-based Cross-Graph is not as high as the GAE-based one. This perhaps, mainly because (1) it has fewer training epochs by default; and (2) the adversarial regularizer brings ARGA a stronger ability in fitting those abnormal edges. This fact also results in an insignificant performance improvement on ARGA, compared to the improvement on GAE by our Cross-Graph.

Parameter Study. We vary the corruption level of the datasets by continuously adding random edges to the original graph structure, until the total number of added edges reaches 50% of the original number. We report the clustering results in Fig. 5.5.

It can be observed that, our Cross-Graph can continuously help improve the performance of the original algorithm, no matter how many edges we add to the graph. Even on the original graph without any redundancy edges, GAE + Cross-

Table 5.4 : Clustering Results on Corrupted **Wiki** Dataset

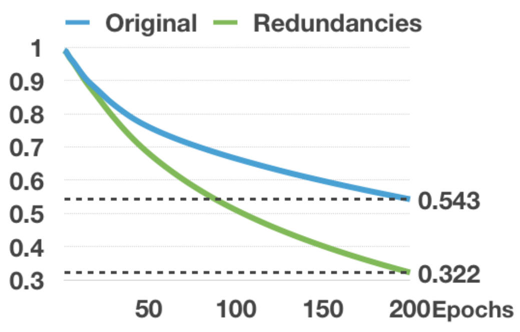
	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
Spectral Clustering	30.06%	23.01%	19.06%	19.71%	18.55%	10.56%
DeepWalk	36.70%	28.77%	24.41%	25.71%	23.31%	16.64%
DNGR	16.76%	1.46%	17.92%	9.85%	98.99%	0.07%
GAE	19.16%	13.09%	2.96%	3.99%	4.37%	4.55%
GAE+Cross-Graph	22.37%	18.97%	3.23%	5.83%	4.49%	5.96%
ARGA	40.45%	41.40%	37.86%	45.62%	39.66%	21.32%
ARGA+Cross-Graph	43.45%	42.97%	39.46%	47.97%	43.05%	22.35%

Graph also slightly outperforms GAE in the clustering task. Generally, the more serious the corruption is, the more effective our Cross-Graph is. This is because our Cross-Graph is designed for robust learning and can retain performance under extremely serious corruption.

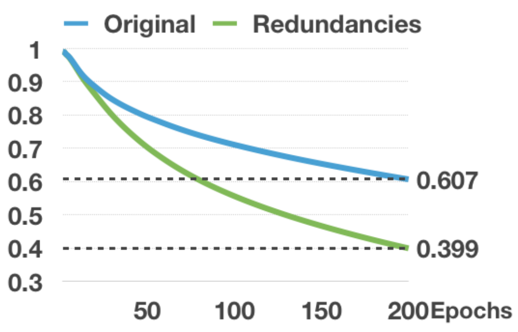
We also vary the coefficient γ in our Cross-Graph framework. This controls the balance between the input and reconstructed structure in the updated graph. The result is reported in Fig. 5.6.

The result from both datasets reveals a similar trend: the performance of the learnt embedding is well when the coefficient γ is small, and the performance reaches its peak when γ is around 0.02. As we increase the value of γ from 0.02, the performance fluctuates and deteriorates overall. This is because (1) the change of the input graph structure is too rapid for the neural network to become stabilized; (2) the reconstructed structure is trusted too much and the graph modification is overdone.

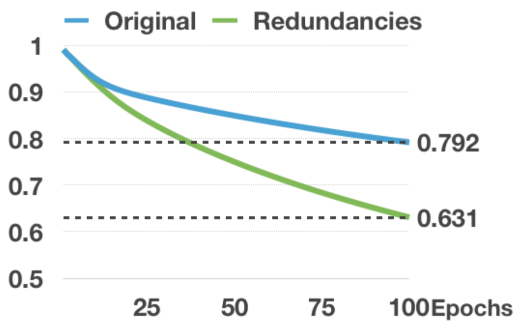
Effectiveness of the Cross-Graph Dual-autoencoders interactive process.



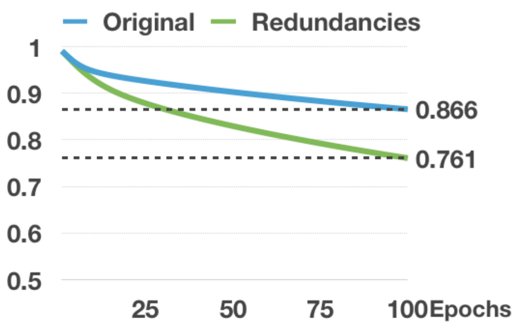
(a) GAE+Cross-Graph on Cora



(b) GAE+Cross-Graph on Citeseer



(c) ARGA+Cross-Graph on Cora

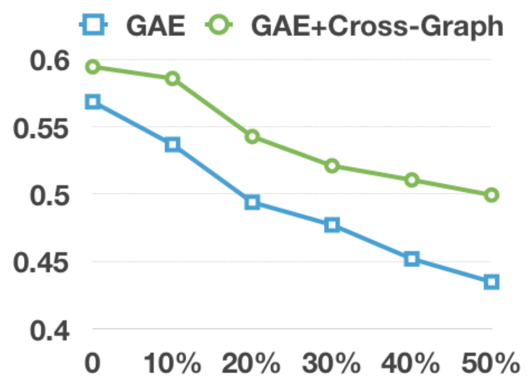


(d) ARGA+Cross-Graph on Citeseer

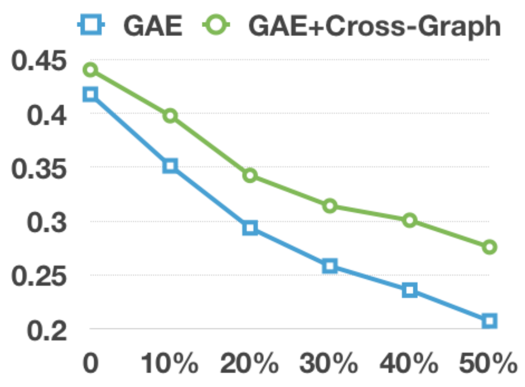
Figure 5.4 : The devalue of two types of edges (original edges and redundancy edges we added) along with the Cross-Graph training process.

As discussed above, a single autoencoder can already work to filter out the corruptions. We use two autoencoders to make the framework more stable. To show the effectiveness of this interactive process, we compare our Cross-Graph with a framework which maintains only one autoencoder. The input graph for this autoencoder is updated according to its own reconstruction result. We keep all the other settings the same, and run each framework for 20 times. The performance evaluated by the clustering accuracy is reported in Fig. 5.7.

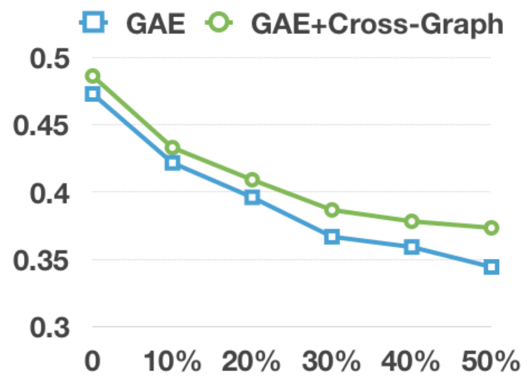
We can observe from the box plots that, though the average performance of the single-autoencoder framework is nearly on par with our Cross-Graph, its 20 results are more dispersive. For example for the Cora dataset, all 20 results from our Cross-Graph lie above 0.45, and half of them are concentrated between 0.50 and 0.54. On



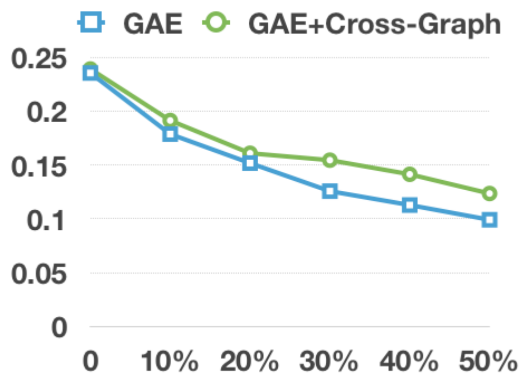
(a) ACC on Cora



(b) NMI on Cora



(c) ACC on Citeseer



(d) NMI on Citeseer

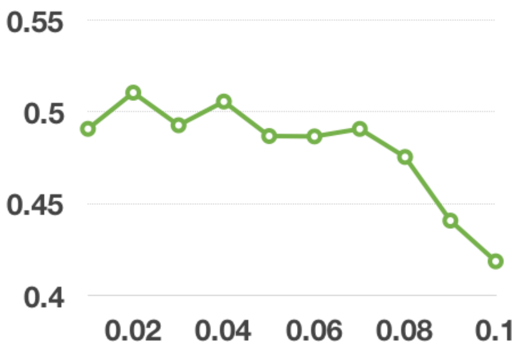
Figure 5.5 : The clustering performance under different percentage of redundancy edge corruption.

the other hand, the results from the single-autoencoder framework vary from 0.40 to 0.57. This show our Cross-Graph interactive process can reduce the impact of individual bias and stabilize the framework.

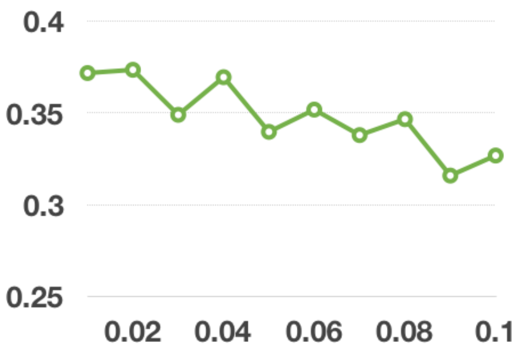
5.4.4 Link Prediction on Corrupted Data

Evaluation Metrics We report the results of AUC score and AP score (average precision) follow Pan et al.(Pan et al., 2018). A better link prediction result should lead to a higher score for both metrics.

Experimental Setup For the link prediction task, we mask 5% edges for hyper-parameter optimization and 10% edges for performance test. We mask the valid

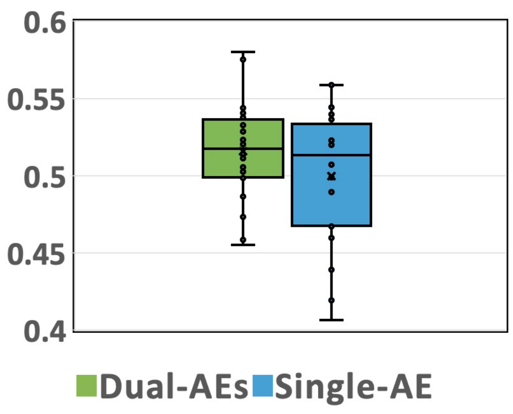


(a) ACC on Cora

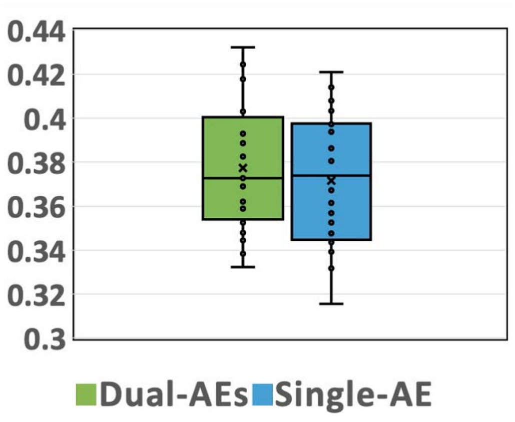


(b) ACC on Citeseer

Figure 5.6 : The clustering performance with different Cross-Graph coefficient γ.



(a) Cora



(b) Citeseer

Figure 5.7 : Box plots of the 20 times' clustering accuracy from the Cross-Graph Dual-autoencoders interactive process, compared with a single autoencoder framework. For each box, the five lines from top to bottom represents the maximum, the first quartiles, the sample median, the third quartiles and the minimum of the clustering results.

Table 5.5 : Link Prediction Results on Corrupted Datasets

	Cora		Citeseer		Wiki	
	AUC(↑)	AP(↑)	AUC(↑)	AP(↑)	AUC(↑)	AP(↑)
GAE	84.03%	86.17%	80.35%	82.88%	75.05%	76.56%
GAE+Cross-Graph	85.72%	87.63%	81.83%	84.07%	76.87%	78.55%
ARGA	86.84%	88.70%	87.29%	89.49%	91.53%	92.71%
ARGA+Cross-Graph	87.74%	89.81%	88.38%	90.49%	92.28%	93.37%

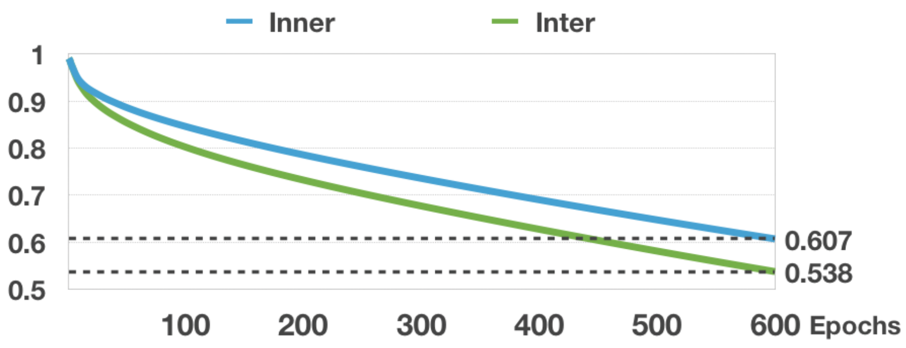


Figure 5.8 : The devalue of inner-edges and inter-edges comparison along with the Cross-Graph training process on uncorrupted Cora dataset.

Table 5.6 : Clustering & Link Prediction Results on Uncorrupted **Cora** Dataset

	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)	AUC(↑)	AP(↑)
GAE	56.85%	41.76%	13.48%	14.55%	14.23%	31.71%	90.93%	92.00%
GAE+Cross-Graph	59.45%	44.04%	13.69%	14.69%	14.64%	33.61%	91.39%	92.43%

and test sets before we corrupt the dataset, to make sure our test sets are clean, as we do not see it as a plus in predicting those redundancy edges. For the baselines, we retain the parameter settings described in the original papers.

Experimental Results The detailed results are shown in Table 5.5. We can observe that our Cross-Graph also helps retain the link prediction performance for the based algorithms under structural corruption. For example, the ARGA algorithm originally can achieve both AUC score and AP score as high as 92% on the Cora and Citeseer dataset. Both scores drop to about $86\% - 89\%$ under the corruption. By incorporating our Cross-Graph, we can recover the performance for about 1%. The effect of Cross-Graph is not as strong as in the clustering task, probably because the structural corruption is much more severe for the link prediction task.

5.4.5 Experiments on Uncorrupted Data

An interesting observation of our framework is that it can also outperform its base method on uncorrupted dataset. To deeper analyze the principle of our Cross-Graph, we also implement our algorithm incorporating GAE on the original "clean" $Cora$ dataset. The result is summarized in Table 5.6.

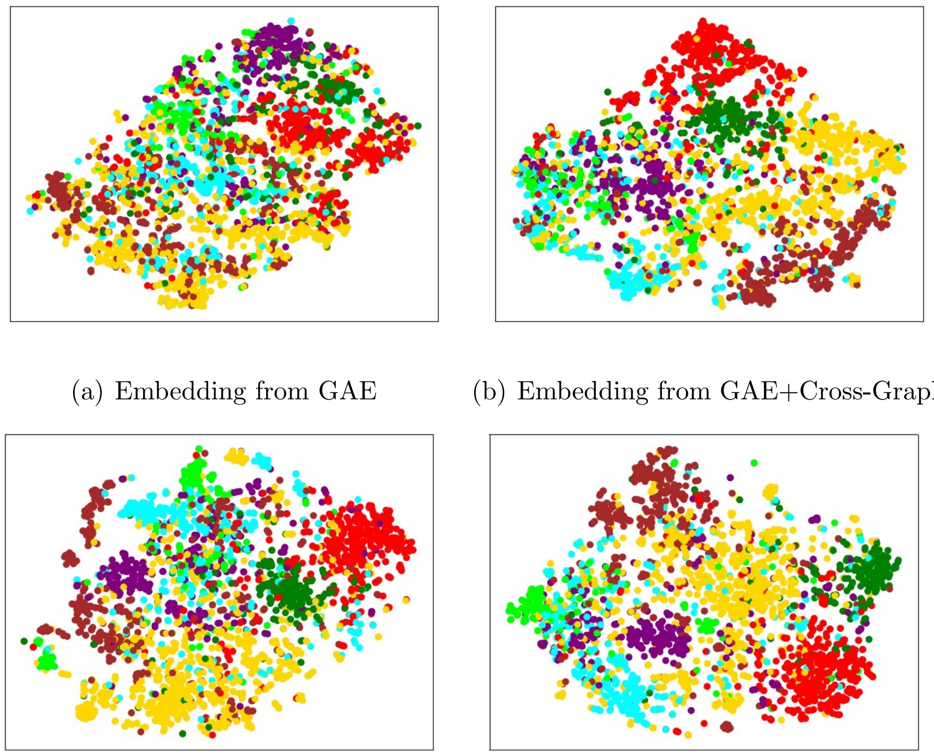
From the node clustering and link prediction task performance, we can see GAE + Cross-Graph can also slightly outperform GAE on the uncorrupted dataset and at least match the performance of GAE along all the evaluation metrics.

Since our Cross-Graph's main ability is to revalue the edges, we also try to monitor two kinds of edges in this "clean" dataset: the edges linking nodes from the same cluster as inner edges, and the edges that link nodes from two different clusters as inter edges. Their average value changes, along with the training process, are simulated in Fig. 5.8.

Fig. 5.8 shows that inter-edges connecting nodes from two different clusters devalue much faster than the inner-edges. This is because each cluster of nodes are closely linked with inner-edges. Inter-edges connecting two nodes from different clusters are rare and distant from the clusters. Not surprisingly, these inter-edges are also called weak links in the network and are hard to reconstruct. This will lead our Cross-Graph to mark them as highly abnormal and devalue these inter-edges faster than the inner-edges which, in return, strengthen the structure of each cluster and improve the clustering performance.

5.4.6 Network Visualization on Corrupted Data

We visualize the corrupted Cora dataset in a two-dimension space by applying the t-SNE algorithm (Van Der Maaten, 2014) on the embedding learned from different algorithms. The visualizations in Fig. 5.9 show that, due to the structural corruption



(a) Embedding from GAE

(b) Embedding from GAE+Cross-Graph

(c) Embedding from ARGA

(d) Embedding from ARGA+Cross-Graph

Figure 5.9 : Visualizations of the corrupted Cora dataset, based on the embedding learned from various algorithms. The dots represent nodes and the seven different colors represent the ground-truth clusters the nodes belongs to.

of the Cora dataset, GAE already can hardly maintain a meaningful embedding, and ARGA's embedding performance is also severely ruined. Fortunately, by applying Cross-Graph to the original graph autoencoders, the embedding could be effectively recovered, allowing us to obtain a more meaningful visualization layout of the graph data.

Chapter 6

Learning from Side Information
Constrained Graph Clustering with Contrastive Regularized Autoencoder

Our last challenge aforementioned is to make use of available side information. To this end, in this chapter we propose a constrained node clustering framework that can improve the clustering performance using pair-wise constraint information.

6.1 Background

Attributed graph is a kind of structured data format, in which the plain node attribute information is combined with the topological structure information of the graph. With the development of network applications, the mining of attributed graph has attracted much more attention, since many real-world networks, such as social networks, citation networks, and protein-protein interaction networks can all be modeled as attributed graphs.

Node clustering aims to partition the nodes in a graph into disjoint groups. It has been a long-standing research topic. In recent years, node clustering for attributed graphs has been more actively researched. A key challenge is to simultaneously model the structural relationship between nodes and exploit the node content information. Various recent attributed node clustering approaches have been proposed to confront it, utilizing non-negative matrix factorization (Li et al., 2018b; Yang et al., 2015), content propagation (Liu et al., 2015), generative models (Yang et al., 2013; He et al., 2017) or GCN-based autoencoders (Kipf and Welling, 2016b; Pan et al., 2018). The results reported by these methods on several classic benchmark datasets have suggested a looming limit to attributed graph clustering performance.

Attributed graph clustering is a more challenging task then classification due to the absence of label supervision, and precisely because of it, node clustering earns its place in real-world applications. In many scenarios, it is expensive to mark labels for the training data. For instance, in a social network, detecting communities is helpful for recommendation and statistics analyzing, but these communities are only ambiguous groups of users with some similarities and hard to define; In citation networks, there are intricate research fields, interdisciplinary research papers are also commonplace.

On the other hand, the situation for constraint information is quite different. in many cases, constraints are much easier to obtain compared with specific labels defining which cluster the nodes belong to. If two users share very similar attributes like affiliation and hobbies in a social network, there should be a high possibility they will be assigned to the same community; If two papers have the same author, they are very likely to belong to the same research field even if there is no citation in between. These kinds of constraint information are also helpful. If we can make use of this information in our clustering, the performance will certainly be improved.

Motivated by these observations, in this paper, we proposed a constrained node clustering framework for attributed graphs which can improve the clustering performance using pair-wise constraint information. To explore the interaction between the graph structure and node content, we employ a graph convolution-based autoencoder, which learns node representation from the graph. A contrastive loss-based graph regularizer is further designed, to manipulate the embedding learning according to the prior pair-wise limitation. The learned embedding could therefore involve the constraint information. The pairs of nodes indicated to belong to the same community by the prior will learn similar embedding in the latent space, and further

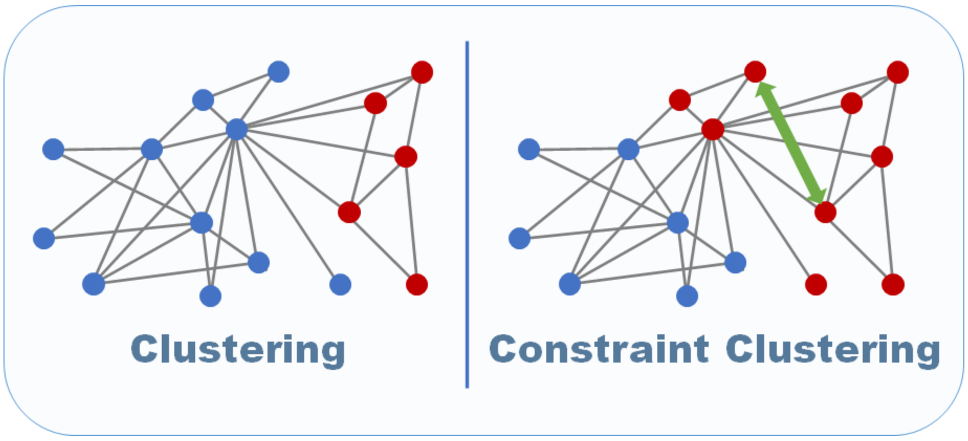


Figure 6.1 : The difference between constrained node clustering and normal node clustering. Constraint information (the green arrow in the Figure) can define some pairs of nodes that should be in the same cluster, and therefore may help partition some equivocal marginal nodes to the right cluster.

naturally assigned to the same cluster.

We summarize the main contributions as follows:

- To the best of our knowledge, we propose the first attributed graph-based node clustering algorithm that takes pair-wise constraint information into account.
- We propose a regularized graph autoencoder, which utilizes a contrastive regularizer to model the prior constraint information and learn deep graph embedding for clustering. The clustering result benefits from the graph structure, the node features as well as the pair-wise constraints.
- We conduct experiments and compare our model with both novel unsupervised node clustering baselines and constrained clustering methods. The results show that our model can sufficiently make use of the constraint information and significantly improve the node clustering performance.

6.2 Problem Definition

We consider the node clustering task on the attributed graphs. An attributed graph is represented as $G = (V, E, X)$, in which $V = \{v_i\}_{i=1,\cdots,n}$ consists of a set of nodes, $E = \{e_{ij}\}$ is a set of edges connecting the nodes. The topological structure of graph G can be represented as an adjacency matrix A, where $A_{i,j} = 1$ if $(v_i, v_j) \in E$; otherwise $A_{i,j} = 0$. $X = \{x_1; \ldots; x_n\}$ are the attribute values where $x_i \in R^m$ is a m-dimension real-value attribute vector associated with node v_i.

The clustering task aims to partition the nodes in the graph G into k disjoint clusters $\{G_1, G_2, \cdots, G_k\}$, so that nodes within the same group are generally: (1) close to each other in terms of graph structure while distant otherwise; and (2) more likely to have similar attribute values.

For our purpose, we add constraints as third party information besides the node attributes X and graph structure A. Such constraints may be represented as a constraint list $C \in R^{o\times 2}$, where o is the number of given node pairs, and each row includes a pair of nodes i and j that should be clustered together.

We aim to make use of these constraints to improve the clustering performance compared with those purely unsupervised clustering schemes.

6.3 Proposed Method

We present our contrastive regularized autoencoder framework in this section. We first briefly introduce our employed graph convolutional autoencoder. Then we present our contrastive loss-based regularization term which handles the pair-wise constraints.

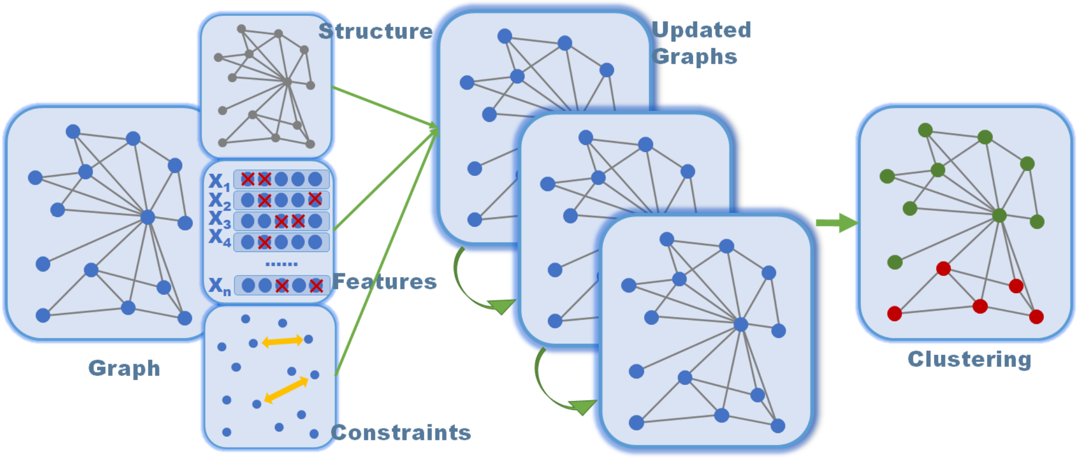


Figure 6.2 : Conceptual framework of the Contrastive Regularized Autoencoder for Constrained Graph Clustering. Given a graph G with graph structure matrix A, node content matrix X, and a list of node pairs that are constrained to the same cluster, we aim to learn effective clustering assignment. We train a graph autoencoder to integrate graph structure and node content information into a latent node embedding. When optimizing the autoencoder reconstruction loss, we together minimize a contrastive loss function that forces the constrained node pairs to learn similar embedding at the same time. The learned embedding is influenced by all three aspects of the information and used for clustering.

6.3.1 Graph Autoencoder

Autoencoders are widely used for unsupervised tasks such as graph clustering. To capture the deep information of the networked data, we employ a classic convolution-based graph autoencoder. The graph autoencoder consists of a graph convolutional encoder and an inner-product decoder.

Graph Convolutional Encoder: The graph convolutional autoencoder integrates the two-sides attributed graph information through an encoder based on the graph convolution operation, and learns latent embedding of the nodes. Its layer-wise transformation can be represented as $f(Z^{(l)}, A)$:

$$Z^{(l+1)} = f(Z^{(l)}, A) = \sigma(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}Z^{(l)}W^{(l)}). \tag{6.1}$$

Here, σ is a nonlinearity function and $W^{(l)}$ is the weight matrix. $Z^{(l)}$ is the input to the l-th layer graph convolution function, and $Z^{(l+1)}$ is the corresponding output of the layer.

We use $\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}$ to approximate the spectral convolution transformation $I + D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, with $\widetilde{A} = A+I$, $\widetilde{D}_{ii} = \sum_j \widetilde{A}_{ij}$, and I being the identity matrix. Thereby, the node attributes are combined with attributes from the neighbor nodes through the convolution operation, and the convoluted node representation contain information from both the node content and graph structure.

The standard graph encoder $Enc(X, A)$ takes the node content X as the input to the first layer $Z^{(1)}$, and stacks two graph convolution layers to obtain a hidden representation Z:

$$Z = Enc(X, A) = f(f(X, A), A). \tag{6.2}$$

Inner Product Decoder: The decoder tries to reconstruct the graph structure information from the hidden representation. Previous works have tried various decoders reconstructing different sides of the graph information, and reconstructing the graph adjacency matrix A has been validated as the best solution, since it also can be more flexible and fit situations where no content information is available. Our inner-product decoder contains only a link prediction layer $Dec(Z)$. Based on the hidden graph representation, its output A' is the reconstructed graph structure, and tries to indicate whether there is an edge between two nodes like the original adjacency matrix A do:

$$A' = Dec(Z) = \text{sigmoid}(Z^{\top}Z), \tag{6.3}$$

Reconstruction Loss: The graph autoencoder minimizes the reconstruction error measuring the binary cross-entropy loss between the original structure A and the reconstructed structure A':

$$\mathcal{L}_r = \text{cross-entropy}(A, A'). \tag{6.4}$$

The weight matrix $W^{(l)}$ is updated through the training and thereby the node representation Z is optimized.

6.3.2 Contrastive Regularizer

Through the graph autoencoder, we obtain a graph embedding Z, which is learned from the node attributes and the graph structure. However, existing graph autoencoders lack the ability to handle constraint information. As we argued above, constraint information is easily accessible and useful when dealing with unsupervised problems like clustering. To make use of the constraint information and improve the unsupervised learning ability of current models, we propose a contrastive regularizer.

Apart from optimizing the reconstruction error of the autoencoder, we also optimize a contrastive loss to model the prior constraints information and manipulate the embedding learning.

Consider prior information that node i and j are highly related and should be assigned to the same cluster. z_i and z_j are the autoencoder-learned embeddings of node i and j (the i-th and j-th row in the hidden representation Z). For the clustering algorithm to cluster these two nodes together, we naturally need to force these two embeddings similar to each other. For this purpose, we minimize the following contrastive loss function:

$$\mathcal{L}_{c(i,j)} = -\log \frac{\exp(z_i \cdot z_j / \tau)}{\sum_{u=1}^{n} \exp(z_i \cdot z_u / \tau)}. \tag{6.5}$$

Here, τ is a temperature hyper-parameter. We measure the similarity between z_i and z_j using dot product. The denominator is summed over all the n nodes of the graph. The whole contrastive loss could be regarded as a n-way softmax classifier that wants to classify z_i to z_j.

Obviously, the contrastive loss is a function whose value is low when the target (z_i) is similar to the positive sample (z_j) and dissimilar to the negative samples (the other nodes). So by minimizing the contrastive loss, we can force z_i and z_j to be closer to each other, and thereby node i and j are more likely to be clustered together.

Considering multiple pairs of highly-related nodes that should be clustered together as prior information. Our contrastive regularizer can be formulated as:

$$\mathcal{L}_c = \sum_{(i,j)\in C} -\log \frac{\exp(z_i \cdot z_j/\tau)}{\sum_{u=1}^{n} \exp(z_i \cdot z_u/\tau)}. \tag{6.6}$$

6.3.3 Joint Embedding Learning

We jointly optimize the autoencoder reconstruction and the contrastive loss. Our total objective function of the framework can be presented as:

$$\mathcal{L} = \mathcal{L}_r + \gamma\mathcal{L}_c, \tag{6.7}$$

where L_r and L_c are the reconstruction loss and contrastive loss respectively, $\gamma \geq 0$ is a coefficient that controls the balance in between.

6.3.4 Framework Description

The algorithm is summarized in Algorithm 5. Our framework is simple but effective, which sufficiently take advantage of the constraint information to improve the autoencoder embedding performance.

It is worth mentioning that, our contrastive regularizer can also be adopted by other graph embedding methods to handle extra constraint information.

Dataset	Nodes	Features	Clusters	Links
Cora	2,708	1,433	7	5,278
Citeseer	3,327	3,703	6	4,552
Pubmed	19,717	500	3	44,338

Table 6.1 : Benchmark Graph Datasets

6.4 Experiments

We evaluate the performance of our framework in this section. We conduct experiments for various methods and settings, to analyze the effect of our model.

6.4.1 Benchmark Datasets

We employ three graph datasets widely-used in attributed graph learning evaluation, summarized in Table 5.1. These datasets are citation networks where nodes are publications categorized by the research sub-fields and edges are the citation relationships.

6.4.2 Baselines

We focus on comparing the performance of different models on the graph clustering task. Constraint-accessible models naturally have an advantage over traditional graph clustering methods. We therefore mainly compare with constrained clustering methods, although there is no constrained clustering baseline perfectly suits the attribute graph data, to the best of our knowledge. We select the following constrained clustering baselines:

- **COP** (Wagstaff et al., 2001) is a variant of the k-means algorithm considering constraint information.

Table 6.2 : Clustering Results on **Cora** Dataset

	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
K-means	F	50.01%	31.67%	37.56%	37.60%	37.63%	23.92%
Spectral	S	39.83%	29.69%	33.16%	31.22%	35.46%	17.44%
TADW	F&S	53.64%	36.63%	40.08%	34.16%	49.24%	24.03%
GAE	F&S	56.72%	41.72%	41.48%	43.17%	40.06%	30.83%
AGC	F&S	68.89%	52.21%	65.60%	67.24%	67.49%	44.81%
COP	F&C	70.42%	73.78%	47.32%	43.48%	54.84%	60.86%
LSGR-NMF	S&C	71.16%	59.20%	61.12%	63.24%	59.14%	52.97%
LSGR-SC	S&C	83.35%	75.21%	81.80%	80.76%	82.87%	77.76%
CRA	**F&S&C**	**90.88%**	**78.24%**	**91.15%**	**89.77%**	**93.04%**	**78.59%**

Table 6.3 : Clustering Results on **Citeseer** Dataset

	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
K-means	F	54.39%	31.16%	41.34%	41.13%	41.62%	28.49%
Spectral	S	30.80%	9.04%	25.71%	24.13%	27.55%	8.24%
TADW	F&S	52.89%	31.95%	43.62%	37.63%	53.15%	28.55%
GAE	F&S	38.02%	17.41%	29.67 %	29.06%	30.44%	14.13%
AGC	F&S	67.28%	41.44%	62.67%	63.47%	63.09%	41.98%
COP	F&C	39.55%	57.55%	21.15%	45.44%	34.53%	34.20%
LSGR-NMF	S&C	53.99%	38.85%	44.70%	39.53%	51.43%	30.71%
LSGR-SC	S&C	42.06%	29.92%	33.32%	27.88%	41.40%	15.24%
CRA	**F&S&C**	**87.25%**	**69.74%**	**86.50%**	**85.95%**	**88.21%**	**72.60%**

- **LSGR** (Yang et al., 2014) is a semi-supervised community detection framework for topological-only network using latent space graph regularization. It can be combined with a base algorithm of Spectral Clustering or Non-negative Matrix Factorization. So we report both results as LSGR-SC and LSGR-NMF.

To have a clear view of the effort of the constraint information, we also list the performance of some representative graph clustering methods as shown in the tables. These methods without the constraint information assistance have been introduced in Chapter 2.

6.4.3 Evaluation Metrics

We follow Xia et al. (Xia et al., 2014), and employ six metrics to validate our clustering performance, namely accuracy (ACC), normalized mutual information (NMI), F-score, precision, recall and average rand index(ARI). A better clustering segmentation should lead to higher scores for all these six metrics.

6.4.4 Experimental Setup

For the baseline methods, we keep their original settings as far as possible, and carefully select parameters following the procedures in the original paper, to achieve the best performance.

For our algorithm, we set the temperature hyper-parameter τ to 0.08. The coefficient γ is simply set to 1. The encoder is constructed with a 32-neuron hidden layer and a 16-neuron embedding layer. We run the algorithms for 10 times to report a relatively steady average result.

As for the constraint information, we randomly generate pair-wise constraints for the constraint-accessible methods. For an undirected graph with n nodes, there are in total $N = n(n-1)/2$ pairs of potential pair-wise relationships. Among these N pairs, the number of possible node-pair prior information can be represented as:

$$N_m = \sum_{k=1}^{K} n_k(n_k - 1)/2,$$

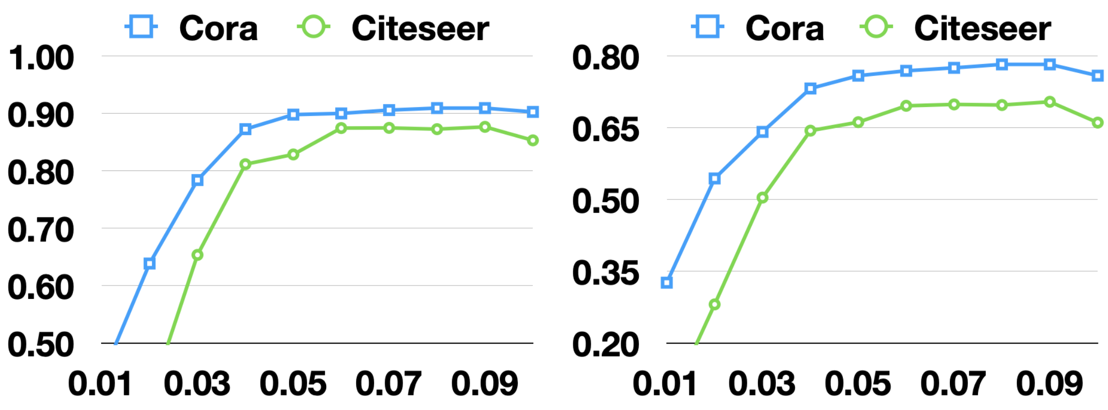
with n_k being the number of nodes belong to the k-th cluster according to the ground truth. The other $N - N_m$ potential node pairs are not supposed to be clustered together.

For convenience, we follow (Yang et al., 2014) and randomly select q node pairs belong to the same cluster as our standard constraint information and provide it to all constraint-accessible methods including our algorithm. q is 0.5 percent of the number of the total potential constraint links N_m for Cora and Citeseer dataset. For example for the Cora dataset with $2,708$ nodes, there are about 3.665 million of potential pair-wise relationships and among them, only $N_m = 657,055$ are potential constraint links. We therefore randomly generate 3,285 pair-wise relationships as our standard constraint information for the Cora dataset. For the Pubmed dataset, we set q to only 0.01 percent of N_m, since Pubmed has too many potential pair-wise relationships.

6.4.5 Experimental Results

The clustering results of various algorithms on the datasets are reported in Tables 5.2, 5.3 and 5.4 respectively. F, S and C indicate if the algorithm has taken advantage of the node feature, graph structure, and pair-wise constraint information.

It can be observed from the results that constraint-accessible methods have a general advantage over the traditional graph clustering methods. It's natural since more information is provided. For example on the Cora dataset, our contrastive regularized autoencoder represents a relative increase of 60.23% and 87.54% w.r.t. accuracy and NMI against the basic graph autoencoder (GAE); LSGR-SC also outperforms the original Spectral Clustering apparently. COP k-means and LSGR failed to perform well on the Citeseer dataset, may because they are not designed



(a) ACC (b) NMI

Figure 6.3 : Average clustering performance evaluated by ACC and NMI, with different temperature parameter τ, from 0.01 to 0.1.

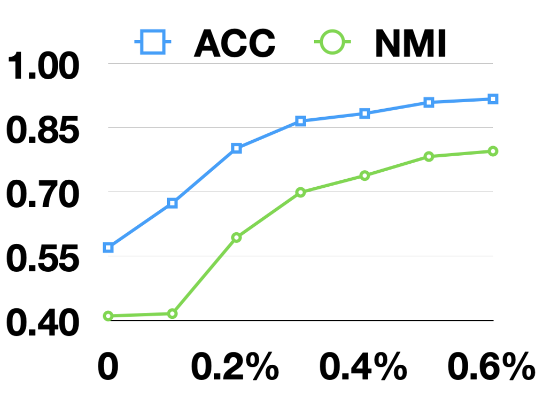
for attributed graphs and cannot well model the complex relationship among the different aspects of information. Citeseer is a relatively delicate dataset and the embedding could become unstable due to some constraint-based operation.

Our CRA obtain satisfactory results on all three datasets, with only very few pair-wise constraints information. The reasons may include: (1) the constraint information is powerful helping with unsupervised tasks like clustering; (2) the contrastive loss efficiently force constrained nodes to be similar to each other; and (3) Our model is specially designed for constrained clustering for attributed graph, and can properly exploit the interaction among the three aspects information.

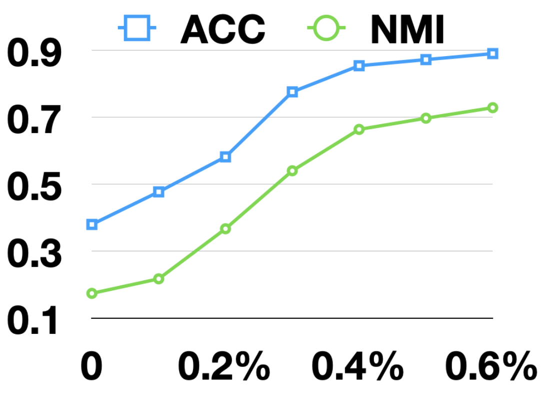
6.4.6 Parameter Study

We vary the temperature hyper-parameter from 0.01 to 0.1 and report the results from both Cora and Citeseer datasets in Fig. 6.3.

The results show that, when increasing the parameter τ from 0.01 to about 0.05, the clustering performance of the learned embedding improves significantly. When τ increase from 0.05 to 0.1, the clustering performance is not much influenced. The performance reaches its peak when τ is set to about 0.08 for both Cora and Citeseer



(a) Cora dataset



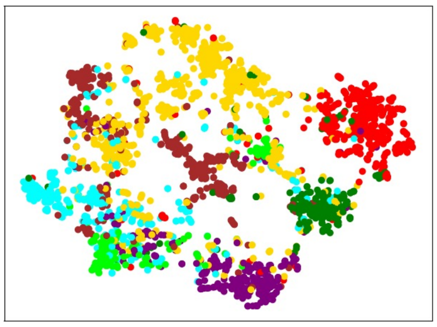
(b) Citeseer dataset

Figure 6.4 : Average clustering performance with different number of constrained node pairs. The percentage is based on the total potential constraint links N_m.

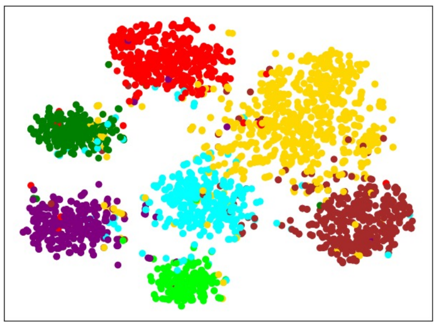
Table 6.4 : Clustering Results on **Pubmed** Dataset

	Info.	ACC(↑)	NMI(↑)	F(↑)	P(↑)	R(↑)	ARI(↑)
K-means	F	57.99%	27.82%	54.43%	48.82%	62.07%	24.56%
Spectral	S	49.67%	14.69%	47.09%	40.66%	56.13%	9.82%
TADW	F&S	56.47%	22.40%	48.11%	46.48%	49.96%	17.73%
GAE	F&S	62.39%	24.92%	51.17%	51.82%	50.56%	24.66%
AGC	F&S	67.92%	30.63%	68.84%	73.32%	69.49%	31.16%
COP	F&C	60.27%	29.00%	57.79%	51.56%	61.35%	2.02%
CRA	**F&S&C**	**86.56%**	**56.14%**	**86.15%**	**85.50%**	**87.41%**	**63.67%**

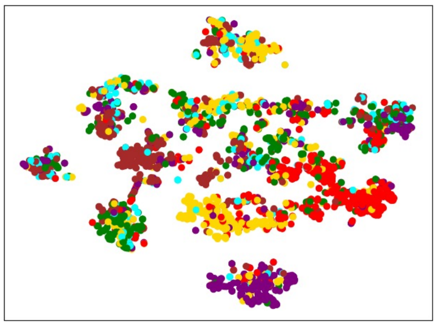
[1] The Pubmed dataset is too large for the LSGR algorithm to run.



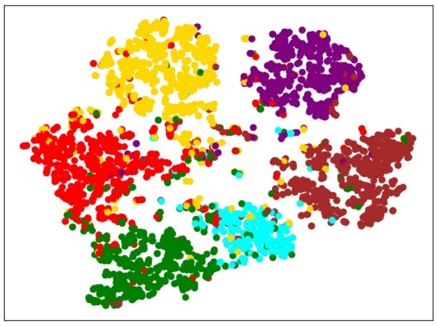
(a) Cora learned from GAE



(b) Cora learned from CRA



(c) Citeseer learned from GAE



(d) Citeseer learned from CRA

Figure 6.5 : Visualizations of the Cora and Citeseer datasets, based on the embedding learned from GAE and our CRA. The dots represent nodes and the seven different colors represent the ground-truth clusters the nodes belong to.

datasets.

We also vary the number of pair-wise constraints provided as prior information to our method, from no constraint provided to 0.6 percent of the total potential pairs N_m. The results are reported in Fig. 6.4.

The results from both Cora and Citeseer dataset reveal a similar trend: Providing the constraint information can significantly help to improve the quality of the learned embedding. The improvement is tremendous with little constrained pairs, and the increase rate gradually decays with more pairs provided (but the performance keeps improving).

6.4.7 Network Visualization

We also visualize the Cora and Citeseer datasets in a two-dimension space by applying the t-SNE algorithm (Van Der Maaten, 2014) on the embedding learned from both our CRA and a basic graph autoencoder. The visualization in Fig. 5.9 shows that with the help of constraint information, CRA has learnt obviously better node embedding compared with the original autoencoder.

Algorithm 5 Contrastive Regularized Autoencoder

```
Require:
  $G = V, E, X$: a graph with links and node features;
  $C = [(i_1, j_1), (i_2, j_2), \ldots, (i_o, j_o)]$: Constraint Node-Pair List;
  $k$: Number of clusters;
  $Iter$: Number of iterations;
  $\gamma$: Loss Balance Coefficient;
Ensure:
  Final constrained clustering results.

  for $l = 0$ to $Iter - 1$ do
    Calculate the convoluted graph embedding $Z$ according to Eq.(5.2);
    Calculate the reconstructed graph structure $A'$ according to Eq.(6.3) and the
    reconstruction loss $\mathcal{L}_r$ according to Eq.(6.4);
    Construct the query matrix for the contrastive loss based-on the hidden embed-
    ding $Qry = (Z_{i_1}, Z_{i_2}, \ldots, Z_{i_o})$; the dictionary matrix $Dic = (Z_{j_1}, Z_{j_2}, \ldots, Z_{j_o})$;
    calculate the contrastive loss $\mathcal{L}_c$ according to Eq.(6.6)
    update the model by minimizing Eq.(4.22)
  end for
  Get the clustering results with final $Z$ by the K-means algorithm
```

Chapter 7

Conclusion

This thesis aimed to study the topic of unsupervised graph learning with deep neural networks. We especially focused on autoencoder-based algorithms for attributed graphs. While previous works were mostly shallow methods or one side information-based approaches, this thesis tried to explore this research area from the following perspectives: (1) How to integrate both graph structure and node content information for graph learning; (2) How to learn deep informative representation for graph data; (3) How to design goal-directed frameworks to avoid the inconsistency between the learned embedding architecture and the downstream tasks; (4) How to deal with different unconventional conditions like corrupted structure of the graph data; and (5) How to make use of available side information.

Specially, we proposed four frameworks to deal with these aforementioned challenges. We first proposed MGAE, which integrates the two aspects of information with a specially designed single-layer autoencoder. To learn better deep representation, we further added random noise to the graph and proposed a marginalized process upon it. Such structure could be stacked multiple layers to learn representative embedding for graph clustering. We then proposed DNEGC, which combines graph autoencoder with a self-training module, and can learn graph embedding and perform clustering simultaneously in a unified framework, thereby achieve clustering-oriented graph embedding for better clustering performance. To deal with corrupted graph structure information, we further proposed Cross-Graph, which can learn robust graph embedding against structure corruption. Two graph autoencoders were maintained, which learn from each other the reliability of each edge and filter out the redundancy edges. Lastly, to make use of side information, we also proposed a constrained graph clustering method, which use a contrastive regularized graph autoencoder to model the pair-wise constraint information and improve the clustering performance. Experiments on real-world datasets have shown the effectiveness of these frameworks.

For future works, we would like to extend our research to more complex graph data conditions. For example, we would like to take cannot-link constraints into consideration when dealing with side information; for corrupted data, we would like to explore more realistic data from real-world scenarios with noise or fragment; we would also like to try to learn from more challenging graphs like multi-view graphs, spatial-temporal graphs, heterogeneous graphs or knowledge graphs.

Bibliography

Al Hasan, M. & Zaki, M. J., 2011, ‘A survey of link prediction in social networks’, *Social network data analytics*, Springer, pp. 243–275.

Arpit, D., Jastrzebski, S., Ballas, N., Krueger, D., Bengio, E., Kanwal, M. S., Maharaj, T., Fischer, A., Courville, A., Bengio, Y. et al., 2017, ‘A closer look at memorization in deep networks’, *International Conference on Machine Learning*, PMLR, pp. 233–242.

Atwood, J. & Towsley, D., 2016, ‘Diffusion-convolutional neural networks’, *NIPS*, pp. 1993–2001.

Blum, A. & Mitchell, T., 1998, ‘Combining labeled and unlabeled data with co-training’, *COLT*, ACM, pp. 92–100.

Bojchevski, A. & Günnemann, S., 2018, ‘Bayesian robust attributed graph clustering: Joint learning of partial anomalies and group structure’, *Proceedings of AAAI*, .

Bruna, J., Zaremba, W., Szlam, A. & LeCun, Y., 2013, ‘Spectral networks and locally connected networks on graphs’, *arXiv preprint arXiv:1312.6203*.

Cai, D., He, X., Wu, X. & Han, J., 2008, ‘Non-negative matrix factorization on manifold’, *Proc. of ICDM*, IEEE, pp. 63–72.

Cai, H., Zheng, V. W. & Chang, K. C.-C., 2018, ‘A comprehensive survey of graph embedding: Problems, techniques, and applications’, *TKDE*, vol. 30, no. 9, pp. 1616–1637.

Cai, Q., Gong, M., Ma, L., Ruan, S., Yuan, F. & Jiao, L., 2015, ‘Greedy discrete particle swarm optimization for large-scale social network clustering’, *Information Sciences*, vol. 316, pp. 503–516.

Cao, S., Lu, W. & Xu, Q., 2016, ‘Deep neural networks for learning graph representations’, *Proc. of AAAI*, AAAI Press, pp. 1145–1152.

Chang, J. & Blei, D. M., 2009, ‘Relational topic models for document networks.’, *AIStats*, , vol. 9pp. 81–88.

Chen, M., Xu, Z., Sha, F. & Weinberger, K. Q., 2012, ‘Marginalized denoising autoencoders for domain adaptation’, *ICML*, pp. 767–774.

Chen, P.-Y. & Wu, L., 2017, ‘Revisiting spectral graph clustering with generative community models’, *ICDM*, IEEE, pp. 51–60.

Chiang, W.-L., Liu, X., Si, S., Li, Y., Bengio, S. & Hsieh, C.-J., 2019, ‘Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks’, *KDD*, pp. 257–266.

Cohn, D. & Hofmann, T., 2001, ‘The missing link-a probabilistic model of document content and hypertext connectivity’, *Advances in neural information processing systems*, pp. 430–436.

Dahl, G. E., Yu, D., Deng, L. & Acero, A., 2012, ‘Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition’, *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42.

Defferrard, M., Bresson, X. & Vandergheynst, P., 2016, ‘Convolutional neural networks on graphs with fast localized spectral filtering’, *NIPS*, pp. 3844–3852.

Diamond, S., Sitzmann, V., Heide, F. & Wetzstein, G., 2017, ‘Unrolled optimization with deep priors’, *arXiv preprint arXiv:1705.08041*.

Dizaji, K. G., Herandi, A., Deng, C., Cai, W. & Huang, H., 2017, ‘Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization’, *ICCV*, IEEE, pp. 5747–5756.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A. & Adams, R. P., 2015, ‘Convolutional networks on graphs for learning molecular fingerprints’, *NIPS*, pp. 2224–2232.

Eaton, E. & Mansbach, R., 2012, ‘A spin-glass model for semi-supervised community detection’, *Twenty-Sixth AAAI Conference on Artificial Intelligence*, .

Fortunato, S., 2010, ‘Community detection in graphs’, *Physics reports*, vol. 486, no. 3-5, pp. 75–174.

Gao, L., Yang, H., Zhou, C., Wu, J., Pan, S. & Hu, Y., 2018, ‘Active discriminative network representation learning’, *Proc. of IJCAI*, pp. 2142–2148.

Girvan, M. & Newman, M. E., 2002, ‘Community structure in social and biological networks’, *Proc. of NAS*, vol. 99, no. 12, pp. 7821–7826.

Grover, A. & Leskovec, J., 2016, ‘node2vec: Scalable feature learning for networks’, *KDD*, pp. 855–864.

Gu, Q. & Zhou, J., 2009, ‘Co-clustering on manifolds’, *Proc. of SIGKDD*, ACM, pp. 359–368.

Guo, T., Pan, S., Zhu, X. & Zhang, C., 2018, ‘Cfond: consensus factorization for co-clustering networked data’, *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 706–719.

Guo, X., Gao, L., Liu, X. & Yin, J., 2017a, ‘Improved deep embedded clustering with local structure preservation’, *IJCAI*, pp. 1753–1759.

Guo, X., Liu, X., Zhu, E. & Yin, J., 2017b, 'Deep clustering with convolutional autoencoders', *International Conference on Neural Information Processing*, Springer, pp. 373–382.

Hadsell, R., Chopra, S. & LeCun, Y., 2006, 'Dimensionality reduction by learning an invariant mapping', *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, , vol. 2IEEE, pp. 1735–1742.

Hammond, D. K., Vandergheynst, P. & Gribonval, R., 2009, 'Wavelets on graphs via spectral graph theory', *arXiv preprint arXiv:0912.3848*.

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

Han, B., Yao, Q., Yu, X., Niu, G., Xu, M., Hu, W., Tsang, I. & Sugiyama, M., 2018, 'Co-teaching: Robust training of deep neural networks with extremely noisy labels', *NeurIPS*, pp. 8527–8537.

Hastings, M. B., 2006, 'Community detection as an inference problem', *Physical Review E*, vol. 74, no. 3, p. 035102.

He, D., Feng, Z., Jin, D., Wang, X. & Zhang, W., 2017, 'Joint identification of network communities and semantics via integrative modeling of network topologies and node contents', *Thirty-First AAAI Conference on Artificial Intelligence*, .

He, K., Fan, H., Wu, Y., Xie, S. & Girshick, R., 2020, 'Momentum contrast for unsupervised visual representation learning', *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738.

Henaff, M., Bruna, J. & LeCun, Y., 2015, 'Deep convolutional networks on graph-structured data', *arXiv preprint arXiv:1506.05163*.

Hu, P., Chan, K. C. & He, T., 2017, ‘Deep graph clustering in social network’, *Proc. of WWW*, pp. 1425–1426.

Hu, R., Pan, S., Long, G., Zhu, X., Jiang, J. & Zhang, C., 2016, ‘Co-clustering enterprise social networks’, *IJCNN*, pp. 107–114.

Ji, S., Pan, S., Cambria, E., Marttinen, P. & Yu, P. S., 2020, ‘A survey on knowledge graphs: Representation, acquisition and applications’, *arXiv preprint arXiv:2002.00388*.

Jiang, L., Zhou, Z., Leung, T., Li, L.-J. & Fei-Fei, L., 2018, ‘Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels’, *ICML*, .

Kim, S.-Y., Jung, T.-S., Suh, E.-H. & Hwang, H.-S., 2006, ‘Customer segmentation and strategy development based on customer lifetime value: A case study’, *Expert systems with applications*, vol. 31, no. 1, pp. 101–107.

Kipf, T. N. & Welling, M., 2016a, ‘Semi-supervised classification with graph convolutional networks’, *arXiv preprint arXiv:1609.02907*.

Kipf, T. N. & Welling, M., 2016b, ‘Variational graph auto-encoders’, *arXiv preprint arXiv:1611.07308*.

Kipf, T. N. & Welling, M., 2017, ‘Semi-supervised classification with graph convolutional networks’, *ICLR*, .

Lawrence, S., Giles, C. L., Tsoi, A. C. & Back, A. D., 1997, ‘Face recognition: A convolutional neural-network approach’, *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113.

Leskovec, J. & Mcauley, J. J., 2012, ‘Learning to discover social circles in ego networks’, *Proc. of NIPS*, pp. 539–547.

Li, J., Dani, H., Hu, X. & Liu, H., 2017, 'Radar: Residual analysis for anomaly detection in attributed networks.', *IJCAI*, pp. 2152–2158.

Li, Y., Sha, C., Huang, X. & Zhang, Y., 2018a, 'Community detection in attributed graphs: An embedding approach', *Proceedings of AAAI*, .

Li, Y., Sha, C., Huang, X. & Zhang, Y., 2018b, 'Community detection in attributed graphs: An embedding approach', *Thirty-Second AAAI Conference on Artificial Intelligence*, .

Liang, D., Cheng, J., Ke, Z. & Ying, L., 2019, 'Deep mri reconstruction: Unrolled optimization algorithms meet neural networks', *arXiv preprint arXiv:1907.11711*.

Liu, L., Xu, L., Wangy, Z. & Chen, E., 2015, 'Community detection based on structure and content: A content propagation perspective', *Proc. of ICDM*, IEEE, pp. 271–280.

Liu, N., Huang, X. & Hu, X., 2017, 'Accelerated local anomaly detection via resolving attributed networks.', *IJCAI*, pp. 2337–2343.

Ma, X., Gao, L., Yong, X. & Fu, L., 2010, 'Semi-supervised clustering algorithm for community structure detection in complex networks', *Physica A: Statistical Mechanics and its Applications*, vol. 389, no. 1, pp. 187–197.

Maaten, L. v. d. & Hinton, G., 2008, 'Visualizing data using t-sne', *JMLR*, pp. 2579–2605.

Malach, E. & Shalev-Shwartz, S., 2017, 'Decoupling" when to update" from" how to update"', *NeurIPS*, pp. 960–970.

Newman, M. E., 2006a, 'Finding community structure in networks using the eigenvectors of matrices', *Physical review E*, vol. 74, no. 3, p. 036104.

Newman, M. E., 2006b, ‘Modularity and community structure in networks’, *Proc. of NAS*, vol. 103, no. 23, pp. 8577–8582.

Oord, A. v. d., Li, Y. & Vinyals, O., 2018, ‘Representation learning with contrastive predictive coding’, *arXiv preprint arXiv:1807.03748*.

Ou, M., Cui, P., Pei, J., Zhang, Z. & Zhu, W., 2016, ‘Asymmetric transitivity preserving graph embedding’, *KDD*, pp. 1105–1114.

Pan, S., Hu, R., Fung, S.-f., Long, G., Jiang, J. & Zhang, C., 2019, ‘Learning graph embedding with adversarial training methods’, *IEEE Transactions on Cybernetics*, vol. 50, no. 6, pp. 2475–2487.

Pan, S., Hu, R., Long, G., Jiang, J., Yao, L. & Zhang, C., 2018, ‘Adversarially regularized graph autoencoder for graph embedding’, *Proc. of IJCAI*, pp. 2609–2615.

Pan, S., Wu, J., Zhu, X., Zhang, C. & Wang, Y., 2016, ‘Tri-party deep network representation’, *Proc. of IJCAI*, pp. 1895–1901.

Pathak, D., Krahenbuhl, P. & Darrell, T., 2015, ‘Constrained convolutional neural networks for weakly supervised segmentation’, *Proceedings of the IEEE international conference on computer vision*, pp. 1796–1804.

Perozzi, B., Al-Rfou, R. & Skiena, S., 2014, ‘Deepwalk: Online learning of social representations’, *Proc. of SIGKDD*, ACM, pp. 701–710.

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

Reihanian, A., Feizi-Derakhshi, M.-R. & Aghdasi, H. S., 2018, ‘Overlapping community detection in rating-based social networks through analyzing topics, ratings and links’, *Pattern Recognition*, vol. 81, pp. 370–387.

Ren, Y., Hu, K., Dai, X., Pan, L., Hoi, S. C. & Xu, Z., 2019, ‘Semi-supervised deep embedded clustering’, *Neurocomputing*, vol. 325, pp. 121–130.

Schmidt, U. & Roth, S., 2014, ‘Shrinkage fields for effective image restoration’, *Proc. of the CVPR*, pp. 2774–2781.

Shao, M., Li, S., Ding, Z. & Fu, Y., 2015, ‘Deep linear coding for fast graph clustering.’, *Proc. of IJCAI*, pp. 3798–3804.

Shen, X., Pan, S., Liu, W., Ong, Y. & Sun, Q., 2018, ‘Discrete network embedding’, *Proc. of IJCAI*, pp. 3549–3555.

Sun, Y., Han, J., Gao, J. & Yu, Y., 2009, ‘itopicmodel: Information network-integrated topic modeling’, *Proc. of ICDM*, IEEE, pp. 493–502.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J. & Mei, Q., 2015, ‘Line: Large-scale information network embedding’, *Proc. of WWW*, ACM, pp. 1067–1077.

Tian, F., Gao, B., Cui, Q., Chen, E. & Liu, T.-Y., 2014, ‘Learning deep representations for graph clustering.’, *AAAI*, pp. 1293–1299.

Van Der Maaten, L., 2014, ‘Accelerating t-sne using tree-based algorithms’, *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 3221–3245.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P. & Bengio, Y., 2017, ‘Graph attention networks’, *arXiv preprint arXiv:1710.10903*.

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A., 2008, ‘Extracting and composing robust features with denoising autoencoders’, *Proc. of ICML*, pp. 1096–1103.

Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S. et al., 2001, 'Constrained k-means clustering with background knowledge', *Icml*, , vol. 1pp. 577–584.

Wang, C., Pan, S., Long, G., Zhu, X. & Jiang, J., 2017a, 'Mgae: Marginalized graph autoencoder for graph clustering', *Proc. of CIKM*, ACM, pp. 889–898.

Wang, H., Zhou, C., Chen, X., Wu, J., Pan, S. & Wang, J., 2020a, 'Graph stochastic neural networks for semi-supervised learning', *Advances in Neural Information Processing Systems*, vol. 33.

Wang, L., Yu, Z., Han, Q., Yang, D., Pan, S., Yao, Y. & Zhang, D., 2020b, 'Compact scheduling for task graph oriented mobile crowdsourcing', *IEEE Transactions on Mobile Computing.*

Wang, L., Yu, Z., Xiong, F., Yang, D., Pan, S. & Yan, Z., 2019, 'Influence spread in geo-social networks: a multiobjective optimization perspective', *IEEE Transactions on Cybernetics.*

Wang, X., Cui, P., Wang, J., Pei, J., Zhu, W. & Yang, S., 2017b, 'Community preserving network embedding', *Proceedings of the AAAI Conference on Artificial Intelligence*, , vol. 31.

Wang, Y., Zhang, W., Wu, L., Lin, X., Fang, M. & Pan, S., 2016, 'Iterative views agreement: an iterative low-rank based structured optimization method to multi-view spectral clustering', *arXiv preprint arXiv:1608.05560.*

Wang, Z., Chen, C. & Li, W., 2017c, 'Predictive network representation learning for link prediction', *SIGIR*, pp. 969–972.

Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K. & Zhu, L., 2019a, 'Adversarial examples for graph data: Deep insights into attack and defense', *IJCAI*, pp. 4816–4823.

Wu, M., Pan, S., Zhou, C., Chang, X. & Zhu, X., 2020a, 'Unsupervised domain adaptive graph convolutional networks', *Proceedings of The Web Conference 2020*, pp. 1457–1467.

Wu, M., Pan, S. & Zhu, X., 2020b, 'Openwgl: Open-world graph learning', *2020 IEEE International Conference on Data Mining (ICDM)*, IEEE, pp. 681–690.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Yu, P. S., 2019b, 'A comprehensive survey on graph neural networks', *arXiv preprint arXiv:1901.00596*.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X. & Zhang, C., 2020c, 'Connecting the dots: Multivariate time series forecasting with graph neural networks', *arXiv preprint arXiv:2005.11650*.

Wu, Z., Xiong, Y., Yu, S. X. & Lin, D., 2018, 'Unsupervised feature learning via non-parametric instance discrimination', *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3733–3742.

Xia, R., Pan, Y., Du, L. & Yin, J., 2014, 'Robust multi-view spectral clustering via low-rank and sparse decomposition.', *Proc. of AAAI*, pp. 2149–2155.

Xie, J., Girshick, R. & Farhadi, A., 2016, 'Unsupervised deep embedding for clustering analysis', *ICML*, pp. 478–487.

Xie, Y., Gong, M., Wang, S. & Yu, B., 2018, 'Community discovery in networks with deep sparse filtering', *Pattern Recognition*, vol. 81, pp. 50–59.

Yang, C., Liu, Z., Zhao, D., Sun, M. & Chang, E. Y., 2015, 'Network representation learning with rich text information.', *Proc. of IJCAI*, pp. 2111–2117.

Yang, J. & Leskovec, J., 2012, 'Community-affiliation graph model for overlapping network community detection', *Proc. of ICDM*, pp. 1170–1175.

Yang, J. & Leskovec, J., 2013, ‘Overlapping community detection at scale: a nonnegative matrix factorization approach’, *Proceedings of the sixth ACM international conference on Web search and data mining*, ACM, pp. 587–596.

Yang, J., McAuley, J. & Leskovec, J., 2013, ‘Community detection in networks with node attributes’, *Proc. of ICDM*, IEEE, pp. 1151–1156.

Yang, L., Cao, X., Jin, D., Wang, X. & Meng, D., 2014, ‘A unified semi-supervised community detection framework using latent space graph regularization’, *IEEE transactions on cybernetics*, vol. 45, no. 11, pp. 2585–2598.

Zhang, C., Bengio, S., Hardt, M., Recht, B. & Vinyals, O., 2016, ‘Understanding deep learning requires rethinking generalization’, *arXiv:1611.03530.*

Zhang, D., Yin, J., Zhu, X. & Zhang, C., 2018, ‘Network representation learning: A survey’, *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–28.

Zhang, H., Kiranyaz, S. & Gabbouj, M., 2017, ‘Outlier edge detection using random graph generation models and applications’, *JBD*, vol. 4, no. 1, p. 11.

Zhang, X., Liu, H., Li, Q. & Wu, X.-M., 2019, ‘Attributed graph clustering via adaptive graph convolution’, *Proc. of IJCAI*, pp. 4327–4333.

Zhang, Z.-Y., 2013, ‘Community structure detection in complex networks with partial background information’, *EPL (europhysics letters)*, vol. 101, no. 4, p. 48005.

Zhang, Z.-Y., Sun, K.-D. & Wang, S.-Q., 2013, ‘Enhanced community structure detection in complex networks with partial background information’, *Scientific reports*, vol. 3, p. 3241.

Zhou, C. & Paffenroth, R. C., 2017, ‘Anomaly detection with robust deep autoencoders’, *Proc. of KDD*, ACM, pp. 665–674.

Zhou, Y., Cheng, H. & Yu, J. X., 2009, ‘Graph clustering based on structural/attribute similarities’, *Proc. of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729.

Zhu, D., Zhang, Z., Cui, P. & Zhu, W., 2019a, ‘Robust graph convolutional networks against adversarial attacks’, *KDD*, pp. 1399–1407.

Zhu, S., Pan, S., Zhou, C., Wu, J., Cao, Y. & Wang, B., 2020, ‘Graph geometry interaction learning’, *arXiv preprint arXiv:2010.12135*.

Zhu, S., Zhou, C., Pan, S., Zhu, X. & Wang, B., 2019b, ‘Relation structure-aware heterogeneous graph neural network’, *2019 IEEE International Conference on Data Mining (ICDM)*, IEEE, pp. 1534–1539.

Zhu, X., Loy, C. C. & Gong, S., 2015, ‘Constrained clustering with imperfect oracles’, *IEEE transactions on neural networks and learning systems*, vol. 27, no. 6, pp. 1345–1357.

Zügner, D., Akbarnejad, A. & Günnemann, S., 2018, ‘Adversarial attacks on neural networks for graph data’, *KDD*, pp. 2847–2856.