UNIVERSITY OF CALIFORNIA

Los Angeles

Deep Representation Learning on Complex Graphs

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

by

Cheng Zheng

2020

ABSTRACT OF THE DISSERTATION

Deep Representation Learning on Complex Graphs

by

Cheng Zheng
Doctor of Philosophy in Electrical and Computer Engineering
University of California, Los Angeles, 2020
Professor Wei Wang, Co-Chair
Professor Jonathan Chau-Yan Kao, Co-Chair

Graph representation learning serves as the core of many important tasks on graphs, ranging from friendship recommendation, name disambiguation, drug discovery, and fraud detection. Recently, deep learning has revolutionized various domains such as computer vision, natural language processing, speech recognition, etc. Inspired by the success of deep neural networks, there has been an increasing interest to learn graph representations with deep learning models such as autoencoders, convolutional neural networks, etc. However, graphs in real-life applications usually have complex structures such as sparse connections, task-irrelevant information, and rapidly evolving structures. The complexity poses great challenges to the existing frameworks, such as network embedding models with random walks and graph neural networks based on the neighborhood aggregation.

In this dissertation, we propose several deep learning frameworks to tackle the aforementioned problems of graph representation learning on complex graphs. We propose a novel model to learn network representations with adversarially regularized autoencoders to overcome the sparse sampling issue of random walks on graphs. To resolve the task-irrelevant

noise, we propose a general framework that is trained to simultaneously select task-relevant edges and learn graph representations by the feedback signals from downstream tasks. To learn from the dynamic evolving graphs, we propose to extract local features by performing convolutions in nodes' neighborhoods defined in joint temporal-structural space. The methodologies presented in these frameworks span different research areas, including deep network embedding, graph representation learning, temporal graph modeling, and node classification on graphs. As a result, these methodologies not only tackle specific challenges in the graph learning tasks mentioned above but also shed light on other applications like social network analysis.

The dissertation of Cheng Zheng is approved.

Yizhou Sun

Vwani P. Roychowdhury

Jonathan Chau-Yan Kao, Committee Co-Chair

Wei Wang, Committee Co-Chair

University of California, Los Angeles

2020

To my wife Yingze, for her
inspiration, support and love.

TABLE OF CONTENTS

1 Introduction **1**

1.1 Scope of the Research 1

1.2 Contribution 4

1.3 Overview 7

2 Related Work **8**

2.1 Network Embedding 8

2.2 Graph Neural Networks 9

2.3 Graph Sparsification 10

2.4 Temporal Graph Modeling 11

2.5 Adversarial Regularization 12

3 Network Embedding with Adversarial Regularization **13**

3.1 Background 13

3.2 Preliminaries 16

3.2.1 Autoencoder Neural Networks 16

3.2.2 Generative Adversarial Networks 17

3.2.3 Network Embedding 18

3.3 The NetRA Model 18

3.3.1 Random Walk Generator 19

3.3.2 Embedding with Adversarially regularized Autoencoders 20

3.4 Evaluation 26

3.4.1 Datasets . . . 26
3.4.2 Comparing Algorithms . . . 27
3.4.3 Visualization . . . 29
3.4.4 Link Prediction . . . 30
3.4.5 Network Reconstruction . . . 32
3.4.6 Multi-label Classification . . . 34
3.4.7 Parameter Sensitivity . . . 35
3.5 Summary . . . 37

4 Robust Graph Representation Learning via Neural Sparsification . . . 38
4.1 Background . . . 38
4.2 Proposed Method: NeuralSparse . . . 40
4.3 Sparsification Network . . . 43
4.4 Evaluation . . . 47
4.4.1 Datasets . . . 48
4.4.2 Experimental Setup . . . 49
4.4.3 Experimental Results . . . 51
4.5 Summary . . . 57

5 Temporal Graph Modeling with Temporal Structural Convolution . . . 58
5.1 Background . . . 58
5.2 TSNet Overview . . . 60
5.2.1 A Two-step Framework . . . 61
5.2.2 Architecture . . . 62

5.3 Sparsification network 62
5.3.1 Design Goals 62
5.3.2 Edge Representations 63
5.3.3 Sampling Sparsified Subgraphs 63
5.3.4 Making Samples Differentiable 64
5.4 Temporal-Structural Convolutional Network 66
5.4.1 Temporal-structural Neighborhood 67
5.4.2 Temporal-structural Convolutional Layer 67
5.4.3 Network Architecture 68
5.5 Evaluation 69
5.5.1 Datasets 69
5.5.2 Baseline Methods 72
5.5.3 Experimental Settings 73
5.5.4 Experimental Results 75
5.6 Summary 81

6 Applications in Social Network Analysis 83

6.1 Social Media User Geolocation via Hybrid Attention 83
6.1.1 Introduction 83
6.1.2 Hybrid-attentive User Geolocation 85
6.1.3 Experiments 89
6.1.4 Summary 92
6.2 On-demand Influencer Discovery on Social Media 94
6.2.1 Introduction 94

6.2.2 on-Demand Influencer Discovery 96
6.2.3 Experiments 100
6.2.4 Summary 104

7 Conclusion 105

7.1 Conclusion 105

LIST OF FIGURES

3.1 Illustration of the adversarially regularized autoencoders 15

3.2 Sparsity of network sampling. 20

3.3 Visualization results of the compared methods on JDK dependency network. . . 29

3.4 Link prediction using vertex representation on UCI and JDK. 30

3.5 Link prediction using vertex representation on BLOG and DBLP. 31

3.6 Network reconstruction results on UCI and BLOG 33

3.7 Multi-label classification on PPI and Wikipedia 34

3.8 Parameter sensitivity analysis 35

3.9 Performance on different λ_1 for $\mathcal{L}_{\text{LE}}$ 36

3.10 Performance on different NetRA architectures 36

4.1 Example of task-irrelevant edges in graphs. 41

4.2 The overview of NeuralSparse 42

4.3 Performance vs hyper-parameters 52

4.4 Graph visualizations with orginal and sparsified graphs. 53

4.5 Convergence analysis 55

4.6 Node classification performance when adding noise to graph structure. 56

4.7 Impact from hyper-parameter k on validation and testing on the Transaction dataset 57

5.1 An example of node classification in a temporal graph from the financial domain. 59

5.2 The frameworks of TSNet. 61

5.3 An illustration of the proposed sparsification network. 64

5.4 Accuracy vs hyper-parameter k 77
5.5 Accuracy vs hyper-parameter l on DBLP-3 and DBLP-5 78
5.6 One-hop neighborhood of *Thomas S. Huang* in DBLP-5. 79
5.7 Classification accuracy vs ratio of nodes in training set. 82

6.1 The overview of the proposed framework, HUG. 85
6.2 Attention weight analysis. 93
6.3 Hashtag distribution of 1% US English Tweets in 11/01/2019-12/31/2019. . . . 95
6.4 The overview of the proposed framework, DID 96
6.5 Seed user pool size and specificity impact. 103

LIST OF TABLES

3.1 Statistics of the real-world network datasets 27

3.2 AUC score of link prediction 32

4.1 Dataset statistics 48

4.2 Node classification performance 50

4.3 Node classification performance with κ-NN graphs 52

4.4 Percentage of edges connecting nodes of the same labels 54

4.5 Node classification performance with complete graphs 56

5.1 Dataset statistics 70

5.2 Node classification performance 73

6.1 Dataset statistics. 90

6.2 Twitter user geolocation prediction performance. 91

6.3 Ablation study on Twitter-US. 91

6.4 Dataset statistics. 100

6.5 Topic-specific influencer detection performance. 101

6.6 Ablation study on HIV. 101

6.7 Influencers detected in US-English dataset. 104

ACKNOWLEDGMENTS

It has been a wonderful experience to have my graduate studies in UCLA ScAi lab. I would love to especially thank my advisor, Prof. Wei Wang, for her continuous guidance, support, and encouragement in my Ph.D. research. She has been leading me to this fascinating field of deep graph learning, guiding me to overcome the various difficulties in the research and inspiring me to come up with better ideas. I have learned a lot from Prof. Wang about the philosophy of managing the team and collabrating with different people, which would benefit deeply for my future career.

I also own my sincere gratitude to my departmental advisor Prof. Jonathan Kao for his continuous help during my Ph.D. program. His deep learning course in the ECE department has inspired me to continue exploring the challenging problems in graph mining fields with deep learning techniques. I also would like to acknowledge Prof. Vwani Roychowdhury and Prof. Yizhou Sun for serving as my doctoral committee members and giving me insightful comments and advice that greatly help me improve my work.

I am deeply grateful to my outstanding collaborators for their indispensable support and contribution to my research work. In particular, I would like to thank Dr. Bo Zong, Dr. Wei Cheng, and Dr. Wenchao Yu at NEC Labs for their substantial support. I acknowledge Dr. Qin Zhang at the University of Technology Sydney for collaboration in social network analysis projects. The discussions with Dr. Dongjin Song, Dr. Jingchao Ni, and Dr. Haifeng Chen have also provided me a different scientific mindset.

I have the great pleasure to work with many talented students in ScAi Lab at UCLA. I would like to thank Ruirui Li, Chelsea Ju, Yichao Zhou, Jyun-yu Jiang, Zeyu Li, Guangyu Zhou, Junheng Hao, Xiusi Chen, and Yu Yan for their help and support through this journey. I am grateful to work with them together which left me with precious memories.

Last but not least, to the most important people in my life, my wife (Yingze Qu) and my parents (Qingfu Zheng and Jingwen Wei). There are no other words in the world to

express my deepest gratitude to my family who supports, encourages, and helps me to break through anything even seemingly impossible.

In this dissertation, Chapter 3 is based on our paper titled "Learning Deep Network Representations with Adversarially Regularized Autoencoders". Chapter 4 is based on our paper titled "Robust Graph Representation Learning via Neural Sparsification". Chapter 5 is based on our paper titled "Node Classification in Temporal Graphs through Stochastic Sparsification and Temporal Structural Convolution". Chapter 6 is based on our papers titled "Social Media User Geolocation via Hybrid Attention" and "On-demand Influencer Discovery on Social Media".

VITA

2011-2015	B.S. in Physics, Tsinghua University, Beijing, China
2017-2018	M.S. in Computer Science, UCLA, Los Angeles, California
2018	Summer Research Assistance, NEC Laboratories America, Princeton, New Jersey
2019	Software Engineer Intern, Facebook, Menlo Park, California
2020	Software Developer Intern, Google, Remotely in Vancouver, British Columbia, Canada
2015-2017	Graduate Student Researcher, Electrical and Computer Engineering Department, UCLA, Los Angeles, California
2017-2020	Graduate Student Researcher, Computer Science Department, UCLA, Los Angeles, California

PUBLICATIONS

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, Wei Wang. Robust Graph Representation Learning via Neural Sparsification. In *International Conference on Machine Learning* (ICML 2020).

Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, Wei Wang. Node Classification in Temporal Graphs through Stochastic Sparsification

and Temporal Structural Convolution. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (ECML-PKDD 2020).

Cheng Zheng, Qin Zhang, Sean D. Young, Wei Wang. On-demand Influencer Discovery on Social Media. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management* (CIKM 2020).

Cheng Zheng, Jyun-Yu Jiang, Yichao Zhou, Sean D. Young, Wei Wang. Social Media User Geolocation via Hybrid Attention. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR 2020).

Cheng Zheng, Qin Zhang, Guodong Long, Chengqi Zhang, Sean D. Young, Wei Wang. Measuring Time-Sensitive and Topic-Specific Influence in Social Networks with LSTM and Self-Attention. In *IEEE Access 2020*.

Wenchao Yu, **Cheng Zheng**, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, Wei Wang. Automatic Speaker Recognition with Limited Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (KDD 2018).

Yabin Fan, Qiming Shao, Lei Pan, Xiaoyu Che, Qinglin He, Gen Yin, **Cheng Zheng**, et al. Unidirectional magneto-resistance in modulation-doped magnetic topological insulators. In *Nano letters 2019*.

Qiming Shao, Guoqiang Yu, Yann-Wen Lan, Yumeng Shi, Ming-Yang Li, **Cheng Zheng**, et al. Strong Rashba-Edelstein Effect-Induced Spin–Orbit Torques in Monolayer Transition Metal Dichalcogenide/Ferromagnet Bilayers. In *Nano Letters 2016*.

CHAPTER 1

Introduction

1.1 Scope of the Research

As a kind of data structure that models a set of objects (nodes) and their relationships (edges) [46], graphs can be used as the denotation of a large number of systems across various areas including social science [165], physical systems [62], biology [144], knowledge graphs [28] and many others. The central problem in machine learning on graphs is finding a way to incorporate information of graph structure into the machine learning frameworks. Utilizing machine learning on graphs is important and ubiquitous for applications [115, 109, 104, 157] ranging from friendship recommendation in social networks, name disambiguation in citation networks, drug discovery, and so on. The primary challenge in this domain is to find the most effective and efficient way to learn the representations that can encode the structural information about the graph. Traditional approaches [49, 120, 9, 98] rely heavily on summary graph statistics (e.g. node degrees or clustering coefficients), kernel functions, or carefully engineered features to compare local neighborhood structures. However, these approaches are limited since the hand-engineered features cannot generalize to most of the real-life graphs that are evolving fast and have complex structures.

In recent years, deep learning has revolutionized many machine learning domains [38], ranging from computer vision, natural language processing, speech recognition, to name a few. The data in these tasks are typically represented in the Euclidean space. More recently, there has been an increasing interest to learn graph representations with deep learning models such as autoencoder [122], convolutional neural network (CNN) [64], recurrent neural networks (RNN) [73], graph neural networks (GNNs) [165, 46, 132]. However, there are a large number of graphs from real-life applications that are constructed with complex prop-

erties, such as sparse graphs, over-densified graphs, dynamically evolving graphs, etc. The complexity of graph data has imposed significant challenges on existing machine learning algorithms.

- The problem of network representation learning, also known as network embedding [40], arises in many machine learning tasks assuming that there exist a small number of variabilities in the vertex representations which can capture the semantics of the original network structure. Most existing network embedding models [86, 104, 41], with shallow or deep architectures, learn vertex representations from the sampled vertex sequences such that the low-dimensional embeddings preserve the locality property and/or global reconstruction capability. The resultant representations, however, are difficult for model generalization due to the intrinsic sparsity of sampled sequences from the input network. As such, an ideal approach to address the problem is to generate vertex representations by learning a probability density function over the sampled sequences. However, in many cases, such a distribution in a low-dimensional manifold may not always have an analytic form.

- Although representation learning has been in the center of machine learning tasks on graphs, the underlying motivation why two nodes get connected may have no relation to a target downstream task, and such task-irrelevant edges could hurt neighborhood aggregation as well as the performance of GNNs. When task-irrelevant edges are mixed into neighborhood aggregation, the trained GNN fails to learn better representations, and it becomes difficult to learn a subsequent classifier with strong generalization power. Conventional methods, such as graph sparsification [75, 67, 99], are unsupervised such that the resulting sparsified graphs may not favor downstream tasks. Several other works focus on downsampling under predefined distributions [152, 45, 22]. As the predefined distributions may not well adapt to subsequent tasks, such methods could suffer suboptimal prediction performance. Multiple recent efforts strive to make use of supervision signals to remove noise edges [125]. However, the proposed methods

are either transductive with difficulty to scale [34] or of high gradient variance bringing increased training difficulty [96].

- Temporal graphs, as a data structure that carries both temporal and structural information from real-world data, has been widely adopted in applications from various domains. Here, we focus on the problem of node classification in temporal graphs [135]: Given a set of nodes with rich features and a temporal graph that records historical activities between nodes, the goal is to predict the label of every node. Consider the following application scenario. While node representation lies at the core of this problem, we face two main challenges from temporal graphs. Temporal graphs from real-life applications are large with high complexity. Local features in the temporal-structural domain are the key to node classification in temporal graphs. Although existing techniques have investigated how to build convolutional operators to automatically learn and extract local features from either temporal domain [7] or structural domain [117], a naïve method that simply stacks temporal and structural operators could lead to suboptimal performance. An effective method that learns and extracts local features from joint temporal-structural space is still missing.

- Determining user geolocation is vital to various real-world applications on the internet, such as online marketing and event detection. To identify the geolocations of users, their behaviors on social media like published posts and social interactions can be strong evidence. However, most of the existing social media based approaches individually learn from text contexts and social networks. This separation can not only lead to sub-optimal performance but also ignore the distinct importance of two resources for different users. In another problem, identifying influencers on social media, such as Twitter, has played a central role in many applications, including online marketing and political campaigns. Compared with social media celebrities, domain-specific influencers are less expensive to hire and more engaged in spreading messages such as new treatment or timely prevention for HIV. However, most of the existing topic

modeling based approaches fail to identify influencers who are dedicated to the rare yet important topics such as HIV and suicide.

1.2 Contribution

In this dissertation, we propose the following deep learning frameworks to tackle the problem of graph representation learning in complex graphs, including learning network embedding with adversarial regularization, robust graph representation via neural sparsification, and temporal graph modeling with temporal-structural convolution. We detail the contributions of these research projects as follows.

The first research issue comes from the unrepresentative sampled vertex sequences by the random walk on graphs, which fail to preserve the locality property and/or global reconstruction capability. To address these challenges we propose a novel model to learn the network representations with adversarially regularized autoencoders (NetRA). NetRA jointly minimizes network locality-preserving loss and the reconstruction error of autoencoder which utilizes a long short-term memory network (LSTM) as an encoder to map the input sequences into a fixed-length representation. The joint embedding inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution. The model employs a discrete LSTM autoencoder to learn continuous vertex representations with sampled sequences of vertices as inputs. In this model, besides minimizing the reconstruction error in the LSTM autoencoder, the locality-preserving loss at the hidden layer is also minimized simultaneously. Meanwhile, the continuous space generator is also trained by constraining to agree in distribution with the encoder. The generative adversarial training can be regarded as a complementary regularizer to the network embedding process. We present experimental results to show the embedding capability of NetRA on a variety of tasks, including network reconstruction, link prediction, and multi-label classification.

The second research work addresses how to utilize supervision signals to remove task-irrelevant edges to achieve robust graph representation learning. We propose Neural Sparsification (NeuralSparse), a general framework that simultaneously learns to select task-relevant edges and graph representations by feedback signals from downstream tasks. The NeuralSparse consists of two major components: sparsification network and GNN. For the sparsification network, we utilize a deep neural network to parameterize the sparsification process: how to select edges from the one-hop neighborhood given a fixed budget. In the training phase, the network learns to optimize a sparsification strategy that favors downstream tasks. In the testing phase, the network sparsifies input graphs following the learned strategy, instead of sampling subgraphs from a predefined distribution. Unlike conventional sparsification techniques, our technique takes both structural and non-structural information as input and optimizes the sparsification strategy by feedback from downstream tasks, instead of using (possibly irrelevant) heuristics. For the GNN component, the NeuralSparse feeds the sparsified graphs to GNNs and learns graph representations for subsequent prediction tasks. Under the NeuralSparse framework, by the standard stochastic gradient descent and backpropagation techniques, we can simultaneously optimize graph sparsification and representations. Experimental results on both public and private datasets demonstrate that NeuralSparse consistently provides improved performance for existing GNNs on node classification tasks, yielding up to 7.2% improvement.

The third research work focuses on the node classification problem in temporal graphs. We propose Temporal Structural Network (TSNet), a deep learning framework that performs supervised node classification in sparsified temporal graphs. TSNet also leverage the supervised sparsification technique and consists of two major sub-networks: sparsification network and temporal-structural convolutional network. The sparsification network aims to sparsify input temporal graphs by sampling edges from the one-hop neighborhood following a distribution that is learned from the subsequent supervised classification tasks. The temporal-structural convolutional network takes sparsified temporal graphs as input and

extracts local features by performing convolution in nodes' neighborhoods defined in joint temporal-structural space. As both sub-networks are differentiable, we can leverage standard stochastic gradient descent and backpropagation techniques to iteratively learn better parameters to sparsify temporal graphs and extract node representations. Experimental results on both public and private datasets show that TSNet can offer competitive performance on node classification tasks. Using a case study, we demonstrate the potential of TSNet to improve model interpretation and visualization of temporal graphs.

We also demonstrate the applications of deep graph representation learning on social network analysis problems. We propose a novel end-to-end framework, Hybrid-attentive User Geolocation (HUG), to jointly model post texts and user interactions in social media. The hybrid attention mechanism is introduced to automatically determine the importance of texts and social networks for each user while social media posts and interactions are modeled by a graph attention network and a language attention network. Extensive experiments conducted on three benchmark geolocation datasets using Twitter data demonstrate that HUG significantly outperforms competitive baseline methods. The in-depth analysis also indicates the robustness and interpretability of HUG. For the topic-specific influencer detection problem, we investigate an on-Demand Influencer Discovery (DID) framework that is able to identify influencers on any subject depicted by a few user-specified keywords, regardless of its popularity on social media. The DID model employs an iterative learning process that integrates the language attention network as a subject filter and the influence convolution network built on user interactions. Comprehensive evaluations on Twitter datasets show that the DID model can reliably identify influencers even on rare subjects such as HIV and suicide, outperforming existing topic-specific influencer detection models.

1.3 Overview

The rest of the dissertation is organized as follows: Chapter 2 summarizes the relevant works for each research problem. Chapter 3 describes our method of utilizing generative regularized autoencoders to learn smoothly regularized vertex representations that well capture the network structure. Chapter 4 presents our work on the supervised graph sparsification technique that improves generalization power by learning to remove potentially task-irrelevant edges from input graphs. Chapter 5 discusses our proposed methods to jointly learns temporal and structural features for node classification from the sparsified temporal graphs. Chapter 6 presents two models that utilize the deep graph representation learning to study social network analysis problems. Chapter 7 concludes this dissertation with a summary of our work.

CHAPTER 2

Related Work

2.1 Network Embedding

Matrices are the most straightforward representation of a network. Earlier work has focused on the factorization of different matrix representations, including but not limited to adjacency matrix [98], Laplacian matrix [107], and transition probability matrix [16]. Matrix factorization approaches usually have the time complexity of $\mathcal{O}(|V|^2)$, which restricts their applications. Thus one challenge facing matrix factorization is its poor scalability as the data volume explodes with time, as demonstrated in [164] on a real-world network of Alibaba Group, which has 290 million vertices and 18 billion edges.

Recently, we have witnessed the emergence of random walk based methods [30, 86, 41], inspired by the success of natural language processing [86]. These models build connections between network structure and natural language. The training input of these algorithms changes from matrices to sentence-like vertex sequences generated by random walks among connected vertices. The skip-gram algorithm [81] maximizes the co-occurrence probability among the vertices within a certain window in a random walk. DeepWalk [86] obtains effective embeddings using truncated random walks. Node2vec [41] extends the model with flexibility between homophily and structural equivalence [153]. These last two methods motivate the study of network embedding taking advantage of language models.

Deep learning embedding models [110, 17, 122] have also been applied to solve the network embedding problem. Autoencoder based approaches [122, 17] were proposed, utilizing its ability to learn highly non-linear properties. By carefully constructing the learning objective, [122] preserves the first and second proximity of networks which delivers state-of-the-art performance. Recent works on graph convolutional networks [27, 64] have demonstrated ef-

fective convolution operation on network data. There are multiple recent works [45, 118] to train unsupervised network embedding models with the neighborhood aggregation technique.

2.2 Graph Neural Networks

Driven by the success of convolutional neural networks in the computer vision and natural language processing domains, graph neural networks (GNNs) are proposed to automate node representation learning in both supervised and unsupervised settings. The basic idea behind the GNNs to perform the convolution by aggregating the neighbor nodes' information in the neighborhood.

Under the supervised setting, early studies [27, 13] investigate convolutional filters in graph spectral domain under transductive settings. Since that time, there have been increasing improvements, extensions, and approximations on spectral-based graph convolutional networks [156, 167]. However, as the spectral methods usually handle the whole graph simultaneously and are difficult to parallel or scale to large graphs, convolutional filters in graph domain are proposed [101, 83, 64, 137]. Veličković et al. [117] proposes the graph attentional network which performs the neighborhood aggregations based on the attention weights among the node neighborhood. There are also a few studies [45, 66] explore how to differentiate neighborhood filtering by sequential models. Multiple recent studies [134, 151, 145] explore other types of aggregation approaches under the graph neural network framework.

Under the unsupervised setting, graph neural networks aims to learn node representations that best preserve local proximity [17, 104, 45, 58]. The learned node representations can serve as an important source of features for prediction tasks, such as node classification [86, 20, 41], link prediction [72, 122], and community detection [93]. Veličković et al. [118] maximizes the mutual information between patch representations and corresponding high-level summaries of graphs which is applicable to both transductive and inductive learning setups. Kipf and Welling [63] introduces a framework for unsupervised learning on

graph-structured data based on the variational auto-encoder (VAE). To some extent, the deep learning approaches for network embedding at the same time belong to graph neural networks, which include graph autoencoder-based algorithms [122] and graph convolution neural networks with unsupervised training [45].

2.3 Graph Sparsification

Real-world graphs are usually large and noisy, rendering problems from the perspectives of overfitting risk, interpretable visualization, and scalability. The goal of graph sparsification is to find small subgraphs from input large graphs that best preserve desired properties. Existing techniques are mainly unsupervised and deal with simple graphs without node/edge attributes for preserving predefined graph metrics [52], information propagation traces [80], graph spectrum [3, 14, 99, 19, 2], node degree distribution [31, 121], node distance distribution [67], or clustering coefficient [78]. Importance based edge sampling has also been studied in a scenario where we could predefine the importance of edges [158, 22]. The detailed discussion on graph sparsification can be found in [75, 154]. SPINE proposed by Mathioudakis et al. [80] sparsifies social graphs by maximizing the likelihood of observing information propagation traces. Spectral sparsification methods discover reduced subgraphs that best approximate the spectrum of original graphs [3, 14, 99, 19, 102, 2]. Graph sampling methods extract small subgraphs that approximates statistic information in graphs, such as node degree distribution [71, 31, 121], node distance distribution [67, 92], and clustering coefficient [78].

Besides the spectral graph sparsification approaches, there are multiple recent works exploring downsampling the graphs in GNNs under predefined distributions [152, 45, 22]. Chen et al. [22] proposes the importance sampling method in graph convolutional networks for efficient training and better generalization. As the predefined distributions may not well adapt to subsequent tasks, such methods could suffer suboptimal prediction performance.

Multiple recent efforts strive to make use of supervision signals to remove noise edges [125]. However, the proposed methods are either transductive with difficulty to scale [34] or of high gradient variance bringing increased training difficulty [96].

2.4 Temporal Graph Modeling

Temporal graphs are commonly defined in two ways: snapshot sequences [69] and timestamped graph [114]. The snapshot sequence is a collection of evolving graph snapshots at multiple discrete time steps and the timestamped graph is a single graph with continuous-valued timestamped links. In this dissertation, we focus on the representation learning over graph snapshots.

The traditional approaches uses matrix factorization to extract features in a sequence of graphs [68, 108, 97, 148]. Yu et al. [148] proposes the structural factorization model which fully characterizes the structure of the network over time or directly express the network structure as a function of time, once the features have been extracted. This fully parameterized model can be used to reconstruct the approximate future structure in the network at any point in time.

Recent studies also attempts to model dynamics in temporal graphs using generative methods [36] and other deep learning approaches [65, 138, 146]. The model in [146] fused the sequential and spatial graph convolution in a "sandwich" structure, which is yet dependent on the convolution order of sub-structures [133]. The ST-GCN model in [138] learned both spatial and temporal patterns by adding all temporally connected nodes into temporal kernels and conduct similar convolution as GCN [64] on skeleton graph with static topology. There are recent works [100, 44] based on the recurrent neural networks to capture temporal dynamics by maintaining hidden states to summarize historical snapshots, and achieve state-of-the-art results on dynamic link prediction. However, the computation cost of the recurrent methods is the main bottleneck to be applied to the large scale real-world graphs.

2.5 Adversarial Regularization

The rapid advances in deep learning research in the last decades have provided novel methods for studying highly non-linear data like natural language or graphs. One such model is the Generative adversarial networks (GANs) [39] which has achieved great success in generating and learning the latent presentation of high dimensional data, such images [88]. The success of GANs on images has led many researchers to consider applying GANs to discrete data such as text and discrete images. Although there have been several successful attempts [43, 91, 59], using GANs to learn the representation of discrete contents like natural languages and social networks remains a challenging problem due to the difficulty in back-propagation through discrete random variables.

Another notable technique is adversarial autoencoders (AAE) [79] which attempt to imbue the model a more flexible prior implicitly through adversarial training. Recent work on Wasserstein autoencoders [112] provides a theoretical foundation for the AAE and shows that AAE minimizes the Wasserstein distance between the data/model distributions. Recent work on GANs such as GraphGAN [124] and ANE [25] for discrete data is either through the use of discrete structures [147, 21] or the improved autoencoders.

CHAPTER 3

Network Embedding with Adversarial Regularization

3.1 Background

Network analysis has been attracting many research interests with its enormous potential in mining useful information which benefits the downstream tasks such as link prediction, community detection and anomaly detection on social network [115], biological networks [109] and language networks [104], to name a few.

To analyze network data, one fundamental problem is to learn a low-dimensional vector representation for each vertex, such that the network structure is preserved in the learned vector space [86]. For this problem, there are two major challenges: (1) *preservation of complex structure property.* The objective of network embedding is to train a model to "fit" the training networks, that is, to preserve the structure property of networks [86, 94]. However, the latent structure of the network is too complex to be portrayed by an explicit form of probability density which can capture both the local network neighborhood information and global network structure. (2) *sparsity of network sampling.* Current research on network embedding employs network sampling techniques, including random walk sampling, breadth-first search etc., to derive vertex sequences as training datasets. However the sampled data represent only a small proportion of all the vertex sequences. An alternative approach is to encode these discrete structures in a continuous code space [122]. Unfortunately, learning continues latent representations of discrete networks remains a challenging problem since in many cases, the prior distribution may not exist in a low dimensional manifold [94].

Recent work on network embedding has shown fruitful progress in learning vertex representations of complex networks [86, 94, 122]. These representations employ nonlinear transformations to capture the "semantics" of the original networks. Most existing methods first

employ a random walk technique to sample a bunch of vertex sequences from the input network, then feed a learning model with these sequences to infer the optimal low-dimensional vertex embeddings. However, the sampling strategy suffers from the data sparsity problem since the total amount of vertex sequences is usually very large in real networks and it is often intractable to enumerate all. Subsequently, learning on a sparse sample set tends to produce an overly complex model to explain the sampled dataset, which eventually causes overfitting. Though autoencoders are adopted to encode the inputs into continuous latent representations [122], regularizations are still desirable to force the learned representations remain on the latent manifold. Ideally we could generate the continuous vertex representations with a prior distribution. However, in many cases, it is difficult, if not impossible, to pre-define an explicit form of the prior distribution in a low-dimensional manifold. For example, Dai et al. [25] proposed to train a discriminator to distinguish samples generated from a fixed prior distribution and the input encoding, and thereby pushing the embedding distribution to match the fixed prior. While this gives more flexibility, it suffers from the mode-collapse problem [59]. Moreover, most network embedding models with deep architectures usually do not consider the order of the vertices in the sampled vertex sequences [122]. Thus, the information of proximity orders cannot be well considered.

To address the aforementioned challenges, in this study, we propose a novel model to learn the network representations with adversarially regularized autoendoers (NetRA). NetRA jointly minimizes network locality-preserving loss and the reconstruction error of autoencoder which utilizes a long short-term memory network (LSTM) as an encoder to map the input sequences into a fixed length representation. The joint embedding inference is encapsulated in a generative adversarial training process to circumvent the requirement of an explicit prior distribution. As visually depicted in Figure 3.1, our model employs a discrete LSTM autoencoder to learn continuous vertex representations with sampled sequences of vertices as inputs. In this model, besides minimizing the reconstruction error in the LSTM autoencoder, the locality-preserving loss at the hidden layer is also minimized simultaneously. Meanwhile,

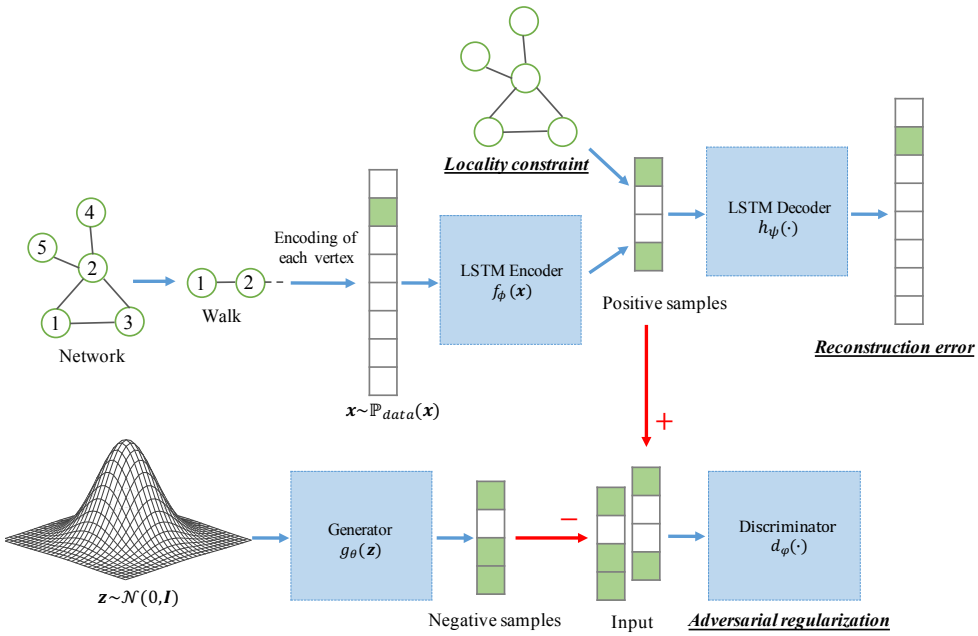


Figure 3.1: Illustration of the adversarially regularized autoencoders

the continuous space generator is also trained by constraining to agree in distribution with the encoder. The generative adversarial training can be regarded as a complementary regularizer to the network embedding process.

NetRA exhibits desirable properties that a network embedding model requires: 1) *structure property preservation*, NetRA leverages LSTM as an encoder to capture the neighborhood information among vertices in each sequence sampled from the network. Additionally, the model is also trained simultaneously with the locality-preserving constraint. 2) *generalization capability*, the generalization capability requires a network embedding model to generalize well on unseen vertex sequences which follow the same distribution as the population. The generative adversarial training process enables the proposed model to learn smoothly regularized representations without pre-defining an explicit density distribution which overcomes the sparsity issue from the input sequences of vertices. We present experimental results to show the embedding capability of NetRA on a variety of tasks, including network reconstruction, link prediction and multi-label classification. To summarize, the

main contributions of this work are as follows:

- We propose a novel deep network embedding model with adversarially regularized autoencoders, NetRA, to learn vertex representations by jointly minimizing locality-preserving loss and global reconstruction error using generative adversarial training process. The resultant representations are robust to the sparse inputs derived from the network.
- NetRA learns an adversarially regularized LSTM encoder that can produce useful vertex representations from discrete inputs, without a pre-defined explicit latent-space prior.
- We conduct extensive experiments on tasks of network reconstruction, link prediction and multi-label classification using real-world information networks. Experimental results demonstrate the effectiveness and efficiency of NetRA.

The rest of this chapter is organized as follows. In Section 3.2, we review the preliminary knowledge of autoencoders, generative adversarial networks and network embedding algorithms. In Section 3.3, we describe NetRA framework of learning a low dimensional mapping with generative adversarial training process. In Section 3.4, we demonstrate the performance of NetRA by adapting this joint learning framework on tasks of network reconstruction, link prediction and multi-label classification. Finally, in Section 3.5 we conclude this study and mention several directions for future work.

3.2 Preliminaries

3.2.1 Autoencoder Neural Networks

An autoencoder neural network is trained to set the target values to be equal to the inputs. The network consists of two parts: an encoder $f_\phi(\cdot)$ that maps inputs ($\mathbf{x} \in \mathbb{R}^n$) to latent

low-dimensional representations and a decoder $h_\psi(\cdot)$ that produces a reconstruction of the inputs. Specifically, given a data distribution $\mathbb{P}_{data}$, from which $\mathbf{x}$ is drawn from, i.e., $\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})$, we want to learn representations $f_\phi(\mathbf{x})$ such that the output hypotheses $h_\psi(f_\phi(\mathbf{x}))$ is approximately equal to $\mathbf{x}$. The learning process is described simply as minimizing a cost function

$$\min \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{3.1}$$

where $\text{dist}(\cdot)$ is some similarity metric in the data space. In practice, there are many options for the distance measure. For example, if we use ℓ_2 norm to measure the reconstruction error, then the objective function can be defined as $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}\|\mathbf{x} - h_\psi(f_\phi(\mathbf{x}))\|^2$. Similarly the objective function for cross-entropy loss can be defined as,

$$-\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\mathbf{x}\log h_\psi(f_\phi(\mathbf{x})) + (\mathbf{1}-\mathbf{x})\log(\mathbf{1} - h_\psi(f_\phi(\mathbf{x})))] \tag{3.2}$$

The choice of encoder $f_\phi(\cdot)$ and decoder $h_\psi(\cdot)$ may vary across different tasks. In this chapter, we use LSTM autoencoders [103] which are capable of dealing with sequences as inputs.

3.2.2 Generative Adversarial Networks

The Generative Adversarial Networks (GANs) [39] build an adversarial training platform for two players, namely *g*enerator $g_\theta(\cdot)$ and *d*iscriminator $d_w(\cdot)$, to play a minimax game.

$$\min_\theta \max_w \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\log d_w(\mathbf{x})] + \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[\log(1 - d_w(g_\theta(\mathbf{z})))] \tag{3.3}$$

The *generator* $g_\theta(\cdot)$ tries to map the noise to the input space as closely as the true data, while the *discriminator* $d_w(\mathbf{x})$ represents the probability that $\mathbf{x}$ came from the data rather than the noise. It aims to distinguish *real* data distribution $\mathbb{P}_{data}(\mathbf{x})$ and *fake* sample distribution $\mathbb{P}_g(\mathbf{z})$, e.g. $\mathbf{z} \sim \mathcal{N}(0, \mathbb{I})$. Wasserstein GANs [5] overcome unstable training problem by replacing Jensen-Shannon divergence with Earth-Mover (Wasserstein-1) distance, which

considers solving the problem

$$\min_{\theta} \max_{w \in \mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathbb{P}_{data}(\mathbf{x})} [d_w(\mathbf{x})] - \mathop{\mathbb{E}}_{\mathbf{z} \sim \mathbb{P}_g(\mathbf{z})} [d_w(g_\theta(\mathbf{z}))]. \tag{3.4}$$

The Lipschitz constraint $\mathcal{W}$ on discriminator has been kept by clipping the weights of the discriminator within a compact space [-c, c].

3.2.3 Network Embedding

Network embedding approaches seek to learn representations that encode structural information about the network. These approaches learn a mapping that embeds vertices as points into a low-dimensional space. Given the encoded vertex set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\}$, finding an embedding $f_\phi(\mathbf{x}^{(i)})$ of each $\mathbf{x}^{(i)}$ can be formalized as an optimization problem [149, 129]

$$\min_{\phi} \sum_{1 \leq i < j \leq n} L(f_\phi(\mathbf{x}^{(i)}), f_\phi(\mathbf{x}^{(j)}), \varphi_{ij}), \tag{3.5}$$

where $f_\phi(\mathbf{x}) \in \mathbb{R}^d$ is the embedding result for a given input $\mathbf{x}$. $L(\cdot)$ is the loss function between a pair of inputs. φ_{ij} is the weight between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

We consider the Laplacian Eigenmaps (LE) that well fits the framework. LE enables the embedding to preserve the locality property of network structure. Formally, the embedding can be obtained by minimizing the following objective function

$$\mathcal{L}_{LE}(\phi; \mathbf{x}) = \sum_{1 \leq i < j \leq n} \|f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)}\|^2 \varphi_{ij}. \tag{3.6}$$

3.3 The NetRA Model

In this section, we present NetRA, a deep network embedding model using adversarially regularized autoencoders, to learn smoothly regularized vertex representations with sequences of vertices as inputs. The resultant representations can be used in the downstream tasks, such as link prediction, network reconstruction and multi-class classification.

3.3.1 Random Walk Generator

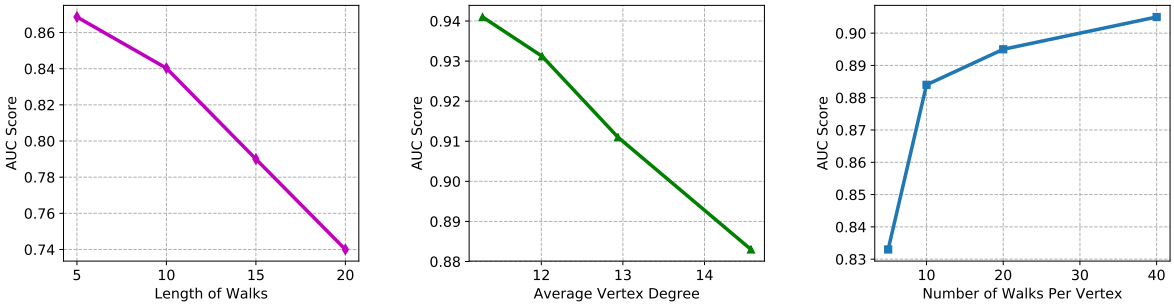
Given network $G(V, E)$, the random walk generator in DeepWalk [86] is utilized to obtain truncated random walks (i.e. sequences of vertices) rooted on each vertex $v \in V$ in $G(V, E)$. A walk is sampled randomly from the neighbors of the last visited vertex until the preset maximum length is reached.

The random walk sampling technique is widely adopted in network embedding research [41, 86, 122]. However, it suffers from the sparsity problem in network sampling. For each vertex in given network, if we assume that the average node degree is $\bar{d}$, the walk length is l and the number of samples is k, then the sampling fraction of walks can be calculated by

$$p_{frac} \propto \frac{|V| \times k}{|V| \times \bar{d}^l} = \frac{k}{\bar{d}^l} \times 100\%. \tag{3.7}$$

The effect of the sampling fraction is presented in Figure 3.2. In the example, DeepWalk is used to perform link prediction task on the UCI message network described in Section 3.4.1. Figure 3.2(a) and Figure 3.2(b) show that if the walk length or the average vertex degree increases, the performance decreases dramatically[1]. According to Eq. (3.7), obviously, when l or $\bar{d}$ increases, the sampling fraction of walks is getting smaller. Thereby, the trained model is prone to overfitting because of the sparse inputs. On the contrary, if the number of samples k increases, the performance is getting better as shown in Figure 3.2(c). However, more sampled walks also call for more computing burden on model training. Therefore, it is desirable to develop effective models with better capabilities of generalization on sparsely sampled network walks.

[1]In Figure 3.2(a), the window size of DeepWalk is set to be equal to the walk length. The reason is that, if the window size is set to a small value against a long walk length, it turns out to be equivalent to increase the samples per vertex with a short walk length. In Figure 3.2(b), we reduce the degree of the dataset by removing vertices with large degrees [11].



(a) Walk length (b) Vertex degree (c) Number of samples

Figure 3.2: Sparsity of network sampling.

3.3.2 Embedding with Adversarially regularized Autoencoders

In this chapter, we propose NetRA, a network embedding model with adversarially regularized autoencoders, to address the sparsity problem. Autoencoders are popularly used for data embedding, such as images and documents. It provides informative low dimensional representations of input data by mapping the them to the latent space. Unfortunately, if the encoder and decoder are allowed too much capacity, the autoencoder can learn to perform the copying task without extracting useful information about the distribution of the data [38]. We proposed to use a generative adversarial training process as a complementary regularizer. The process has two advantages. On one hand, the regularizer can guide the extraction of useful information about data [38]. On the other hand, the generative adversarial training provides more robust discrete-space representation learning that can well address the overfitting problem on sparsely sampled walks [79]. Specifically, in NetRA, the discriminator updates by comparing the samples from the latent space of the autoencoder with the fake samples from the generator, as shown in Figure 3.1. The latent space of autoencoder provides optimal embedding for the vertices in the network with the simultaneous update of encoder and discriminator. In this study, we use the LSTM as the encoder and decoder networks [103] because it takes the order information of the sampled walks into consideration.

This joint architecture requires dedicated training objective for each part. The autoencoder can be trained individually by minimizing the negative log-likelihood of reconstruction, which is indicated by cross entropy loss in the implementation

$$\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[\text{dist}(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))], \tag{3.8}$$

where $\text{dist}(\mathbf{x}, \mathbf{y}) = \mathbf{x}\log\mathbf{y} + (\mathbf{1}-\mathbf{x})\log(\mathbf{1}-\mathbf{y})$. Here $\mathbf{x}$ is the sampled batch from training data. $f_\phi(\mathbf{x})$ is embedded latent representation of $\mathbf{x}$, which is also the positive samples for discriminator, indicated by the arrow with "+" in Figure 3.1. ϕ and ψ are parameters of the encoder and decoder functions, respectively. In the training iteration of autoencoder, not only the encoder and decoder are updated, the locality-preserving loss (Eq. (3.6)) is jointly minimized.

As depicted in Figure 3.1, NetRA minimizes the distributions between the learned representations from the encoder function $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$, and the representations from the continuous generator model $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$. The dual form of the Earth Mover distance between $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$ can be described as follows [5]

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\|d(\cdot)\|_{L\leq 1}} \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[d(\mathbf{y})] \tag{3.9}$$

where $\|d(\cdot)\|_{L\leq 1}$ is the Lipschitz continuity constraint (with Lipschitz constant 1). If we have a family of functions $\{d_w(\cdot)\}_{w\in\mathcal{W}}$ that are all K-Lipschitz for some K, then we have

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \propto \max_{w\in\mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathop{\mathbb{E}}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.10}$$

We can separate the training of generator and discriminator. As for the generator, the cost function can be defined as,

$$\mathcal{L}_{\text{GEN}}(\theta; \mathbf{x}, \mathbf{z}) = \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.11}$$

and the cost function for discriminator is,

$$\mathcal{L}_{\text{DIS}}(w; \mathbf{x}, \mathbf{z}) = -\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[d_w(f_\phi(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \tag{3.12}$$

NetRA learns smooth representations by jointly minimizing the autoencoder reconstruction error and the locality-preserving loss in an adversarial training process. Specifically, we consider solving the joint optimization problem with objective function

$$\mathcal{L}_{\text{NetRA}}(\phi, \psi, \theta, w) = \mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x}) + \lambda_1 \mathcal{L}_{\text{LE}}(\phi; \mathbf{x}) + \lambda_2 W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) \tag{3.13}$$

Theorem 1. *Let $\mathbb{P}_\phi(\mathbf{x})$ be any distribution. Let $\mathbb{P}_\theta(\mathbf{z})$ be the distribution of $g_\theta(\mathbf{z})$ with $\mathbf{z}$ being a sample drawn from distribution $\mathbb{P}_g(\mathbf{z})$ and $g_\theta(\cdot)$ being a function satisfying the local Lipschitz constants $\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[L(\theta, \mathbf{z})] < +\infty$. Then we have*

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\text{NetRA}} &= -\lambda_2 \nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] && (3.14)\\
\nabla_w \mathcal{L}_{\text{NetRA}} &= -\lambda_2 \nabla_w \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] \\
&\quad +\lambda_2 \nabla_w \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\boldsymbol{z}))] && (3.15)\\
\nabla_\phi \mathcal{L}_{\text{NetRA}} &= \lambda_1 \nabla_\phi \sum_{1\le i<j\le n} \|f_\phi(\mathbf{x}^{(i)}) - f_\phi(\mathbf{x}^{(j)})\|^2 \varphi_{ij} \\
&\quad -\nabla_\phi \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[dist(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] \\
&\quad +\lambda_2 \nabla_\phi \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_{data}(\boldsymbol{x})}[d_w(f_\phi(\boldsymbol{x}))] && (3.16)\\
\nabla_\psi \mathcal{L}_{\text{NetRA}} &= -\nabla_\psi \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}(\mathbf{x})}[dist(\mathbf{x}, h_\psi(f_\phi(\mathbf{x})))] && (3.17)
\end{aligned}$$

Proof. Let $\mathcal{X} \subseteq \mathbb{R}^n$ be a compact set, and

$$\begin{aligned}
V(\tilde{d}, \theta) &= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\theta(\mathbf{z})}[\tilde{d}(\mathbf{y})] \\
&= \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[\tilde{d}(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[\tilde{d}(g_\theta(\mathbf{z}))] && (3.18)
\end{aligned}$$

where $\tilde{d}$ lies in $\mathcal{D} = \{\tilde{d} : \mathcal{X} \to \mathbb{R}, \tilde{d} \text{ is continuous and bounded}, \|\tilde{d}\| \le 1\}$. Since $\mathcal{X}$ is compact, we know by the Kantorovich-Rubinstein duality [5] that there exists a $d \in \mathcal{D}$ that attains the value

$$W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \sup_{\tilde{d}\in\mathcal{D}} V(\tilde{d}, \theta) = V(d, \theta) \tag{3.19}$$

and $D^*(\theta) = \{d \in \mathcal{D} : V(d, \theta) = W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z}))\}$ is non-empty. According to the envelope theorem [82], we have

$$\nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) = \nabla_\theta V(d, \theta) \tag{3.20}$$

for any $d \in D^*(\theta)$. Then we get

$$\begin{aligned}\nabla_\theta W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z})) &= \nabla_\theta V(d, \theta) \\ &= \nabla_\theta \mathbb{E}_{\mathbf{y}\sim\mathbb{P}_\phi(\mathbf{x})}[d(\mathbf{y})] - \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d(g_\theta(\mathbf{z}))] \\ &= -\nabla_\theta \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))] \qquad (3.21)\end{aligned}$$

Therefore, we have $\nabla_\theta \mathcal{L}_{\text{NETRA}} = -\lambda_2 \nabla_\theta \mathbb{E}_{\mathbf{z}\sim\mathbb{P}_g(\mathbf{z})}[d_w(g_\theta(\mathbf{z}))]$.

Eq.(3.15)-(3.17) are straightforward applications of the derivative definition. □

We now have all the derivatives needed. To train the model, we use a block coordinate descent to alternate between optimizing different parts of the model: (1) locality-preserving loss and autoencoder reconstruction error (update ϕ and ψ), (2) the discriminator in the adversarial training process (update w), and (3) the generator (update θ). Pseudocode of the full approach is given in Algorithm 1.

The training process of NETRA consists of the following steps: Firstly, given a network $G(V, E)$, we run random walk generator acquiring random walks of length l. Then, one hot representation $\mathbf{x}^{(i)}$ of each vertex is taken as input to LSTM cells. We pass the random walks through encoding layers and obtain the vector representations of vertices. After the decoder network, the vertex representations will be transformed back into n dimensions. Cross-entropy loss is calculated between the inputs and outputs by minimizing the reconstruction error in autoencoder operation. Meanwhile, locality-preserving constraint ensures that the adjacent vertices are in close proximity (Step 2-7 in Algorithm 1). The latent representation of encoder and the output of generator will be fed into discriminator to get adversarial loss (Step 10-17). Additionally, the generator transforms Gaussian noise into the latent space as closely as the true data, by passing through multilayer perceptron (Step 20-23). After the training of NETRA, we obtain the vertex representations $f_\phi(\mathbf{x})$ of the network by passing the input walks through the encoder function.

Optimality Analysis. NETRA, as illustrated in Figure 3.1, can be interpreted as minimizing the divergence between two distributions, namely $\mathbb{P}_\phi(\mathbf{x})$ and $\mathbb{P}_\theta(\mathbf{z})$. We provide

Algorithm 1 NetRA Model Training

Require: the walks generated from input graph, maximum training epoch n_{epoch}, the number of discriminator training per generator iteration n_D.

for $epoch = 0; epoch < n_{epoch}$ **do**
  Minimizing $\mathcal{L}_{\text{LE}}(\phi; \mathbf{x})$ **with autoencoder** $\mathcal{L}_{\text{AE}}(\phi, \psi; \mathbf{x})$
  Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
  Compute latent representation $f_\phi(\mathbf{x}^{(i)})$
  Compute reconstruction output $h_\psi(f_\phi(\mathbf{x}^{(i)}))$
  Compute $\mathcal{L}_{\text{AE}}(\phi, \psi)$ and $\mathcal{L}_{\text{LE}}(\phi)$ using Eq.(3.8) and Eq.(3.6)
  Backpropagate loss and update ϕ and ψ using Eq.(3.16)-(3.17)

  Discriminator training
  for $n = 0, n < n_D$ **do**
    Sample $\{\mathbf{x}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{data}(\mathbf{x})$ a batch from the walks
    Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
    Compute representations $f_\phi(\mathbf{x}^{(i)})$ and $g_\theta(z^{(i)})$
    Compute $\mathcal{L}_{\text{DIS}}(w)$ using Eq.(3.12)
    Backpropagate loss and update w using Eq.(3.15)
    clip the weight w within $[-c, c]$
  end for

  Generator training
  Sample $\{\mathbf{z}^{(i)}\}_{i=1}^{B} \sim \mathbb{P}_{g}(\mathbf{z})$ a batch from the noise
  Compute the representation $g_\theta(\mathbf{z}^{(i)})$
  Compute $\mathcal{L}_{\text{GEN}}(\theta)$ using Eq.(3.11)
  Backpropagate loss and update θ using Eq.(3.14)
end for

the following proposition which shows that under our parameter settings, if the Wasserstein distance converges, the encoder distribution $f_\phi(\mathbf{x}) \sim \mathbb{P}_\phi(\mathbf{x})$ converges to the generator distribution $g_\theta(\mathbf{z}) \sim \mathbb{P}_\theta(\mathbf{z})$.

Proposition 1. *Let $\mathbb{P}$ be a distribution on a compact set $\mathcal{X}$, and $(\mathbb{P}_n)_{\in\mathbb{N}}$ be a sequence of distributions on $\mathcal{X}$. Considering $W(\mathbb{P}_n, \mathbb{P}) \to 0$ as $n \to \infty$, the following statements are equivalent:*

1. *$\mathbb{P}_n \overset{\mathcal{D}}{\rightsquigarrow} \mathbb{P}$ where $\overset{\mathcal{D}}{\rightsquigarrow}$ represents convergence in distribution for random variables.*
2. *$\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$, where $F(\mathbf{x}) = \Pi_{i=1}^n \mathbf{x}_i^{p_i}, \mathbf{x} \in \mathbb{R}^n, \sum_{i=1}^n p_i = k, k > 1, k \in \mathbb{N}$.*

Proof. (1) As shown in [119], $\mathbb{P}_n$ converges to $\mathbb{P}$ is equivalent to $W(\mathbb{P}_n, \mathbb{P}) \to 0$.

(2) According to the *Portmanteau Theorem* [119], $\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[F(\mathbf{x})] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[F(\mathbf{x})]$ holds if $F : \mathbb{R}^n \to \mathbb{R}$ is a bounded continuous function. Our encoder $f_\phi(\cdot)$ is bounded as the inputs are normalized to lie on the unit sphere, and our generator $g_\theta(\cdot)$ is also bounded to lie in $(-1, 1)^n$ by *tanh* function. Therefore, $F(\mathbf{x}) = \Pi_{i=1}^n \mathbf{x}_i^{p_i}$ is a bounded continuous function for all $p_i > 0$, and

$$\mathbb{E}_{\mathbf{x}\sim\mathbb{P}_n}[\Pi_{i=1}^n \mathbf{x}_i^{p_i}] \to \mathbb{E}_{\mathbf{x}\sim\mathbb{P}}[\Pi_{i=1}^n \mathbf{x}_i^{p_i}] \tag{3.22}$$

such that $\sum_{i=1}^n p_i = k, \forall k > 1, k \in \mathbb{N}$. □

Computational Analysis. Given a network $G(V, E)$, where $|V| = n, |E| = m$, according to the definition in Eq.(3.6), the overall complexity of Laplacian Eigenmaps embedding is $\mathcal{O}(n^2)$. In our implementation, we only consider the vertex pairs $(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ that have edges between them, thus the size of the sampled pairs is $\mathcal{O}(m)$, which is much smaller than $O(n^2)$ because real networks are sparse in real settings.

The computational complexity of learning LSTM autoencoders is proportional to the number of parameters $|\phi|$ and $|\psi|$ in each iteration. Therefore, the learning computational complexity for LSTM autoencoders is $\mathcal{O}(n_{epoch} \times (|\phi| + |\psi|))$. Similarly, for the generator

and discriminator, each invocation of backpropagation is typically linear in the number of parameters $\mathcal{O}(|\theta|)$ and $\mathcal{O}(|w|)$. Thus the computational complexity for generator and discriminator is $\mathcal{O}(n_{epoch} \times (n_D \times |w| + |\theta|))$. It is basically quadratic if the input and hidden layers are of roughly the same size. However, if we set the size of embedding layers much less than that of the inputs, the time complexity reduces to $\mathcal{O}(n)$.

3.4 Evaluation

We evaluate the performance of our model with extensive experiments on tasks including network reconstruction, link prediction and multi-label classification, using a variety of network datasets.

3.4.1 Datasets

To verify the performance of the proposed network embedding model, we conduct experiments on a variety of networks from different domains including the social network, software dependency network, biological network and language network, as summarized in Table 3.1.

- UCI message (UCI) [84] is a directed communication network containing sent messages (edges) between the users (vertices) of an online community of students from the University of California Irvine.
- JDK dependency (JDK)[2] is the software class dependency network of the JDK 1.6.0.7 framework. The network is directed, with vertices representing Java classes and an edge between two vertices indicating there exists a dependency between the two classes.
- Blogcatalog (BLOG) [106] is an undirected social network from BlogCatalog website which manages the bloggers and their blogs. The vertices represent users and edges

[2]http://konect.uni-koblenz.de/networks/subelj_jdk

Table 3.1: Statistics of the real-world network datasets

Dataset	$\|V\|$	$\|E\|$	**Avg. degree**	**#label**	**Type**
UCI	1,899	27,676	14.57	-	Directed
JDK	6,434	53,892	8.38	-	Directed
BLOG	10,312	333,983	32.96	-	Undirected
DBLP	180,768	382,732	4.23	-	Undirected
PPI	3,890	76,584	19.69	50	Directed
WIKI	4,777	184,812	38.69	40	Directed

represent friendship between users.

- DBLP[3] is an undirected collaboration graph of authors from the DBLP computer science bibliography. The vertices in this network represent the authors, and the edges represent the co-authorships between two authors.

- Wikipedia (WIKI) [41] is a directed word network. Vertex labels represent the Part-of-Speech (POS) tags inferred using the Stanford POS-Tagger [113].

- Protein-Protein Interactions (PPI) [12] is a subgraph of the PPI network for Homo Sapiens, which is a network depicting interactions between human proteins. The vertex label indicates biological states of proteins.

3.4.2 Comparing Algorithms

To evaluate the performance of our network embedding model, the competitors used in this chapter are summarized as follows.

- Spectral Clustering (SC) [107]: SC is an approach based on matrix factorization, generating the vertex representation with the smallest d eigenvectors of the normalized

[3] `http://dblp.uni-trier.de/xml`

Laplacian matrix of the graph.

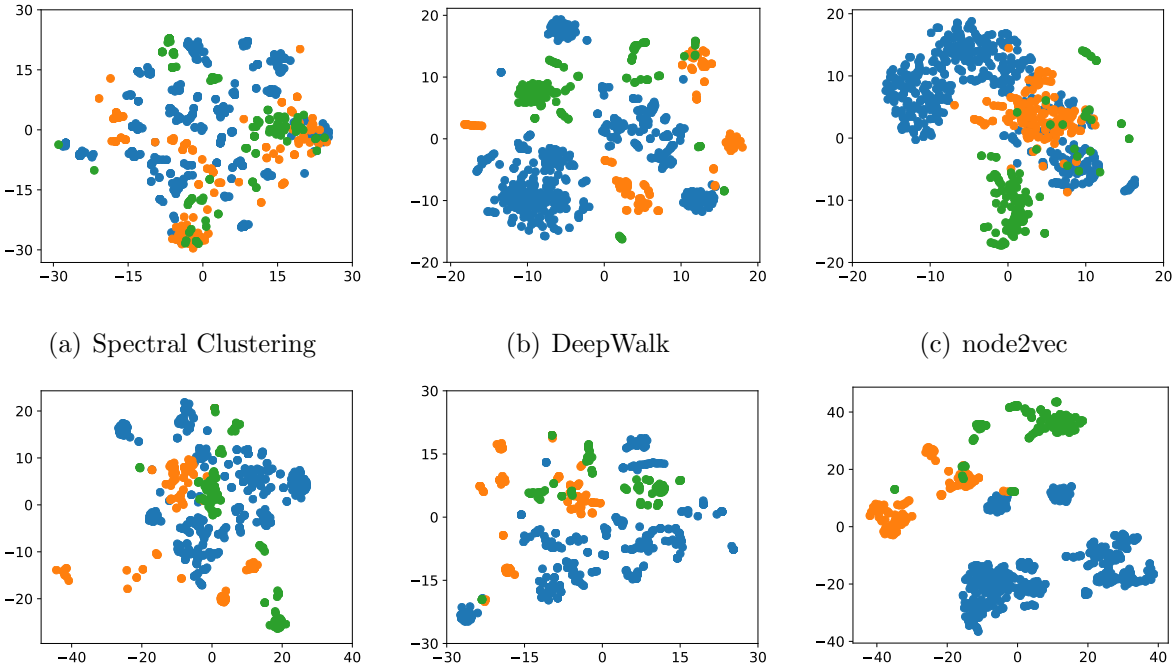
- DeepWalk [86]: DeepWalk is a skip-gram [81] based model which learns the graph embedding with truncated random walks.

- node2vec [41]: This approach combines the advantage of breadth-first traversal and depth-first traversal algorithms. The random walks generated by node2vec can better represent the structural equivalence.

- Structural Deep Network Embedding (SDNE) [122]: SDNE is a deep learning based network embedding model which uses autoencoder and locality-preserving constraint to learn vertex representations that capture the highly non-linear network structure.

- Adversarial Network Embedding (ANE) [25]:ANE proposes to train a discriminator to push the embedding distribution to match the fixed prior.

For fair comparison [70], we run each algorithm to generate 300 dimensional vertex representations on different datasets, unless noted otherwise. The number of walks per vertex in DeepWalk and node2vec is set to 10 with walk length 30, which is the same as the random walk generation step of NetRA. The window size of DeepWalk and node2vec is optimized to 10. node2vec is optimized with grid search over its return and in-out parameters $(p, q) \in \{0.25, 0.50, 1, 2, 4\}$. For SDNE, we utilize the default parameter setting as described in [122]. For NetRA, the gradient clipping is performed in every training iteration to avoid the gradient explosion, and we use stochastic gradient descent as the optimizer of autoencoder networks. The multilayer perceptron (MLP) is used in the generator and discriminator. The evaluation of different algorithms is based on applying the embeddings they learned to the downstream tasks, such as link prediction, network reconstruction, and multi-label classification as will be illustrated in the subsequent sections.



(a) Spectral Clustering (b) DeepWalk (c) node2vec

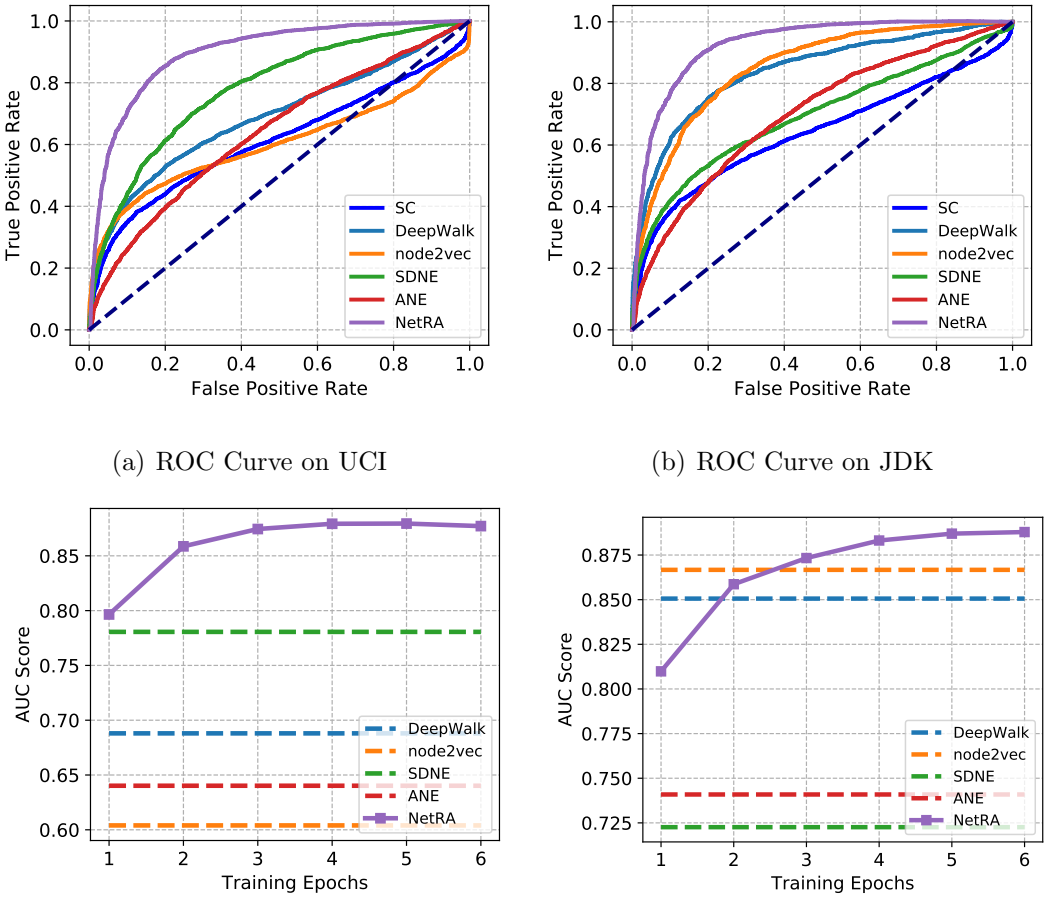
(d) SDNE (e) ANE (f) NetRA

Figure 3.3: Visualization results of the compared methods on JDK dependency network. The red points belong to class *org.omg*; the green points belong to class *org.w3c*; the blue points belong to class *java.beans*.

3.4.3 Visualization

In order to demonstrate how well key properties of network structure are captured by the network embedding models, we visualize the embeddings of each compared method. We run different embedding algorithms described in Section 3.4.2 to obtain low dimensional representations of each vertex and map vertex vectors onto a two dimensional space using t-SNE [116]. With vertex colored by its label, we perform the visualization task on JDK dependency network, as shown in Figure 3.3.

As observed in Figure 3.3, three classes are presented: red points for *org.omg*, green points for *org.w3c* and blue points for *java.beans*. It can be seen that the eigenvector-based method Spectral Clustering cannot effectively identify different classes. Other baselines can



(a) ROC Curve on UCI

(b) ROC Curve on JDK

(c) UCI

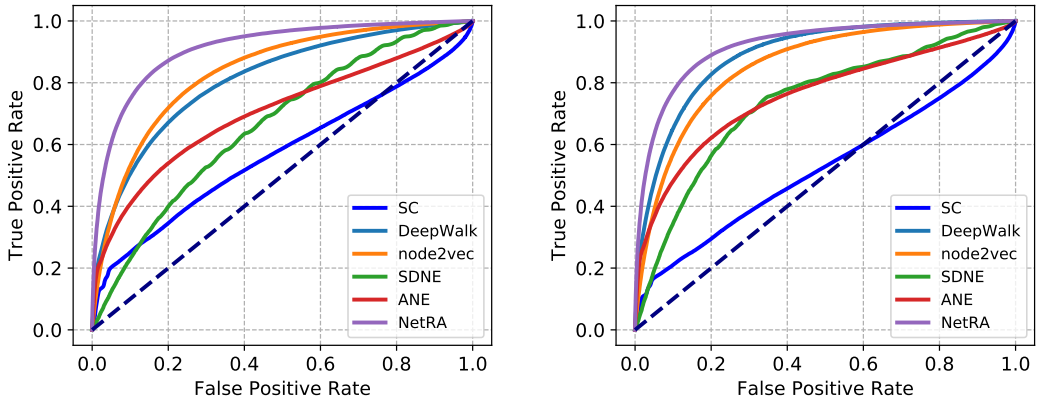
(d) JDK

Figure 3.4: Link prediction using vertex representation on UCI and JDK. Evaluated with AUC ROC score versus training epochs.

detect the classes to varying extents. NetRA performs best as it can separate these three classes with large boundaries, except for a small overlap between green and red vertices.

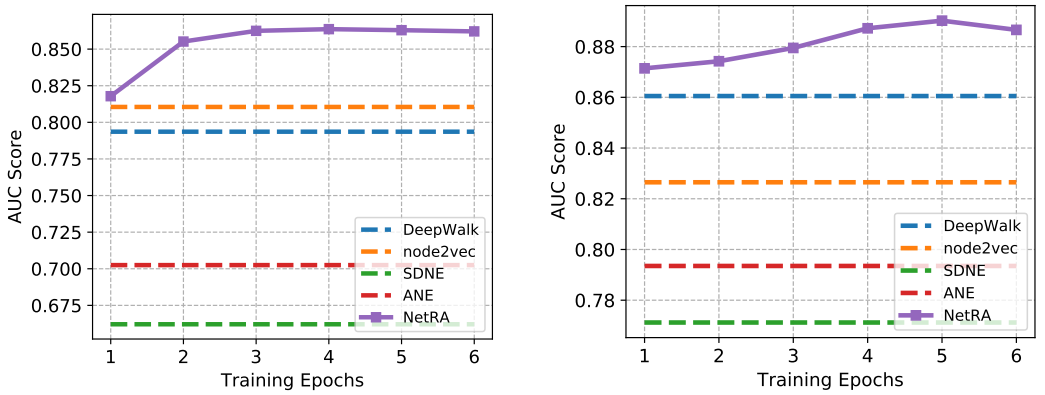
3.4.4 Link Prediction

The objective of link prediction task is to infer missing edges given a network with a certain fraction of edges removed. We randomly remove 50% of edges from the network, which serve as positive samples, and select an equal number of vertex pairs without linkage between them



(a) ROC Curve on BLOG

(b) ROC Curve on DBLP



(c) BLOG

(d) DBLP

Figure 3.5: Link prediction using vertex representation on BLOG and DBLP. Evaluated with AUC ROC score versus training epochs.

as negative samples. With vertex representation learned by network embedding algorithms, we obtain the edge feature from the ℓ_2 norm of two vertex vectors, and use it directly to predict missing edges. Because our focus is network embedding model, this simple experimental setup can evaluate the performance based on the assumption that the representations of two connected vertices should be closer in the Euclidean space. We use the area under curve (AUC) score for evaluation on link prediction task. The results are shown in Table 3.2.

Obviously, we observe that NetRA outperforms the baseline algorithms across all datasets

Table 3.2: AUC score of link prediction

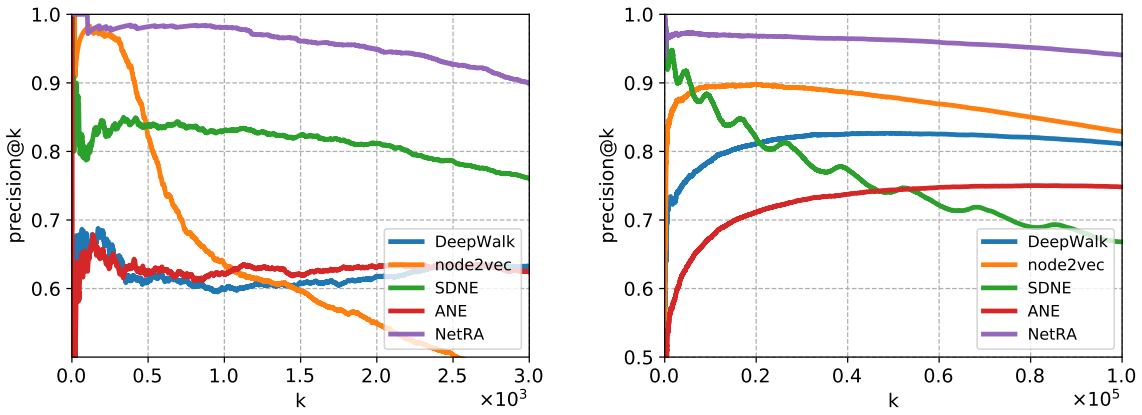
Method	UCI	JDK	BLOG	DBLP
SC	0.6128	0.6686	0.6014	0.5740
DeepWalk	0.6880	0.8506	0.7936	0.8605
node2vec	0.6040	0.8667	0.8105	0.8265
SDNE	0.7806	0.7226	0.6621	0.7712
ANE	0.6402	0.7409	0.7025	0.7935
NetRA	**0.8879**	**0.8913**	**0.8627**	**0.8902**

by a large margin. It can be seen that NetRA achieves 3% to 32% improvement based on the AUC score on the four datasets. By comparing NetRA, node2vec and DeepWalk, which all use random walks as inputs, we can see the effectiveness of generative adversarial regularization for improving the generalization performance in NetRA model. With same random walk sequences, NetRA can overcome the sparsity issue from the sampled sequences of vertices.

We also plot the ROC curve of these four datasets, as shown in Figure 3.4(a)-(d). The ROC curve of NetRA dominates other approaches and is very close to the $(0, 1)$ point. We train the NetRA model with different epochs for different datasets and embed the vertices to get representations after each training epoch. The results are shown in Figure 3.5(a)-(d). Generally, we can observe that NetRA converges pretty fast with high AUC score almost after the first epoch. When comparing with Deepwalk, node2vec, SDNE and ANE, we can clearly see the better performance of NetRA on these datasets.

3.4.5 Network Reconstruction

Network embeddings are considered as effective representations of the original network. The vertex representations learned by networking embedding maintain the edge information for



(a) UCI (b) BLOG

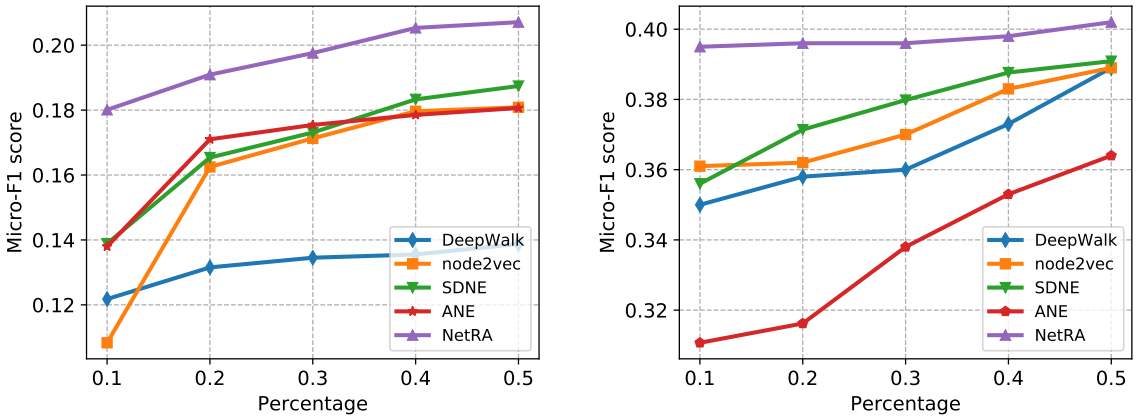
Figure 3.6: Network reconstruction results on UCI and BLOG

network reconstruction. We randomly select vertex pairs as edge candidates and calculate the Euclidean distance between the vertices. We use the *precision@k*, the fraction of correct predictions in the top k predictions, for evaluation.

$$precision@k = \frac{1}{k} \times |E_{pred}(1:k) \cap E_{obs}|, \tag{3.23}$$

where $E_{pred}(1:k)$ represents the top k predictions and E_{obs} represents observed edges in original network. In the evaluation, the UCI message and Blogcatalog datasets have been utilized to illustrate the performance of NetRA, with results shown in Figure 3.6.

As it can be seen from the *precision@k* curves, the NetRA model achieves higher precision in the network reconstruction task. The total number of edge candidates selected in this task is $8k$ for UCI message and $300k$ for Blogcatalog. The reconstruction given by NetRA is very accurate in predicting most positive samples (results on JDK and DBLP datasets show similar trends which haven't been included here). DeepWalk and node2vec can give reasonable reconstruction but the results are worse than NetRA for most k's. By learning smoothly regularized vertex representations using generative adversarial training process [39], our model well integrates the locality-preserving and global reconstruction constraints to learn embeddings that capture the "semantic" information.



(a) PPI

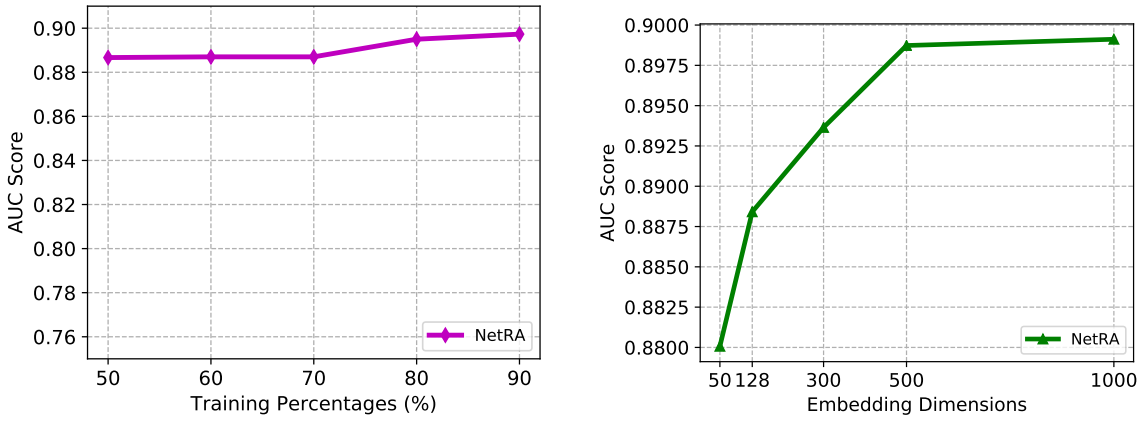
(b) WIKI

Figure 3.7: Multi-label classification on PPI and Wikipedia

3.4.6 Multi-label Classification

The task of predicting vertex labels with representations learned by network embedding algorithms is widely used in recent studies for performance evaluation [86, 41, 122]. An effective network embedding algorithm should capture network topology and extract most useful features for downstream machine learning tasks. In this section, we use vertex features as input to a one-vs-rest logistic regression using the LIBLINEAR [33] package to train the classifiers. For the Wikipedia and PPI datasets, we randomly sample 10% to 50% of the vertex labels as the training set and use the remaining vertices as the test set. We report *Micro-F1* [122]as evaluation metrics. Each result is averaged by five runs, as shown in Figure 3.7.

It is evident from the figure that NetRA outperforms the state-of-the-art embedding algorithms on multi-label classification task. In the PPI dataset, NetRA achieves higher *Micro-F1* scores than the baseline models by over 10% in all experiment settings. In the Wikipedia dataset, NetRA model performs better even with lower percentage training set. This well illustrates the good generalization performance when the training set is sparse. The multi-label classification task shows that, with adversarially regularized LSTM autoencoders,



(a) Edge percentage for training

(b) Embedding dimensions

Figure 3.8: Parameter sensitivity analysis

the neighborhood information can be well captured by the low dimensional representations.

3.4.7 Parameter Sensitivity

In this section, we investigate the parameter sensitivity in NetRA for link prediction. We study how the training set size, embedding dimension and locality-preserving constraint parameter λ_1 will affect the performance of link prediction. Also by changing the architecture of the NetRA model, we can investigate roles of different components in NetRA. Note that similar observations can be made on multi-label classification and network reconstruction tasks.

In Figure 3.8(a), we vary the training percentage of edges in the UCI message network. As it can be seen, the performance increases as the training ratio increases. Comparing with other algorithms, NetRA can capture the network topology even with a small proportion of edges for training, which demonstrates the generalization capability of the NetRA model. In Figure 3.8(b), we vary the embedding dimension from 50 to 1000. The prediction performance gets saturated as the dimension increases. Considering that the embedding dimension is related to the parameter volume in NetRA, there exists a trade off between

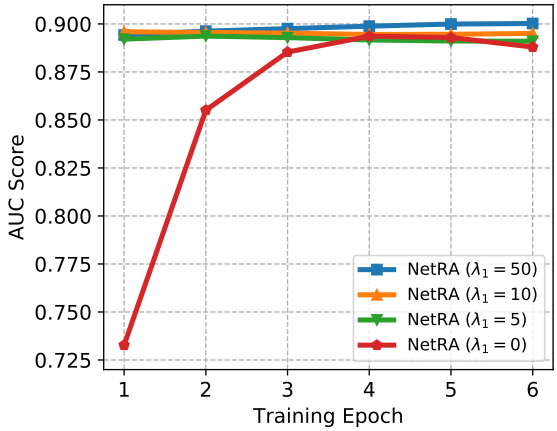


Figure 3.9: Performance on different λ_1 for $\mathcal{L}_{\text{LE}}$

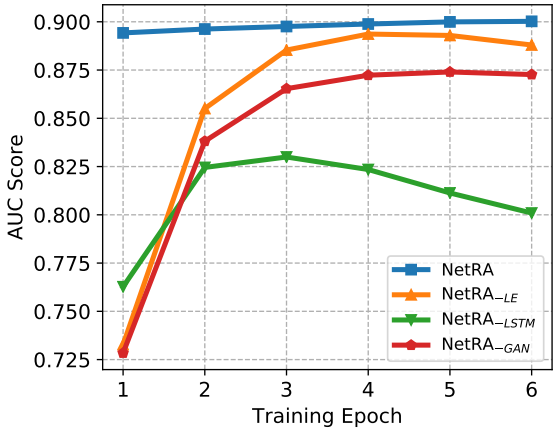


Figure 3.10: Performance on different NetRA architectures

the performance and the efficiency during model training.

The parameter λ_1 is defined by the relative strength between locality-preserving constraint and autoencoder constraint. The higher the λ_1, the larger the gradient comes from the locality-preserving constraint. As observed from the Figure 3.9, a higher λ_1 enhances the link prediction performance on the UCI message network, indicating the important role of local proximity.

We also include three variants of NetRA to demonstrate the importance of individual components in NetRA, including NetRA$_{-LE}$, NetRA$_{-LSTM}$, and NetRA$_{-GAN}$. NetRA$_{-LE}$ and NetRA$_{-GAN}$ remove the locality-preserving constraint $\mathcal{L}_{\text{LE}}$ and adversarial regularization $W(\mathbb{P}_\phi(\mathbf{x}), \mathbb{P}_\theta(\mathbf{z}))$, respectively. As for NetRA$_{-LSTM}$, we replace LSTM with multilayer perceptron. It's evident from Figure 3.10 that LSTM autoencoder, locality-preserving constraint, and adversarial regularization play important roles in NetRA model. The overfitting becomes obvious in the training of NetRA$_{-LSTM}$ and NetRA$_{-GAN}$.

3.5 Summary

In this study we proposed NetRA, a deep network embedding model for encoding each vertex in a network as a low-dimensional vector representation with adversarially regularized autoencoders. Our model demonstrated the ability of generative adversarial training process in extracting informative representations. The proposed model has better generalization capability, without requiring an explicit prior density distribution for the latent representations. Specifically, we leveraged LSTM autoencoders that take the sampled sequences of vertices as input to learn smooth vertex representations regularized by locality-preserving constraint and generative adversarial training process. The resultant representations are robust to the sparse vertex sequences sampled from the network. Empirically, we evaluated the learned representations with a variety of network datasets on different tasks such as network reconstruction, link prediction and multi-label classification. The results showed substantial improvement over the state-of-the-art network embedding competitors.

CHAPTER 4

Robust Graph Representation Learning via Neural Sparsification

4.1 Background

Representation learning has been in the center of many machine learning tasks on graphs, such as name disambiguation in citation networks [157], spam detection in social networks [4], recommendations in online marketing [143], and many others [150, 74, 162]. As a class of models that can simultaneously utilize non-structural (*e.g.*, node and edge features) and structural information in graphs, Graph Neural Networks (GNNs) construct effective representations for downstream tasks by iteratively aggregating neighborhood information [73, 45, 64]. Such methods have demonstrated state-of-the-art performance in classification and prediction tasks on graph data [117, 22, 136, 142].

Meanwhile, the underlying motivation why two nodes get connected may have no relation to a target downstream task, and such task-irrelevant edges could hurt neighborhood aggregation as well as the performance of GNNs. Consider the following example shown in Figure 4.1. Blue and Red are two classes of nodes, whose two-dimensional features are generated following two independent Gaussian distributions, respectively. As shown in Figure 4.1(a), the overlap between their feature distributions makes it difficult to find a good boundary that well separates the Blue and Red nodes by node features only. Blue and Red nodes are also inter-connected forming a graph. For each node (either Blue or Red), it randomly selects 10 nodes as its one-hop neighbors, and the resulting edges may not be related to node labels. On such a graph, we train a two-layer GCN [64], and the node representations output from the two-layer GCN is illustrated in Figure 4.1(b). When task-irrelevant edges

are mixed into neighborhood aggregation, the trained GCN fails to learn better representations, and it becomes difficult to learn a subsequent classifier with strong generalization power.

In this chapter, we study how to utilize supervision signals to remove task-irrelevant edges in an inductive manner to achieve robust graph representation learning. Conventional methods, such as graph sparsification [75, 154, 67, 99, 121], are unsupervised such that the resulting sparsified graphs may not favor downstream tasks. Several works focus on downsampling under predefined distributions [152, 45, 22]. As the predefined distributions may not well adapt to subsequent tasks, such methods could suffer suboptimal prediction performance. Multiple recent efforts strive to make use of supervision signals to remove noise edges [125]. However, the proposed methods are either transductive with difficulty to scale [34] or of high gradient variance bringing increased training difficulty [96].

Present work. We propose Neural Sparsification (NeuralSparse), a general framework that simultaneously learns to select task-relevant edges and graph representations by feedback signals from downstream tasks. The NeuralSparse consists of two major components: sparsification network and GNN. For the sparsification network, we utilize a deep neural network to parameterize the sparsification process: how to select edges from the one-hop neighborhood given a fixed budget. In the training phase, the network learns to optimize a sparsification strategy that favors downstream tasks. In the testing phase, the network sparsifies input graphs following the learned strategy, instead of sampling subgraphs from a predefined distribution. Unlike conventional sparsification techniques, our technique takes both structural and non-structural information as input and optimizes the sparsification strategy by feedback from downstream tasks, instead of using (possibly irrelevant) heuristics. For the GNN component, the NeuralSparse feeds the sparsified graphs to GNNs and learns graph representations for subsequent prediction tasks. Under the NeuralSparse framework, by the standard stochastic gradient descent and backpropagation techniques, we can simultaneously optimize graph sparsification and representations. As shown in Figure 4.1(d),

with task-irrelevant edges automatically excluded, the node representations learned from the NeuralSparse suggest a clearer boundary between Blue and Red with promising generalization power, and the sparsification learned by NeuralSparse could be more effective than the regularization provided by layer-wise random edge dropping [96] shown in Figure4.1(c).

Experimental results on both public and private datasets demonstrate that NeuralSparse consistently provides improved performance for existing GNNs on node classification tasks, yielding up to 7.2% improvement.

4.2 Proposed Method: NeuralSparse

In this section, we introduce the core idea of our method. We start with the notations that are frequently used in this chapter. We then describe the theoretical justification behind NeuralSparse and our architecture to tackle the supervised node classification problem.

Notations. We represent an input graph of n nodes as $G = (V, E, \mathbf{A})$: (1) $V \in \mathbb{R}^{n \times d_n}$ includes node features with dimensionality d_n; (2) $E \in \mathbb{R}^{n \times n}$ is a binary matrix where $E(u, v) = 1$ if there is an edge between node u and node v; (3) $\mathbf{A} \in \mathbb{R}^{n \times n \times d_e}$ encodes input edge features of dimensionality d_e. Besides, we use Y to denote the prediction target in downstream tasks (*e.g.,* $Y \in \mathbb{R}^{n \times d_l}$ if we are dealing with a node classification problem with d_l classes).

Theoretical justification. From the perspective of statistical learning, the key of a defined prediction task is to learn $P(Y \mid G)$, where Y is the prediction target and G is an input graph. Instead of directly working with original graphs, we would like to leverage sparsified subgraphs to remove task-irrelevant information. In other words, we are interested in the following variant,

$$P(Y \mid G) \approx \sum_{g \in \mathbb{S}_G} P(Y \mid g) P(g \mid G), \tag{4.1}$$

where g is a sparsified subgraph, and $\mathbb{S}_G$ is a class of sparsified subgraphs of G.

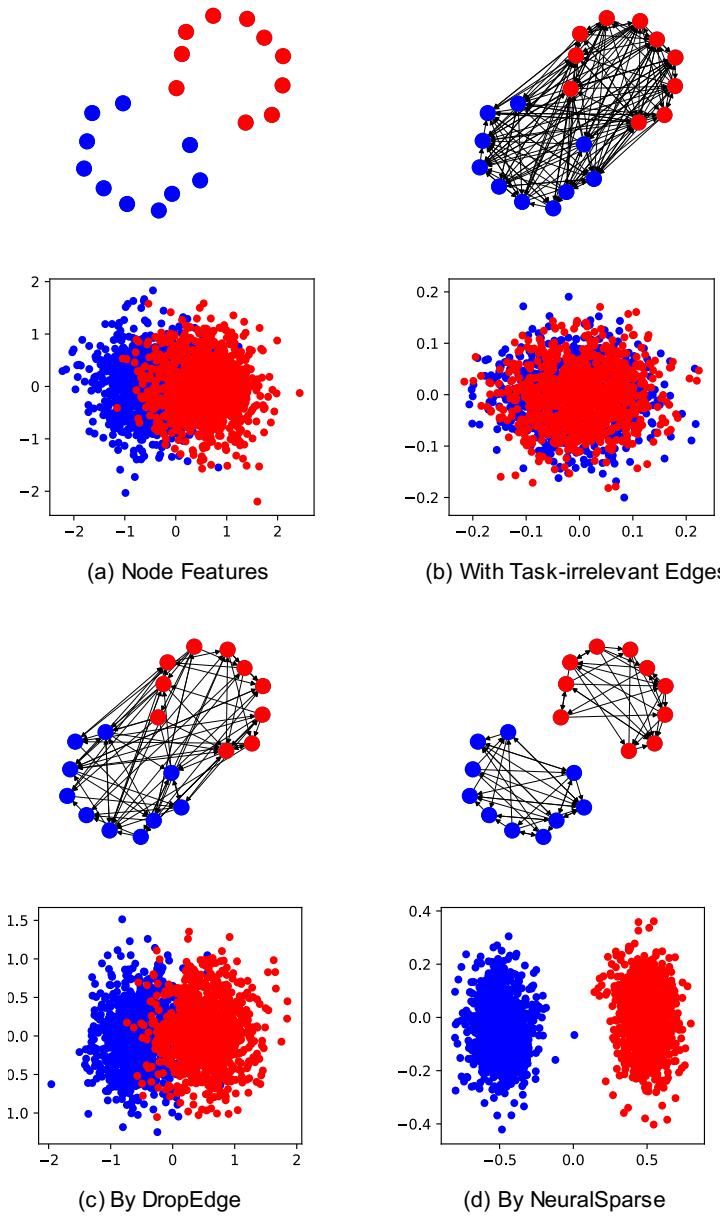


Figure 4.1: Example of task-irrelevant edges in graphs.

Top: Samples of graphs. Bottom: Visualization of node representations that are (a) input two-dimensional node features. (b) learned from a two-layer GCN on top of graphs with task irrelevant edges. (c) learned from DropEdge. (d) learned from NeuralSparse.

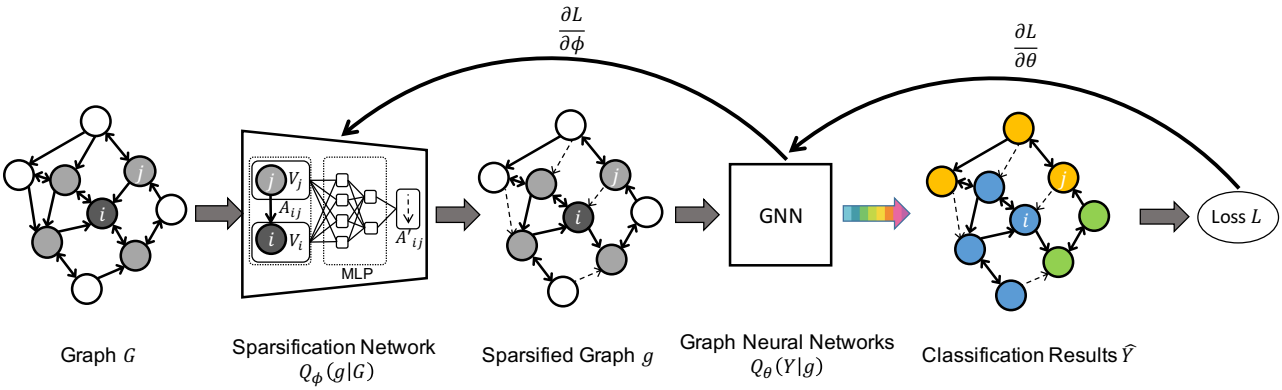


Figure 4.2: The overview of NeuralSparse

In general, because of the combinatorial complexity in graphs, it is intractable to enumerate all possible g as well as estimate the exact values of $P(Y \mid g)$ and $P(g \mid G)$. Therefore, we approximate the distributions by tractable functions,

$$\sum_{g \in \mathbb{S}_G} P(Y \mid g) P(g \mid G) \approx \sum_{g \in \mathbb{S}_G} Q_\theta(Y \mid g) Q_\phi(g \mid G) \tag{4.2}$$

where Q_θ and Q_ϕ are approximation functions for $P(Y \mid g)$ and $P(g \mid G)$ parameterized by θ and ϕ, respectively.

Moreover, to make the above graph sparsification process differentiable, we employ reparameterization tricks [53] to make $Q_\phi(g \mid G)$ directly generate differentiable samples, such that

$$\sum_{g \in \mathbb{S}_G} Q_\theta(Y \mid g) Q_\phi(g \mid G) \propto \sum_{g' \sim Q_\phi(g|G)} Q_\theta(Y \mid g') \tag{4.3}$$

where $g' \sim Q_\phi(g \mid G)$ means g' is a random sample drawn from $Q_\phi(g \mid G)$.

To this end, the key is how to find appropriate approximation functions $Q_\phi(g \mid G)$ and $Q_\theta(Y \mid g)$.

Architecture. In this chapter, we propose Neural Sparsification (NeuralSparse) to implement the theoretical framework discussed in Equation 4.3. As shown in Figure 4.2, NeuralSparse consists of two major components: the sparsification network and GNNs.

- The sparsification network is a multi-layer neural network that implements $Q_\phi(g \mid G)$: Taking G as input, it generates a random sparsified subgraph of G drawn from a learned distribution.

- GNNs implement $Q_\theta(Y \mid g)$ that takes the sparsified subgraph as input, extracts node representations, and makes predictions for downstream tasks.

Algorithm 2 Training algorithm for NeuralSparse

1: **Input:** graph $G = (V, E, \mathbf{A})$, integer l, and training labels Y.
2: **while** stop criterion is not met **do**
3: Generate sparsified subgraphs $\{g_1, g_2, \cdots, g_l\}$ by sparsification network (Section 4.3);
4: Produce prediction $\{\hat{Y}_1, \hat{Y}_2, \cdots, \hat{Y}_l\}$ by feeding $\{g_1, g_2, \cdots, g_l\}$ into GNNs;
5: Calculate loss function J;
6: Update ϕ and θ by descending J
7: **end while**

As the sparsified subgraph samples are differentiable, the two components can be jointly trained using the gradient descent based backpropagation techniques from a supervised loss function, as illustrated in Algorithm 2. While the GNNs have been widely investigated in recent works [64, 45, 117], we focus on the practical implementation for the sparsification network in the remaining of this chapter.

4.3 Sparsification Network

Following the theory discussed above, the goal of the sparsification network is to generate sparsified subgraphs for input graphs, serving as the approximation function $Q_\phi(g \mid G)$. Therefore, we need to answer the following three questions in the sparsification network. **i**). What is $\mathbb{S}_G$ in Equation 4.1, the class of subgraphs we focus on? **ii**). How to sample sparsified subgraphs? **iii**). How to make the sparsified subgraph sampling process differentiable for

the end-to-end training? In the following, we address the questions one by one.

k-neighbor subgraphs. We focus on k-neighbor subgraphs for $\mathbb{S}_G$ [99]: Given an input graph, a k-neighbor subgraph shares the same set of nodes with the input graph, and each node in the subgraph can select no more than k edges from its one-hop neighborhood. Although the concept of the sparsification network is not limited to a specific class of subgraphs, we choose k-neighbor subgraphs for the following reasons.

- We are able to adjust the estimation on the amount of task-relevant graph data by tuning the hyper-parameter k. Intuitively, when k is an under-estimate, the amount of task-relevant graph data accessed by GNNs could be inadequate, leading to inferior performance. When k is an over-estimate, the downstream GNNs may overfit the introduced noise or irrelevant graph data, resulting in sub-optimal performance. It could be difficult to set a golden hyper-parameter that works all the time, but one has the freedom to choose the k that is the best fit for a specific task.

- k-neighbor subgraphs are friendly to parallel computation. As each node selects its edges independently from its neighborhood, we can utilize tensor operations in existing deep learning frameworks, such as tensorflow [1], to speed up the sparsification process for k-neighbor subgraphs.

Sampling k-neighbor subgraphs. Given k and an input graph $G = (V, E, \mathbf{A})$, we obtain a k-neighbor subgraph by repeatedly sampling edges for each node in the original graph. Without loss of generality, we sketch this sampling process by focusing on a specific node u in graph G. Let $\mathbb{N}_u$ be the set of one-hop neighbors of the node u.

1. $v \sim f_\phi(V(u), V(\mathbb{N}_u), \mathbf{A}(u))$, where $f_\phi(\cdot)$ is a function that generates a one-hop neighbor v from the learned distribution based on the node u's attributes, node attributes of u's neighbors $V(\mathbb{N}_u)$, and their edge attributes $\mathbf{A}(u)$. In particular, the learned distribution is encoded by parameters ϕ.

2. Edge $E(u, v)$ is selected for the node u.

3. The above two steps are repeated k times.

Note that the above process performs sampling without replacement. Given a node u, each of its adjacent edges is selected at most once. Moreover, the sampling function $f_\phi(\cdot)$ is shared among nodes; therefore, the number of parameters ϕ is independent of the input graph size.

Making samples differentiable. While conventional methods are able to generate discrete samples [99], these samples are not differentiable such that it is difficult to utilize them to optimize sample generation. To make samples differentiable, we propose a Gumbel-Softmax based multi-layer neural network to implement the sampling function $f_\phi(\cdot)$ discussed above.

To make the discussion self-contained, we briefly discuss the idea of Gumbel-Softmax. Gumbel-Softmax is a reparameterization trick used to generate differentiable discrete samples [53, 77]. Under appropriate hyper-parameter settings, Gumbel-Softmax is able to generate continuous vectors that are as "sharp" as one-hot vectors widely used to encode discrete data.

Without loss of generality, we focus on a specific node u in a graph $G = (V, E, \mathbf{A})$. Let $\mathbb{N}_u$ be the set of one-hop neighbors of the node u. We implement $f_\phi(\cdot)$ as follows.

1. $\forall v \in \mathbb{N}_u$,

$$z_{u,v} = \mathrm{MLP}_\phi(V(u), V(v), \mathbf{A}(u, v)), \tag{4.4}$$

   where MLP_ϕ is a multi-layer neural network with parameters ϕ.

2. $\forall v \in \mathbb{N}_u$, we employ a softmax function to compute the probability to sample the edge,

$$\pi_{u,v} = \frac{\exp(z_{u,v})}{\sum_{w \in \mathbb{N}_u} \exp(z_{u,w})} \tag{4.5}$$

3. Using Gumbel-Softmax, we generate differentiable samples

$$x_{u,v} = \frac{\exp((\log(\pi_{u,v}) + \epsilon_v)/\tau)}{\sum_{w \in \mathbb{N}_u} \exp((\log(\pi_{u,w}) + \epsilon_w)/\tau)} \tag{4.6}$$

where $x_{u,v}$ is a scalar, $\epsilon_v = -\log(-\log(s))$ with s randomly drawn from Uniform$(0, 1)$, and τ is a hyper-parameter called *temperature* which controls the interpolation between the discrete distribution and continuous categorical densities.

Note that when we sample k edges, the computation for $z_{u,v}$ and $\pi_{u,v}$ only needs to be performed once. For the hyper-parameter τ, we discuss how to tune it as follows.

Discussion on temperature tuning. The behavior of Gumbel-Softmax is governed by a hyper-parameter τ called temperature. In general, when τ is small, the Gumbel-Softmax distribution resembles the discrete distribution, which induces strong sparsity; however, small τ also introduces high-variance gradients that block effective backpropagation. A high value of τ cannot produce expected sparsification effect. Following the practice in [53], we adopt the strategy by starting the training with a high temperature and anneal to a small value with a guided schedule.

Sparsification algorithm and its complexity. As shown in Algorithm 3, given hyper-parameter k, the sparsification network visits each node's one-hop neighbors k times. Let m be the total number of edges in the graph. The complexity of sampling subgraphs by the sparsification network is $O(km)$. When k is small in practice, the overall complexity is $O(m)$.

Comparison with multiple related methods. Unlike FastGCN [22], GraphSAINT [152] and DropEdge [96] that incorporate layer-wise node samplers to reduce the complexity of GNNs, NeuralSparse samples subgraphs before applying GNNs. As for the computation complexity, the sparsification in NeuralSparse is more friendly to parallel computation than the layer-conditioned approaches such as AS-GCN. Compared with the graph attentional models [117], the NeuralSparse can produce sparser neighborhoods, which effectively remove task-irrelevant information on original graphs. Unlike LDS [34], NeuralSparse learns graph sparsification under inductive setting, and its graph sampling is constrained by input graph topology.

Algorithm 3 Sampling subgraphs by sparsification network

Input: graph $G = (V, E, \mathbf{A})$ and integer k.
Edge set $\mathbb{H} = \emptyset$
for $u \in \mathbb{V}$ **do**
    for $v \in \mathbb{N}_u$ **do**
        $z_{u,v} \leftarrow \mathrm{MLP}_\phi(V(u), V(v), \mathbf{A}(u, v))$
    end for
    for $v \in \mathbb{N}_u$ **do**
        $\pi_{u,v} \leftarrow \exp(z_{u,v}) / \sum_{w \in \mathbb{N}_u} \exp(z_{u,w})$
    end for
    for $j = 1, \cdots, k$ **do**
        for $v \in \mathbb{N}_u$ **do**
            $x_{u,v} \leftarrow \frac{\exp((\log(\pi_{u,v}) + \epsilon_v)/\tau)}{\sum_{w \in \mathbb{N}_u} \exp((\log(\pi_{u,w}) + \epsilon_w)/\tau)}$
        end for
        Add the edge represented by vector $[x_{u,v}]$ into $\mathbb{H}$
    end for
end for

4.4 Evaluation

In this section, we evaluate our proposed NeuralSparse on the node classification task with both inductive and transductive settings. The experimental results demonstrate that NeuralSparse achieves superior classification performance over state-of-the-art GNN models. Moreover, we provide a case study to demonstrate how the sparsified subgraphs generated by NeuralSparse could improve classification compared against other sparsification baselines.

Table 4.1: Dataset statistics

	Reddit	**PPI**	**Transaction**	**Cora**	**Citeseer**
Task	Inductive	Inductive	Inductive	Transductive	Transductive
Nodes	232,965	56,944	95,544	2,708	3,327
Edges	11,606,919	818,716	963,468	5,429	4,732
Features	602	50	120	1,433	3,703
Classes	41	121	2	7	6
Training Nodes	152,410	44,906	47,772	140	120
Validation Nodes	23,699	6,514	9,554	500	500
Testing Nodes	55,334	5,524	38,218	1,000	1,000

4.4.1 Datasets

We employ five datasets from various domains and conduct the node classification task following the settings as described in Hamilton et al. [45] and Kipf and Welling [64]. The dataset statistics are summarized in Table 4.1.

Inductive datasets. We utilize the Reddit and PPI datasets and follow the same setting in Hamilton et al. [45]. The Reddit dataset contains a post-to-post graph with word vectors as node features. The node labels represent which community Reddit posts belong to. The protein-protein interaction (PPI) dataset contains graphs corresponding to different human tissues. The node features are positional gene sets, motif gene sets, and immunological signatures. The nodes are multi-labeled by gene ontology.

Graphs in the Transaction dataset contains real transactions between organizations in two years, with the first year for training and the second year for validation/testing. Each node represents an organization and each edge indicates a transaction between two organizations. Node attributes are side information about the organizations such as account balance, cash

reserve, etc. On this dataset, the objective is to classify organizations into two categories: *promising* or *others* for investment in the near future. Under the inductive experimental setting, We use the 47,772 organization data of the year 2014 for training and remaining data are hidden from the model. The 9,554 organizations are used for validation and 38,218 for testing. Validation and testing node sets are from the year 2015 and are not connected to the nodes in the training set. Like the PPI dataset, models need to generalize to the unseen graph when testing on the Transaction dataset.

In the inductive setting, models can only access training nodes' attributes, edges, and labels during training. In the PPI and Transaction datasets, the models have to generalize to completely unseen graphs.

Transductive datasets. We use two citation benchmark datasets in Yang et al. [140] and Kipf and Welling [64] with the transductive experimental setting. The citation graphs contain nodes corresponding to documents and edges as citations. Node features are the sparse bag-of-words representations of the documents and node labels indicate the topic class of the documents. In transductive learning, the training methods have access to all node features and edges, with a limited subset of node labels.

4.4.2 Experimental Setup

Baseline models. We incorporate four state-of-the-art methods as the base GNN components, including GCN [64], GraphSAGE [45], GAT [117], and GIN [136]. Besides evaluating the effectiveness and efficiency of NeuralSparse against base GNNs, we leverage three other categories of methods in the experiments: (1) We incorporate the two unsupervised graph sparsification models, the spectral sparsifier [SS, 99] and the Rank Degree [RD, 121]. The input graphs are sparsified before sent to the base GNNs for node classification. (2) We compare against the random layer-wise sampler DropEdge [96]. Similar to the Dropout trick [48], DropEdge randomly removes connections among node neighborhood in each GNN layer. (3) We also incorporate LDS [34], which works under a transductive setting and learns

Table 4.2: Node classification performance

Sparsifier	Method	Reddit	PPI	Transaction	Cora	Citeseer
		Micro-F1	Micro-F1	AUC	Accuracy	Accuracy
N/A	GCN	0.922 ± 0.041	0.532 ± 0.024	0.564 ± 0.018	0.810 ± 0.027	0.694 ± 0.020
	GraphSAGE	0.938 ± 0.029	0.600 ± 0.027	0.574 ± 0.029	0.825 ± 0.033	0.710 ± 0.020
	GAT	-	0.973 ± 0.030	0.616 ± 0.022	0.821 ± 0.043	0.721 ± 0.037
	GIN	0.928 ± 0.022	0.703 ± 0.028	0.607 ± 0.031	0.816 ± 0.020	0.709 ± 0.037
SS/ RD	GCN	0.912 ± 0.022	0.521 ± 0.024	0.562 ± 0.035	0.780 ± 0.045	0.684 ± 0.033
	GraphSAGE	0.907 ± 0.018	0.576 ± 0.022	0.565 ± 0.042	0.806 ± 0.032	0.701 ± 0.027
	GAT	-	0.889 ± 0.034	0.614 ± 0.044	0.807 ± 0.047	0.686 ± 0.034
	GIN	0.901 ± 0.021	0.693 ± 0.019	0.593 ± 0.038	0.785 ± 0.041	0.706 ± 0.043
DropEdge	GCN	0.961 ± 0.040	0.548 ± 0.041	0.591 ± 0.040	0.828 ± 0.035	0.723 ± 0.043
	GraphSAGE	0.963 ± 0.043	0.632 ± 0.031	0.598 ± 0.043	0.821 ± 0.048	0.712 ± 0.032
	GAT	-	0.851 ± 0.030	0.604 ± 0.043	0.789 ± 0.039	0.691 ± 0.039
	GIN	0.931 ± 0.031	0.783 ± 0.037	0.625 ± 0.035	0.818 ± 0.044	0.715 ± 0.039
LDS	GCN	-	-	-	0.831 ± 0.017	0.727 ± 0.021
Neural Sparse	GCN	0.966 ± 0.020	0.651 ± 0.014	0.610 ± 0.022	0.837 ± 0.014	**0.741** ± 0.014
	GraphSAGE	**0.967** ± 0.015	0.696 ± 0.023	0.649 ± 0.018	0.841 ± 0.024	0.736 ± 0.013
	GAT	-	**0.986** ± 0.015	**0.671** ± 0.018	**0.842** ± 0.015	0.736 ± 0.026
	GIN	0.959 ± 0.027	0.892 ± 0.015	0.634 ± 0.023	0.838 ± 0.027	0.738 ± 0.015

Bernoulli variables associated with individual edges.

Temperature tuning. We anneal the temperature with the schedule defined as $\tau = \max(0.05, \exp(-rp))$, where p is the training epoch and $r \in 10^{\{-5,-4,-3,-2,-1\}}$. τ is updated every N steps and $N \in \{50, 100, ..., 500\}$. Compared with the MNIST VAE model in Jang et al. [53], smaller hyper-parameter τ fits NeuralSparse better in practice.

Metrics. We evaluate the performance on the transductive datasets with accuracy [64]. For inductive tasks on the Reddit and PPI datasets, we report micro-averaged F1 scores [45]. Due to the imbalanced classes in the Transaction dataset, models are evaluated with AUC value [51]. The results show the average of 10 runs.

4.4.3 Experimental Results

Classification Performance

Table 4.2 summarizes the classification performance of NeuralSparse and the baseline methods on all datasets. For Reddit, PPI, Transaction, Cora, and Citeseer, the hyper-parameter k is set as 30, 15, 10, 5, and 3 respectively. The hyper-parameter l is set as 1. Note that the result of GAT on Reddit is missing due to the out-of-memory error and LDS only works under the transductive setting. For simplicity, we only report the better performance with SS or RD sparsifiers.

Overall, NeuralSparse is able to help GNN techniques achieve competitive generalization performance with sparsified graph data. We make the following observations. (1) Compared with basic GNN models, NeuralSparse can enhance the generalization performance on node classification tasks by utilizing the sparsified subgraphs from the sparsification network, especially in the inductive setting. Indeed, large neighborhood size in the original graphs could increase the chance of introducing noise into the aggregation operations, leading to sub-optimal performance. (2) With different GNN options, the NeuralSparse can consistently achieve comparable or superior performance, while other sparsification approaches tend to favor a certain GNN structure. (3) Compared with DropEdge, NeuralSparse achieves up to 13% of improvement in terms of accuracy with lower variance. (4) In comparison with the two NeuralSparse variants SS-GNN and RD-GNN, NeuralSparse outperforms because it can effectively leverage the guidance from downstream tasks.

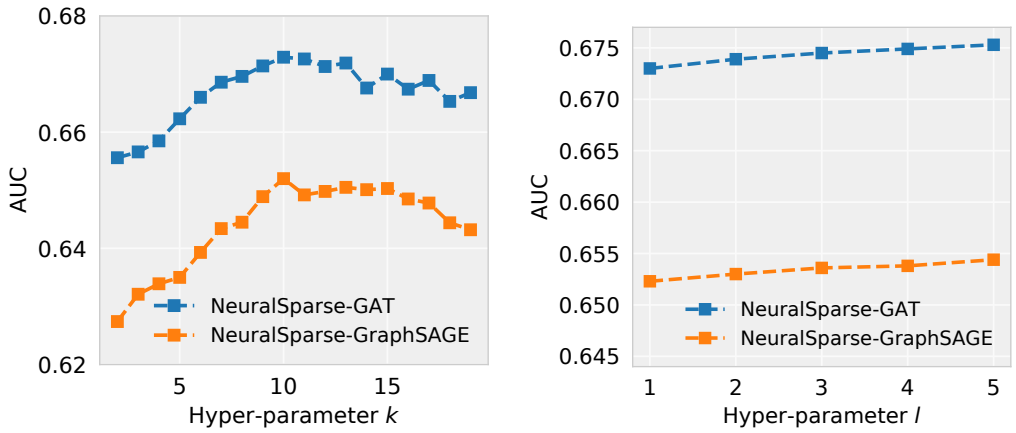
In the following, we discuss the comparison between NeuralSparse and LDS [34] on the Cora and Citeseer datasets. Note that the row labeled with LDS in Table 4.2 presents the classification results on original input graphs. In addition, we adopt κ-nearest neighbor (κ-NN) graphs suggested in [34] for more comprehensive evaluation. In particular, κ-NN graphs are constructed by connecting individual nodes with their top-κ similar neighbors in terms of node features, and κ is selected from $\{10, 20\}$. In Table 4.3, we summarize the

Table 4.3: Node classification performance with κ-NN graphs

Dataset(κ)	**LDS**	**NeuralSparse**
Cora(10)	0.715 $\pm$ 0.035	0.723 $\pm$ 0.025
Cora(20)	0.703 $\pm$ 0.029	0.719 $\pm$ 0.021
Citeseer(10)	0.691 $\pm$ 0.031	0.723 $\pm$ 0.016
Citeseer(20)	0.715 $\pm$ 0.026	0.725 $\pm$ 0.019

classification accuracy of LDS (with GCN) and NeuralSparse (with GCN). On both original and κ-NN graphs, NeuralSparse outperforms LDS in terms of classification accuracy. As each edge is associated with a Bernoulli variables, the large number of parameters for graph sparsification could impact the generalization power in LDS.

Sensitivity to Hyper-parameters and the Sparsified Subgraphs

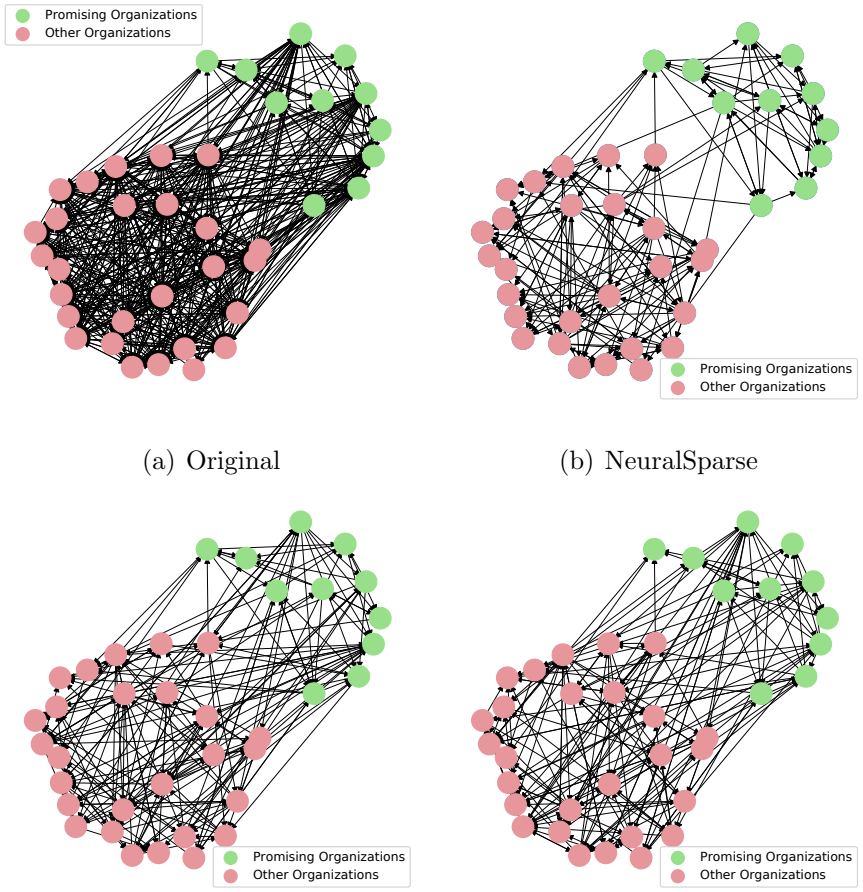


(a) Hyperparameter k

(b) Hyperparameter l

Figure 4.3: Performance vs hyper-parameters

Figure 4.3(a) demonstrates how classification performance responds when k increases on the Transaction dataset. There exists an optimal k that delivers the best classification AUC score. The similar trend on the validation set is also observed. When k is small, NeuralSparse



(a) Original (b) NeuralSparse

(c) Spectral Sparsifier (d) RD Sparsifier

Figure 4.4: Graph visualizations with orginal and sparsified graphs. (a) Original graph from the Transaction dataset and sparsified subgraphs by (b) NeuralSparse, (c) Spectral Sparsifier, and (d) RD Sparsifier.

can only make use of little relevant structural information in feature aggregation, which leads to inferior performance. When k increases, the aggregation convolution involves more complex neighborhood aggregation with a higher chance of overfitting noise data, which negatively impacts the classification performance for unseen testing data. Figure 4.3(b) shows how hyper-parameter l impacts classification performance on the Transaction dataset. When l increases from 1 to 5, we observe a relatively small improvement in classification AUC score. As the parameters in the sparsification network are shared by all edges in the

Table 4.4: Percentage of edges connecting nodes of the same labels

	Reddit	**PPI**	**Transaction**	**Cora**	**Citeseer**
Original	53.1%	55.0%	67.3%	82.2%	73.1%
SS	50.9%	52.8%	62.8%	79.8%	75.6%
RD	49.8%	53.5%	63.4%	84.8%	72.3%
NeuralSparse	59.6%	61.5%	76.8%	93.1%	87.4%

graph, the estimation variance from random sampling could already be mitigated to some extent by a number of sampled edges in a sparsified subgraph. Thus, when we increase the number of sparsified subgraphs, the incremental gain could be small.

In Figure 4.4(a), we present a sample of the graph from the Transaction dataset which consists of 38 nodes (promising organizations and other organizations) with an average node degree 15 and node feature dimension 120. As shown in Figure 4.4(b), the graph sparsified by the NeuralSparse has lower complexity with an average node degree around 5. In Figure 4.4(c, d), we also present the sparsified graphs output by the two baseline methods, SS and RD.

By comparing the four plots in Figure 4.4, we make the following observations: First, the NeuralSparse sparsified graph tends to select edges that connect nodes of identical labels, which favors the downstream classification task. The observed clustering effect could further boost the confidence in decision making. Second, instead of exploring all the neighbors, we can focus on the selected connections/edges, which could make it easier for human experts to perform model interpretation and result visualization.

Convergence Analysis

We analyze the convergence properties of NeuralSparse and DropEdge on Citeseer. The results, as shown in Figure 4.5, demonstrate that NeuralSparse converges faster and achieves better performance than DropEdge.

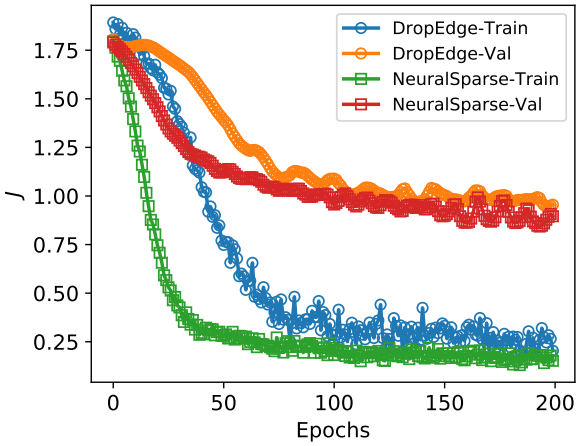


Figure 4.5: Convergence analysis

More Empirical Comparison between NeuralSparse and LDS

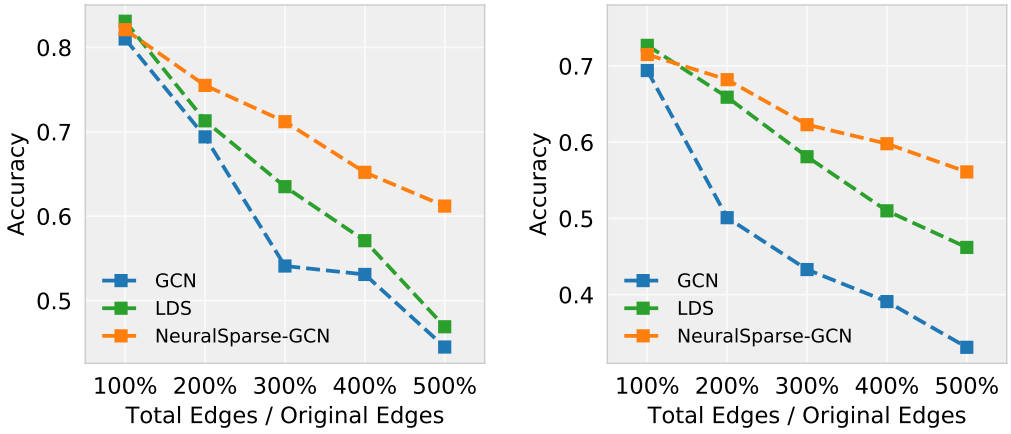
We further compare NeuralSparse and LDS [34] on the node classification tasks where original graph structure is available but more random edges are introduced as noise. Starting from the original graphs, we add edges by randomly sampling two nodes u, v from node set $\mathbb{V}$ and connecting them.

The results are shown in Figure 4.6. In both datasets, NeuralSparse achieves better performance compared with LDS as the noise level goes beyond 200%. When the amount of noise increases, the classification accuracy of LDS drops significantly. This result confirms our conjecture that NeuralSparse is more robust to random edges, compared to LDS.

We compare the NeuralSparse and LDS in the case of complete graphs suggested in [34]. As shown in Table 4.5, we observe that NeuralSparse consistently performs better.

Validation Performance as hyper-parameter k Changes

In this section, we demonstrate how the hyper-parameter k impacts the performance of NeuralSparse-GAT and NeuralSparse-GraphSAGE in both validation and testing on the Transaction dataset. In terms of validation, as shown in Figure 4.7, the validation performance increases when k ranges from 2 to 10 with more available graph data. After k exceeds



(a) Cora (b) Citeseer

Figure 4.6: Node classification performance when adding noise to graph structure.

Table 4.5: Node classification performance with complete graphs

	Cora	Citeseer
GCN	0.580 ± 0.037	0.493 ± 0.026
LDS	0.684 ± 0.029	0.656 ± 0.039
NeuralSparse-GCN	0.691 ± 0.016	0.679 ± 0.033

10, the increase in validation performance slows down and turns to be saturated. In terms of testing performance, it shares a similar trend when k ranges from 2 to 10. Meanwhile, the testing performance drops more after k exceeds 10.

Quantitative Edge Sampling Evaluation

We qualitatively demonstrate the difference by Figure 4.4(a) original graph, (b) NeuralSparse, (c) SS, and (d) RD. In addition, we provide quantitative analysis in Table 4.4, where we report the percentage of edges that connect nodes of same class labels in sparsified graphs. Both qualitative and quantitative results suggest a common trend: NeuralSparse prefers to select neighbors with the same labels compared with the baseline methods.

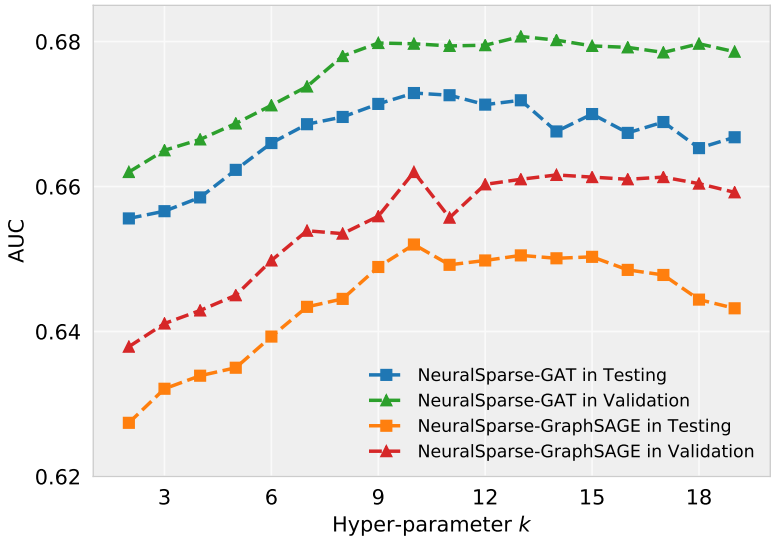


Figure 4.7: Impact from hyper-parameter k on validation and testing on the Transaction dataset

4.5 Summary

In this chapter, we propose Neural Sparsification (NeuralSparse) to address the noise brought by the task-irrelevant information on real-life large graphs. NeuralSparse consists of two major components: (1) The sparsification network sparsifies input graphs by sampling edges following a learned distribution; (2) GNNs take sparsified subgraphs as input and extract node representations for downstream tasks. The two components in NeuralSparse can be jointly trained with supervised loss, gradient descent, and backpropagation techniques. The experimental study on real-life datasets shows that the NeuralSparse consistently renders more robust graph representations, and yields up to 7.2% improvement in accuracy over state-of-the-art GNN models.

CHAPTER 5

Temporal Graph Modeling with Temporal Structural Convolution

5.1 Background

Temporal graphs, as a data structure that carries both temporal and structural information from real-world data, has been widely adopted in applications from various domains, such as online social media [160], biology [135], action recognition [138], and so on. In this chapter, we study the problem of node classification in temporal graphs [135]: Given a set of nodes with rich features and a temporal graph that records historical activities between nodes, the goal is to predict the label of every node. Consider the following application scenario.

Example. In the financial domain, investors are eager to know which companies are promising for investment. As shown in Figure 5.1, companies and their historical transactions naturally form a temporal graph, shown as a sequence of graph snapshots. Each snapshot encodes companies as nodes and transactions as edges within a month. The side information of companies (*e.g.,* industry sector and cash reserve) and transactions (*e.g.,* transaction amount) is represented by the node and edge attributes, respectively. In this task, we aim to predict each company's label: *promising* or *others* for future investment, with interpretable evidence for domain experts.

While node representation lies at the core of this problem, we face two main challenges from temporal graphs.

Temporal graph sparsification. Temporal graphs from real-life applications are large with high complexity. For example, the social graph on Facebook [23] and the financial transaction graph on Venmo [155] are densely connected with average node degrees of 500

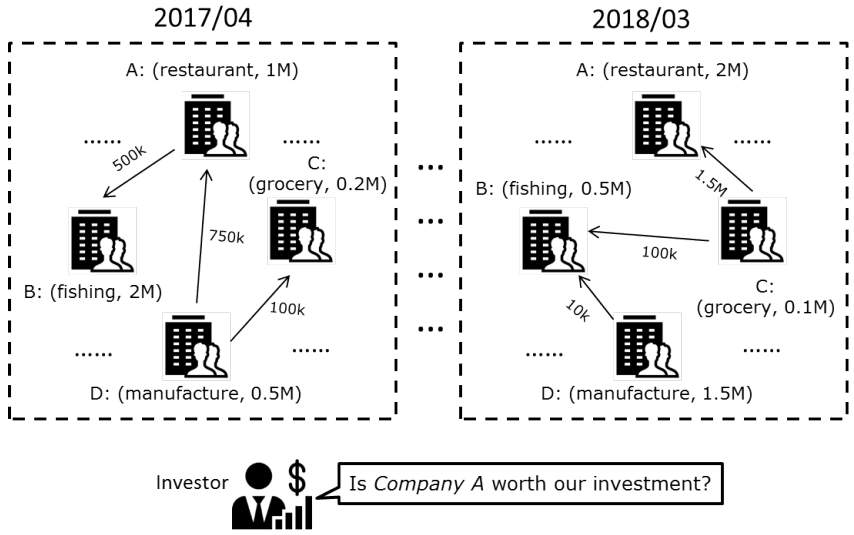


Figure 5.1: An example of node classification in a temporal graph from the financial domain. Nodes are companies, and edges indicate monthly transactions. The goal is to predict which companies are promising for investment in the near future.

and 111, respectively. Such complexity poses a high overfitting risk to existing machine learning techniques [80, 76], and makes it difficult for domain experts to interpret and visualize learned models. While graph sparsification [75] suggests a promising direction to reduce graph complexity, existing methods perform sparsification by sampling subgraphs from predefined distributions [67, 45, 22, 150]. The sparsified graphs may miss important information for classification because the predefined distributions could be irrelevant to subsequent tasks. Several recent efforts [34, 96, 163] strive to utilize supervision signals to remove noise edges and regularize the graph model training. However, the proposed methods are either transductive with difficulty to scale or of high gradient variance bringing increased training difficulty.

Temporal-structural convolution. Local features in the temporal-structural domain are the key to node classification in temporal graphs. Although existing techniques have investigated how to build convolutional operators to automatically learn and extract local features from either temporal domain [7] or structural domain [117], a naïve method that

simply stacks temporal and structural operators could lead to suboptimal performance. An effective method that learns and extracts local features from joint temporal-structural space is still missing.

Our contribution. We propose Temporal Structural Network (TSNet), a deep learning framework that performs supervised node classification in sparsified temporal graphs. TSNet consists of two major sub-networks: *sparsification network* and *temporal-structural convolutional network*.

1. The *sparsification network* aims to sparsify input temporal graphs by sampling edges from the one-hop neighborhood following a distribution that is learned from the subsequent supervised classification tasks.

2. The *temporal-structural convolutional network* takes sparsified temporal graphs as input and extracts local features by performing convolution in nodes' neighborhood defined in joint temporal-structural space.

As both sub-networks are differentiable, we can leverage standard stochastic gradient descent and backpropagation techniques to iteratively learn better parameters to sparsify temporal graphs and extract node representations. Experimental results on both public and private datasets show that TSNet can offer competitive performance on node classification tasks. Using a case study, we demonstrate the potential of TSNet to improve model interpretation and visualization of temporal graphs.

5.2 TSNet Overview

In this section, we start with a theoretical overview of the proposed TSNet.

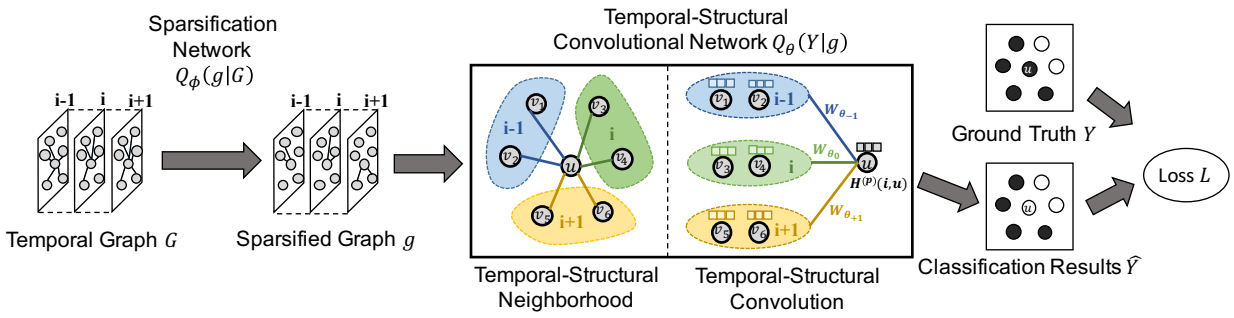


Figure 5.2: The frameworks of TSNet.

We utilize the two-step formulation of node classification problem. The sparsification network takes the temporal graph as input and generates sparsified subgraphs drawn from a learned distribution. The temporal-structural network extracts temporal and structural features simultaneously with the sparsified subgraph as input.

5.2.1 A Two-step Framework

Given input temporal graph G and node label matrix Y, our objective is to learn $P(Y \mid G)$. Current Graph Neural Networks (GNNs) [64, 45, 117] learn node representation by aggregating node neighborhood features. However, in large and complex temporal graphs, node neighborhood tends to be dense with much noise which introduces high overfitting risk to existing approaches. To tackle the challenge, we leverage the two-step framework proposed in [163] to break node classification problem down into two steps: graph sparsification step and representation learning step.

$$P(Y \mid G) \approx \sum_{g \in \mathbb{S}_G} P(Y \mid g) P(g \mid G) \approx \sum_{g \in \mathbb{S}_G} Q_\theta(Y \mid g) Q_\phi(g \mid G) \tag{5.1}$$

where g is a sparsified subgraph, and $\mathbb{S}_G$ is a class of sparsified subgraphs of G. We approximate the distributions by tractable functions Q_θ and Q_ϕ. With reparameterization tricks [42], we could differentiate the graph sparsification step to make efficient backpropagation. In the following, we will introduce our framework to find approximation function $Q_\phi(g \mid G)$ and $Q_\theta(Y \mid g)$.

5.2.2 Architecture

As shown in Figure 5.2, the proposed TSNet consists of two major sub-networks: **sparsification network** and **temporal-structural convolutional network**.

- The **sparsification network** is a multi-layer neural network that implements $Q_\phi(g \mid G)$: Taking temporal graph G as input, it generates a random sparsified subgraph of G drawn from a learned distribution.
- The **temporal-structural convolutional network** implements $Q_\theta(Y \mid g)$ that takes a sparsified subgraph as input, extracts node representations by convolutional filtering on the temporal-structural neighborhood of each node, and makes predictions on node labels.

With differentiable operations in both sub-networks, our TSNet is an end-to-end supervised framework, which is trainable using gradient-based optimization.

5.3 Sparsification network

In this section, we present the sparsification network, which optimizes temporal graph sparsification for subsequent node classification tasks.

5.3.1 Design Goals

The goal of sparsification network is to generate sparsified subgraphs for temporal graphs, serving as the approximation function $Q_\phi(g \mid G)$. Therefore, we need to answer the following three questions in the sparsification network.

1. As the essence of sparsification is to sample a subset of edges, how should we represent each edge so that we can differentiate edges for edge sampling?
2. What is the class of sparsified subgraphs $\mathbb{S}_G$? How to sample such sparsified subgraphs?

3. How to make sparsified subgraphs differentiable for end-to-end training?

5.3.2 Edge Representations

Given a temporal graph $G = (\mathbb{V}, \mathbf{V}, \mathbf{E}, \mathbf{A})$, an expected edge representation could consist of its edge attributes and certain information from the two connected nodes. Let $\mathbb{N}_{u,i}$ be the set of one-hop neighbors with respect to node u's incoming edges at time i. The expected edge representation $\mathbf{X}(i, u, v)$ for the edge from v to u at time i is calculated as follows.

$$\mathbf{X}(i, u, v) = \mathbf{V}'(i, u)||\mathbf{V}'(i, v)||\mathbf{A}(i, u, v) \tag{5.2}$$

where $||$ indicates vector concatenation and $\mathbf{A}(i, u, v)$ denotes edge attributes. $\mathbf{V}'(i, u)$ ($\mathbf{V}'(i, v)$) is the representation of node u (v), which we calculate with mean aggregation [45] to capture both attribute and structural information,

$$\mathbf{V}'(i, u) = \sigma(W_{\phi_1} \cdot \mathbf{V}(i, u)||\text{MEAN}(\mathbf{V}(i, u'), \forall u' \in \mathbb{N}_{u,i})) \tag{5.3}$$

where W_{ϕ_1} is the weights to be learned and σ is a nonlinear activation function.

5.3.3 Sampling Sparsified Subgraphs

We focus on k-neighbor subgraphs for $\mathbb{S}_G$. The concept of k-neighbor subgraph is originally proposed in the context of spectral sparsification for static graphs [99]: Given an input graph, each node of a k-neighbor subgraph can select no more than k edges from its one-hop neighborhood. In this work, we extend the concept of k-neighbor subgraph to temporal graphs: Given a temporal graph G, each node of a k-neighbor subgraph can select no more than k incoming edges *from its one-hop neighborhood in each graph snapshot of* G. Without loss of generality, we sketch this sampling process by focusing on a specific node u in graph snapshot at time i. Let $\mathbb{N}_{u,i}$ be the set of one-hop neighbors with respect to u's incoming edges at time i and the cardinality of $\mathbb{N}_{u,i}$ is d.

1. For $v \in \mathbb{N}_{u,i}$, $r_{uv} = f_{\phi_2}(\mathbf{X}(i, u, v))$, where r_{uv} is a scalar denoting the ranking score of the

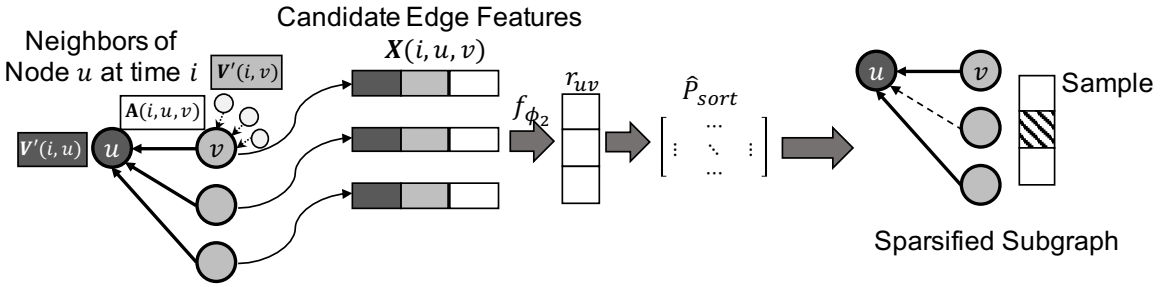


Figure 5.3: An illustration of the proposed sparsification network.
In this example, we focus on the node u at time i with 3 neighbor nodes and set k as 2. Edge representations consist of both edge attributes and node representations. We implement the sparsification by a continuous relaxation of sorting and top-k important incoming edge sampling.

edge from node v to u at time i, and f_{ϕ_2} is a feedforward neural network (parameterized by ϕ_2) that generates the score based on the edge representation $\mathbf{X}(i,u,v)$.

2. We sort the incoming edges based on their ranking scores, and select the top-k edges with the largest ranking scores.

3. The above two steps are repeated for each node in each graph snapshot.

The parameters in f_{ϕ_2} are shared among all nodes in all graph snapshots; therefore, the number of parameters is independent to the size of temporal graphs.

5.3.4 Making Samples Differentiable

The conventional sorting operators are not differentiable such that it is difficult to utilize them for parameter optimization. To make sorting differentiable, we propose to implement the subgraph sampling based on the continuous relaxation of sorting operator [42]. Without loss of generality, we focus on a specific node u at time i in a temporal graph G. We implement the subgraph sampling in Section 5.3.3 as follows.

1. Let $\mathbb{N}_{u,i}$ be the set of one-hop neighbors with respect to u's incoming edges at time i. We apply the reparameterization trick and introduce a fixed source of randomness [42] to the ranking score r_{uv}, $\forall v \in \mathbb{N}_{u,i}$,

$$\pi_{uv} = \log r_{uv} + g_{uv} \tag{5.4}$$

where π_{uv} is a reparameterized scalar indicating the importance of the edge from node v to u. g_{uv} is a sample drawn from Gumbel$(0, 1)$ and $g_{uv} = -\log(-\log(u))$ with $u \sim$ Uniform$(0, 1)$. The reparameterization trick refines the stochastic computational graph for smooth gradient backward pass.

2. We relax the permutation matrix of the edge sorting operator $\hat{P}_{sort} \in \mathbb{R}^{d\times d}$ for node u at time i, and its j-th row is

$$\hat{P}_{sort}(j, :)(\tau) = \text{softmax}[((d + 1 - 2j)\pi_{u:} - A_\pi \mathbf{1})/\tau] \tag{5.5}$$

where d is the cardinality of $\mathbb{N}_{u,i}$ and $\mathbf{1}$ denotes the column vector of all ones. A_π is the matrix of absolute pairwise differences of the elements in $\{v \in \mathbb{N}_{u,i} \mid \pi_{uv}\}$, and the element at x-row and y-column is $A_\pi(x, y) = |\pi_{ux} - \pi_{uy}|$. τ is a hyper-parameter called *temperature* which controls the interpolation between discrete distribution and continuous categorical densities.

3. Before sparsification, the feature tensor of $\mathbb{N}_{u,i}$ is $\mathbf{V}_U(i, u) = \{\mathbf{V}(i, u, u'_1), \ldots, \mathbf{V}(i, u, u'_d)\}$, where $\mathbf{E}(i, u, u'_j) = 1$ and $\mathbf{V}_U(i, u) \in \mathbb{R}^{d\times d_n}$. By applying the relaxed sort operator $\hat{P}_{sort}$ to the unsparsified node features $\mathbf{V}_U(i, u)$, we then select first k rows as the output

$$\mathbf{V}_S(i, u) = [\hat{P}_{sort}\mathbf{V}_U(i, u)](: k, :) \tag{5.6}$$

If $|\mathbb{N}_{u,i}| \leq k$ for node u, we will skip its sparsification and take all in $\mathbf{V}_U(i, u)$.

We sketch the full algorithm of sparsification network in a combinatorial manner in Algorithm 4. Let $\bar{d}$ be the average degree, n be the total number of nodes in a temporal graph, and t be the number of snapshots. The sparsification network visits each node's

Algorithm 4 Sampling subgraphs by sparsification network

Require: Temporal graph $G = (\mathbb{V}, \mathbf{V}, \mathbf{E}, \mathbf{A})$ and integer k.

for $i = 1, \cdots, t$ **do**
  for $u \in \mathbb{V}$ **do**
    if $|\mathbb{N}_{u,i}| > k$ **then**
      for $v \in \mathbb{N}_{u,i}$ **do**
        compute $\mathbf{X}(i, u, v)$ by Equation (5.2)
        compute π_{uv} by Equation (5.4)
      end for
      compute $\hat{P}_{sort}$ by Equation (5.5)
      compute $\mathbf{V}_S(i, u)$ by Equation (5.6)
    end if
  end for
end for

one-hop neighborhood and makes $\bar{d}^2$ calculations. The complexity of sampling subgraphs by the sparsification network is $O(\bar{d}^2 nt)$.

5.4 Temporal-Structural Convolutional Network

As discussed in Section 5.2.1, the goal of the temporal-structural convolutional network (TSCN) is to serve as $Q_\theta(Y \mid g)$: it extracts node representations from the sparsified subgraphs generated by the sparsification network and leverages the vector representations to perform node classification. Inspired by the success of convolutional aggregation in the graph domain [45, 64, 22, 117], the core idea behind the temporal-structural convolutional network is to simultaneously extract local temporal and structural features for node representations by convolutional aggregation in individual nodes' *temporal-structural neighborhood*.

5.4.1 Temporal-structural Neighborhood

Unlike the neighborhood defined in static graphs that only tells "who are close to me", temporal-structural neighborhood stores information about "who and when are close to me". To accomplish this, we extend the notion of neighborhood to the temporal domain by aggregating the structural neighborhood across several preceding and/or subsequent snapshots of any given snapshot. Given a node u at time i, its temporal-structural neighborhood can be represented by a matrix $F_{u,i} \in \mathbb{R}^{t \times n}$, where $F_{u,i}(j, v) = 1$ if node v is in u's (structural) neighborhood at time j; otherwise, $F_{u,i}(j, v) = 0$. In this work, we focus on the *first-order* temporal-structural neighborhood in the sparsified subgraphs. In other words, we have $F_{u,i}(j, v) = 1$ if the following two conditions hold: (1) $|i - j| = 1$, and (2) at time j, there is an incoming edge from node v to u in the sparsified temporal graph. Note that node u at time i is also in its own temporal-structural neighborhood, that is, $F_{u,i}(i, u) = 1$. With the notion of the temporal-structural neighborhood, we are ready to introduce the design of a temporal-structural convolutional layer.

5.4.2 Temporal-structural Convolutional Layer

A temporal-structural convolutional layer performs feature aggregation in individual nodes' temporal-structural neighborhood. One could stack multiple convolutional layers to extract higher-order temporal-structural features.

Without loss of generality, we discuss the technical details of temporal-structural convolutional layer by focusing on a specific node u at time i in the p-th convolutional layer. The input is a temporal graph $G = (\mathbb{V}, \mathbf{V}, \mathbf{E}, \mathbf{A})$, node representations $\mathbf{H}^{(p-1)} \in \mathbb{R}^{t \times n \times d_n^{(p-1)}}$ and a relaxed sort operator $\hat{P}_{sort}$ from Section 5.3.4. With the same sort operator in Equation 5.6 that sparsifies the node features of the first convolution layer, we obtain the sparsified node features of the p-th convolutional layer as

$$\mathbf{V}_U^{(p)}(i, u) = \{\mathbf{H}^{(p-1)}(i, u, u'_1), \ldots, \mathbf{H}^{(p-1)}(i, u, u'_d)\} \tag{5.7}$$

$$\mathbf{V}_S^{(p)}(i,u) = [\hat{P}_{sort}\mathbf{V}_U^{(p)}(i,u)](:k,:) \tag{5.8}$$

The temporal-structural convolution performs as follows.

$$\mathbf{H}^{(p)}(i,u) = \sigma(\sum_{\{(j,v)|F_{u,i}(j,v)=1\}} \mathbf{V}_S^{(p)}(j,u,v)W_{i,u,j,v}^{(p)}) \tag{5.9}$$

where $\sigma(\cdot)$ is a non-linear activation function, $\mathbf{H}^{(p)} \in \mathbb{R}^{t\times n\times d_n^{(p)}}$ is the output node representations, and $W_{i,u,j,v}^{(p)} \in \mathbb{R}^{d_n^{(p-1)}\times d_n^{(p)}}$ is a customized convolution filter generated by

$$W_{i,u,j,v}^{(p)} = \text{MLP}_{\theta_{i-j}^{(p)}}(\mathbf{V}_S^{(p)}(i,u), \mathbf{V}_S^{(p)}(j,v), \mathbf{A}(j,u,v)) \tag{5.10}$$

where $\text{MLP}_{\theta_{i-j}^{(p)}}(\cdot)$ is a multi-layer neural network with parameters $\theta_{i-j}^{(p)}$ that generates customized convolutional filters based on node and edge features. In other words, in the case of first-order temporal-structural neighborhood, we utilize three networks $\text{MLP}_{\theta_{-1}^{(p)}}$, $\text{MLP}_{\theta_0^{(p)}}$, and $\text{MLP}_{\theta_1^{(p)}}$ to model the temporal impacts from the temporal-structural neighborhood. Note that $\{\theta_{-1}^{(p)}, \theta_0^{(p)}, \theta_1^{(p)}\}$ are the only parameters in this convolutional layer and are shared by all nodes; therefore, the number of parameters in a temporal-structural convolutional layer is independent of the number of nodes, edges, or time points in a temporal graph.

As described in Equation 5.9 and 5.10, for a single node, the computational cost is determined by the MLP structure as a fixed number (c). Therefore, the complexity of convolution layer is $O(cdtn)$ and is generally proportional to the number of nodes (n).

5.4.3 Network Architecture

Now we present the full temporal-structural convolutional network.

- **Convolutional layer.** As discussed in Section 5.4.2, one could stack multiple convolutional layers to hierarchically explore high-order temporal-structural neighborhood.
- **Pooling layer.** Let $\mathbf{H}^{(p)} \in \mathbb{R}^{t\times n\times d_n^{(p)}}$ be the output node representations from the p-th convolutional layers. The pooling layer performs another round of aggregation in temporal

domain by $H = \text{Pooling}(\mathbf{H}^{(p)})$, where $H \in \mathbb{R}^{n \times d_n^{(p)}}$. Possible pooling operations include *max*, *average*, and *sum* [45].

- **Output layer**. This layer employs a multi-layer neural network and the final output of TSNet is $\hat{Y} = \text{MLP}_{\theta_o}(H)$, where θ_o denotes the parameters.

- **Objective function**. To handle the estimation variance brought by random sampling, the sparsification network generates l sparsified subgraphs and we optimize the parameters in TSNet by minimizing the average loss from the l samples. In particular, the objective function is formulated as follows.

$$J = \frac{1}{l} \sum_{i=1}^{l} L(Y, \hat{Y}_i) \tag{5.11}$$

The function L is defined by cross entropy loss.

5.5 Evaluation

In this section, we evaluate the performance of TSNet using real-life temporal graph datasets from multiple domains. In particular, we compare TSNet with state-of-the-art techniques in terms of classification accuracy and analyze its sensitivity to to the amount of training labels as well as the hyper-parameters. Moreover, we provide a case study to demonstrate how sparsified subgraphs generated by TSNet could improve visualization. The supplementary material contains more detailed information.

5.5.1 Datasets

We employ four temporal graph datasets from different domains, including collaboration network, online social media, and financial marketing. The dataset statistics are summarized in Table 5.1.

DBLP-3. This temporal graph records co-author relationships between authors in the

Table 5.1: Dataset statistics

	DBLP-3	**DBLP-5**	**Reddit**	**Finance**
# nodes	1,662	5,994	128,858	45,542
# edges	33,808	113,062	29,009,401	661,586
# snapshots / # in training	10/5	10/5	31/16	36/18
time granularity	1 Year	1 Year	1 Day	1 Month
# node attributes	5	1,000	20	5
# edge attributes	1	1	1	2
# classes	3	5	10	2

DBLP computer science bibliography[1] from 2001 to 2010, where nodes represent active computer science researchers and edges denote co-author relationships between authors. There are 10 graph snapshots and each snapshot stores co-author relationships within one year. To generate the attributes for each node in one snapshot, we aggregate titles and abstracts of the corresponding author's papers published in that year into one document, represent this document by the bag-of-words model, and then reduce the dimensionality to 5 by PCA [131]. Otherwise, if the author has no paper in one snapshot, we initialize the node attribute with random initialization. The prediction task of this dataset is to classify authors into three academic areas: *data mining*, *computer vision*, and *computer architectures*. Only the first 5 snapshots are in G_{train}.

DBLP-5. This temporal graph also records co-author relationships from 2001 to 2010. For node attributes, we represent the documents with the bag-of-words feature vectors from the most frequent 1,000 words, aiming to evaluate the model's ability to handle high-dimensional sparse features. On this dataset, we aim to classify authors into five academic

[1]https://www.aminer.cn/citation

areas: *data mining*, *computer vision*, *computer architectures*, *computer networks*, and *theoretical computer science*.

Reddit. Reddit is a large online forum, where users contribute original posts or make comments/upvotes to existing posts. We extract posts and comments in 10 mid-sized subreddits in May 2015 [2]. Following the procedure in [45], we build a post-to-post temporal graph, where nodes are posts, and two posts become connected if they are both commented by at least one identical user. For node attributes, we aggregate the post and comment texts in each day into one document, represent it by the bag-of-words model, and reduce the dimensionality to 20 by PCA. On this dataset, our goal is to classify each post into one of the 10 subreddit categories. Only the first 16 snapshots are in G_{train}. To sample posts in communities, we rank subreddits by their total number of posts and select communities that are ranked [11, 20]. We skip the top 10 subreddits to avoid the skewed class distribution. In this dataset, our goal is to classify each post into one of the 10 subreddit categories. The selected subreddits are *DestinyTheGame, worldnews, soccer, DotA2, AdviceAnimals, WTF, GlobalOffensive, hockey, movies, SquaredCircle*.

Finance. This private dataset contains temporal graphs that record transaction history between companies from April 2014 to March 2017. Each node represents a company and each edge indicates a transaction between two companies. Node attributes are side information about the companies such as industry sector, account balance, cash reserve, etc., which may change from year to year. In this dataset, we aim to classify companies into two categories: *promising* or *unpromising* for investment in the near future. We put the first 18 snapshots in G_{train}.

[2] https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/

5.5.2 Baseline Methods

We implement four categories of baselines: (1) node classification methods for static graphs, (2) stacking structural and temporal feature learning models, (3) temporal graph learning models, and (4) variants of the proposed TSNet models which employ a conventional graph sparsification method followed by the temporal-structural convolutional network.

- **GCN** (Graph Convolutional Networks) [64] is a deep method that supports node classification on static graphs.
- **GraphSAGE** [45] classifies nodes in static graphs by randomly sampling and aggregating features from the local neighborhood.
- **GAT** [117] aggregates node local neighborhood feature with weighted mean and coefficients are learned by attention mechanism.
- **LDS** [34] simultaneously learns the graph structure and the parameters of GCNs by learning Bernoulli variables associated with individual edges.
- **TempCNN-GCN** first extracts temporal features with temporal CNN [7] and then trains GCN model on the static graphs for structural feature extraction. This baseline is fully supervised in both steps.
- **GCN-TempCNN** first extracts structural features by GCN in the individual snapshots, and then makes use of temporal CNN to aggregate structural features in temporal domain.
- **Deepwalk-LSTM** extracts structure features by DeepWalk [86], and then leverages LSTM to learn temporal features.
- **DynamicTriad** [166] is an unsupervised model that learns node embedding for temporal graphs.

Table 5.2: Node classification performance

	DBLP-3		**DBLP-5**		**Reddit**		**Finance**	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
GCN	0.862	0.859	0.679	0.678	0.357	0.290	0.480	0.464
GraphSAGE	0.850	0.847	0.814	0.814	0.399	0.336	0.496	0.448
GAT	0.875	0.863	0.821	0.830	-	-	0.509	0.495
LDS	0.876	0.869	0.797	0.794	-	-	0.499	0.474
TempCNN-GCN	0.851	0.857	0.720	0.710	0.411	0.340	0.532	0.548
GCN-TempCNN	0.676	0.691	0.720	0.711	0.384	0.313	0.440	0.397
DeepWalk-LSTM	0.913	0.913	0.772	0.777	0.370	0.303	0.493	0.446
DynamicTriad	0.753	0.745	0.713	0.717	0.393	0.324	0.419	0.430
STAR	0.908	0.908	0.811	0.815	0.439	0.367	0.541	0.502
TSCN	0.942	0.929	0.850	0.845	0.466	0.406	0.559	0.537
SS-TSCN	0.839	0.835	0.807	0.805	0.418	0.351	0.465	0.445
DE-TSCN	0.875	0.888	0.801	0.792	0.391	0.343	0.538	0.487
TSNet	**0.954**	**0.955**	**0.859**	**0.860**	**0.475**	**0.416**	**0.630**	**0.610**

- **STAR** [135] is a spatio-temporal GRU with dual attention model for node classification in temporal graphs.
- **TSCN** is a variant of TSNet that only uses temporal-structural convolutional network without sparsification network.
- **SS-TSCN** is a variant of TSNet that utilizes a state-of-the-art spectral sparsification technique [99] to generate sparsified subgraphs, and then we train TSCN using the sparsified subgraphs.
- **DE-TSCN** is a variant of TSNet that employs DropEdge [96] as graph sampling method to generate subgraphs.

5.5.3 Experimental Settings

TSNet. We implement the proposed TSNet in tensorflow framework [1] for efficient GPU computation. The hyper-parameter k is searched between 3 and 15 for the optimal per-

formance. For the temporal-structural convolutional network, it starts with two temporal-structural convolutional layers, with an internal single-layer feedforward network to generate convolutional filters. Then the output features pass a max-pooling layer over time and a non-linear layer which produces the logit for label prediction. We employ cross-entropy to formulate the loss function and apply Adam [60] optimizer for training. The learning rate of Adam optimizer is initially set to be $\alpha = 1.0 \times 10^{-3}$. We initial the weight matrices in the proposed TSNet model with Xavier initialization [37].

Baseline methods. For fair comparison, we compare against the baseline methods by tuning hyper-parameters and network structures for their best performance. For GCN, We stack two graph convolutional layers with hidden unit dimensions of 5, 128, 10, and 5, for DBLP-3, DBLP-5, Reddit, and Finance, respectively. In GraphSAGE, we select the max-pooling aggregation with 2-hop neighborhood of sample sizes 25. We fix the number of attention heads for GAT to 8. For GCN-TempCNN, the temporal CNN is implemented by 1-D convolution with kernel size 3 followed by a max-pooling layer. For LSTM, its hidden dimension is set to 3 for all the datasets. For DynamicTriad, we enumerate choices of $\beta_0 \in \{0.01, 0.1, 1, 10\}$ and $\beta_1 \in \{0.01, 0.1, 1, 10\}$ using grid search as suggested in [166]. For spectral sparsifier in SS-TSCN, ϵ is set to 0.4 for all the datasets. For the Rank Degree algorithm [121] in RD-TSCN, we select 1% of nodes as the initial seeds and adopt $\rho \in \{0.1, 0.2, \cdots, 0.8\}$ for the best results.

Temperature tuning. In the sparsification network, the temperature τ in Equation 5.5 controls the smoothness of sort operator relaxation. In general, when τ is small, the relaxed sort operator $\hat{P}_{sort}$ resembles the discrete operator, which induces strong sparsity; however, small τ introduces high variance gradient that blocks effective backpropagation. A high value of τ cannot produce expected sparsification effect. Following the practice in [42, 53], we adopt the strategy by starting the training with a high temperature and anneal to a small value with a guided schedule. In particular, we anneal the temperature with the schedule $\tau = \max(1, 4 \times \exp(-\beta s))$, where s is the training epoch and $\beta \in \{1 \times 10^{-3}, 5 \times 10^{-3}\}$. τ is

updated every 50 epochs during the training.

Dataset split and accuracy metrics. We prepare the training and testing temporal graphs following the similar setting in [45]. We split the snapshots of temporal graphs into G_{train} and G_{test}: test graphs remain unseen during training. We randomly sample 80% nodes in G_{train} as the training nodes and provide their labels to the models. The validation set consists of the other 10% of the nodes in G_{train}. We evaluate the performance of the models with all nodes in G_{test}. By monitoring the model performance on the validation set for each epoch, we execute early stopping if the validation loss does not decrease for 10 consecutive epochs. We evaluate the classification accuracy using macro-F1 and micro-F1 scores[3]. The results show the average of 10 runs with random initializations.

5.5.4 Experimental Results

Classification accuracy

Table 5.2 summarizes the classification performance of TSNet and the baseline methods on all datasets. For DBLP-3, DBLP-5, Reddit, and Finance, the hyper-parameter k is set as 8, 9, 10, and 5, respectively. The hyper-parameter l is set as 1 in this experiment. Note some results on Reddit is missing due to the out-of-memory error.

Overall, TSNet consistently outperforms all of the baseline methods in terms of macro-F1 and micro-F1 over all of the datasets. We make the following observations. (1) Compared with the deep learning techniques for static graphs, including GCN, GraphSAGE and GAT, TSNet achieves better performance by effectively utilizing temporal features in graphs. (2) In GCN-TempCNN and Deepwalk-LSTM, structural and temporal features are extracted from separate components: temporal features are extracted based on the structural features obtained from graph convolution [64] or skip-gram [86]. Because of the inter-component dependency, it becomes harder to adjust the parameters in structural feature learning than

[3]http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

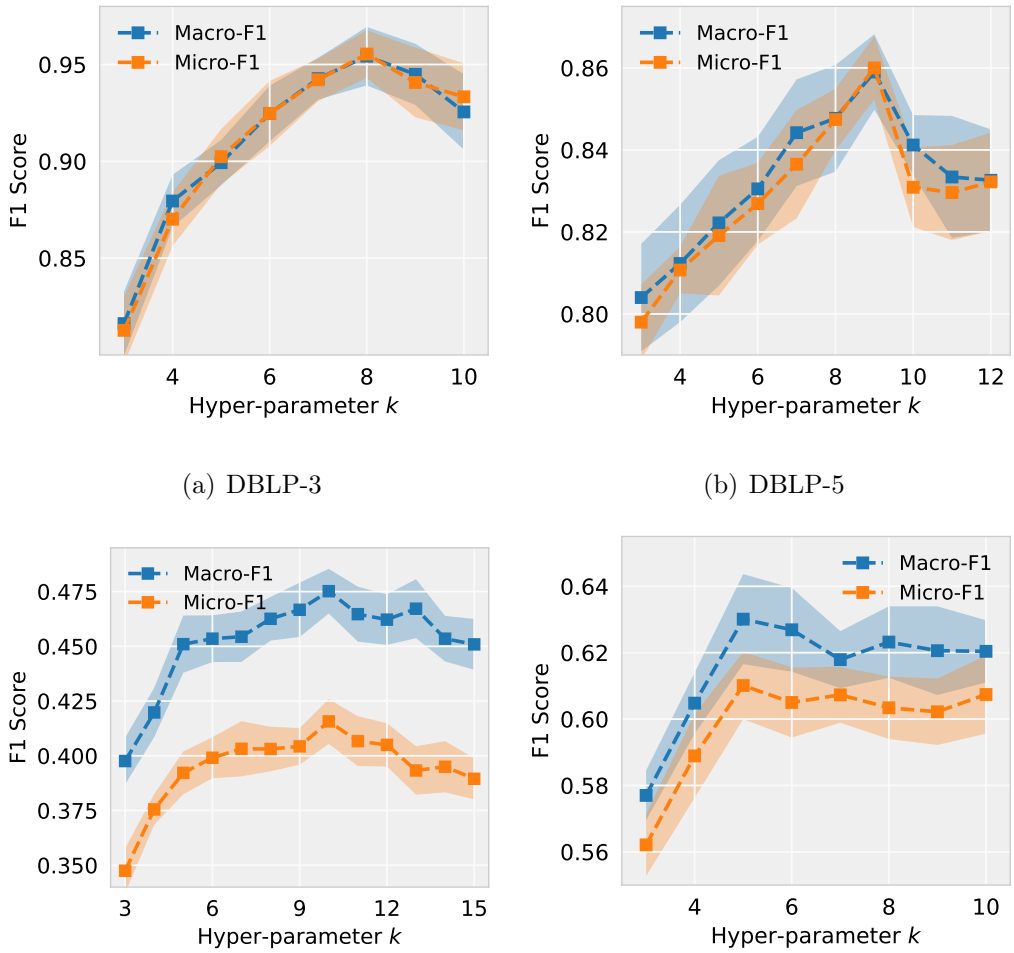
in temporal feature learning. Similarly, in TempCNN-GCN, parameters in temporal feature learning are harder to get trained. With temporal-structural convolution, TSNet can simultaneously adjust parameters for temporal and structural feature learning, and generate more effective features for better classification accuracy. (3) In comparison with the two TSNet variants SS-TSCN and RD-TSCN, TSNet outperforms because of the automatically learned graph sparsification, which highlights the importance of sparsification network in TSNet. (4) The comparison with TSCN is interesting: using the sparsified subgraphs from the sparsification network, it is easier to make TSCN generalized to unseen testing data with improved classification performance. Indeed, large neighborhood size in the original temporal graphs significantly increases the complexity of the convolutional operations, which brings a higher risk of overfitting. (5) Compared with DynamicTriad, TSNet is more effective in node representations customized by the end-to-end supervised training.

Sensitivity to hyper-parameters

In this section, we investigate how hyper-parameter k (*i.e.,* number of sampled edges per node) and l (*i.e.,* number of sparsified subgraphs) impact the classification accuracy in TSNet.

Figure 5.4 demonstrates how classification accuracy responds when k increases from 3. Over the four datasets, we observe a common phenomenon: there exists an optimal k that delivers the best classification accuracy. When k is small, TSNet can only make use of little structural information in temporal-structural convolution, which leads to suboptimal performance. When k gets larger, the temporal-structural convolution involves more complex neighborhood aggregation with higher overfitting risk, which negatively impacts the classification performance for unseen testing data. By comparing across datasets, we observe that the optimal k is associated with the average node degrees of the temporal graphs: higher k in dense Reddit graph and lower k in sparse Finance graph.

Figure 5.5 shows how hyper-parameter l impacts classification accuracy on DBLP-3 and DBLP-5. When l increases from 1 to 5, we observe a relatively small improvement in



(a) DBLP-3

(b) DBLP-5

(c) Reddit

(d) Finance

Figure 5.4: Accuracy vs hyper-parameter k

classification accuracy. As the parameters in the sparsification network are shared by all edges in the temporal graphs, the estimation variance from random sampling could already be mitigated to some extent by a number of sampled edges in a sparsified subgraph. Thus, when we increase the number of sparsified subgraphs, the incremental gain could be small.

Case Study

In this section, we present a case study to demonstrate the potential of TSNet in enhancing model interpretation and visualization. As proof of concept, we focus on a specific node

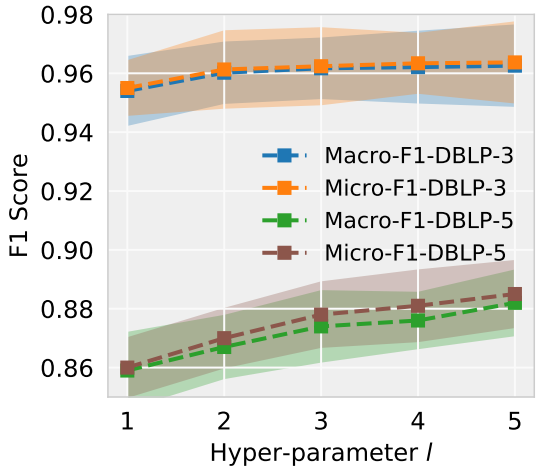


Figure 5.5: Accuracy vs hyper-parameter l on DBLP-3 and DBLP-5

in the DBLP-5 temporal graph. Figure 5.6 visualizes the one-hop neighborhood of *Thomas S. Huang* with k set as 7, and we only present snapshots from 2005, 2007, 2008, and 2010 due to space limitation. In the figure, node colors indicate node labels from the ground truth, solid red edges are ones selected by the sparsification network, and the remaining dotted edges also appear in the original temporal graph.

We make the following observations from Figure 5.6. First, the central author is from the area of *computer vision*, and all selected edges also connect to authors from *computer vision*. This selection seems reasonable from the perspective of feature learning. When neighbors share identical labels consistently over time, temporal-structural features could boost the confidence of making classification decision. Second, instead of exploring all the neighbors, we can only focus on a subset of selected neighbors, which could make it easier for human experts to conduct the effort on model interpretation and result visualization.

Implementations of TSNet

We implement the proposed TSNet in tensorflow framework for efficient GPU computation. In particular, the multi-layer neural network (Equation 5.4) in the sparsification network is implemented by two-layer feed-forward neural networks in all experiment, where

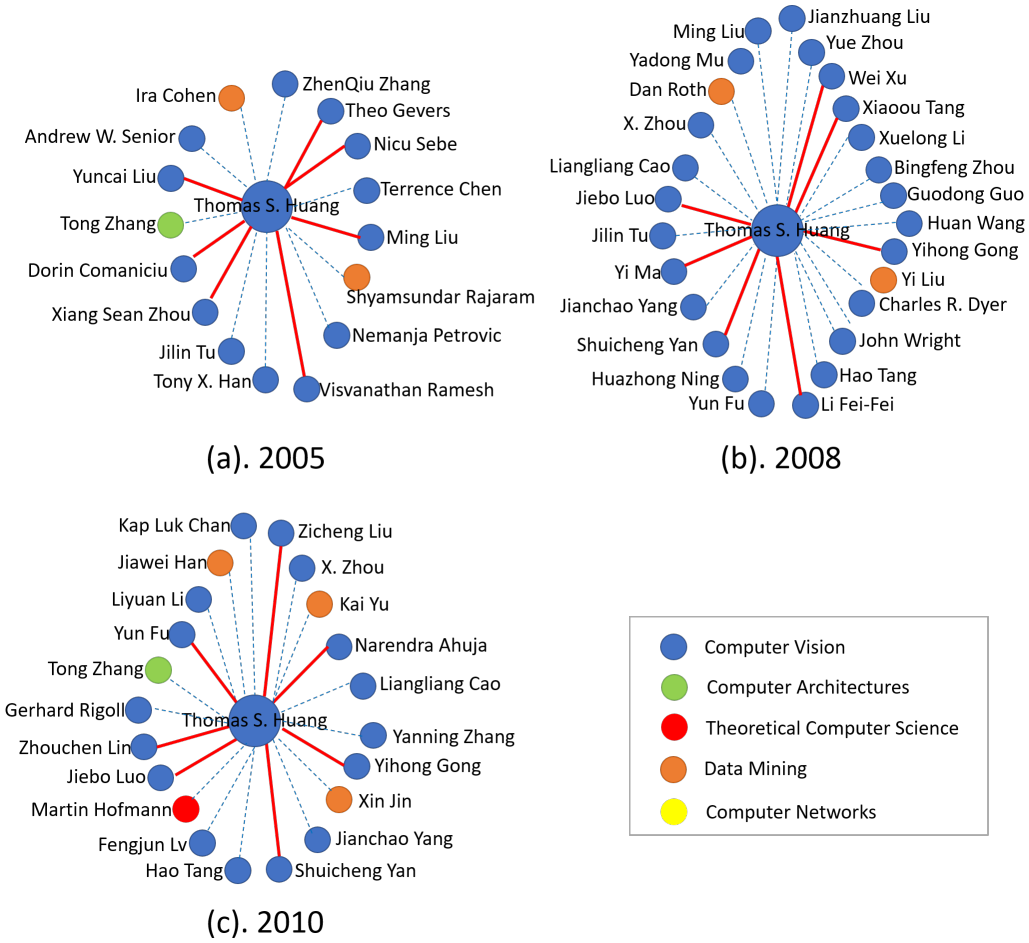


Figure 5.6: One-hop neighborhood of *Thomas S. Huang* in DBLP-5.
The visualization presents snapshots from (a) 2005, (b) 2007, (c) 2008, and (d) 2010. Node colors indicate node labels from ground truth. Sparsification network selects edges with solid red lines. The sparsified graph supports downstream classification as well as model interpretation.

the hyper-parameter k is searched between 3 and 15 for the optimal performance. For the temporal-structural convolutional network, it starts with two temporal-structural convolutional layers, with an internal single-layer feedforward network to generate convolutional filters. Then the output features pass a max-pooling layer over time and a non-linear layer which produces the logit for label prediction. We employ cross-entropy to formulate the loss

function and apply Adam optimizer for training. The learning rate of Adam optimizer is initially set to be $\alpha = 1.0 \times 10^{-3}$. We initial the weight matrices in the proposed TSNet model with Xavier initialization.

In the following, we detail the network structures of TSNet used on individual datasets. FC(a, b, f) means a fully-connected layer with a input neurons and b output neurons activated by function f (none means no activation function is used). TSC(a, b, c, $d \times e$, f) means a temporal-structural convolutional layer with input dimension a, output dimension b, c input neurons for the convolution generator, convolution filter size $d \times e$ and activation function f.

DBLP-3 The sparsification network runs with: FC(10, 5, ReLU)-FC(5, 1, Gumbel-Softmax). The structure of TSCN sub-network is TSC(5, 5, 10, 5×5, ReLU)-TSC(5, 3, 10, 5×3, ReLU)-MaxPool-FC(3, 3, softmax).

DBLP-5 The sparsification network runs with: FC(2000, 128, ReLU)-FC(128, 1, Gumbel-Softmax). The structure of TSCN sub-network is TSC(1000, 10, 2000, 1000×10, ReLU)-TSC(10, 5, 20, 10×5, ReLU)-MaxPool-FC(5, 5, softmax).

Reddit The sparsification network runs with: FC(40, 10, ReLU)-FC(10, 1, Gumbel-Softmax). The structure of TSCN sub-network is TSC(20, 10, 40, 20×10, ReLU)-TSC(10, 10, 20, 10×10, ReLU)-MaxPool-FC(10, 10, softmax).

Finance The sparsification network runs with: FC(11, 5, ReLU)-FC(5, 1, Gumbel-Softmax). The structure of TSCN sub-network is TSC(5, 5, 11, 5×5, ReLU)-TSC(5, 2, 11, 5×2, ReLU)-MaxPool-FC(2, 2, softmax). Note that there is one-dimensional edge attribute in this dataset.

Temperature tuning In the sparsification network, the temperature τ in Equation 5.5 controls the smoothness of sort operator relaxation. In general, when τ is small, the relaxed sort operator $\hat{P}_{sort}$ resembles the discrete operator, which induces strong sparsity; however, small τ introduces high variance gradient that blocks effective backpropagation. A high

value of τ cannot produce expected sparsification effect. Following the practice in [42], we adopt the strategy by starting the training with a high temperature and anneal to a small value with a guided schedule. In particular, we anneal the temperature with the schedule $\tau = \max(1, 4 \times \exp(-\beta s))$, where s is the training epoch and $\beta \in \{1 \times 10^{-3}, 5 \times 10^{-3}\}$. τ is updated every 50 epochs during the training.

Sensitivity to ratio of nodes in training set

In this set of experiments, we study how the ratio of nodes in training set impacts the classification performance of TSNet over all datasets. As shown in Figure 5.7, we vary the ratio of nodes in the training set from 10% to 90% of each dataset and plot the average of F-1 scores. TSNet and its baseline methods share a common trend: classification accuracy increases with more labels in the training process. Compared with the baselines, TSNet consistently provides the most robust classification performance, even when the training ratio is small, which means TSNet is more useful in real practice with sparse labels in temporal graphs.

5.6 Summary

In this chapter, we propose Temporal Structural Network (TSNet) for node classification in temporal graphs. TSNet consists of two major sub-networks: (1) the sparsification network sparsifies input temporal graphs by sorting and sampling edges following a learned distribution; (2) the temporal-structural convolutional network performs convolution on the sparsified graphs to extract local features from the joint temporal-structural space. As an end-to-end model, the two sub-networks in TSNet are trained jointly and iteratively with supervised loss, gradient descent, and backpropagation techniques. In the experimental study, TSNet demonstrates superior performance over four categories of baseline models on public and private benchmark datasets. The qualitative case study suggests a promising direction for the interpretability of temporal graph learning.

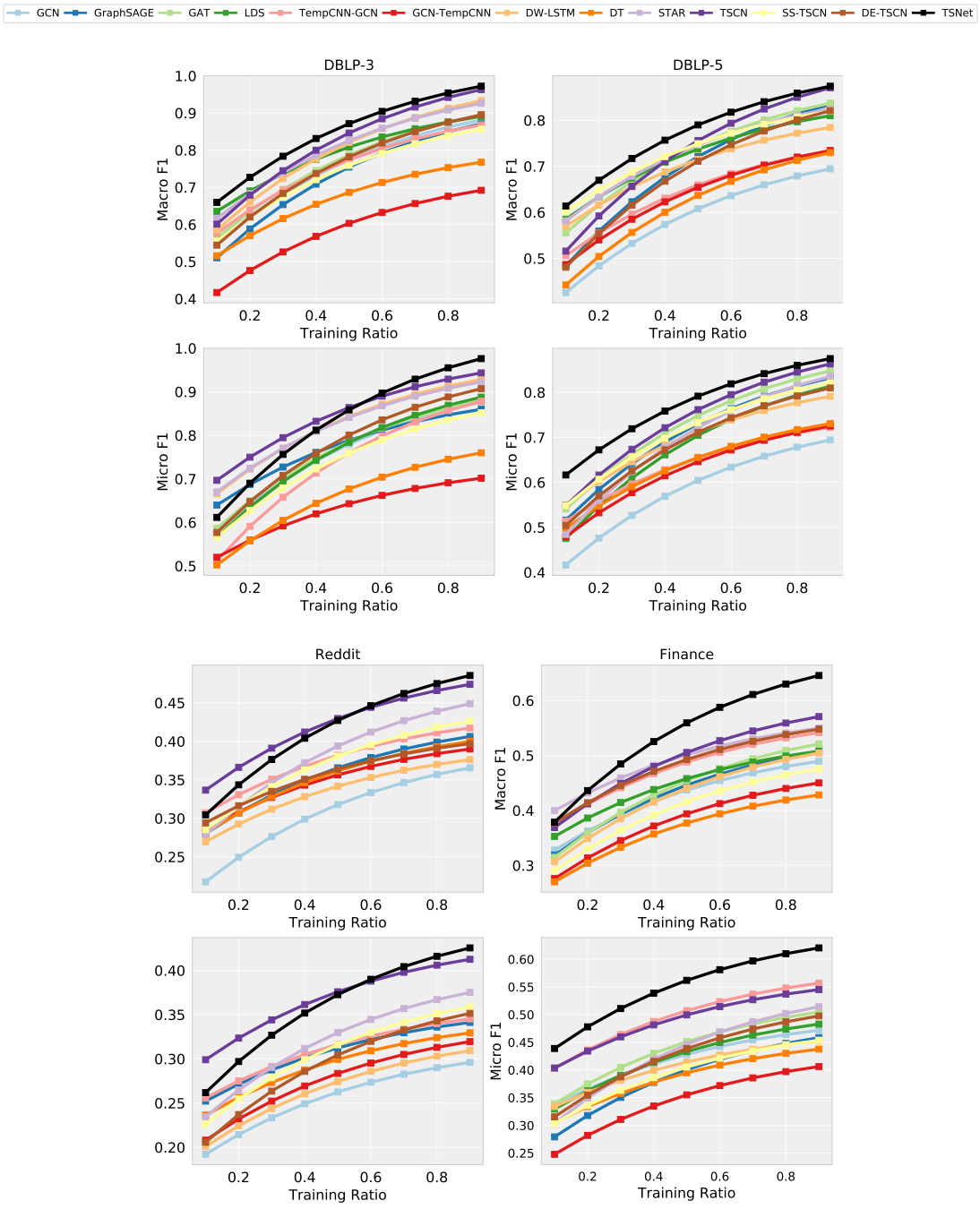


Figure 5.7: Classification accuracy vs ratio of nodes in training set.

CHAPTER 6

Applications in Social Network Analysis

6.1 Social Media User Geolocation via Hybrid Attention

6.1.1 Introduction

Nowadays, social media has become one of the most powerful tools for myriad real-world applications, such as online marketing [61, 161] and event detection [127]. To facilitate those applications, the geolocations of social media users are usually required. For example, online marketing needs to decide the target audience based on their locations. A real-world event may be only related to the users within a certain geographical region. However, there are only a limited amount of social media posts annotated with posting geolocations because position sensors and services can be unavailable or prevented. In addition, most of the social media users also do not denote their locations in the user profiles due to the data privacy issue. Hence, it is important to identify user geolocations with only their behaviors on social media.

One of the most intuitive approaches for user geolocation is to analyze the natural languages utilized in social media posts. Users can mention specific entities or events related to geolocations and people living in a certain region may reveal noticeable habits or patterns in their languages. For example, Rahimi et al. [89] extract bag-of-words features from user posts; Wing and Baldridge [130] estimate the word distributions for different regions; Han et al. [47] conduct feature selection to discover location indicative words. However, user language usage sometimes can be too vague and ambiguous to recognize their locations, especially in social media posts with only limited and noisy texts. Several less active users that rarely publish posts may also have insufficient data for geolocation.

In addition to social media posts published by users, social interactions with other users can also be applied to user geolocation. More precisely, a user can be more likely to reach out to the users living in closer areas. For instance, Davis Jr et al. [26] and Jurgens [55] exploit label propagation and rely on the location redundancy through user relationships. Wang et al. [123] derive node embeddings of social networks and location networks as features for user geolocation. Nevertheless, the sparsity of social networks can still lead to unsatisfactory performance. Social connections can be relationships with users living in other locations. Although previous studies utilize text frequency in social networks [90] and train machine learning models with heterogeneous features [29], conventional approaches are significantly affected by the network structures. Moreover, the importance of social networks can be distinct across different users.

In this section, the framework, Hybrid-attentive User Geolocation (HUG), is proposed to tackle the above issues. Social media posts of each user are first encoded by a hierarchical language attention network. The social network of users is modeled by a graph attention network so that the relations between users can be leveraged in representation learning. Finally, the hybrid attention mechanism is applied to automatically decide the individual importance scores of user posts and the social network for each user, thereby identifying her geolocation. To improve the prediction performance of tail locations, we also propose a novel location regularizer that leverages the knowledge from other locations. In the end, we conduct extensive experiments to show the effectiveness of HUG with in-depth analysis. We also demonstrate the interpretability of HUG with several concrete examples.

In the literature, social media user geolocation has attracted increasing attention in recent years. Some of the conventional methods focus on modeling social media posts [130, 61, 89], while several studies rely on social network information [6, 26, 55, 123, 90]. Although some approaches [29, 90, 50] simultaneously consider language and network knowledge, texts and social networks are individually and evenly modeled without considering distinct importance for different users. To the best of our knowledge, this work is the first study that dynamically

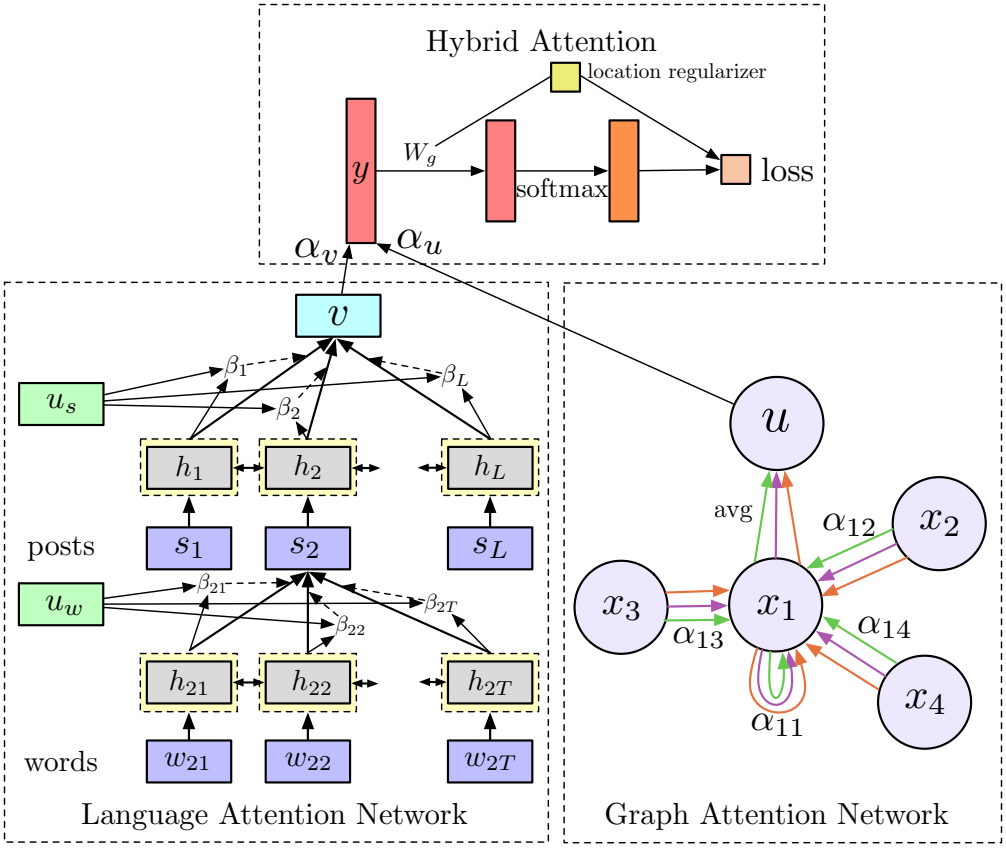


Figure 6.1: The overview of the proposed framework, HUG.

adjusts the influence of different resources upon the model for social media user geolocation.

6.1.2 Hybrid-attentive User Geolocation

In this section, we first formally define the task of social media user geolocation and then introduce our proposed approach, Hybrid-attentive User Geolocation (HUG).

Problem Statement. Suppose we have a set of social media users U and their social network G. For each user $k \in U$, $W^k = \{w_{ij}^k\}$ denotes the social media posts published by the user, where w_{ij}^k represents the j-th word of the i-th post in W^k. The social network $G = (U, A)$ treats each user $k \in U$ as a node and models their relations with an edge set $A \subseteq U \times U$ that indicates the relations between users. Given the social media posts W^k of

a user k and the social network G, the goal of this work is to predict the geolocation of the user $L_k \in \mathcal{L}$, where $\mathcal{L}$ is a set of candidate locations.

6.1.2.1 Multi-head Graph Attention Network

To learn the structural knowledge from social networks, we employ the graph attention network [117] to derive node representations as user graph vectors.

Node Features. For each user k, the input node features x_k^{in} are the node attributes such as user profiles or bag-of-words features. We use a fully-connected layer to learn the hidden node features as $x_k^0 = \mathcal{F}(W^0 x_k^{in})$, where W^0 is the weight matrix and $\mathcal{F}(\cdot)$ is a nonlinear activation function.

Multi-head Graph Attention Layer. The graph attention network consists of several stacked graph attention layers passing node features x_k^i on different levels. For the i-th layer, the importance score s_{jk}^i of each edge $a_{jk} \in A$ between the users j and k can be estimated by the self-attention mechanism [165, 160] as $s_{jk}^i = \langle \mathcal{Q}^i x_j^{i-1}, \mathcal{Q}^i x_k^{i-1} \rangle$, where $\mathcal{Q}^i$ is the weight matrix applied to every node. The node features in the i-th layer x_k^i can then be obtained as:

$$x_k^i = \sigma\left(\sum_{j \in \mathcal{N}(k)} \alpha_{jk}^i \mathcal{W}^i x_j^{i-1}\right), \alpha_{jk}^i = \frac{\exp(s_{jk}^i)}{\sum_{j' \in \mathcal{N}(k)} \exp(s_{j'k}^i)}$$

where $\mathcal{N}(k)$ indicates the neighbors of the user k in the social network; $\mathcal{W}^i$ is a weight matrix for feature projection; $\sigma(\cdot)$ is a nonlinear activation function. Specifically, we utilize the multi-head attention mechanism to have a greater capability of modeling structural knowledge by concatenating the features generated by different weight matrices $\mathcal{Q}_z^i$ and $\mathcal{W}_z^i$, where $1 \leq z \leq H$; H is the number of heads. Finally, in the last layer, we average the multi-head features and delay the employment of the nonlinear activation to derive the user graph vector u_k as:

$$u_k = \sigma\left(\frac{1}{H}\sum_{z=1}^{H}\sum_{j \in \mathcal{N}(k)} \alpha_{jk,z}^N \mathcal{W}_z^i x_j^{N-1}\right),$$

where N is the number of graph attention layers.

6.1.2.2 Language Attention Network

We use a hierarchical language attention network [141] to encode the textual features for each user. The language attention model is composed of several parts, including a word embedding layer, a post encoder, and a user encoder.

Word Embedding Layer. We convert each word w_{ij} into a one-hot encoding representation $\tilde{w}_{ij}$ and embed the words to vectors e with an embedding matrix E, where $e_{ij} = E \cdot \tilde{w}_{ij}$.

Post Encoder. For each post of a user, we feed the word embeddings to a bidirectional Recurrent Neural Network (BiRNN) to learn a hidden state of each word with sequential information as:

$$\overleftarrow{h}_{ij} = \text{GRU}(\overleftarrow{h}_{i,j+1}, e_{ij}), \overrightarrow{h}_{ij} = \text{GRU}(\overrightarrow{h}_{i,j-1}, e_{ij}), h_{ij} = [\overleftarrow{h}_{ij}, \overrightarrow{h}_{ij}],$$

where $\text{GRU}(\cdot)$ is the recurrent neural unit of the BiRNN. Here we choose GRU instead of LSTM because of its computational efficiency. To derive the post representation, we introduce an attention layer to obtain a weighted sum of the hidden states from the BiRNN layer. To be specific, we initialize a context vector u_w and calculate the attention scores β_{ij} for the words in the post as:

$$u_{ij} = \tanh(W_w \cdot h_{ij} + b_w), \beta_{ij} = \frac{exp(u_{ij}^T \cdot u_w)}{\sum_j exp(u_{ij}^T \cdot u_w)}, s_i = \sum_j \beta_{ij} \cdot h_{ij},$$

where W_w and b_w are the weight matrix and the bias to map each word into a hidden space for estimating importance.

User Encoder. Similarly, we employ a BiRNN model using GRU units to derive the hidden representations h_i for the posts of each user according to their published times as:

$$\overleftarrow{h}_i = \text{GRU}(\overleftarrow{h}_{i+1}, s_i), \overrightarrow{h}_i = \text{GRU}(\overrightarrow{h}_{i-1}, s_i), h_i = [\overleftarrow{h}_i, \overrightarrow{h}_i].$$

The other context vector u_s is then learned to estimate the importance score β_i for each post

and aggregate the post representations h_i to form a user language vector v as follows:

$$u_i = \tanh(W_u \cdot h_i + b_u), \beta_i = \frac{exp(u_i^T \cdot u_s)}{\sum_j exp(u_i^T \cdot u_s)}, v = \sum_i \beta_i \cdot h_i,$$

where W_u and b_u are the weight matrix and the bias.

6.1.2.3 Hybrid Attention

To dynamically adjust the importance of two resources for a certain user, we propose the hybrid attention to jointly model texts and social networks. Precisely, a context vector c_h is applied to estimate the importance scores of graph and language user vectors as:

$$\alpha_v = \frac{\exp(o_v \cdot c_h)}{\exp(o_v \cdot c_h) + \exp(o_u \cdot c_h)}, \alpha_u = \frac{\exp(o_u \cdot c_h)}{\exp(o_v \cdot c_h) + \exp(o_u \cdot c_h)},$$

where $o_v = \tanh(W_h \cdot v + b_h)$; $o_u = \tanh(W_h \cdot u + b_h)$; W_h and b_h are the weight matrix and the bias for a nonlinear projection. Therefore, the ultimate feature vector produced by the hybrid attention can be obtained as $y = \alpha_v \cdot v + \alpha_u \cdot u$.

6.1.2.4 Location-regularized User Geolocation

User Geolocation. Based on the feature vector y, we use a fully-connected hidden layer without a bias to estimate the probability of being the geolocation of the user k for each location i as:

$$P(L_k = i) = \text{Softmax}(W_g y),$$

where W_g is the weight matrix for the hidden layer.

Location-based Regularization. To leverage the knowledge across different locations, we regularize the model weights W_g for inferring location probabilities by the corresponding distances. Specifically, we have the location-based regularization loss Loss_R as:

$$\text{Loss}_R = \sum_{j \in \mathcal{L}} \sum_{k \in \mathcal{L}-j} \frac{|W_g(j) - W_g(k)|^2}{D(j,k)},$$

where $D(j,k)$ denotes the distance between the locations j and k.

Learning and Optimization. Finally, the loss function of *HUG* for optimization can be derived by the cross-entropy loss for classification and the location-based regularization loss as:

$$\text{Loss} = \sum_i \mathbb{1}[L_k = i] \cdot P(L_k = i) + \gamma \cdot \text{Loss}_R,$$

where γ is the weight of regularization loss.

6.1.3 Experiments

6.1.3.1 Experimental Setup

Datasets. We employ three public Twitter user geolocation datasets: (1) GeoText [32], (2) Twitter-US [95] and (3) Twitter-World [47]. The datasets are pre-partitioned into training, development and test sets. In each dataset, user tweets are concatenated into single documents. The social graphs are extracted with the mention relations between users, where two users are connected if one mentions the other, or they co-mention a third user. The node attributes in the social graphs are the bag-of-words and TFIDF features. The labels are the discretized geographical coordinates of the training users using a k-d tree [95]. Dataset statistics are summarized in Table 6.1.

Baselines. We compare the proposed HUG against 6 baselines that are trained based on the text and network features to determine user geolocations, including: (1) MLP + k-d tree [89], a text-based multilayer perceptron model; (2) GCN-LP [90], a network-based model using one-hot neighbor encoding as the node attributes; (3) MENET [29], a multiview neural network model that utilize multi-entry data to infer users' locations; (4) MLP-TXT+NET [90], a multilayer perceptron model with the concatenation of text features and adjacent lists as input; (5) GCN [90], a graph convolution network model [64] with the bag-of-words as node attributes; (6) HLPNN [50], a feature fusion model with city and country objectives.

Evaluation. We evaluate the models with three commonly used metrics: (1) **Acc@161**,

Table 6.1: Dataset statistics.

	GeoText	Twitter-US	Twitter-World
Users	9,475	449,200	1,386,766
Classes	129	256	930
Train	5,685	429,200	1,366,766
Dev	1,895	10,000	10,000
Test	1,895	10,000	10,000

the accuracy of predicting a user within 161km or 100 miles from the labeled location; (2) **Mean**, the mean error between the predicted and labeled location; (3) **Median**, the median error between the predicted and labeled location.

Implementation Details. We implement the proposed HUG in PyTorch framework for efficient GPU computation. The language attention network has bidirectional GRUs with hidden dimensions in $\{50, 100, 200\}$ and the word embeddings are initialized with the Glove vectors [85] pre-trained on the Twitter corpus. The entity-level aggregation network has two layers with the hidden dimension $D_e \in \{64, 128, 256\}$. The number of heads in multi-head graph attention is searched in $\{1, 2, 4, 8, 16\}$. We apply Adam optimizer for training and the initial learning rate is set as 5.0×10^{-4}. The activation functions are ELU [24].

6.1.3.2 Experimental Results

Table 6.2 summarizes the model performance of Twitter user geolocation prediction on all datasets. Overall, HUG is able to outperform other baselines across the three datasets on all metrics. We make the following observations. (1) Compared with MLP + k-d tree [89] and GCN-LP [90] that only utilizes a single source of data, e.g. text or network features, HUG outperforms by simultaneously learning the important language features and network

Table 6.2: Twitter user geolocation prediction performance.

	GeoText			Twitter-US			Twitter-World		
	Acc@161 ↑	Mean ↓	Median ↓	Acc@161 ↑	Mean ↓	Median ↓	Acc@161 ↑	Mean ↓	Median ↓
MLP + k-d tree	38%	844	389	54%	554	120	34%	1456	415
GCN-LP	58%	576	56	53%	653	126	45%	2357	279
MENET	62%	532	32	66%	433	45	53%	1044	118
MLP-TXT+NET	58%	554	58	66%	420	56	58%	1030	53
GCN	60%	546	45	62%	485	71	54%	1130	108
HLPNN	-	-	-	71%	362	32	59%	828	60
HUG	**64%**	**516**	**30**	**72%**	**359**	**31**	**62%**	**818**	**49**

structure features. (2) As the feature fusion models, MENET [29], MLP-TXT+NET [90] and HLPNN [50] incorporate the fixed network embeddings as features. In contrast, our proposed HUG can adaptively fine-tune both attention models to favor the geolocation prediction by the hybrid attention mechanism. (3) Compared with GCN [90], our attention-based approach can better understand the hierarchical language features and assign different importance to nodes of the same neighborhood. (4) We conduct the ablation study by removing the graph attention, language attention and location-based regularization one by one at a time. As the results on the Twitter-US dataset shown in Table 6.3, each module contributes to the performance improvement and the proposed HUG benefits from the combination and the hybrid attention mechanism.

Table 6.3: Ablation study on Twitter-US.

	Acc@161 ↑	Mean ↓	Median ↓
HUG	**72%**	**359**	**31**
w/o graph attention	51%	531	57
w/o language attention	58%	612	63
w/o location regularization	59%	562	51

We further investigate the interpretability of the proposed HUG. Figure 6.2 shows the text

and graph examples from the GeoText dataset. In (a) and (b), we show the social media posts of two users (A, B), whose hybrid attention weights for texts are $\alpha_v = 0.643$ and 0.794, respectively. The blue blocks denote the tweet-level attention weights. The orange denotes the word attention weights and our model can select the words with a strong indication of geolocations like *Louisville*, *LA*, *USC* and *West Hollywood*. Figure 6.2 (c) and (d) demonstrate two users (C, D) with the geolocations and attention weights of their one-hop neighbors. The hybrid attention weights for graph vector are $\alpha_u = 0.844$ and 0.942 for user C and D, respectively. We plot the geolocations of user C and D in red dots. The green dots are the geolocations of the one-hop neighbors and the dot sizes denote the attention weights. Our proposed HUG also works in terms of the graph attention and location-based regularization, by assigning the higher weights to closer neighbors and lower weights to farther neighbors.

6.1.4 Summary

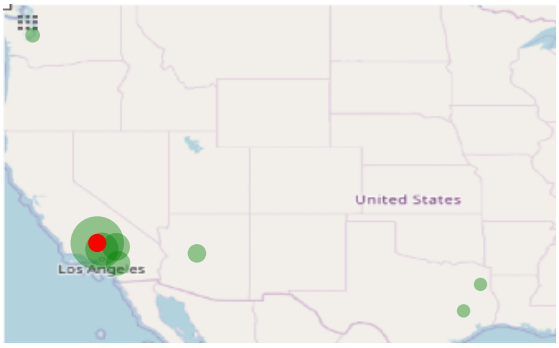
In this section, we propose a novel end-to-end framework, Hybrid-attentive User Geolocation (HUG), to jointly model the post texts and user interactions in social media and predict user geolocations. We introduce the hybrid attention mechanism to automatically determine the importance of texts and social networks while social media posts and interactions are modeled by a graph attention network and a language attention network. The experimental study on three benchmark geolocation datasets from Twitter shows that HUG consistently renders superior prediction performance against baseline approaches. We also demonstrate the interpretability of HUG with in-depth analysis of attention weights.

I just found out I'm doin 2 shows tonite in Louisville becuz the 1st show sold out thats
lol sorry I just saw your screen name and seen the area code! My bad for bothering u.
lol so why gon' be in louisville tonight that's gon' be funny man!
yeah well r spr break actually start the 13 but we dnt leave til the 17th so I'll
omg they got me weak everything they keep sayin and doin is sooo hilarious

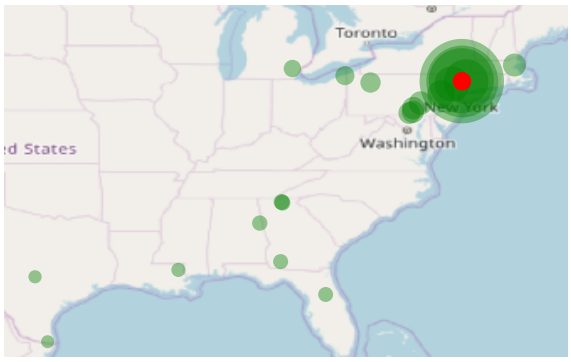
(a) User A

I can't wait to move bck to LA
thanks i did that one a few hours ago i havent had no sleep but there stil more
Why is everyone moving to culver city ugh I wanna move to west hollywood nxt omg 2mc
Live today like its the last don't worry about thee future n don't dwell in the past just
USC gon cost me $42,000 smhshyt there prices went down it use to be 55,000

(b) User B



(c) User C



(d) User D

Figure 6.2: Attention weight analysis.
(a)-(b) Documents of users A and B. (c)-(d) Geolocations and attention weights of users C and D with their one-hop neighbors.

6.2 On-demand Influencer Discovery on Social Media

6.2.1 Introduction

Social influence refers to the ability of a user to change the feelings, attitudes, or behaviors of other users within a network [18]. Detecting influencers has become an essential task in many fields such as online marketing and political campaigns [128, 160]. Many previous works have studied the task by utilizing the rich text and user interactions on social media [10, 159, 139].

It is well accepted that the influence distribution is not homogeneous over topics. An influencer on topics about politics may not have a huge impact on topics about HIV. Compared with social media celebrities, domain-specific influencers are less expensive to hire and more engaged in spreading messages such as new treatment or timely prevention for HIV. However, these important topics are rarely discussed on social media, which is demonstrated with an example from Twitter in Figure 6.3. We crawled 1% US English Tweets during 11/01/2019 - 12/31/2019 and counted the occurrence of hashtags, with the top 5 frequent hashtags in blue and 5 rare but important hashtags on health subjects in orange. Although topic-specific influencer detection algorithms [128, 10, 56, 8] have been proposed, none of them focuses on identifying influencers for rare yet important topics like HIV and suicide. While existing approaches trying to build filters with keywords [54], it is infeasible to enumerate all keywords related to the topic (like $\#PrEP$[1]).

Another key question in identifying topic-specific influencers is how to model the influence propagation. Traditional influencer detection models rely on topic modeling [128, 56] and probabilistic network diffusion [57, 139], which are not ideal for the sparse social network with rare topics. Meanwhile, the recent progress on Graph Neural Networks (GNNs) [64, 136, 150] demonstrates the effective representation learning in graph structures. With a proper design of neighborhood aggregation and objective function, GNNs are capable of modeling the

[1]Pre-exposure prophylaxis (PrEP) is a prevention of HIV/AIDS.

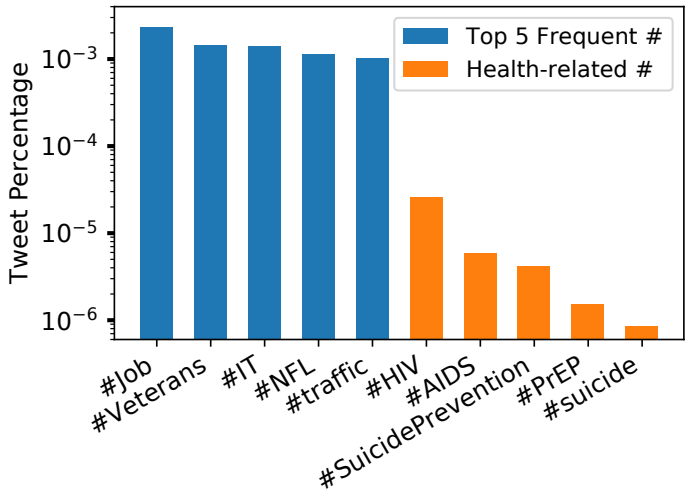


Figure 6.3: Hashtag distribution of 1% US English Tweets in 11/01/2019-12/31/2019. The top 5 frequent hashtags are shown along with 5 health-related hashtags.

influence propagation and identifying topic-specific influencers on social media.

To tackle the challenges of detecting on-demand topic-specific influencers, we propose a new computational framework named on-Demand Influencer Discovery (DID) model. We design an language attention network as a subject filter to iteratively select the social messages related to given keywords. To take full advantage of multiple interactions in social networks, we integrate them with trainable weight functions. The influence propagation is modeled by GNNs with a loss function considering neighborhood and topic concentration. Comprehensive experiments show our approach significantly outperforms all comparative baselines.

In the literature, influencer detection has attracted increasing attention in recent years. Some of the existing methods focus on modeling the influence in all topics [111, 139, 15], while several studies aim to identify topic-specific influencers [105, 128, 126]. Although some approaches [8, 10, 56] also integrate topic discovery and social influence analysis, the influencers are restricted to popular topics in social media only. To the best of our knowledge, this work is the first study that can reliably identify topic-specific influencers even on rare

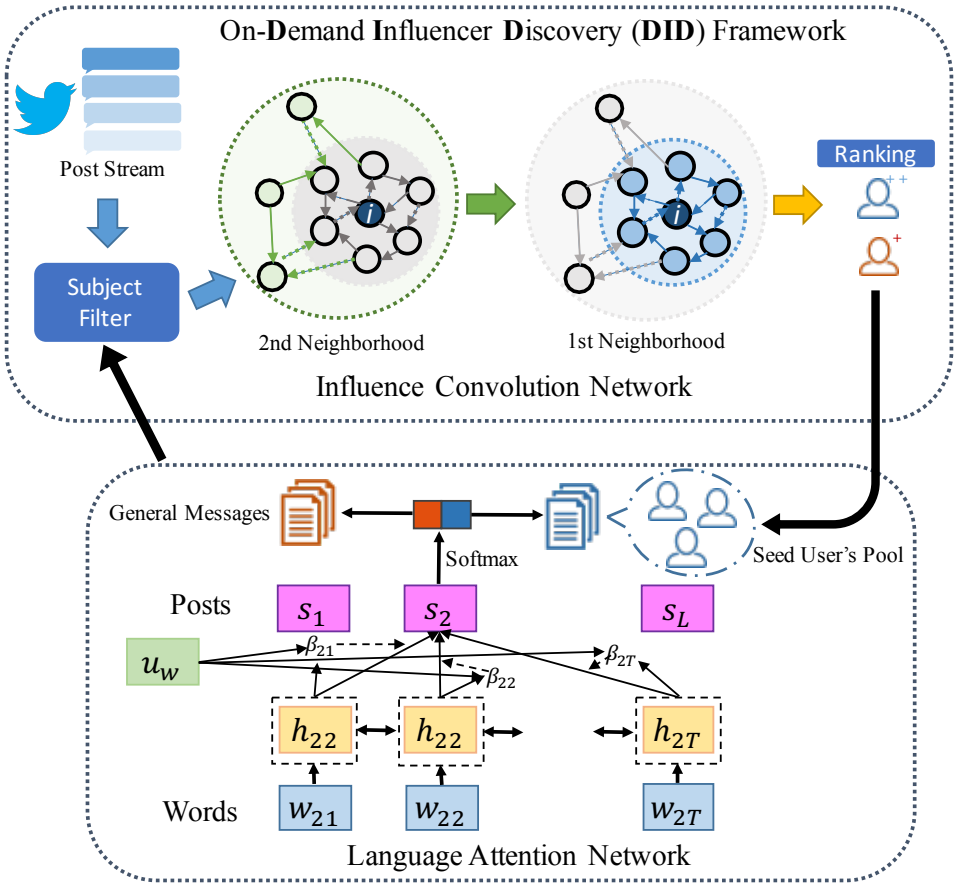


Figure 6.4: The overview of the proposed framework, DID

but important subjects.

6.2.2 on-Demand Influencer Discovery

In this section, we formally define the problem statement and then introduce our proposed approach, on-Demand Influencer Discovery (DID).

Problem Definition. Suppose we have a set of social media users U and their social network G. For each user $k \in U$, $W^k = \{w_{ij}^k\}$ denotes the social media posts published by the user, where w_{ij}^k represents the j-th word of the i-th post in W^k. The social network $G = (U, A)$ treats each user $k \in U$ as a node and models their relations with the adjacency

matrix $A \in \mathbb{R}^{N \times N}$ that indicates the relations between users. N is the total number of users ($|U| = N$). Given the social media posts of users U and the social network G, the goal of this work is to identify top-t influencers with the given keyword set K.

Framework Overview. Here we propose DID for the on-demand influencer detection as shown in Figure 6.4. We design the language attention network that can select subject-related social posts and get trained iteratively with minimum supervision. The influence convolution network simulates the influence propagation in the social network and outputs the topic-specific influence scores of users. Moreover, we integrate different types of social interactions with weight functions that are trained with influence learning.

6.2.2.1 Language Attention Network

We propose the language attention network as a subject filter to select social posts related to the given keyword set K. The language attention network is composed of several parts, including a word embedding layer, a post encoder, and a subject classifier.

Word Embedding Layer. We convert each word w_{ij} into a one-hot encoding representation $\tilde{w}_{ij}$ and embed the words to vectors e with an embedding matrix E, where $e_{ij} = E \cdot \tilde{w}_{ij}$.

Post Encoder. For each post of the user, we feed the word embeddings to a bidirectional Recurrent Neural Network (BiRNN) to learn a hidden state of each word with sequential information as:

$$\overleftarrow{h}_{ij} = \text{GRU}(\overleftarrow{h}_{i,j+1}, e_{ij}), \overrightarrow{h}_{ij} = \text{GRU}(\overrightarrow{h}_{i,j-1}, e_{ij}), h_{ij} = [\overleftarrow{h}_{ij}, \overrightarrow{h}_{ij}],$$

where $\text{GRU}(\cdot)$ is the recurrent neural unit of the BiRNN. Here we choose GRU instead of LSTM because of its computational efficiency. To derive the post representation, we introduce an attention layer to obtain a weighted sum of the hidden states from the BiRNN layer. To be specific, we initialize a context vector u_w and calculate the attention scores β_{ij}

for the words in the post as:

$$u_{ij} = \tanh(W_w \cdot h_{ij} + b_w), \beta_{ij} = \frac{exp(u_{ij}^T \cdot u_w)}{\sum_j exp(u_{ij}^T \cdot u_w)}, s_i = \sum_j \beta_{ij} \cdot h_{ij},$$

where W_w and b_w are the weight matrix and the bias to map each word into a hidden space for estimating importance. With the post encoder, the i-th social post is represented as s_i.

Subject Classifier. As a subject filter, the language attention network aims to select social posts related to the given keywords. We build a subject seed user pool to capture as much subject-related information in the social posts. Subject seed users are defined as users who are dedicated to the topic of given keywords. We initialize the pool by selecting the users that have the highest ratio of posts containing keyword set K. Furthermore, as shown by the right black arrow in Figure 6.4, we iteratively update the seed user pool and subject classifier. More seed users are incorporated into the pool according to the rules: (1) the user has a high ratio of subject-related posts; (2) the user is identified as an influencer. For example in HIV subject on Twitter, the seed user pool is initialized with @*iasociety* and @*CDC_HIVAIDS*, who have the highest 73.4% and 79.5% of tweets containing keyword *HIV*. As the training continues, we have 6 users in the final seed user pool. By collecting posts from seed users as positive samples and general posts as negative samples, we train the subject classifier and iteratively update the seed user pool. Based on the post encoding s_i, we use a fully-connected hidden layer to estimate the probability of being the positive sample of the i-th social post as

$$P = \text{Softmax}(W_c s_i),$$

where W_c is the weight matrix for the subject classifier. With the subject classifier, we obtain the subject-related posts and a vector $\mathbf{c} \in \mathbb{R}^N$ with $\mathbf{c}_i = \frac{\text{\# related posts of } i}{\text{\# total posts of } i}$, describing user i's concentration on topic K. Consequently, the filtered social network only includes interactions from the subject-related posts.

6.2.2.2 Influence Convolution Network

Interaction Integration. Typically, the social network includes multiple types of interactions, such as *follow*, *mention*, *retweet* and *quote* on Twitter. We assume there are L types of interactional graph and the corresponding adjacency matrices are $A^{(1)}, A^{(2)}, \ldots, A^{(L)}$. The majority of existing works only consider a single graph or assign equal weights to different interaction graphs [35]. Instead, we would like to integrate the multiple graphs in a sensible way by learning the weights of the integration. Specifically, we introduce L coefficients $\{\lambda_l, l = 1, 2, \ldots, L\}$ to represent the weights for L adjacency matrices $A^{(l)}$. The integrated adjacency matrix can be formulated as $A = \sum_{l=1}^{L} \lambda_l A^{(l)}$, with λ_l to be learned.

Influence Convolution. On social media, users disseminate posts via multiple rounds of social actions. Based on the closed world assumption [87], social actions are mostly influenced by their near neighbors within the network. As a result, we propose to model the influence propagation on social media with neighborhood aggregation technique from GNNs [64], as shown in Figure 6.4. The post embedding s aggregated by user is the input node representations $H^{(0)}$ to the first layer. The p-th influence convolution layer performs feature aggregation on node i's one-hop neighbors $\mathbb{N}_i$:

$$H^{(p)} = \sigma(\sum_{j \in \mathbb{N}_i} H_j^{(p-1)} W^{(p)})$$

where $H^{(p)} \in \mathbb{R}^{N \times d_n^{(p)}}$ is the output node representations, and $W^{(p)} \in \mathbb{R}^{d_n^{(p-1)} \times d_n^{(p)}}$ is the parameter matrix for this layer. The aggregated feature matrix H from the output of the final layer represents the user influence after propagation. We consider three criteria for the unsupervised objective function. First, users with a larger neighborhood should have a higher influence score. Here we consider the neighborhood within two hops. Second, the higher the concentration $\mathbf{c}$ in keyword topic K, the higher the influence score. Furthermore, we add the third term in the objective function to regularize the weight matrices of P influence convolution layers to prevent overfitting of the model.

$$\max L(W, \lambda_l) = \sum_{i,j=1}^{N} A_{ij}(1 + \sum_{k=1}^{N} A_{jk})H_i^2$$
$$+\zeta_1 \ \mathbf{c}^T H - \zeta_2 \sum_{i=1}^{P} \|W^{(i)}\|^2$$

6.2.3 Experiments

6.2.3.1 Experimental Setup

Datasets. We employ three datasets from Twitter. For the quantitative assessment, we create HIV and SUICIDE datasets. For HIV dataset, we select a sample of users who tweeted about HIV and crawl all their tweets. We label the HIV-related influencers as the users (1) who have more than 50% HIV-related tweets and (2) whose HIV-related tweets have the highest number of retweets in the whole Twitter network. We adopt the same strategy to create SUICIDE dataset focusing on the suicide-related topic. There are 20 and 15 topic-specific influencers in the two labeled datasets. The US-ENGLISH includes randomly sampled 1% US tweets in 11/01/2019 - 12/31/2019 and the hashtag distribution is shown in Figure 6.3. We intend to evaluate the scalability and efficiency with the unlabeled US-ENGLISH dataset. Dataset statistics are summarized in Table 6.4.

Table 6.4: Dataset statistics.

	HIV	**Suicide**	**US-English**
Users	556	639	4,145,234
Tweets	341,167	471,625	75,546,211
Interactions	81,626	89,776	3,848,428
Influencers	20	15	-

Table 6.5: Topic-specific influencer detection performance.

Methods	HIV				Suicide			
	F1	AUC	NDCG	MAP	F1	AUC	NDCG	MAP
TAP	0.415	0.687	0.341	0.612	0.535	0.734	0.423	0.732
TwitterRank	0.212	0.585	0.081	0.578	0.374	0.659	0.186	0.742
TS-SRW	0.159	0.750	0.062	0.500	0.275	0.824	0.106	0.611
LAN-ReF	0.602	0.819	0.415	0.695	0.434	0.727	0.231	0.751
LAN+RR-LT	0.683	0.793	0.500	0.877	0.731	0.863	0.558	0.930
LAN+CoupledGNN	0.719	0.870	0.529	0.910	0.642	0.829	0.683	0.956
DID	**0.882**	**0.912**	**0.771**	**0.970**	**0.859**	**0.936**	**0.739**	**0.996**

Table 6.6: Ablation study on HIV.

	F1	AUC	NDCG	MAP
DID	**0.882**	**0.912**	**0.771**	**0.970**
Non-iterative LAN	0.791	0.852	0.730	0.933
w/o interaction integration	0.823	0.862	0.749	0.942
w/o influence convolution	0.801	0.878	0.753	0.961

Baselines. We compare the proposed DID against six baselines that are trained based on the text and network features. The first three baselines are topic-specific influencer detection models: (1) **TAP** [105] a topical affinity propagation model built on a factor graph to identify the topic-specific social influence; (2) **TwitterRank** [128], an extension of PageRank algorithm with topic modeling; (3) **TS-SRW** [56], a topic-sensitive supervised random walk model. The following three baselines combine the language attention network (LAN) as a subject filter with general influencer detection models: (4) **LAN+ReF** [111], a statistical and analytical model based on influence topology; (5) **LAN+RR-LT** [139], a polling-based method with a sample of random reversely reachable sets to approximate user

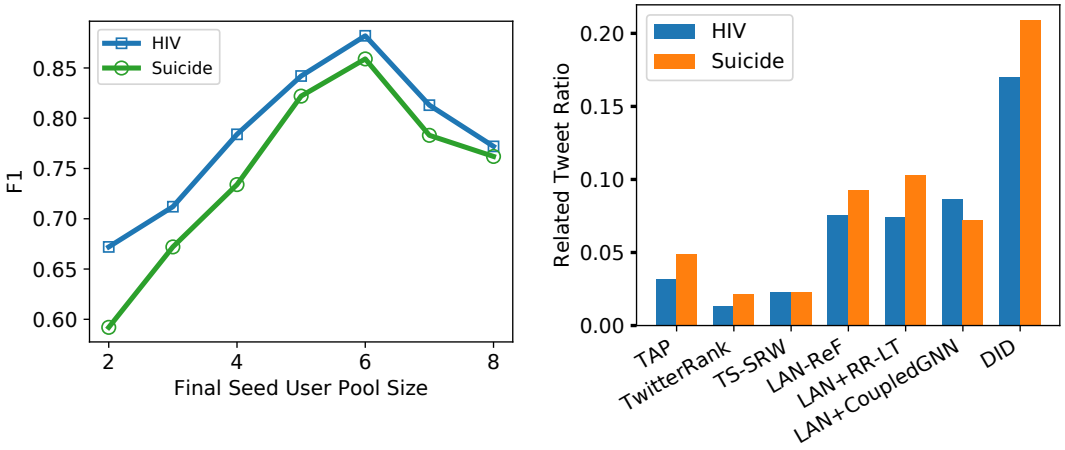
influence; (6) **LAN+CoupledGNN** [15], two coupled graph neural networks to iteratively model and predict user influence.

Evaluation. We evaluate our DID approach against the baselines with four commonly used metrics: (1) F1; (2) Area Under Curve (AUC); (3) Normalized Discounted Cumulative Gain (NDCG); (4) Mean Average Precision (MAP).

Implementation Details. The language attention network has bidirectional GRUs with hidden dimensions in $\{50, 100, 200\}$ and the word embeddings are initialized with the Glove vectors [85] pre-trained on the Twitter corpus. We initialize the pool with 2 seed users and the final size of seed user pool varies in [2, 8]. The influence convolution network is a two-layer network.

6.2.3.2 Experimental Results

Table 6.5 summarizes the model performance of detecting top-t topic-specific influencers, where t is the number of influencers in ground truth ($t = 20$ for HIV and $t = 15$ for SUICIDE). Overall, DID is able to outperform other baselines across the two datasets on all metrics. We make the following observations. (1) Compared with TAP [105], TwitterRank [128] and TS-SRW that directly identify influencers in rare topics from the massive social messages, DID outperforms by iteratively learning an accurate subject filter. (2) As combination models, Filter+ReF [111], Filter+RR-LT [139] and Filter+CoupledGNN [15] train the influencer detection model with fixed social message filters. In contrast, our proposed DID can adaptively update the subject filter for better influencer detection performance. (3) We conduct the ablation study of DID by training a non-iterative LAN, removing interaction integration, and influence convolution one by one at a time. As the results on the HIV dataset shown in Table 6.6, each module contributes to the performance improvement, and the proposed DID benefits from iterative LAN, interaction integration and influence convolution. (4) We also investigate how seed user pool size impacts the detection accuracy in DID, shown in Figure 6.5(a). Over the two datasets, we observe a common phenomenon: there exists an optimal



(a) User Pool Size

(b) Specificity

Figure 6.5: Seed user pool size and specificity impact.
(a) F1 vs. seed user pool final size. (b) The specificity evaluation.

size that delivers the best F1 score.

To confirm that the identified users indeed focus on the target rare topics, we conduct an experiment on their followers' tweets. With the output influencer set of each model, we randomly select 100 followers for each influencer and crawl follower's most recent 100 tweets. Figure 6.5(b) shows the percentage of subject-related tweets to all tweets. The proposed DID model has the highest ratio of related tweets in the two datasets, which indicates these influencers can deliver specific influence to their followers.

To evaluate the scalability of the proposed DID, we show the top 10 topic-specific influencers on large scale US-English dataset in Table 6.7. Account names in bold are topic-specific influencers in HIV or Suicide topic. The results show that DID is capable of detecting these topic-specific influencers out of 3.8 million users in the US-English dataset.

Table 6.7: Influencers detected in US-English dataset.

Topic	Output of DID model
HIV	**CDC_HIVAIDS**, **talkHIV**, **PEPFAR**, **iasociety**, **blackaids**, WHO, **TheBodyDotCom**, **HIV_Insight**, **GlobalFund**, **GreaterThanAIDS**
Suicide	**MentallyAwareNG**,**afspnational**,**samaritans**,**80-0273talk**,**Spreading_L0ve**,**cspyyc**,**papyrus_tweets** CarlDunnJr, **depressionnote**, **CharitySANE**

6.2.4 Summary

This section investigates the problem of detecting on-demand topic-specific influencers. We introduce a computational framework, on-Demand Influencer Discovery model, based on the language attention network and influence convolution network. Comprehensive evaluations are conducted with three Twitter datasets and the results show promising performance of the proposed model against comparative baselines.

CHAPTER 7

Conclusion

7.1 Conclusion

In this dissertation, we investigate the deep representation learning on complex graphs. More specifically, we tackle the three issues of complex graph representation learning with the proposed frameworks, including learning network embedding with adversarial regularization, robust graph representation via neural sparsification, and temporal graph modeling with temporal-structural convolution. We also demonstrate the applications in social network analysis problems.

For the sparse sampling issue in random-walks, we proposed NetRA, a deep network embedding model for encoding each vertex in a network as a low-dimensional vector representation with adversarially regularized autoencoders. Our model demonstrated the ability of generative adversarial training process in extracting informative representations. The proposed model has better generalization capability, without requiring an explicit prior density distribution for the latent representations. Specifically, we leveraged LSTM autoencoders that take the sampled sequences of vertices as input to learn smooth vertex representations regularized by locality-preserving constraint and generative adversarial training process. The resultant representations are robust to the sparse vertex sequences sampled from the network. Empirically, we evaluated the learned representations with a variety of network datasets on different tasks such as network reconstruction, link prediction, and multi-label classification. The experimental results showed substantial improvement over the state-of-the-art network embedding competitors.

We propose Neural Sparsification (NeuralSparse) to address the noise brought by the task-irrelevant information on real-life large graphs. NeuralSparse consists of two major

components: (1) The sparsification network sparsifies input graphs by sampling edges following a learned distribution; (2) GNNs take sparsified subgraphs as input and extract node representations for downstream tasks. The two components in NeuralSparse can be jointly trained with supervised loss, gradient descent, and backpropagation techniques. The experimental study on real-life datasets shows that the NeuralSparse consistently renders more robust graph representations, and yields up to 7.2% improvement in accuracy over state-of-the-art GNN models.

To simultaneously extract features from the temporal-structural neighborhood in temporal graphs, we propose Temporal Structural Network (TSNet) for node classification in temporal graphs. TSNet also leverages our proposed supervised graph sparsification techniques and extends the neighborhood to a broader temporal-structural neighborhood on temporal graphs. The model consists of two major sub-networks: (1) the sparsification network sparsifies input temporal graphs by sorting and sampling edges following a learned distribution; (2) the temporal-structural convolutional network performs convolution on the sparsified graphs to extract local features from the joint temporal-structural space. In the experimental study, TSNet demonstrates superior performance over four categories of baseline models on public and private benchmark datasets. The qualitative case study suggests a promising direction for the interpretability of temporal graph learning.

For social media user geolocation problem, we propose a novel end-to-end framework, Hybrid-attentive User Geolocation (HUG), to jointly model the post texts and user interactions in social media and predict user geolocations. We introduce the hybrid attention mechanism to automatically determine the importance of texts and social networks while social media posts and interactions are modeled by a graph attention network and a language attention network. The experimental study shows that HUG consistently renders superior prediction performance and better interpretability. For topic-specific influencer detection problem, we introduce a computational framework, on-Demand Influencer Discovery model, based on the language attention network and influence convolution network. Comprehen-

sive evaluations are conducted with three Twitter datasets and the results show promising performance of the proposed model against comparative baselines.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning.. In *OSDI*.

[2] Bijaya Adhikari, Yao Zhang, Sorour E Amiri, Aditya Bharadwaj, and B Aditya Prakash. 2018. Propagation-Based Temporal Network Summarization. *TKDE* (2018).

[3] Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. 2013. Spectral sparsification in dynamic graph streams. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques* (2013).

[4] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* (2015).

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *ICML*. 214–223.

[6] Lars Backstrom, Eric Sun, and Cameron Marlow. 2010. Find me if you can: improving geographical prediction with social and spatial proximity. In *WWW*.

[7] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[8] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. 2012. Topic-Aware Social Influence Propagation Models. In *ICDM*.

[9] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. 2011. Node classification in social networks. In *Social network data analytics*. Springer, 115–148.

[10] Bin Bi, Yuanyuan Tian, Yannis Sismanis, Andrey Balmin, and Junghoo Cho. 2014. Scalable topic-specific influence analysis on microblogs. In *WSDM*.

[11] Peter Borg and Kurt Fenech. 2017. Reducing the maximum degree of a graph by deleting vertices. *Australasian Journal Of Combinatorics* 69, 1 (2017), 29–40.

[12] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, et al. 2007. The BioGRID interaction database: 2008 update. *Nucleic acids research* 36, suppl_1 (2007), D637–D640.

[13] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.

[14] Daniele Calandriello, Ioannis Koutis, Alessandro Lazaric, and Michal Valko. 2018. Improved large-scale graph learning through ridge spectral sparsification. In *ICML*.

[15] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. 2020. Popularity Prediction on Social Platforms with Coupled Graph Neural Networks. In *WSDM*.

[16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.

[17] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2016. Deep Neural Networks for Learning Graph Representations.. In *AAAI*.

[18] Meeyoung Cha, Hamed Haddadi, Fabrício Benevenuto, and P. Krishna Gummadi. 2010. Measuring User Influence in Twitter: The Million Follower Fallacy. In *ICWSM*.

[19] Alireza Chakeri, Hamidreza Farhidzadeh, and Lawrence O Hall. 2016. Spectral sparsification in spectral clustering. In *ICPR*.

[20] Shiyu Chang, Wei Han, Jiliang Tang, Guo-Jun Qi, Charu C Aggarwal, and Thomas S Huang. 2015. Heterogeneous network embedding via deep architectures. In *KDD*.

[21] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, and Yoshua Bengio. 2017. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv preprint arXiv:1702.07983* (2017).

[22] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. In *ICLR*.

[23] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. 2015. One trillion edges: Graph processing at facebook-scale. *VLDB* (2015).

[24] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*.

[25] Quanyu Dai, Qiang Li, Jian Tang, and Dan Wang. 2017. Adversarial Network Embedding. *arXiv preprint arXiv:1711.07838* (2017).

[26] Clodoveu A Davis Jr, Gisele L Pappa, Diogo Rennó Rocha de Oliveira, and Filipe de L. Arcanjo. 2011. Inferring the location of twitter messages based on user relationships. *Transactions in GIS* (2011).

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*.

[28] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[29] Tien Huu Do, Duc Minh Nguyen, Evaggelia Tsiligianni, Bruno Cornelis, and Nikos Deligiannis. 2017. Multiview deep learning for predicting twitter users' location. *arXiv preprint arXiv:1712.08091* (2017).

[30] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.

[31] Talya Eden, Shweta Jain, Ali Pinar, Dana Ron, and C. Seshadhri. 2018. Provable and Practical Approximations for the Degree Distribution Using Sublinear Graph Samples. In *WWW*.

[32] Jacob Eisenstein, Brendan O'Connor, Noah A Smith, and Eric P Xing. 2010. A latent variable model for geographic lexical variation. In *EMNLP*.

[33] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *JMLR* 9, Aug (2008), 1871–1874.

[34] Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. 2019. Learning Discrete Structures for Graph Neural Networks. In *ICML*.

[35] Kiran Garimella, Ingmar Weber, and Munmun De Choudhury. 2016. Quote rts on twitter: usage of the new feature for political discourse. In *WebSci*.

[36] Elahe Ghalebi, Baharan Mirzasoleiman, Radu Grosu, and Jure Leskovec. 2018. Dynamic Network Model from Partial Observations. In *NIPS*.

[37] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*.

[38] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[39] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.

[40] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[41] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[42] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. 2019. Stochastic optimization of sorting networks via continuous relaxations. In *ICLR*.

[43] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028* (2017).

[44] Ehsan Hajiramezanali, Arman Hasanzadeh, Krishna Narayanan, Nick Duffield, Mingyuan Zhou, and Xiaoning Qian. 2019. Variational graph recurrent neural networks. In *Advances in neural information processing systems*. 10701–10711.

[45] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*.

[46] William L Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* (2017).

[47] Bo Han, Paul Cook, and Timothy Baldwin. 2012. Geolocation prediction in social media data by finding location indicative words. In *COLING*.

[48] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[49] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. 2004. Spin: mining maximal frequent subgraphs from graph databases. In *KDD*. ACM, 581–586.

[50] Binxuan Huang and Kathleen M Carley. 2019. A Hierarchical Location Prediction Neural Network for Twitter User Geolocation. In *EMNLP*.

[51] Jin Huang and Charles X Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. *TKDE* (2005).

[52] Christian Hübler, Hans-Peter Kriegel, Karsten Borgwardt, and Zoubin Ghahramani. 2008. Metropolis algorithms for representative subgraph sampling. In *ICDM*.

[53] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with gumbel-softmax. In *ICLR*.

[54] Jyun-Yu Jiang, Xue Sun, Wei Wang, and Sean Young. 2019. Enhancing Air Quality Prediction with Social Media and Natural Language Processing. In *ACL*.

[55] David Jurgens. 2013. That's what friends are for: Inferring location in online social media platforms based on social relationships. In *ICWSM*.

[56] Georgios Katsimpras, Dimitrios Vogiatzis, and Georgios Paliouras. 2015. Determining influential users with supervised random walks. In *WWW*.

[57] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *KDD*.

[58] Junghwan Kim, Haekyu Park, Ji-Eun Lee, and U Kang. 2018. Side: representation learning in signed directed networks. In *WWW*.

[59] Yoon Kim, Kelly Zhang, Alexander M Rush, Yann LeCun, et al. 2017. Adversarially Regularized Autoencoders for Generating Discrete Structures. *arXiv preprint arXiv:1706.04223* (2017).

[60] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[61] Sheila Kinsella, Vanessa Murdock, and Neil O'Hare. 2011. " I'm eating a sandwich in Glasgow" modeling locations with tweets. In *SMUC*.

[62] Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. 2018. Neural relational inference for interacting systems. *arXiv preprint arXiv:1802.04687* (2018).

[63] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[64] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[65] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2018. Learning Dynamic Embeddings from Temporal Interactions. *arXiv preprint arXiv:1812.02289* (2018).

[66] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. 2018. Graph classification using structural attention. In *KDD*.

[67] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In *KDD*.

[68] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*.

[69] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* 1, 1 (2007), 2–es.

[70] Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics* 3 (2015), 211–225.

[71] Rong-Hua Li, Jeffrey Xu Yu, Lu Qin, Rui Mao, and Tan Jin. 2015. On random walk based graph sampling. In *ICDE*.

[72] Xiaoyi Li, Nan Du, Hui Li, Kang Li, Jing Gao, and Aidong Zhang. 2014. A deep learning approach to link prediction in dynamic networks. In *SDM*.

[73] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated Graph Sequence Neural Networks. In *ICLR*.

[74] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *ICLR*.

[75] Yike Liu, Tara Safavi, Abhilash Dighe, and Danai Koutra. 2018. Graph Summarization Methods and Applications: A Survey. *Comput. Surveys* (2018).

[76] Andreas Loukas and Pierre Vandergheynst. 2018. Spectrally approximating large graphs with smaller graphs. In *ICML*.

[77] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The concrete distribution: A continuous relaxation of discrete random variables. In *ICLR*.

[78] Arun S Maiya and Tanya Y Berger-Wolf. 2010. Sampling community structure. In *WWW*.

[79] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. 2016. Adversarial Autoencoders. In *ICLR*.

[80] Michael Mathioudakis, Francesco Bonchi, Carlos Castillo, Aristides Gionis, and Antti Ukkonen. 2011. Sparsification of influence networks. In *KDD*.

[81] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*. 3111–3119.

[82] Paul Milgrom and Ilya Segal. 2002. Envelope theorems for arbitrary choice sets. *Econometrica* 70, 2 (2002), 583–601.

[83] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning convolutional neural networks for graphs. In *ICML*.

[84] Tore Opsahl and Pietro Panzarasa. 2009. Clustering in weighted networks. *Social networks* (2009), 155–163.

[85] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

[86] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.

[87] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. 2018. DeepInf: Social Influence Prediction with Deep Learning. In *KDD*.

[88] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[89] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2017. A Neural Model for User Geolocation and Lexical Dialectology. In *ACL*.

[90] Afshin Rahimi, Trevor Cohn, and Timothy Baldwin. 2018. Semi-supervised User Geolocation via Graph Convolutional Networks. In *ACL*.

[91] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. 2017. Adversarial Generation of Natural Language. *arXiv preprint arXiv:1705.10929* (2017).

[92] Bruno Ribeiro and Don Towsley. 2010. Estimating and sampling graphs with multidimensional random walks. In *IMC*.

[93] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning Node Representations from Structural Identity. In *KDD*.

[94] Leonardo F.R. Ribeiro, Pedro H.P. Saverese, and Daniel R. Figueiredo. 2017. Struc2Vec: Learning Node Representations from Structural Identity. In *KDD* (Halifax, NS, Canada). ACM, 385–394.

[95] Stephen Roller, Michael Speriosu, Sarat Rallapalli, Benjamin Wing, and Jason Baldridge. 2012. Supervised text-based geolocation using language models on an adaptive grid. In *EMNLP*.

[96] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. 2020. DropEdge: Towards Deep Graph Convolutional Networks on Node Classification. In *ICLR*.

[97] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. 2013. Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 667–676.

[98] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[99] Veeru Sadhanala, Yu-Xiang Wang, and Ryan Tibshirani. 2016. Graph sparsification approaches for laplacian smoothing. In *AISTATS*.

[100] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*. Springer, 362–373.

[101] Martin Simonovsky and Nikos Komodakis. 2017. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *CVPR*.

[102] Daniel A Spielman and Nikhil Srivastava. 2011. Graph sparsification by effective resistances. *SIAM J. Comput.* (2011).

[103] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*. 3104–3112.

[104] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. WWW, 1067–1077.

[105] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. 2009. Social influence analysis in large-scale networks. In *KDD*.

[106] Lei Tang and Huan Liu. 2009. Relational learning via latent social dimensions. In *KDD*. ACM, 817–826.

[107] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23, 3 (2011), 447–478.

[108] Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. 2008. Community evolution in dynamic multi-mode networks. In *KDD*.

[109] Athanasios Theocharidis, Stjin Van Dongen, Anton J Enright, and Tom C Freeman. 2009. Network visualization and analysis of gene expression data using BioLayout Express3D. *Nature protocols* 4, 10 (2009), 1535–1550.

[110] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. 2014. Learning Deep Representations for Graph Clustering.. In *AAAI*.

[111] Ramine Tinati, Leslie Carr, Wendy Hall, and Jonny Bentwood. 2012. Identifying communicator roles in twitter. In *WWW*.

[112] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. 2017. Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558* (2017).

[113] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. Association for Computational Linguistics, 173–180.

[114] Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. 2017. Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. *arXiv preprint arXiv:1705.05742* (2017).

[115] Tomasz Tylenda, Ralitsa Angelova, and Srikanta Bedathur. 2009. Towards time-aware link prediction in evolving social networks. In *Proceedings of the 3rd workshop on social network mining and analysis*. ACM, 9.

[116] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *JMLR* 9 (2008), 2579–2605.

[117] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[118] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. 2019. Deep graph infomax. In *ICLR*.

[119] Cédric Villani. 2008. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media.

[120] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11, Apr (2010), 1201–1242.

[121] Elli Voudigari, Nikos Salamanos, Theodore Papageorgiou, and Emmanuel J Yannakoudakis. 2016. Rank degree: An efficient algorithm for graph sampling. In *ASONAM*.

[122] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *KDD*.

[123] Fengjiao Wang, Chun-Ta Lu, Yongzhi Qu, and S Yu Philip. 2017. Collective geographical embedding for geolocating social network users. In *PAKDD*.

[124] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. 2018. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. *AAAI* (2018).

[125] Lu Wang, Wenchao Yu, Wei Wang, Wei Cheng, Wei Zhang, Hongyuan Zha, Xiaofeng He, and Haifeng Chen. 2019. Learning Robust Representations with Graph Denoising Policy Network. In *ICDM*.

[126] Wei Wei, Gao Cong, Chunyan Miao, Feida Zhu, and Guohui Li. 2016. Learning to find topic experts in twitter via different relations. *TKDE* (2016).

[127] Jianshu Weng and Bu-Sung Lee. 2011. Event detection in twitter. In *ICWSM*.

[128] Jianshu Weng, Ee-Peng Lim, Jing Jiang, and Qi He. 2010. Twitterrank: finding topic-sensitive influential twitterers. In *WSDM*.

[129] J. Weston, F. Ratle, and R. Collobert. 2008. Deep learning via semi-supervised embedding. In *ICML*.

[130] Benjamin P Wing and Jason Baldridge. 2011. Simple supervised document geolocation with geodesic grids. In *ACL*.

[131] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* (1987).

[132] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[133] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, and Chengqi Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*.

[134] Louis-Pascal AC Xhonneux, Meng Qu, and Jian Tang. 2019. Continuous Graph Neural Networks. *arXiv preprint arXiv:1912.00967* (2019).

[135] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Xiao Liu, and Xiang Zhang. 2019. Spatio-temporal attentive rnn for node classification in temporal attributed graphs. In *IJCAI*.

[136] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks? *ICLR* (2019).

[137] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *ICML*.

[138] Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *AAAI*.

[139] Yu Yang, Zhefeng Wang, Jian Pei, and Enhong Chen. 2017. Tracking Influential Individuals in Dynamic Networks. *TKDE* (2017).

[140] Zhilin Yang, William W Cohen, and Ruslan Salakhutdinov. 2016. Revisiting semi-supervised learning with graph embeddings. In *ICML*.

[141] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *NAACL*.

[142] Rex Ying, Dylan Bourgeois, Jiaxuan You, Marinka Zitnik, and Jure Leskovec. 2019. GNN Explainer: A Tool for Post-hoc Explanation of Graph Neural Networks. In *NIPS*.

[143] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD*.

[144] Jiaxuan You, Bowen Liu, Zhitao Ying, Vijay Pande, and Jure Leskovec. 2018. Graph convolutional policy network for goal-directed molecular graph generation. In *Advances in neural information processing systems*. 6410–6421.

[145] Jiaxuan You, Rex Ying, and Jure Leskovec. 2019. Position-aware graph neural networks. *arXiv preprint arXiv:1906.04817* (2019).

[146] Bing Yu, Haoteng Yin, and Zhanxing Zhu. 2018. Spatio-Temporal Graph Convolutional Networks: a Deep Learning Framework for Traffic Forecasting. In *IJCAI*.

[147] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. In *AAAI*. 2852–2858.

[148] Wenchao Yu, Charu C Aggarwal, and Wei Wang. 2017. Temporally factorized network modeling for evolutionary network analysis. In *WSDM*.

[149] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2013. Embedding with autoencoder regularization. In *ECMLPKDD*. Springer, 208–223.

[150] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. 2018. Learning Deep Network Representations with Adversarially Regularized Autoencoders. In *KDD*.

[151] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. In *Advances in Neural Information Processing Systems*. 11983–11993.

[152] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *ICLR*.

[153] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2016. Homophily, Structure, and Content Augmented Network Representation Learning. In *ICDM*. IEEE, 609–618.

[154] L-C Zhang and M Patone. 2017. Graph sampling. *Metron* (2017).

[155] Xinyi Zhang, Shiliang Tang, Yun Zhao, Gang Wang, Haitao Zheng, and Ben Y Zhao. 2017. Cold hard E-cash: Friends and vendors in the Venmo digital payments system. In *ICWSM*.

[156] Yizhou Zhang, Yun Xiong, Xiangnan Kong, Shanshan Li, Jinhong Mi, and Yangyong Zhu. 2018. Deep Collective Classification in Heterogeneous Information Networks. In *WWW*.

[157] Yutao Zhang, Fanjin Zhang, Peiran Yao, and Jie Tang. 2018. Name Disambiguation in AMiner: Clustering, Maintenance, and Human in the Loop.. In *KDD*.

[158] Peixiang Zhao. 2015. gSparsify: Graph Motif Based Sparsification for Graph Clustering. In *CIKM*.

[159] Cheng Zheng, Jyun-Yu Jiang, Yichao Zhou, Sean D Young, and Wei Wang. 2020. Social Media User Geolocation via Hybrid Attention. In *SIGIR*.

[160] Cheng Zheng, Qin Zhang, Guodong Long, Chengqi Zhang, Sean D Young, and Wei Wang. 2020. Measuring Time-Sensitive and Topic-Specific Influence in Social Networks with LSTM and Self-Attention. *IEEE Access* (2020).

[161] Cheng Zheng, Qin Zhang, Sean D Young, and Wei Wang. 2020. On-demand Influencer Discovery on Social Media. In *CIKM*.

[162] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Node Classification in Temporal Graphs through Stochastic Sparsification and Temporal Structural Convolution. In *ECML-PKDD*.

[163] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, Jingchao Ni, Wenchao Yu, Haifeng Chen, and Wei Wang. 2020. Robust Graph Representation Learning via Neural Sparsification. In *ICML*.

[164] Chang Zhou, Yuqiong Liu, Xiaofei Liu, Zhongyi Liu, and Jun Gao. 2017. Scalable Graph Embedding for Asymmetric Proximity.. In *AAAI*. 2942–2948.

[165] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph neural networks: A review of methods and applications. *arXiv:1812.08434* (2018).

[166] Le-kui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process.. In *AAAI*.

[167] Chenyi Zhuang and Qiang Ma. 2018. Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. In *WWW*.